STM32F407 Parallel I/O Ports

Cortex-M4 memory map

Cortex-M Peripherals

STM32xx Peripherals
STMicroelectronics
STM32F40x microcontroller

Cortex-M:
NVIC, System Tick Timer

CPU

Flash Memory
External memory

STM32F40x Peripherals
STM32F40x system architecture

**Bus Masters**

- 64-Kbyte CCM data RAM
- ARM Cortex-M4
- DMA
- DMA MEM1
- DMA MEM2
- DMA_P1
- DMA_P2
- MAC Ethernet
- USB OTG HS

**Slaves**

- S0, S1, S2, S3, S4, S5, S6, S7
- Bus matrix-S

Parallel instruction/data accesses by CPU
STM32F407 flash memory

Main memory = 1Mbyte
= 7x128K + 1x64K + 4x16K “sectors”
(Command erase one “sector” or entire memory)

<table>
<thead>
<tr>
<th>Block</th>
<th>Name</th>
<th>Block base addresses</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main memory</td>
<td>Sector 0</td>
<td>0x0800 0000 - 0x0800 3FFF</td>
<td>16 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 1</td>
<td>0x0800 4000 - 0x0800 7FFF</td>
<td>16 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 2</td>
<td>0x0800 8000 - 0x0800 BFFF</td>
<td>16 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 3</td>
<td>0x0800 C000 - 0x0800 FFFF</td>
<td>16 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 4</td>
<td>0x0801 0000 - 0x0801 FFFF</td>
<td>64 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 5</td>
<td>0x0802 0000 - 0x0803 FFFF</td>
<td>128 Kbytes</td>
</tr>
<tr>
<td></td>
<td>Sector 6</td>
<td>0x0804 0000 - 0x0805 FFFF</td>
<td>128 Kbytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System memory</td>
<td>0x1FFF 0000 - 0x1FFF 77FF</td>
<td>30 Kbytes</td>
</tr>
<tr>
<td></td>
<td>OTP area</td>
<td>0x1FFF 7800 - 0x1FFF 7A0F</td>
<td>528 bytes</td>
</tr>
<tr>
<td></td>
<td>Option bytes</td>
<td>0x1FFF C000 - 0x1FFF C00F</td>
<td>16 Kbytes</td>
</tr>
</tbody>
</table>
STM32F407 SRAM blocks

- Byte, half-word, and word addressable
- 192Kbytes of system SRAM
  - 112Kbyte and 16Kbyte blocks at 0x2000_0000
    - Accessible by all AHB masters
  - 64Kbyte block at 0x1000_0000
    - Accessible by CPU only via D-bus
- Concurrent SRAM accesses possible to separate blocks
- 4Kbytes of battery-backup SRAM
  - Configure with control registers
STM32F4xx microcontroller
General-Purpose Input/Output (GPIO)

- Up to 144 GPIO pins, individually configurable
  - # GPIO ports varies among microcontroller parts
- Each port (GPIOA through GPIOI) comprises 16 GPIO pins
- Pin options (each pin configurable via GPIO registers):
  - Output: push-pull or open-drain+pull-up/down
    - Selectable output speed
  - Input: floating, pull-up/down
  - Analog: input or output
  - Alternate functions: up to 16 per pin
    - Data to/from peripheral functions (Timers, I2C/SPI, USART, USB, etc.)
- Digital data input/output via GPIO registers
  - Input data reg. (IDR) – parallel (16-bit) data from pins
  - Output data reg. (ODR) – parallel (16-bit) data to pins
  - Bit set/reset registers (BSRR) for bitwise access to pins
STM32F4xx GPIO pin structure

- **IDR**: Input data register
- **ODR**: Output data register
- **BSRR**: Bit set/reset registers
- **STR**: Read/write from on-chip peripheral
- **LDR**: Read analog input
- **Alt. Function**: Alternate function
- **Analog IO**: Analog input/output
- **PIN**: I/O pin
- **Alt. Function**: Alternate function output

The diagram illustrates the internal structure of a GPIO pin, showing how it can be configured for different functions such as analog input or output, and how it interacts with various control registers and signals.
Addressing I/O registers (in C)
(from \textit{stm32f4xx.h} header file)

```
#define PERIPH_BASE ((uint32_t)0x40000000) /*!< Peripheral base address in the alias region */
#define AHB1PERIPH_BASE (PERIPH_BASE + 0x00020000) /*!< AHB1 bus peripherals */
#define GPIOD_BASE (AHB1PERIPH_BASE + 0x0C00) /*!< GPIO Port D base address */
#define GPIOD ((GPIO_TypeDef *) GPIOD_BASE) /*!< GPIO Port D pointer */
/* General Purpose I/O */
typedef struct /* Treat GPIO register set as a “record” data structure */
{
    __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 uint32_t BSRR; /*!< GPIO port bit set/reset register, Address offset: 0x18 */
    __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;
```

```
GPIOD->ODR = value; /*!< write data to ODR of GPIOD */
```
GPIO data registers

