Parallel Input/Output

Textbook Chapter 14 – General-Purpose Input/Output
STM32F407 Reference Manual, Chapter 8 (general-purpose IOs)
Computer Architecture (Chapter 1)

Von-Neumann

Instructions and data are stored in the same memory.

Harvard

Data and instructions are stored into separate memories.

Memory-mapped I/O: I/O peripherals assigned addresses in memory address map
Isolated I/O: I/O peripherals have a separate (isolated) address map
Embedded system components

STM32F407VG

Microcontroller
- Processor
- RAM
- ROM

I/O Ports
- ADC

Bus

Electrical, mechanical, chemical, or optical devices
- DAC
- Analog signals
ARM Cortex-M4 based system

- ARM Cortex-M4 processor
- *Harvard* architecture
  - Different busses for instructions and data
- RISC machine
  - *Pipelining* effectively provides single cycle operation for many instructions
  - Thumb-2 configuration employs both 16 and 32 bit instructions

Bard, Gerstlauer, Valvano, Yerraballi
Arm Cortex-M4 processor

- Interrupt Signals
- Interrupt Control
- ST-Link Debugger (on the board)

- Optional WIC
- Optional FPU
- Optional Memory protection unit
- Optional Debug Access Port
- Optional Flash patch
- Optional Data watchpoints
- Optional Serial Wire viewer

- Bus matrix
- Code interface
- SRAM and peripheral interface

- Program memory
- Data memory
- I/O devices
STM32F407VG Microcontroller
1. Flash memory @0x08000000
2. SRAM @0x20000000
3. ST peripheral modules*
4. Cortex-M4 peripherals*

*Memory-Mapped I/O
### STM32F407 flash memory

Main memory = 1Mbyte = 7x128K + 1x64K + 4x16K “sectors”

(Commands 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>Sector 11</td>
<td>0x080E 0000 - 0x080F FFFF</td>
<td>128 Kbytes</td>
</tr>
<tr>
<td>System memory</td>
<td></td>
<td>0x1FFF 0000 - 0x1FFF 77FF</td>
<td>30 Kbytes</td>
</tr>
<tr>
<td>OTP area</td>
<td></td>
<td>0x1FFF 7800 - 0x1FFF 7A0F</td>
<td>528 bytes</td>
</tr>
<tr>
<td>Option bytes</td>
<td></td>
<td>0x1FFF C000 - 0x1FFF C00F</td>
<td>16 Kbytes</td>
</tr>
</tbody>
</table>

Program code, Constant data

Separate, one-time programmable
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
- Supports *concurrent* SRAM accesses to separate blocks
- 4Kbytes of battery-backed-up SRAM
  - Configure with control registers
Input/Output (I/O) Overview

- **Issues**
  - Device selection: I/O addressing/address decoding
  - Data transfer: amount, rate, to/from device
  - Synchronization: CPU and external device

- **Bus structures**
  - Links: Internal bus, system bus, data link
  - Memory-mapped I/O

- **Synchronization**
  - Programmed I/O
  - Interrupt-driven I/O
  - DMA (Direct Memory Access) I/O
Interface between CPU and external device

- “Device” may include digital and/or non-digital components.
- Two paths: CPU-to-interface; interface-to-device
  - Can be different data widths and speeds (data rates)
  - Might not be ready “at the same time”
- Typical digital interface to CPU is via addressable registers
  - Registers assigned memory-mapped (or isolated I/O) addresses
Simple input: on/off switch

**Negative Logic s**
- pressed, 0V, false
- not pressed, 3.3V, true

**Positive Logic t**
- pressed, 3.3V, true
- not pressed, 0V, false
Example – 10-key matrix keypad

**Drive (output pins)**

- Pin 1
- Pin 2
- Pin 3
- Pin 4
- Pin 5
- Pin 6

**Read (input pins)**

- Pin 0
- Pin 1
- Pin 2
- Pin 3
- Pin 4
- Pin 5

Uses both input and output pins.

**Pull-up Resistors**

- 10K
- +3v

**Pressed Key**

Column 0

Column 1

Column 2

Column 3
Simple output: LED

anode (+)  

cathode (-)  

“big voltage connects to big pin”

(a) LED curve  

(b) Positive logic interface  

(c) Negative logic interface  

Bard, Gerstlauer, Valvano, Yerraballi
LED interfaces

**Positive logic, low current**
- TM4C Out
- R → LED
- 2mA

**Positive logic, high current**
- TM4C Out
- 7405 or 7406
- +5V
- 10mA
- R → LED
- high
- 0.5V

**Negative logic, low current**
- TM4C Out
- 3.3V
- R → LED
- 2mA

**Negative logic, high current**
- TM4C Out
- 7407
- +5V
- 10mA
- R → LED
- low
- 0.5V
7-segment LED/LCD display

- May use parallel or multiplexed port outputs.
Discovery board button and LEDs

- **PAx =** port GPIOA pin
- **PDx =** port GPIOD pin

- The user button is positive logic
  - Uses external pull-down resistor (outside the uC)
  - Reset (black button) – NRST pin

