Lecture 7: STM32 GPIO Programming and ARM CMSIS

Seyed-Hosein Attarzadeh-Niaki

Some slides due to ARM and Mazidi

Review

- Software development flow
- C For Embedded Systems
  - Data types
  - Bitwise operations
- Application Binary Interface (ABI)
  - Memory requirements
  - Calling procedures

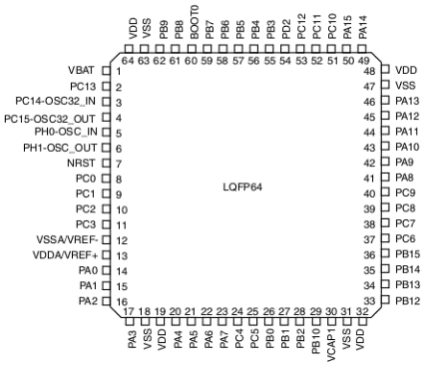
Outline

- STM32F401 General Purpose IO
  - Port Circuitry
  - Registers
- Cortex Microcontroller System Interface Standard (CMSIS)
  - Components
- GPIO programming using CMSIS
  - Accessing Hardware Registers in C
  - Clocking and Muxing
- Circuit Interfacing
  - Inputs
  - Outputs

STM32F401 GENERAL PURPOSE IO

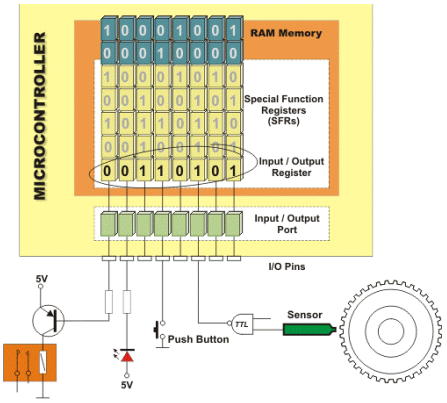
STM32F40x LQFP64 pinout

- Port A (PA) through Port E (PE)
- Not all port bits are available
- Quantity depends on package pin count



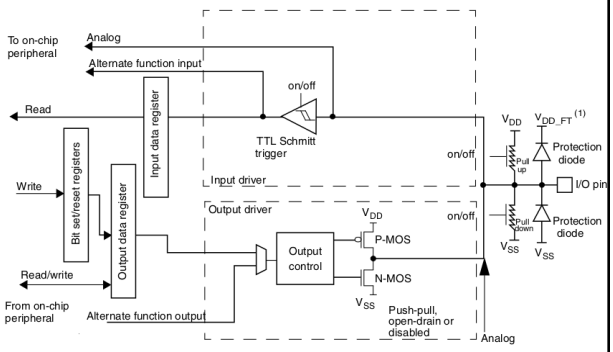
What Can We Expect From A GPIO?

- Configure it
  - Set its mode (digital/analog, input/output, etc.)
  - Set its speed
  - Set its electrical operating mode (output driver)
- Read/write data from/to it
- Set/reset its bits individually
- Set alternate functions in case of limited package pins
- …



GPIO Port Bit Circuitry in MCU

- Configuration
  - Direction
  - MUX
  - Modes
  - Speed
- Data
  - Output (different ways to access it)
  - Input
  - Analogue
- Locking



GPIO Registers

- Each general-purpose I/O port has
  - four 32-bit configuration registers
    - **GPIOx_MODER** (input, output, AF, analog)
    - **GPIOx_OTYPER** (output type: push-pull or open drain)
    - **GPIOx_OSPEEDR** (speed)
    - **GPIOx_PUPDR** (pull-up/pull-down)
  - 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**)
  - two 32-bit alternate function selection register (**GPIOx_AFRH** and **GPIOx_AFRL**)
- One set of control registers (10 in total) per port
- Each bit in a control register corresponds to a port bit
- All registers have to be accessed as 32-bit word

Address (Offset)	Name	Description	Type
0x00	GPIOx_MODER	GPIOx Port Mode(Direction) Register	R/W
0x04	GPIOx_OTYPER	Output Type Register	R/W
0x08	GPIOx_OSPEEDR	Output Speed Register	R/W
0x0C	GPIOx_PUDR	Pull-Up / Down Register	R/W
0x10	GPIOx_IDR	Port Input Data Register	R/W
0x14	GPIOx_ODR	Port Output Data Register	R/W
0x18	GPIOx_BSRR	Bit Set / Reset Register	R/W
Where x=A, B, C, ... for ports			

