Kuwait University
College of Engineering and Petroleum

جامعة الكويت
KUWAIT UNIVERSITY

ME319 MECHATRONICS

Part I: The Brains – Microcontrollers, Software and Digital Logic

Lecture 5: General Purpose Input Output Peripheral

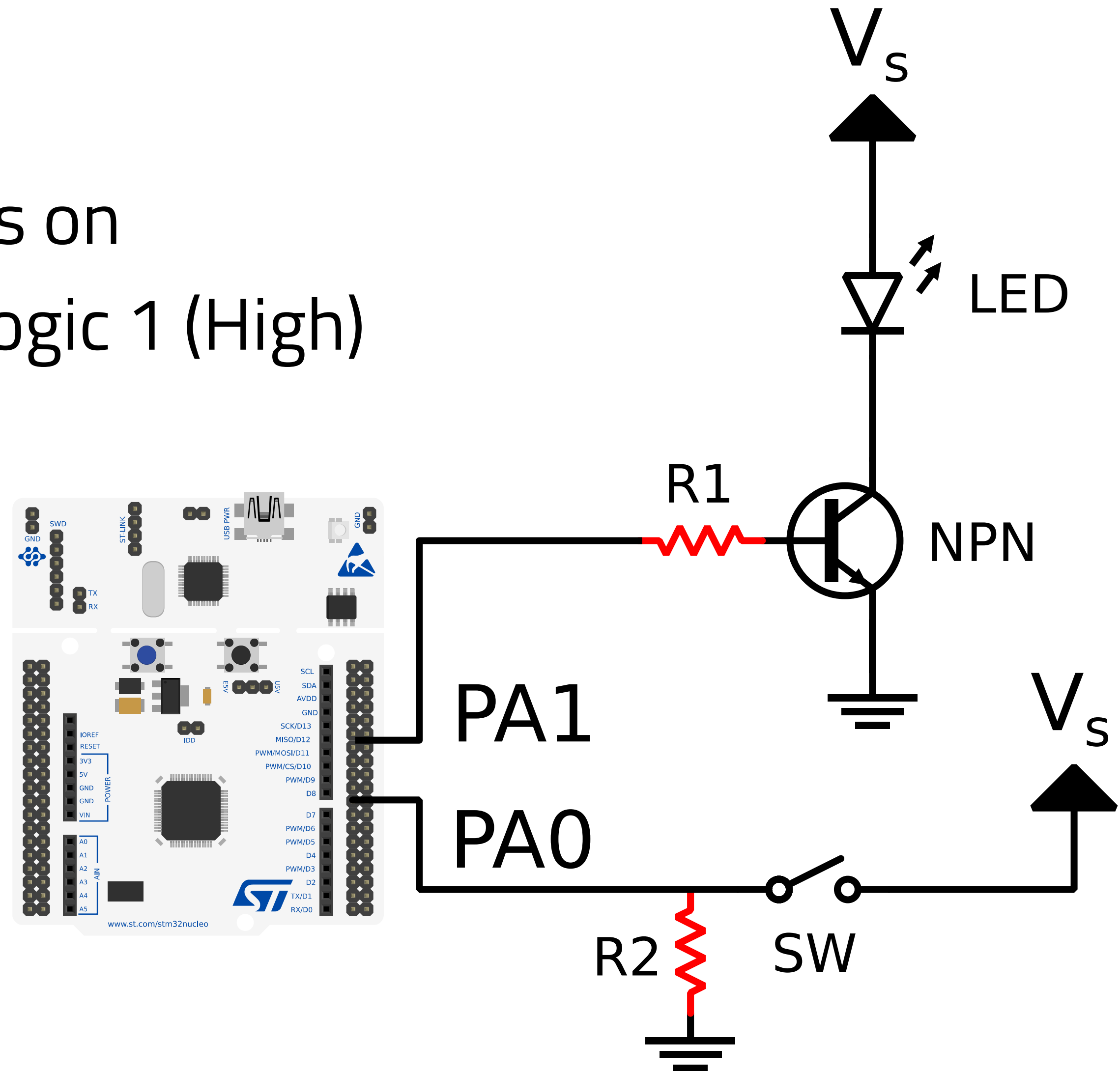
Spring 2021

Ali AlSaibie

- Objectives:
  - Review the basic components of the GPIO Peripheral
  - Become familiar with the microcontroller reference manual
  - Walk through a blinky routine at different abstraction levels

