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.
- ARM Cortex-M4 processor
- *Harvard* architecture internally
  - 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
Embedded system components

STM32F407VG
Memory-mapped vs Isolated I/O

Memory-Mapped I/O

<table>
<thead>
<tr>
<th>One Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFFFC</td>
</tr>
<tr>
<td>0xFFFFFFFFF8</td>
</tr>
<tr>
<td>0xFFFFFFFFF4</td>
</tr>
<tr>
<td>0x000000008</td>
</tr>
<tr>
<td>0x000000004</td>
</tr>
<tr>
<td>0x000000000</td>
</tr>
</tbody>
</table>

- Some addresses allocated to memory, others to I/O devices.
- Access both with LDR/STR instructions.

<table>
<thead>
<tr>
<th>Memory Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFFFC</td>
</tr>
<tr>
<td>0xFFFFFFFFF8</td>
</tr>
<tr>
<td>0xFFFFFFFFF4</td>
</tr>
<tr>
<td>0x000000008</td>
</tr>
<tr>
<td>0x000000004</td>
</tr>
<tr>
<td>0x000000000</td>
</tr>
</tbody>
</table>

Isolated I/O

<table>
<thead>
<tr>
<th>Memory Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFFFC</td>
</tr>
<tr>
<td>0xFFFFFFFFF8</td>
</tr>
<tr>
<td>0xFFFFFFFFF4</td>
</tr>
<tr>
<td>0x000000008</td>
</tr>
<tr>
<td>0x000000004</td>
</tr>
<tr>
<td>0x000000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFFFC</td>
</tr>
<tr>
<td>0x000000008</td>
</tr>
<tr>
<td>0x000000004</td>
</tr>
<tr>
<td>0x000000000</td>
</tr>
</tbody>
</table>

- Access with memory-reference instructions. (LDR/STR)
- Access with special IN/OUT instructions.

Arm CPUs

Intel x86 CPUs
Arm Cortex-M4 memory map

STM32F407VG Microcontroller
1. Flash memory @0x08000000
2. SRAM @0x20000000
3. ST peripheral modules*
4. Cortex-M4 peripherals*

*Memory-Mapped I/O
ST Microelectronics
STM32F40x
microcontroller

General-Purpose Inputs/Outputs (GPIO)

CPU

ROM

RAM

Other Peripheral Modules

Other Peripheral Modules
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></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>Sector 11</td>
<td>0x080E 0000 - 0x080F FFFF</td>
<td>128 Kbytes</td>
</tr>
<tr>
<td>System memory</td>
<td>0x1FFF 0000 - 0x1FFF 77FF</td>
<td>30 Kbytes</td>
<td></td>
</tr>
<tr>
<td>OTP area</td>
<td>0x1FFF 7800 - 0x1FFF 7A0F</td>
<td>528 bytes</td>
<td></td>
</tr>
<tr>
<td>Option bytes</td>
<td>0x1FFF C000 - 0x1FFF C00F</td>
<td>16 Kbytes</td>
<td></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, direction (to/from device)
  - Synchronization: CPU and external device

• **Bus structures**
  - Links: Internal bus, system bus, data link
  - Memory-mapped (use LDR/STR) or isolated I/O

• **Synchronization**
  - Programmed I/O
  - Interrupt-driven I/O
  - DMA (Direct Memory Access) I/O
“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) 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

Bard, Gerstlauer, Valvano, Yerraballi
Example – 10-key matrix keypad

Uses both input and output pins.

Drive (output pins)

1 Pin 8
0 Pin 5
1 Pin 3
1 Pin 1

Read (input pins)

1 Pin 2
0 Pin 4
1 Pin 6

Column 3
Column 2
Column 1
Column 0

Uses both input and output pins.
Simple output: LED (light-emitting diode)

anode (+)

---

cathode (-)

“big voltage connects to big pin”

(a) LED curve

(b) Positive logic interface

(c) Negative logic interface

Bard, Gerstlauer, Valvano, Yerraballi
7-segment LED/LCD display

- May use parallel or multiplexed port outputs.

Common anode or Common cathode

Select digit

Data[6:0]

One wire each to LEDs a-b-c-d-e-f-g
**Target Board – STM32F4Discovery**

- **32-bit Cortex-M4 Processor Core**
- **STM32F407VG Microcontroller**
  - DSP and FPU
  - 168 MHz max clock
  - Up to 1MB Flash/192KB RAM
  - Wide range of peripherals, including USB on-the-go
- **STM32F4Discovery Board**
  - Reset and user push-buttons
  - Four LEDs
  - Peripherals: 3-axis accelerometer, LED, microphone, audio codec and speaker driver
  - Quick breadboard connection and easy probing with compatible expansion headers
  - Rich examples, libraries and extra expansion boards available from ST and other third parties
The user button is positive logic
- Uses external pull-down resistor (outside the uC)
- Reset (black button) – NRST pin

LED3-LED6 (user LEDs) are positive logic
- LED7 (USB OTG, Vbus) – PA9
- LED8 (USB OTG, overcurrent) – PD5
- LED1 – USB communication
- LED2 – 3.3v power
Parallel input/output ports

- **Parallel port** => multiple bits read/written in parallel by 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 in most microcontrollers
  - Some uCs support additional I/O ports via expansion buses

- Each port configured and accessed via one or more **registers**
  - Each register is assigned a unique memory address
  - CPU reads/writes data via **data registers** in the port hardware
  - **Mode Register** (or data direction register) for a 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
- Each pin configurable via GPIO mode register (MODER)
  - 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
GPIOx registers effectively a 10-word array