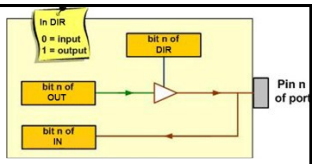
GPIO Configuration Registers

- Each bit can be configured differently
- Reset clears port bit direction to 0
- Output modes: push-pull or open drain + pull-up/down
- Output data from output data register (**GPIOx_ODR**) or peripheral (alternate function output)
- Input states: floating, pull-up/down, analog
- Input data to input data register (**GPIOx_IDR**) or peripheral (alternate function input)

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]	I/O configuration
01	0	SPEED [B:A]	0 0	GP output PP
	0		0 1	GP output PP + PU
	0		1 0	GP output PP + PD
	0		1 1	Reserved
	1		0 0	GP output OD
	1		0 1	GP output OD + PU
	1		1 0	GP output OD + PD
	1		1 1	Reserved (GP output OD)
10	0	SPEED [B:A]	0 0	AF PP
	0		0 1	AF PP + PU
	0		1 0	AF PP + PD
	0		1 1	Reserved
	1		0 0	AF OD
	1		0 1	AF OD + PU
	1		1 0	AF OD + PD
	1		1 1	Reserved
00	x	x x	0 0	Input Floating
	x	x x	0 1	Input PU
	x	x x	1 0	Input PD
	x	x x	1 1	Reserved (input floating)
11	x	x x	0 0	Input/output Analog
	x	x x	0 1	Reserved
	x	x x	1 0	
	x	x x	1 1	

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

The Data and Direction Registers: A Simplified View of an I/O pin



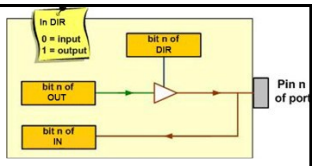
- GPIOx_ODR Output Data Register (x=A, B,C, ..)

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

- GPIOx_MODER (GPIO Mode) for Direction Register (x=A, B, C, …)

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

The Data and Direction Registers: A Simplified View of an I/O pin



- GPIOx_BSRR (GPIO Bit Set Reset) Register

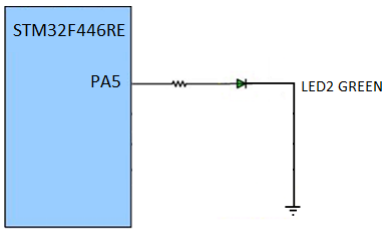
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- GPIOx_IDR (Input Data) Register

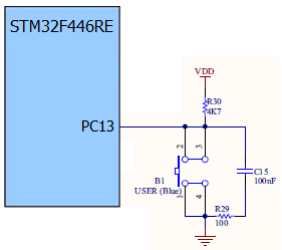
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