- Basic way to interface MCU with outside world
- Direction: Input or Output
- Data (Value): Logic High or Low – Written (Output). Read (Input)
- GPIO Pins belong to Ports, On STM32F401x there are up to 16 pins per port
- STM32F401x
    - Most Pins are GPIO by default (on reset)
        - Some Pins are set for special functions on reset (JTAG)
    - 5V-Tolerant
    - 15 GPIO Blocks. Ports A - Port Q [No Port I or Port O]
    - Internal weak pull-up or pull-down resisters

- PA1 and PA0 are both configured as GPIO Pins
  - PA1 reads: *Port A Pin 1*
- PA1 Set as output and PA0 as Input
- When PA1 is High (Logic 1): LED turns on
- When Switch is pressed: PA0 reads logic 1 (High)
- The GPIO Peripheral usually has:
  - Multiple ports, which have:
    - Multiple pins



Alternate Functions

- GPIO Pins also refer to programmable pins in general (non-fixed func pins: GND, 5V, etc)
- GPIO pins can be configured for alternative functions such as:
  - UART,I2C,ADC,DAC etc.
  - Table 9 in the datasheet lists the alternate functions each pin can have
  - A pin can serve only one function at a time.
- E.g. Pins PA0 and PA1 can have one of 4 alternative digital functions
  - Timer2 Ch 1, Timer 5 Ch 2, USART Clear To Send, or Event Out (Interrupt Pin)