- LED3-LED6 are positive logic
  - LED7 (USB OTG, Vbus) – PA9
  - LED8 (USB OTG, overcurrent) – PD5
  - LED1 – USB communication
  - LED2 – 3.3v power

From Discovery Board User Manual
Parallel input/output ports

- **Parallel port** => multiple bits read/written in parallel by the CPU
  - Parallel **input** port = portal through which a CPU can access information FROM an external device
  - Parallel **output** port = portal through which a CPU can send information TO an external device
- Multiple I/O ports are contained in the most microcontrollers
  - Some microcontrollers allow for additional I/O ports to be added via “expansion buses”
- Each port is configured and accessed via one or more registers
  - Each register is assigned a unique memory address
  - CPU reads/writes data via port **data registers**
  - **Mode Register** (or data direction register) for each port determines whether each pin is input or output
The input/output direction of a bidirectional port is specified by its “mode” register (sometimes called “data direction” register).

- **GPIOx_MODER**: designate each pin as input, output, analog, or “alternate function”

**Question**: What if the program reads from GPIOA_ODR?
STM32F4xx microcontroller
General-Purpose I/O (GPIO) ports

- 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 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 control of output pins
STM32F4xx GPIO pin structure
GPIO "mode" register

- **GPIOx_MODE** selects operating mode for each pin
  - \( x = A \ldots I \)  \( (\text{GPIOA, GPIOB, \ldots, GPIOI}) \)

- 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 \( \text{(see later slide)} \)
  
  - **11 – Analog mode:**
    - Disable output buffer, input Schmitt trigger, pull resistors
    - \( \text{(so as not to alter the analog voltage on the pin)} \)
GPIO data registers

- 16-bit memory-mapped data registers for each port GPIOx
  \[ x = A \ldots I \] (GPIOA, GPIOB, …, GPIOI)

- GPIOx_IDR
  - Data input through the 16 pins
  - Read-only

- GPIOx_ODR
  - 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] = Port x set bit y (y = 15..0) (BSRRL)
  - Bits [31..16] = Port x reset bit y (y = 15..0) (BSRRH)
  - Bits are write-only
    - 1 = Set/reset the corresponding GPIOx bit
    - 0 = 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
  ```
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

Modify these registers for other than default configuration

- **GPIOx_OTYPER** – output type
  - 0 = push/pull (reset state)
  - 1 = open drain

- **GPIOx_PUPDR** – pull-up/down
  - 00 – no pull-up/pull-down (reset state)
  - 01 – pull-up
  - 10 – pull-down

- **GPIOx_OSPEEDER** – output speed
  - 00 – 2 MHz low speed (reset state)
  - 01 – 25 MHz medium speed
  - 10 – 50 MHz fast speed
  - 11 – 100 MHz high speed (on 30 pf)
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
  - GPIOx_MODER = 0x00 pin direction/mode register
  - GPIOx_OTYPER = 0x04 pin output type register
  - GPIOx_OSPEEDR = 0x08 pin output speed register
  - GPIOx_PUPDR = 0x0C pull=up/pull-down register
  - GPIOx_IDR = 0x10 input data register
  - GPIOx_ODR = 0x14 output data register
  - GPIOx_BSRR = 0x18 bit set/reset register
  - GPIOx_BSRRL = 0x18 BSRR low half – set bits
  - GPIOx_BSRRH = 0x1A BSRR high half – reset bits
  - GPIOx_LCKR = 0x1C lock register
  - GPIOx_AFRIL = 0x20 alt. function register – low
  - GPIOx_AFRH = 0x24 alt. function register – high
Assembly language example

; Symbols for GPIO register block and register offsets
GPIOA EQU 0x40020000 ; GPIOA base address
GPIO_ODR EQU 0x14 ; ODR reg offset
GPIO_IDR EQU 0x10 ; IDR reg offset

; Alternative – create 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
STRH r1,[r0,#GPIOx_ODR] ; GPIOA base + ODR offset
LDRH r1,[r0,#GPIOx_IDR] ; GPIOA base + IDR offset

; Using separate address for each GPIO register
LDR r0,=GPIOA_ODR ; GPIOA_ODR address
STRH r1,[r0]
LDR r0,=GPIOA_IDR ; GPIOA_IDR address
LDRH 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. Configure\(^1\) speed of each output pin in `GPIOx_OSPEEDER`
  4. Configure\(^1\) type of each pin in `GPIOx_OTYPER`
  5. Configure\(^1\) pull-up/pulldown of each pin in `GPIOx_PUPDR`
     \(^1\) if other than reset state required

- **Input from switches, output to LEDs**
  Read/write 16-bit data via `GPIOx_IDR/ODR`
  Set/clear output pins via `GPIOx_BSRRL/BSRRH`
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.