I/O Connections for Nucleo board

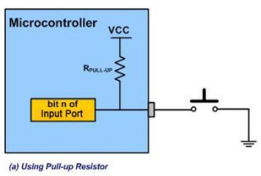
User LED

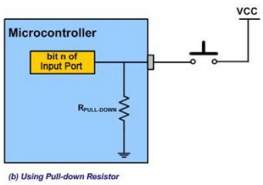


Switch

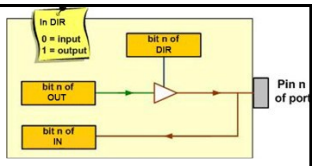


External switches





The Data and Direction Registers: A Simplified View of an I/O pin



- GPIOx_OTYPER Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

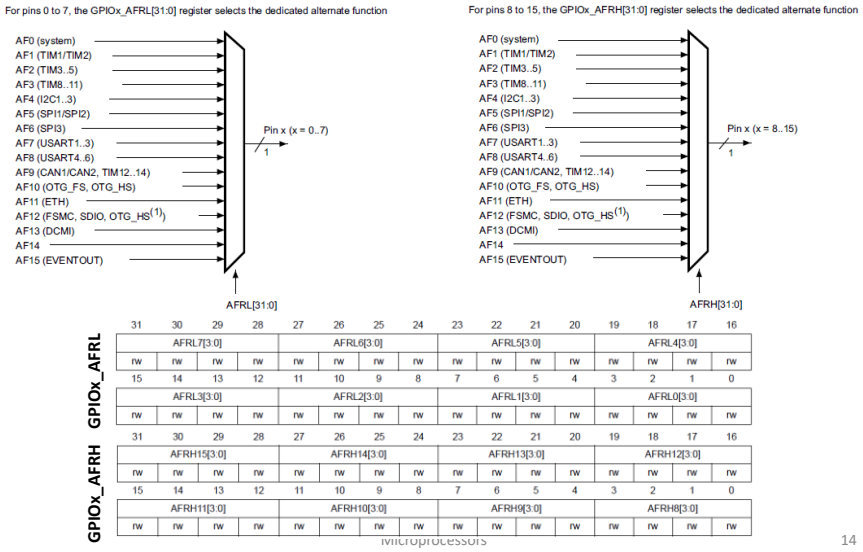
- GPIOx_OSPEEDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [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
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3[1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- GPIOx_PUPDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[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
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

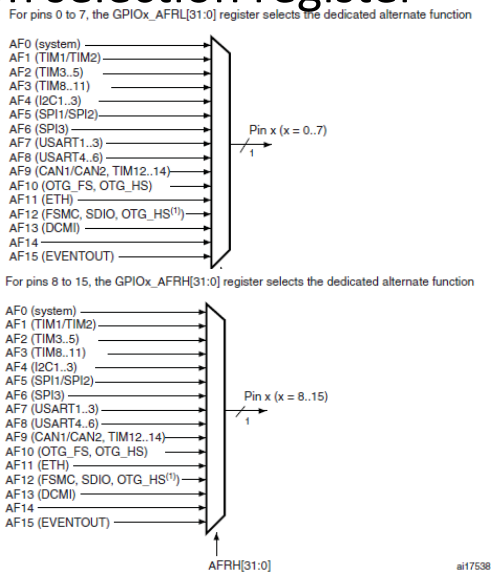
STM32 Arm Pin Multiplexing of Alternate Functions



Alternate function selection register

- In AF mode, AFRL or AFRH needs to be configured to be driven by specific peripheral
- Can be seen as a select signal to the Mux

- EVENTOUT is not mapped onto the following I/O pins: PC13, PC14, PC15, PH0, PH1 and PI8.



STM32F401RE Alternative Functions (AF0-AF15) Pin Selection

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			
Port A	PA0	-	TIM2_CH1/ TIM2_ETR	TIM5_CH1	-	-	-	-	USART2_ CTS	-	-	-	-	-	-	-	EVENT OUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_ RTS	-	-	-	-	-	-	-	EVENT OUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_ TX	-	-	-	-	-	-	-	EVENT OUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-		-	USART2_ RX	-	-	-	-	-	-	-	EVENT OUT
	PA4	-	-	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	USART2_ CK	-	-	-	-	-	-	-	EVENT OUT
	PA5	-	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_SCK	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA6	-	TIM1_BKIN	TIM3_CH1	-	-	SPI1_ MISO	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA7	-	TIM1_CH1N	TIM3_CH2	-	-	SPI1_ MOSI	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA8	MCO_1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_ CK	-	-	OTG_FS_ SOF	-	-	-	-	EVENT OUT
	PA9	-	TIM1_CH2	-	-	I2C3_ SMBA	-	-	USART1_ TX	-	-	OTG_FS_ VBUS	-	-	--	-	EVENT OUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_ RX	-	-	OTG_FS_I D	-	-	-	-	EVENT OUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_ CTS	USART6_ TX	-	OTG_FS_ DM	-	-	-	-	EVENT OUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_ RTS	USART6_ RX	-	OTG_FS_ DP	-	-	-	-	EVENT OUT
	PA13	JTMS_ SWDIO	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA14	JTCK_ SWCLK	-	-	-	-	-	-	-	-	-	-	-	-	-	-	EVENT OUT
	PA15	JTDI	TIM2_CH1/ TIM2_ETR	-	-	-	SPI1_NSS	SPI3_NSS/ I2S3_WS	-	-	-	-	-	-	-	-	EVENT OUT

CORTEX MICROCONTROLLER SYSTEM INTERFACE STANDARD (CMSIS)