Table 9. Alternate function mapping

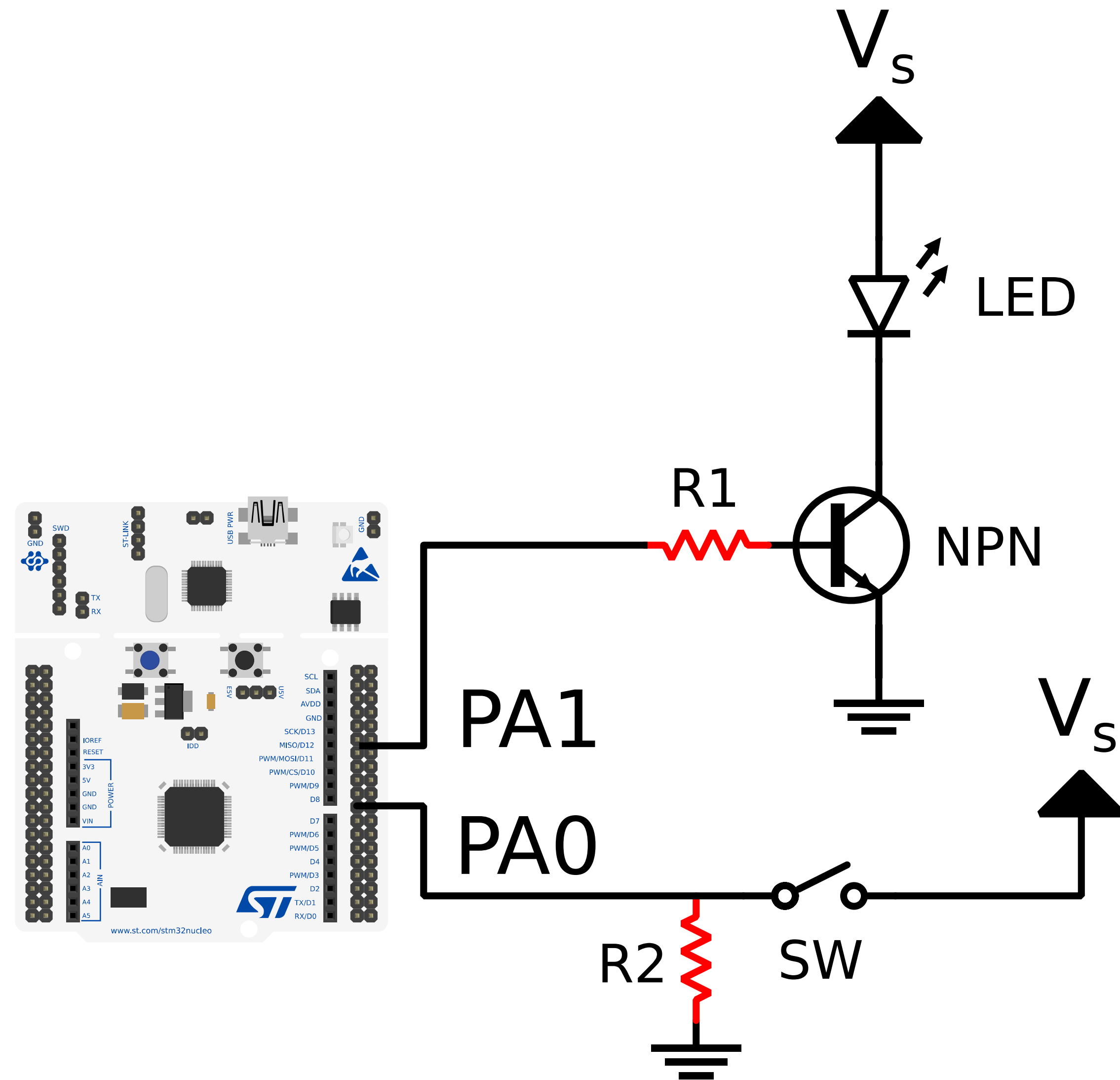
Port	AF00	AF01	AF02	AF03	AF04	AF05	AF06	AF07	AF08	AF09	AF10	AF11	AF12	AF13	AF14	AF15
	SYS_AF	TIM1/TIM2	TIM3/ TIM4/ TIM5	TIM9/ TIM10/ TIM11	I2C1/I2C2/ I2C3	SPI1/SPI2/ I2S2/SPI3/ I2S3/SPI4	SPI2/I2S2/ SPI3/ I2S3	SPI3/I2S3/ USART1/ USART2	USART6	I2C2/ I2C3	OTG1_FS		SDIO			
PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENT OUT
PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	-	-	-	-	-	-	-	EVENT OUT

- Pins can drive low current external devices, such as LEDs or other integrated circuits