```
GPIOD_MODER |= 0x0F; // 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
```

<table>
<thead>
<tr>
<th>c7</th>
<th>c6</th>
<th>c5</th>
<th>c4</th>
<th>c3</th>
<th>c2</th>
<th>c1</th>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c7</td>
<td>c6</td>
<td>c5</td>
<td>c4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

value of R1

0x0F constant

result of the **ORR**
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
```

<table>
<thead>
<tr>
<th>c7 c6 c5 c4 c3 c2 c1 c0</th>
<th>value of R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 0 0 0 0</td>
<td>BIC #x0F = AND #0xFFFFFFFF0</td>
</tr>
<tr>
<td>c7 c6 c5 c4 0 0 0 0</td>
<td>result of the BIC</td>
</tr>
</tbody>
</table>

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:**

```
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>b_7</th>
<th>b_6</th>
<th>b_5</th>
<th>b_4</th>
<th>b_3</th>
<th>b_2</th>
<th>b_1</th>
<th>b_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

value of R1

<table>
<thead>
<tr>
<th>b_7</th>
<th>b_6</th>
<th>b_5</th>
<th>b_4</th>
<th>b_3</th>
<th>b_2</th>
<th>b_1</th>
<th>b_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0x80</td>
</tr>
</tbody>
</table>

0x80 constant

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

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

**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)

```
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
```
Switch Interfacing

The **and** operation to extract, or *mask*, individual bits:

```
Pressed = GPIOA_ODR & 0x10;
//true if PA6 switch pressed (pos. logic)
```

**Assembly:**

```
LDR R0,=GPIOA_IDR
LDRH R1,[R0] ; read port A IDR
AND R1,#0x10 ; clear all bits except bit 6
LDR R0,=Pressed ; update variable
STRH R1,[R0] ; true iff switch pressed
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Mapped Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0x00</td>
</tr>
<tr>
<td>00000001</td>
<td>0x01</td>
</tr>
<tr>
<td>00000010</td>
<td>0x02</td>
</tr>
<tr>
<td>00000011</td>
<td>0x03</td>
</tr>
<tr>
<td>00000100</td>
<td>0x04</td>
</tr>
<tr>
<td>00000101</td>
<td>0x05</td>
</tr>
<tr>
<td>00000110</td>
<td>0x06</td>
</tr>
<tr>
<td>00000111</td>
<td>0x07</td>
</tr>
<tr>
<td>00001000</td>
<td>0x08</td>
</tr>
<tr>
<td>00001001</td>
<td>0x09</td>
</tr>
<tr>
<td>00001010</td>
<td>0x0A</td>
</tr>
<tr>
<td>00001011</td>
<td>0x0B</td>
</tr>
<tr>
<td>00001100</td>
<td>0x0C</td>
</tr>
<tr>
<td>00001101</td>
<td>0x0D</td>
</tr>
<tr>
<td>00001110</td>
<td>0x0E</td>
</tr>
<tr>
<td>00001111</td>
<td>0x0F</td>
</tr>
<tr>
<td>00010000</td>
<td>0x10</td>
</tr>
<tr>
<td>00010001</td>
<td>0x11</td>
</tr>
<tr>
<td>00010010</td>
<td>0x12</td>
</tr>
<tr>
<td>00010011</td>
<td>0x13</td>
</tr>
<tr>
<td>00010100</td>
<td>0x14</td>
</tr>
<tr>
<td>00010101</td>
<td>0x15</td>
</tr>
<tr>
<td>00010110</td>
<td>0x16</td>
</tr>
<tr>
<td>00010111</td>
<td>0x17</td>
</tr>
<tr>
<td>00011000</td>
<td>0x18</td>
</tr>
<tr>
<td>00011001</td>
<td>0x19</td>
</tr>
<tr>
<td>00011010</td>
<td>0x1A</td>
</tr>
<tr>
<td>00011011</td>
<td>0x1B</td>
</tr>
<tr>
<td>00011100</td>
<td>0x1C</td>
</tr>
<tr>
<td>00011101</td>
<td>0x1D</td>
</tr>
<tr>
<td>00011110</td>
<td>0x1E</td>
</tr>
<tr>
<td>00011111</td>
<td>0x1F</td>
</tr>
</tbody>
</table>

Bard, Gerstlauer, Valvano, Yerraballi