CMSIS Aims

Enhanced software reusability

- easier to reuse software code in different Cortex-M projects

Enhanced software compatibility

- by having a consistent software infrastructure

Easy to learn

- easy access to processor core features from the C language

Toolchain independent

- can be used with various compilation tools

Openness

- the source code for CMSIS core files can be downloaded from its public github repository

CMSIS Components

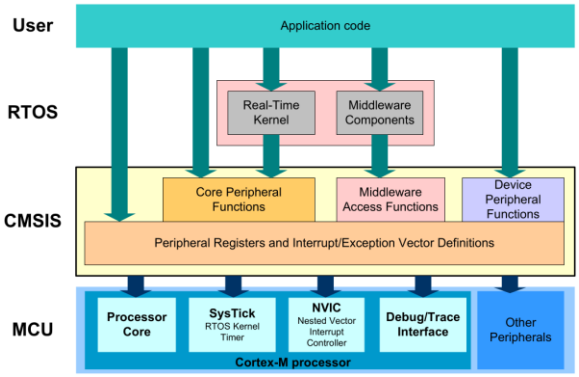
CMSIS-Core (M)	• Standard API for the Cortex-M processor core and peripherals
CMSIS-Core (A)	• API and basic run-time system for the Cortex-A5/A7/A9 processor core and peripherals
CMSIS-Driver	• Generic peripheral driver interfaces for middleware
CMSIS-DSP	• DSP library with fixed and floating point support
CMSIS-RTOS	• Common API for real-time operating systems along with a reference implementation based on RTX
CMSIS-SVD	• Peripheral description of a device for debugger/core header files
CMSIS-DAP	• Firmware for a debug unit
CMSIS-NN	• Collection of efficient neural network kernels

Areas of Standardization in CMSIS-Core

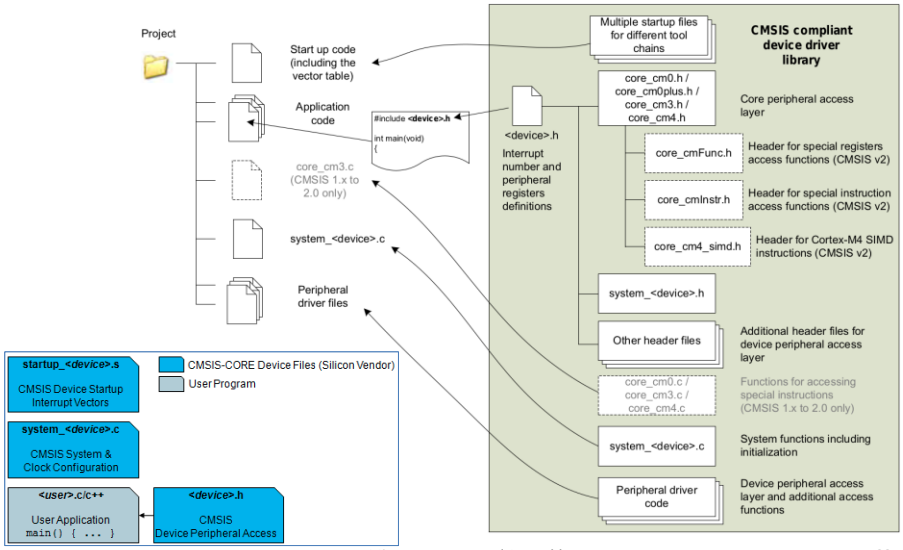
- **Definitions for the processor's peripherals**
  - NVIC registers, SysTick timer, MPU, SCB, etc.
- **Access functions to access processor's features**
  - Interrupt control and special register access
- **Functions for accessing special instructions easily**
  - Special instructions that cannot be generated from standard C (e.g., WFI)
- **Function names for system exception handlers**
  - Easier software development for multiple Cortex-M products
- **Functions for system initialization**
  - "SystemInit()": configuration of clock circuitry and power management registers
- **Software variables for clock speed information**
  - "SystemCoreClock"
- **A common platform for device-driver libraries**

Organization of CMSIS-Core

- Core Peripheral Access Layer
- Device Peripheral Access Layer
- Access Functions for Peripherals



Using CMSIS-Core in A Project



Using CMSIS

GPIO PROGRAMMING

CMSIS - Accessing Hardware Registers in C

- Header file stm32f4xx.h defines C data structure types to represent hardware registers in MCU with CMSIS-Core hardware abstraction layer