Table 12. Current characteristics

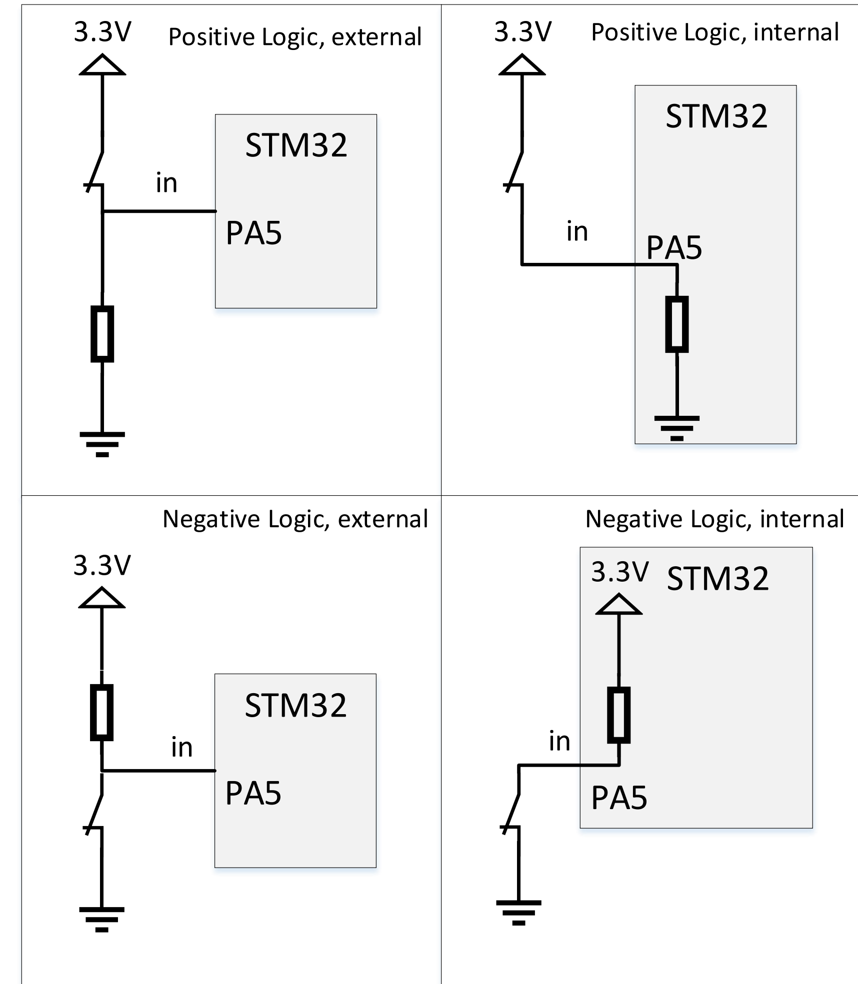
Symbol	Ratings	Max.	Unit
ΣI_{VDD}	Total current into sum of all V_{DD_x} power lines (source)[1]	160	mA
ΣI_{VSS}	Total current out of sum of all V_{SS_x} ground lines (sink)[1]	-160	
I_{VDD}	Maximum current into each V_{DD_x} power line (source)[1]	100	
I_{VSS}	Maximum current out of each V_{SS_x} ground line (sink)[1]	-100	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current sourced by any I/O and control pin	-25	
ΣI_{IO}	Total output current sunk by sum of all I/O and control pins [2]	120	
	Total output current sourced by sum of all I/Os and control pins[2]	-120	

- To drive high current devices such as a light bulb, an external driver is required. Pin acts as signal trigger
- E.g. Use a transistor to light LED
  - Drive the transistor via pin



Pull Up and Pull Down Resistors

- To ensure deterministic binary logic, a pull up or pull down resistor is used.
- Positive Logic: Pin normally connected to ground through *pull-down* resistor, so pin reads low. When source is connected, pin reads high.
- Negative Logic: Pin normally connected to source through *pull-up* resistor, connecting to ground sets pin low.
- Weak:- high resistance (weak current drain)
- The resistor guarantees the logic is inverted when the source is not connected, and that the value is not "floating"



MCU Registers: An Overview

- Program code interacts with hardware through changing bits inside registers
- A register is a memory location inside the microcontroller
- Interface with a microcontroller is done through registers, whether reading or writing to them. Registers can be Read-Only, or Read-Write
- The STM32F401RE is a 32bit microcontroller, and so each register is technically composed of 32 bit-fields.
- A typical register address looks like

0x4002 0000

- The above happens to be the base address for GPIO Port A
- GPIO registers are listed by their offset. The same offset is applied over whichever GPIO Port base address

Example Register

- Here is a the GPIO Mode register (REF02_STM32 Reference Manual)
- Common to all GPIO Ports (every GPIO Port has a mode register)