Example:
Read GPIOA_IDR
LDR R0,=GPIOA
LDRH R1,[R0,#0x10]
STRH  R1,[R0,#0x14]
GPIO “mode” register

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

- 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 data registers for each port GPIOx
  \[ x = A \ldots I \quad (\text{GPIOA}, \text{GPIOB}, \ldots, \text{GPIOI}) \]

- **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 register addresses

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

- **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_AFRL = 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

  LDR    r0,=GPIOA ;GPIOA base address
  STRH   r1,[r0,#GPIO_ODR] ;GPIOA base + ODR offset
  LDRH   r1,[r0,#GPIO_IDR] ;GPIOA base + IDR 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

  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 Enable Register)*  
     - RCC register block base address = 0x4002 3800  
     - AHB1ENR register offset = 0x30  
     - AHB1ENR bits 8-0 enable clocks for GPIOI-GPIOA, respectively

**Example: Turn on clock to GPIOB**

```
RCC_AHB1ENR  EQU  0x40023830 ;RCC base address + AHB1ENR offset
LDR R0,=RCC_AHB1ENR ;point to RCC_AHB1ENR
LDR R1,[R0] ;read current settings
ORR R1,#0x0002 ;set bit 1 to 1 to enable clock for GPIOB
STR  R1,[R0] ;write back to RCC_AHB1ENR
```

2. Configure “mode” of *each pin* in **GPIOx_MODER**  
   - Input/Output/Analog/Alternate Function  
   - Change mode bits for *only the pins to be configured*

3. Write initial values to output pins via **GPIOx_ODR**  
   - Can also use BSRR to set/reset pin values
Optional Setup:

- If other than reset condition required
- NOT NEEDED FOR DISCOVERY BOARD

- 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`
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.

\[
\text{GPIOD}\rightarrow\text{MODER} \mid= 0x0F; \quad // \quad \text{PD1,PD0 analog}
\]

**Assembly:**

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

\[
\begin{array}{cccccccc}
 c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

value of R1

\[
\begin{array}{cccccccc}
 c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

**0x0F** constant

\[
\begin{array}{cccccccc}
 c_7 & c_6 & c_5 & c_4 & c_3 & c_2 & c_1 & c_0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
\]

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
LDR  R1,[R0,#MODER] ; read previous value
BIC  R1,R1,#0x0F    ; clear bits 3-0
STR  R1,[R0,#MODER] ; 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>
<th>value of R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 1 1 1 0 0 0 0</td>
</tr>
</tbody>
</table>

**BIC #x0F = AND #0xFFFFFFFF0**

<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>
<th>result of the <strong>BIC</strong></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>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
To toggle bits

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

```
GPIOD->ODR ^= 0x80; /* toggle PD7 */
```

**Assembly:**

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

```
b_7  b_6  b_5  b_4  b_3  b_2  b_1  b_0  
1 0 0 0 0 0 0 0 0 0x80 constant

~b_7  b_6  b_5  b_4  b_3  b_2  b_1  b_0  
```

value of R1

result of the **EOR**
GPIO port bit set/reset registers

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

- \textbf{GPIOx\_BSRR} (Bit Set/Reset Register)

  - Bits [15..0] = Port x \textbf{set} bit y (y = 15..0) \hspace{1cm} (BSRRL)
  
  - Bits [31..16] = Port x \textbf{reset} bit y (y = 15..0) \hspace{1cm} (BSRRH)

  - Bits are \textit{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:

  - \texttt{GPIOA\_BSRRL} = (1 << 4); \hspace{1cm} // set bit 4 of GPIOA
  
  - \texttt{GPIOA\_BSRRH} = (1 << 5); \hspace{1cm} // reset bit 5 of GPIOA
To set or reset bits using BSSR
(Bit Set/Reset Register)

Write directly to ODR to initialize PD7 to 1 and PD10 to 0

```
LDR    R0,=GPIOD    ; GPIOD base address
LDRH   R1,[R0,#ODR] ; read current ODR
ORR    R1,#0x0080   ; set bit 7 = 1
BIC    R1,#0x0400   ; reset bit 10 = 0
STRH   R1,[R0,#ODR] ; write updated pattern to ODR
```

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

```
LDR    R0,=GPIOD    ; GPIOD base address
MOV    R1,#0x0080   ; select PD7
STRH   R1,[R0,#BSRRH] ; set PD7 = 1
MOV    R1,#0x0400   ; select PD10
STRH   R1,[R0,#BSRRL] ; 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,#BSRR] ; PD10=0 and PD7=1
```
Alternate function selection
(After setting MODER bits to 10)

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

GPIOx_AFRL
(low pins 0..7)

GPIOx_AFRH
(high pins 8..15)
Other GPIO pin options

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

```c
if ((GPIOA->ODR & 0x40) != 0) {
    //Result 0x40 if PA6 switch pressed (pos. logic)
}
```

**Assembly:**

```
LDR    R0,=GPIOA
LDRH   R1,[R0,#IDR]  ; read port A IDR
TST    R1,#0x40       ; clear all bits except bit 6
BNE    SwitchPressed ; branch if switch pressed (Z flag 0)
```

<table>
<thead>
<tr>
<th>a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0</th>
<th>value of R1</th>
<th>0x40 constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0x40</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0 a_6 0 0 0 0 0 0</th>
<th>result of the <strong>AND</strong>: Z flag set if a_6=0</th>
</tr>
</thead>
</table>

Bard, Gerstlauer, Valvano, Yerraballi