Address (Offset)	Name	Description	Type
0x00	GPIOx_MODER	GPIOx Port Mode(Direction) Register	R/W
0x04	GPIOx_OTYPER	Output Type Register	R/W
0x08	GPIOx_OSPEEDR	Output Speed Register	R/W
0x0C	GPIOx_PUDR	Pull-Up / Down Register	R/W
0x10	GPIOx_IDR	Port Input Data Register	R/W
0x14	GPIOx_ODR	Port Output Data Register	R/W
0x18	GPIOx_BSRR	Bit Set / Reset Register	R/W
Where x=A, B, C, ... for ports			

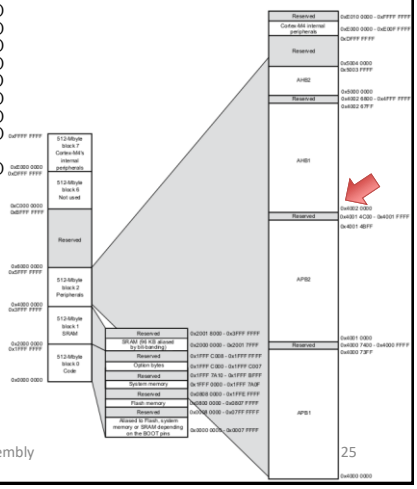
```
/**
  * @brief General Purpose I/O
  */

typedef struct
{
  __IO uint32_t MODER;    /*!< GPIO port mode register,               Address offset: 0x00      */
  __IO uint32_t OTYPER;   /*!< GPIO port output type register,        Address offset: 0x04      */
  __IO uint32_t OSPEEDR;  /*!< GPIO port output speed register,       Address offset: 0x08      */
  __IO uint32_t PUPDR;    /*!< GPIO port pull-up/pull-down register,  Address offset: 0x0C      */
  __IO uint32_t IDR;      /*!< GPIO port input data register,         Address offset: 0x10      */
  __IO uint32_t ODR;      /*!< GPIO port output data register,        Address offset: 0x14      */
  __IO uint16_t BSRRL;    /*!< GPIO port bit set/reset low register,  Address offset: 0x18      */
  __IO uint16_t BSRRH;    /*!< GPIO port bit set/reset high register, Address offset: 0x1A      */
  __IO uint32_t LCKR;     /*!< GPIO port configuration lock register, Address offset: 0x1C      */
  __IO uint32_t AFR[2];   /*!< GPIO alternate function registers,     Address offset: 0x20-0x24 */
} GPIO_TypeDef;
```

CMSIS C Support

- Header file stm32f4xx.h defines pointers to GPIO_Type registers

```
#define GPIOA_BASE            (AHB1PERIPH_BASE + 0x0000)
#define GPIOB_BASE            (AHB1PERIPH_BASE + 0x0400)
#define GPIOC_BASE            (AHB1PERIPH_BASE + 0x0800)
#define GPIOD_BASE            (AHB1PERIPH_BASE + 0x0C00)
#define GPIOE_BASE            (AHB1PERIPH_BASE + 0x1000)
#define GPIOF_BASE            (AHB1PERIPH_BASE + 0x1400)
#define GPIOG_BASE            (AHB1PERIPH_BASE + 0x1800)
#define GPIOH_BASE            (AHB1PERIPH_BASE + 0x1C00)
#define GPIOI_BASE            (AHB1PERIPH_BASE + 0x2000)
……
#define AHB1PERIPH_BASE       (PERIPH_BASE + 0x00020000)
……
#define PERIPH_BASE           ((uint32_t)0x40000000)
```



Clocking Logic

- Need to enable clock to GPIO module
- By default, GPIO modules are disabled to save power
- Writing to an unclocked module triggers a hardware fault!
- Control register RCC_AHB1ENR gates clocks to GPIO ports
- Enable clock to Port D

```
RCC->AHB1ENR |= (1UL << 3);
```

- Header file stm32f4xx.h has definitions

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
```

Initializing GPIO

1. Enable clock for Port
2. Set the mode
3. Set the output type
4. Set the speed
5. Set the pull-up or pull down
6. Set the AF

- Not all of these are necessary, default setting is OK (usually all bits cleared after reset)
- Need to access the entire 32 registers

- Simple example for initializing an LED connected to Port D pin 12