Example: we write 01 in bits [11:10] if we want to set Pin 5 as an output pin

This is basically what the Arduino function:

pinMode(PA5, OUTPUT);

does

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

- On STM32 MCUs, a base address is given and the list of offsets
- Given GPIO Port A base address: 0x4002 0000
- And the GPIOs mode registers (GPIOx_MODER) address offset is 0x00, and the output data register (GPIOx_ODR) address offset is 0x14

Then:

- GPIO mode register address for Port A (GPIOA_MODER) is

  0x4002 0000 (0x4002 0000 + 0x00)

- GPIO output type register address for Port A (GPIOA_ODR) is
  0x4002 0014 (0x4002 0000 + 0x14)
- If the base address of GPIO Port B is: 0x40020400,
  - What is the GPIO**B**_ODR address?

Blinky Example on STM32Nucleo

- To execute a basic blinky routine, we need to do the following on STM32F401RE. The LED is connected to PA5: Port A Pin 5
    1. Enable the GPIO Port A clock (See RCC Register)
    2. Set the GPIO Port A Pin 5 is output (See GPIOA_MODER Register)
    3. Set the GPIO Port A Pin 5 output to 1 (High) to turn LED On, or set it to 0 (Low) to switch it off
    4. Have some delay routine in between the Ons and Offs

Let’s see the relevant registers and see how we can execute a blinky code.

The concepts learned will extend to advanced peripherals.

RCC Register

- RCC: Reset and Clock Control. Base address: 0x4002 3800
- By default, peripherals are switched off (clock source disabled)
- We can turn each peripheral clock on/off separately.
- Specifically, the RCC_AHB1 peripheral clock enable register is where GPIO Port A is enabled.

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

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

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

- Through the GPIO Mode register we set individual pins either as: Input, Output, Analog or Alternate Function (AF, e.g. UART, USB, PWM, TIM, etc)

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

- ODR: Output Data Register
- If the pin is set as output, write to this register to set respective bit high/low

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).

GPIO IDR

- There are other registers for manipulating GPIO, which you can review on the Reference Manual (REF02)
- The GPIO_IDR: Input Data Register for example, is where you would read the state of an input pin.

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

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**
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **IDRy**: Port input data (y = 0..15)
These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

GPIO Registers

- GPIOx_MODER
  - Mode Register (Input, Output, AF, Analog)
- GPIOx_OTYPER
  - Output Type (Push-pull or Open-drain)
- GPIOx_OSPEEDR
  - Output Speed (Low, Medium, High, Very High Speed)
- GPIOx_PUPDR
  - Pull-up/Pull-down Register
- GPIOx_IDR
  - Input Data Register
- GPIOx_ODR
  - Output Data Register
- GPIOx_BSRR
  - Bit Set / Reset
- GPIOx_LCKR
  - Port Configuration Lock
- GPIOx_AFRL
  - Alternate Function Low Register
- GPIOx_AFRH
  - Alternate Function High Register

جامعة الكويت
KUWAIT UNIVERSITY

Headerless Blinky Example on STM32Nucleo

- In C, this is the code to perform a blinky routine. Program Size: 220 Bytes

```
/* Look Ma!, no headers */
#define GPIOARCCR (*(volatile int *)(0x40023800 + 0x30))
#define GPIOAMODER (*(volatile int *)0x40020000)
#define GPIOAODR (*(volatile int *)(0x40020000 + 0x14))

int main(void) {
    /* Enable GPIOA Clock */
    GPIOARCCR |= 1; /* Ref RCC_AHB2ENR register */
    /* Set Port A Pin 5 as Output */
    GPIOAMODER |= (1 << 10); /* Ref GPIOx_MODER register */
    while (1) {
        /* Set LED Pin High */
        GPIOAODR |= (1 << 5); /* Ref GPIOx_ODR register*/
        /* Dumb Delay: wait x number of clock cycles */
        for (int k = 0; k<1000000; k++){__asm("nop");}
        /* Set LED Pin Low */
        GPIOAODR &= ~(1 << 5); /* Ref GPIOx_ODR register*/
        /* Dumb Delay */
        for (int k = 0; k<1000000; k++){__asm("nop");}
    }
}
```