- 16-bit memory-mapped data registers for each port GPIOx
  \[ x = A \ldots I \] (GPIOA, GPIOB, …, GPIOI)
- \texttt{GPIOx_IDR} (Input Data Register)
  - Data input through the 16 pins
  - Read-only
- \texttt{GPIOx_ODR} (Output Data Register)
  - Write data to be output to the 16 pins
  - Read last value written to ODR
  - Read/write (for read-modify-write operations)
- C examples:

  ```c
  GPIOA->ODR = 0x45; // send data to output pins
  N = GPIOA->IDR; // copy data from in pins to N
  ```
GPIO port bit set/reset registers

- GPIO output bits can be individually set and cleared
  
  (without affecting other bits in that port)

- **GPIOx_BSRR** (Bit Set/Reset Register)
  - Bits $[15..0] = \text{Port x set bit } y \ (y = 15..0)$  (BSRRL)
  - Bits $[31..16] = \text{Port x reset bit } y \ (y = 15..0)$  (BSRRH)
  - Bits are write-only
    - $1 = \text{Set/reset the corresponding GPIOx bit}$
    - $0 = \text{No action on the corresponding GPIOx bit}$
      
      (“set” has precedence if bit=1 in both BSSRL and BSSRH)

- C examples:
  
  ```c
  GPIOA->BSRRL = (1 << 4); //set bit 4 of GPIOA
  GPIOA->BSRRH = (1 << 5); //reset bit 5 of GPIOA
  ```
GPIO “mode” register

- **GPIOx_MODER** selects operating mode for each pin
  - \( x = A \ldots I \) (GPIOA, GPIOB, …, GPIOI)

- 2 bits per pin:
  - 00 – Input mode (reset state):
    - Pin value captured in IDR every bus clock (through Schmitt trigger)
  - 01 – General purpose output mode:
    - Write pin value to ODR
    - Read IDR to determine pin state
    - Read ODR for last written value
  - 10 – Alternate function mode:
    - Select alternate function via AF mux/register (see later slide)
  - 11 – Analog mode:
    - Disable output buffer, input Schmitt trigger, pull resistors
      (so as not to alter the analog voltage on the pin)
Alternate function selection

Each pin defaults to GPIO pin at reset (mux input 0)

GPIOx_AFRL
(low pins 0..7)

GPIOx_AFRH
(high pins 8..15)
GPIO pin option registers

- **GPIOx_OTYPER** – output type
  - 0 = push/pull, 1 = open drain

- **GPIOx_PUPDR** – pull-up/down
  - 00 = no pull-up/pull-down
  - 01 = pull-up
  - 10 = pull-down

- **GPIOx_OSPEEDR** – output speed
  - 00 = 2 MHz low speed
  - 01 = 25 MHz medium speed
  - 10 = 50 MHz fast speed
  - 11 = 100 MHz high speed (on 30 pf)
Discovery Board connections

- **User LEDs**
  - LED3 (orange) – PD13
  - LED4 (green) – PD12
  - LED5 (red) – PD14
  - LED6 (blue) – PD15
  - LED7 (USB OTG, Vbus) – PA9
  - LED8 (USB OTG, overcurrent) – PD5
  - LED1 – USB communication
  - LED2 – 3.3v power

- **Push buttons**
  - User (blue button) – PA0
  - Reset (black button) – NRST pin

PAx = port GPIOA pin
PDx = port GPIOD pin
Discovery board button and LEDs

- The user button is positive logic
  - Uses external pull-down resistor (outside the uC)
- LED3-LED6 are positive logic
GPIO register addresses

- Base addresses of GPIOx register “blocks”
  - GPIOA = 0x4002_0000
  - GPIOB = 0x4002_0400
  - GPIOC = 0x4002_0800
  - GPIOD = 0x4002_0C00
  - GPIOE = 0x4002_1000
  - GPIOF = 0x4002_1400
  - GPIOG = 0x4002_1800
  - GPIOH = 0x4002_1C00
  - GPIOI = 0x4002_2000

- Register address offsets within each GPIOx register block
  - MODER = 0x00 pin direction/mode register
  - OTYPER = 0x04 pin output type register
  - OSPEEDR = 0x08 pin output speed register
  - PUPDR = 0x0C pull=up/pull-down register
  - IDR = 0x10 input data register
  - ODR = 0x14 output data register
  - BSRR = 0x18 bit set/reset register
  - LCKR = 0x1C lock register
  - AFRL = 0x20 alt. function register – low
  - AFRH = 0x24 alt. function register – high
Assembly language example

; Symbols for GPIO register block and register offsets
GPIOA   EQU   0x40020000 ; GPIOA base address
ODR     EQU   0x14 ; ODR reg offset
IDR     EQU   0x10 ; IDR reg offset
; Alternative – symbol for each register address
GPIOA_ODR EQU   GPIOA + GPIO_ODR ; addr of GPIOA_ODR
GPIOA_IDR EQU   GPIOA + GPIO_IDR ; addr of GPIOA_IDR