```
void LED_Init (void) {

  RCC->AHB1ENR  |=  (1UL <<    3) ;

  GPIOD->MODER  |=  (1UL << 2*12) ;

  GPIOD->OTYPER |=  (0UL <<   12) ;

  GPIOD->OSPEEDR |= (2UL << 2*12) ;

  GPIOD->PUPDR   |= (1UL << 2*12) ;
```


CMSIS C Support

- Header file stm32f4xx.h also has bits definition for GPIO register

```
#define GPIO_MODER_MODER0         ((uint32_t)0x00000003)
#define GPIO_MODER_MODER0_0       ((uint32_t)0x00000001)
#define GPIO_MODER_MODER0_1       ((uint32_t)0x00000002)

#define GPIO_OTYPER_OT_0          ((uint32_t)0x00000001)

#define GPIO_OSPEEDER_OSPEEDR0    ((uint32_t)0x00000003)
#define GPIO_OSPEEDER_OSPEEDR0_0  ((uint32_t)0x00000001)
#define GPIO_OSPEEDER_OSPEEDR0_1  ((uint32_t)0x00000002)

#define GPIO_PUPDR_PUPDR0         ((uint32_t)0x00000003)
#define GPIO_PUPDR_PUPDR0_0       ((uint32_t)0x00000001)
#define GPIO_PUPDR_PUPDR0_1       ((uint32_t)0x00000002)
```

Writing/Reading Output/Input Port Data

- Direct: write value GPIOx_ODR
- Clear (to 0): Write 1 to BSRRL
- Set (to 1): write 1 to BSRRH
  - `GPIOD->ODR|=(1<<12);`
  - Equivalent to: `GPIOD->BSRRL=(1<<12);`
  - Or with CMSIS: `GPIOD->ODR|=GPIO_ODR_ODR_12`
  - `GPIOD->ODR&=~(1<<12);`
  - Equivalent to: `GPIOD->BSRRH=(1<<12);`
  - Or with CMSIS: `GPIOD->ODR&=~GPIO_ODR_ODR_12`
- Read from IDR
  - `data=GPIOD->IDR&(1<<12)`
  - Or with CMSIS: `data=GPIOD->IDR&GPIO_IDR_IDR_12`

Coding Style and Bit Access

- Easy to make mistakes dealing with literal binary and hexadecimal values
  - "To set bits 13 and 19, use
    0000 0000 0000 1000 0010 0000 0000 0000 or **0x00082000**"
- Make the literal value from shifted bit positions

```
n = (1UL << 19) | (1UL << 13);
```

- Define names for bit positions

```
#define POS_0 (13)
#define POS_1 (19)
n = (1UL << POS_0) | (1UL << POS_1);
```

- Create macro to do shifting to create mask

```
#define MASK(x) (1UL << (x))
n = MASK(POS_0) | MASK(POS_1);
```

Using Masks

- Overwrite existing value in **n** with mask
  ```
  n = MASK(foo);
  ```
- Set in **n** all the bits which are one in mask, leaving others unchanged
  ```
  n |= MASK(foo);
  ```
- Complement the bit value of the mask
  ```
  ~MASK(foo);
  ```
- Clear in n all the bits which are zero in mask, leaving others unchanged
  ```
  n &= MASK(foo);
  ```


Using Masks with CMSIS

```
BIT = MASK(foo);
```

- `#define SET_BIT(REG, BIT)  ((REG) |= (BIT))`
- `#define CLEAR_BIT(REG, BIT)  ((REG) &= ~(BIT))`
- `#define READ_BIT(REG, BIT)  ((REG) & (BIT))`
- `#define CLEAR_REG(REG)  ((REG) = (0x0))`
- `#define WRITE_REG(REG, VAL)  ((REG) = (VAL))`
- `#define READ_REG(REG)  ((REG))`
- ```
  #define MODIFY_REG(REG, CLEARMASK, SETMASK)
          WRITE_REG((REG), (((READ_REG(REG)) &
          (~(CLEARMASK))) | (SETMASK)))
  ```
  ```