This is the lowest level programming in C, any lower and you will have to program in assembly

Not portable to other MCUs in the same family

- What is this gibberish: `#define GPIOAODR (*(volatile int *)(0x40020000 + 0x14))`
- This is creating a macro: GPIOAODR to access the GPIO Port A Output Data R
- The address itself is 0x40020014, but in C/C++ we need to tell the compiler that we want to represent the value **in** that address, so

1. Cast the hex number to 32bit pointer, now we have a pointer (address only)

```
#define GPIOAODR (int *)(0x40020000 + 0x14)
```

2. Make it volatile, to tell compiler that its value might change by hardware

```
#define GPIOAODR (volatile int *)(0x40020000 + 0x14)
```

3. Then dereference it using *, to act on the value INSIDE the address

```
#define GPIOAODR (*(volatile int *)(0x40020000 + 0x14))
```

جامعة الكويت
KUWAIT UNIVERSITY

Blinky Example on STM32Nucleo with stm32f401xe definitions

- The same code as before, but we use the provided macro definitions for the addresses, address shifts and bitmasks (exactly similar binary as before)

```
#include "stm32f401xe.h"

int main(void) {
    /* Enable GPIOA Clock */
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN; /* Ref RCC_AHB2ENR register */
    /* Set Port A Pin 5 as Output */
    GPIOA->MODER |= (1 << GPIO_MODER_MODE0_Pos); /* Ref GPIOx_MODER register */
    while (1) {
        /* Set LED Pin High */
        GPIOA->ODR |= (1 << GPIO_ODR_OD5_Pos); /* Ref GPIOx_ODR register*/
        /* Dumb Delay: wait x number of clock cycles */
        for (int k = 0; k<1000000; k++){__asm("nop");}
        /* Set LED Pin Low */
        GPIOA->ODR &= ~(1 << GPIO_ODR_OD5_Pos); /* Ref GPIOx_ODR register*/
        /* Dumb Delay */
        for (int k = 0; k<1000000; k++){__asm("nop");}
    }
}
```

Still Low-Level C but with the help of macro definitions (at no extra memory overhead charge)

Blinky Example on STM32Nucleo with STM32Cube HAL

- ST provides an abstraction for their MCUs called STM32Cube, this is a blinky routine using their abstraction. Program Size: 404 Bytes

```
#include "stm32f4xx_hal.h" /* We don't include the specific stm32f401xe header, but just the HAL */
#include "stm32f4xx.h"
#define LED_GPIO_CLK_ENABLE()                   __HAL_RCC_GPIOA_CLK_ENABLE()

int main(void){
  HAL_Init(); /* Initialize the HAL Drivers */

  LED_GPIO_CLK_ENABLE(); /* Enable GPIO A Clock */
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct); /* The pin configuration is done through a function */

  while (1)
  {
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5); /* HAL provides a toggle pin function */
    HAL_Delay(1000); /* As well as a delay function (milliseconds) */
  }
}
```

Higher Lower Level C Code, this is the recommended level for commercial/professional programming of MCU

Portable across other STM32 MCUs

Blinky Example on STM32Nucleo with Arduino

- And this is the blinky code using the Arduino framework
- Program Size: 12404 Bytes. Abstraction comes at a cost!

```
#include <Arduino.h>

void setup() { pinMode(LED_BUILTIN, OUTPUT); }

void loop() {
    digitalWrite(LED_BUILTIN, HIGH);
    delay(50);
    digitalWrite(LED_BUILTIN, LOW);
    delay(200);
}
```

A good level of abstraction. Good for education / hobbyist / prototyping / quick lab instruments.