; Using addresses = GPIO base + register offset
    LDR   r0,=GPIOA ; GPIOA base address
    LDR   r1,[r0,#ODR] ; GPIOA base + ODR offset
    STR   r1,[r0,#IDR] ; GPIOA base + IDR offset

; Using separate address for each GPIO register
    LDR   r0,=GPIOA_ODR ; GPIOA_ODR address
    LDR   r1,[r0]
    LDR   r0,=GPIOA_IDR ; GPIOA_IDR address
    STR   r1,[r0]

How would we address GPIOD ODR/IDR?
GPIO port initialization ritual

- Initialization (executed once at beginning)
  1. Turn on GPIOx clock in register **RCC_AHB1ENR**
     (Reset and Clock Control, AHB1 peripheral clock register)
     - RCC register block base address = 0x4002 3800
     - AHB1ENR register offset = 0x30
     - AHB1ENR bits 0-8 enable clocks for GPIOA-GPIOI, respectively
  2. Configure “mode” of each pin in **GPIOx_MODER**
     - Input/Output/Analog/Alternate Function
  3. Optionally (if other than reset configuration required)
     - Configure speed of each output pin in **GPIOx_OSPEEDR**
     - Configure type of each pin in **GPIOx_OTYPER**
     - Configure pull-up/pulldown of each pin in **GPIOx_PUPDR**

- Input from switches, output to LEDs
  Read/write 16-bit data via **GPIOx_IDR/ODR**
  Set/clear output pins via **GPIOx_BSRR**
To set bits

The **or** operation to set bits 3-0 of GPIOD_MODER, to select analog mode for pins PD1 and PD0.

(The other 28 bits of GPIOD_MODER are to remain constant.)

*Friendly* software modifies just the bits that need to be.

\[
\text{GPIOD\_MODER } \text{|= 0x0F; } // \text{ PD1,PD0 analog}
\]

**Assembly:**

```
LDR  R0,=GPIOD\_MODER
LDR  R1,[R0]    ; read previous value
ORR  R1,R1,#0x0F  ; set bits 0-3
STR  R1,[R0]    ; update
```

| c7 c6 c5 c4 c3 c2 c1 c0 | value of R1
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 1 1 1 1 1</td>
<td>0x0F constant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c7 c6 c5 c4 c1 c1 c1 c1</th>
<th>result of the ORR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 1 1 1 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
To clear bits

The **AND** or **BIC** operations to clear bits 3-0 of GPIOD_MODER to select “input mode” for pins PD1 and PD0. (Without altering other bits of GPIOD_MODER.)

*Friendly* software modifies just the bits that need to be.

```
GPIOD_MODER &= ~0x0F; // PD1,PD0 output
```

**Assembly:**

```
LDR R0,=GPIOD_MODER
LDR R1,[R0] ; read previous value
BIC R1,R1,#0x0F ; clear bits 3-0
STR R1,[R0] ; update
```

```
c7  c6  c5  c4  c3  c2  c1  c0  \text{value of R1}
1   1   1   1   0   0   0   0
```

```
\text{BIC #x0F = AND #0xFFFFFFFF0}
```

```
c7  c6  c5  c4  0   0   0   0  \text{result of the BIC}
```

Bard, Gerstlauer, Valvano, Yerraballi
To toggle bits

The **exclusive or** operation can also be used to toggle bits.

```c
GPIOD_ODR ^= 0x80; /* toggle PD7 */
```

**Assembly:**

```assembly
LDR   R0,=GPIOD_ODR
LDRH  R1,[R0]    ; read port D
EOR   R1,R1,#0x80 ; toggle state of pin PD7
STRH  R1,[R0]    ; update port D
```

<table>
<thead>
<tr>
<th>b7 b6 b5 b4 b3 b2 b1 b0</th>
<th>value of R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 0 0 0 0 0</td>
<td>0x80 constant</td>
</tr>
</tbody>
</table>

```
~b7 b6 b5 b4 b3 b2 b1 b0 | result of the EOR |
```
To set or reset bits using BSSR

Use BSSR register to set or reset selected GPIO bits, without affecting the others

```c
GPIOD_ODR |= 0x0080;  // PD7 = 1
GPIOD_ODR &= ~0x0400; // PD10 = 0
```

**Assembly:**

```assembly
LDR R0,=GPIOD            ; GPIOD base address
MOV R1,#0x0080           ; select PD7
STRH R1,[R0,#BSSRH]     ; set PD7 = 1
MOV R1,#0x0400           ; select PD10
STRH R1,[R0,#BSSRL]     ; reset PD10 = 0
```

**Alternative:** write concurrently to BSSRH and BSSRL (as one 32-bit register)

```assembly
LDR R0,=GPIOD            ; GPIOD base address
MOV R1,#0x0400           ; select PD10 in BSSRL
MOVT R1,#0x0080          ; select PD7 in BSSRH
STR R1,[R0,#BSSR]       ; PD10=0 and PD7=1
```