C Code

```
#define LED1_POS (13)
#define LED2_POS (14)
#define SW1_POS  (0)
#define MASK(x)  (1UL << (x))
RCC->AHB1ENR|=RCC_AHB1ENR_GPIODEN;

/* Initialization of GPIO */
…
GPIOD->ODR = MASK(LED1_POS);  // turn on LED1, turn off LED2

while (1) {
  if (GPIOD->IDR & MASK(SW1_POS)) {
    // switch is pressed, then light LED 2
    GPIOD->BSRRL = MASK(LED2_POS);
    GPIOD->BSRRH = MASK(LED1_POS);
  } else {
    // switch is not pressed, so light LED 1
    GPIOD->BSRRL = MASK(LED1_POS);
    GPIOD->BSRRH = MASK(LED2_POS);
  }
}
```


Atomic Access

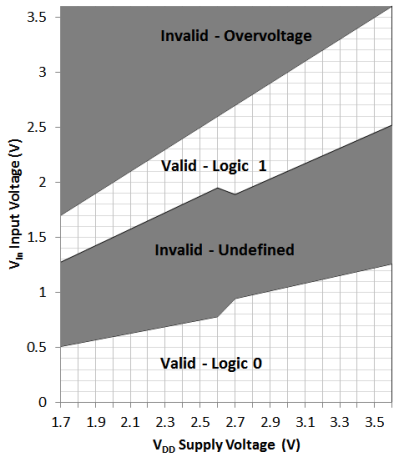
- Unlike some of other MCU, the AHB1 on STM32F4 provides atomic access to one or more bits.
- Which means do not have to disable the interrupt when programming the GPIOx_ODR at bit level.

Inputs and Outputs, Ones and Zeros, Voltages and Currents

INTERFACING

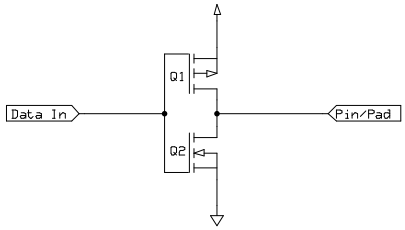
Inputs: What's a One? A Zero?

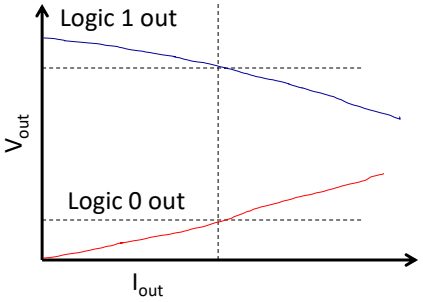
- Input signal's value is determined by voltage
- Input threshold voltages depend on supply voltage VDD
- Exceeding VDD or GND may damage chip



Outputs: What's a One? A Zero?

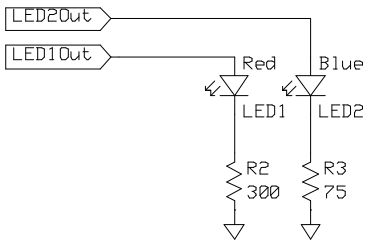
- Nominal output voltages
  - 1: V_{DD}-0.5 V to V_{DD}
  - 0: 0 to 0.5 V
- Note: Output voltage depends on current drawn by load on pin
  - Need to consider source-to-drain resistance in the transistor
  - Above values only specified whencurrent < 5 mA (18 mA for high-drive pads) and V_{DD} > 2.7 V





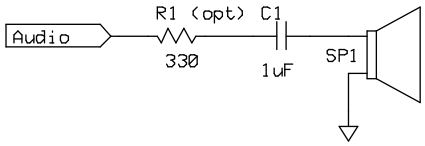
Driving External LEDs

- Need to limit current to a value which is safe for both LED and MCU port driver
- Use current-limiting resistor
  - $R = (V_{DD} – V_{LED})/I_{LED}$
- Set I_{LED} = 4 mA
- V_{LED} depends on type of LED (mainly color)
  - Red: ~1.8V
  - Blue: ~2.7 V
- Solve for R given VDD = ~3.0 V
  - Red: 300 Ω
  - Blue: 75 Ω



Output Example: Driving a Speaker

- Create a square wave with a GPIO output
- Use capacitor to block DC value
- Use resistor to reduce volume if needed




```
void Speaker_Beep(uint32_t
frequency){
  Init_Speaker();
  while(1){
    GPIOD->BSRRL=(MASK(2));
    Delay(frequency);
    GPIOD->BSRRH=(MASK(2));
    Delay(frequency);
  }
}
```