Portable across all MCUs that have an Arduino adoption

Some available frameworks

- CMSIS: Cortex Microcontroller Software Interface Standard
  - Developed by ARM
  - Universal across other ARM Cortex MCU, not just ST
- SPL: Standard Peripheral Library
  - Developed by ST for ST based ARM MCUs
- HAL ST: Hardware Abstraction Layer ST
  - Developed by ST for ST based ARM MCUs, with focus on abstraction
- Arduino
  - Developed for hobbyists and prototypists
  - Become so popular, that manufacturers provide support for it

Standard Peripheral Library or STM32Cube

- Modern microcontrollers pack a lot of features and peripherals. Configuring peripherals through direct register access becomes cumbersome for a programmer.

- Using manufacturers provided standard libraries “framework” is becoming standard practice.

- Libraries abstract away differences between microcontrollers so code can be more portable.
  - HAL: Hardware Abstraction Layer

- Libraries are practically more proof tested with fewer bugs

How to know what to do?

- Configuring and programming an MCU is not a straight-forward or linear process.
- There are multiple references and tools that must be used concurrently
- With STM32Nucleo, we use:
  - Datasheet
  - Reference Manual

    (Datasheet and Reference Manual: Other manufacturers may combine them into one document)
  - User Manual for STM32 Nucleo
  - Framework User Manual (e.g. HAL and LL User Manual)
    - Example Code for framework selected
  - Consult developer community

- Gives "specs"

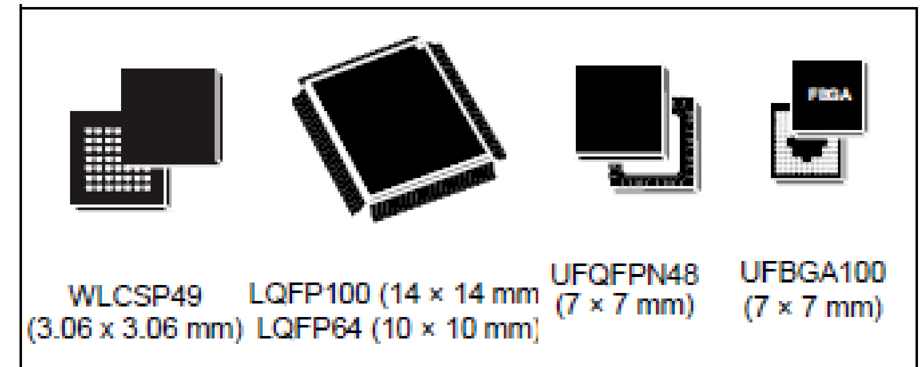
STM32F401xD STM32F401xE

ARM® Cortex®-M4 32b MCU+FPU, 105 DMIPS, 512KB Flash/96KB RAM, 11 TIMs, 1 ADC, 11 comm. interfaces

Datasheet - production data

Features

- Core: ARM® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 84 MHz, memory protection unit, 105 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - up to 512 Kbytes of Flash memory
  - up to 96 Kbytes of SRAM
- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os



- Debug mode
  - Serial wire debug (SWD) & JTAG interfaces
  - Cortex®-M4 Embedded Trace Macrocell™
- Up to 81 I/O ports with interrupt capability
  - Up to 78 fast I/Os up to 42 MHz

- A guide on how to configure and use the MCU
  - Registers Information
  - Possible Configurations
- Used by developers

RM0368
Reference manual

STM32F401xB/C and STM32F401xD/E
advanced Arm®-based 32-bit MCUs

Introduction

This Reference manual targets application developers. It provides complete information on how to use the memory and the peripherals of the STM32F401xB/C and STM32F401xD/E microcontrollers.

STM32F401xB/C and STM32F401xD/E are part of the STM32F401xx family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics refer to the datasheets.

For information on the Arm® Cortex®-M4 with FPU core, refer to the *Cortex®-M4 with FPU Technical Reference Manual*.

Related documents

Available from STMicroelectronics web site (*http://www.st.com*):

- STM32F401xB/C datasheet
- STM32F401xD/E datasheet
- For information on the Arm®-M4 core with FPU, refer to the *STM32F3xx/F4xxx Cortex®-M4 with FPU-M4 programming manual (PM0214)*.

- The HAL and LL User Manual is almost self contained
  - It re-describes the available configurations on the mcu

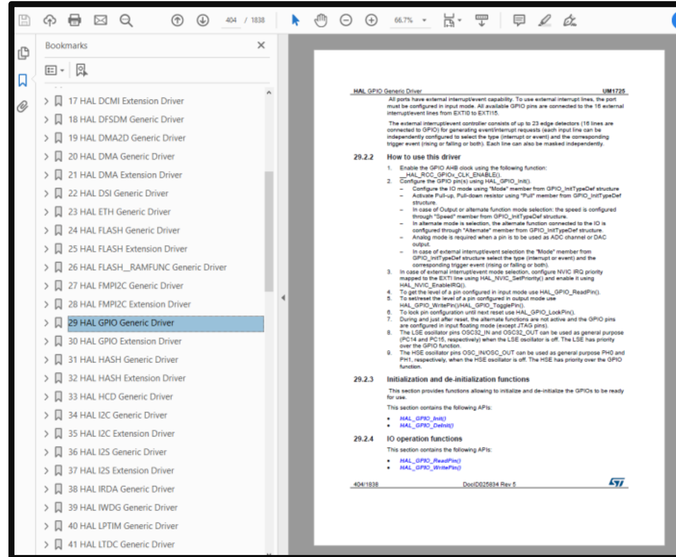
UM1725
User Manual

Description of STM32F4 HAL and LL drivers

Introduction

STM32Cube™ is STMicroelectronics's original initiative to ease developers' life by reducing development efforts, time and cost. STM32Cube™ covers the STM32 portfolio.

STM32Cube™ Version 1.x includes:



29.2.2 How to use this driver

1. Enable the GPIO AHB clock using the following function: __HAL_RCC_GPIOx_CLK_ENABLE().
2. Configure the GPIO pin(s) using HAL_GPIO_Init().
   - Configure the IO mode using "Mode" member from GPIO_InitTypeDef structure
   - Activate Pull-up, Pull-down resistor using "Pull" member from GPIO_InitTypeDef structure.
   - In case of Output or alternate function mode selection: the speed is configured through "Speed" member from GPIO_InitTypeDef structure.
   - In alternate mode is selection, the alternate function connected to the IO is configured through "Alternate" member from GPIO_InitTypeDef structure.
   - Analog mode is required when a pin is to be used as ADC channel or DAC output.
   - In case of external interrupt/event selection the "Mode" member from GPIO_InitTypeDef structure select the type (interrupt or event) and the corresponding trigger event (rising or falling or both).
3. In case of external interrupt/event mode selection, configure NVIC IRQ priority mapped to the EXTI line using HAL_NVIC_SetPriority() and enable it using HAL_NVIC_EnableIRQ().
4. To get the level of a pin configured in input mode use HAL_GPIO_ReadPin().
5. To set/reset the level of a pin configured in output mode use HAL_GPIO_WritePin()/HAL_GPIO_TogglePin().
6. To lock pin configuration until next reset use HAL_GPIO_LockPin().
7. During and just after reset, the alternate functions are not active and the GPIO pins are configured in input floating mode (except JTAG pins).
8. The LSE oscillator pins OSC32_IN and OSC32_OUT can be used as general purpose (PC14 and PC15, respectively) when the LSE oscillator is off. The LSE has priority over the GPIO function.
9. The HSE oscillator pins OSC_IN/OSC_OUT can be used as general purpose PH0 and PH1, respectively, when the HSE oscillator is off. The HSE has priority over the GPIO function.

جامعة الكويت
KUWAIT UNIVERSITY