I/O Programming Basics

LISHA/UFSC

Prof. Dr. Antônio Augusto Fröhlich
Fauze Valério Polpeta
Lucas Francisco Wanner

http://www.lisha.ufsc.br/~guto

March 2009
Input/Output

“The world outside the CPU”

- I/O ports (GIOP, SPI, ...)
- Buses (I2C, CAN, ...)
- Peripheral devices
  - UART
  - Timers
  - EEPROM
  - Sensors
  - Actuators
  - ...
I/O Ports

- Connect external devices
  - Sensors, actuators, UART, ...
- Parallel (e.g. GPIO) or serial (e.g. SPI)
- Input, output or bidirectional
- Synchronous (clock) or asynchronous (strobe)
- Some can directly drive leds, buttons and TTL-style circuitry
I/O Mapping

■ Register mapped
  ● CPU registers directly map I/O ports

■ Memory mapped
  ● I/O ports and device registers are mapped in the processor's memory address space
  ● chip select <- address decoder

■ I/O mapped
  ● I/O ports and device registers are mapped in a separate address space
  ● chip select <- address decoder + I/O line
I/O Operation

- **Polling x Interrupt**
  - **Polling**
    - Processor continuously probes an I/O device's status register
    - Implies in busy waiting
  - **Interrupt**
    - I/O device notifies the processor when its status changes
    - Supports idle waiting

- **PIO x DMA**
  - **Programmed I/O**
    - I/O device <=> CPU <=> memory
  - **Direct Memory Access**
    - I/O device <=> memory
Polling

- “Sequential interrogation of devices for various purposes”
  - Operational status, readiness to send or receive data, ...
- Processor continuously probes an I/O device's status register
  - Implies in busy waiting
- Example
  - When receiving data via an UART, an application can pool the UART status register to check when a byte is ready to be read and then read the corresponding data register
Interrupts

- I/O device notifies the processor when its status changes by means of asynchronous signals named Interrupt Requests (IRQ)
- An interrupt request causes the processor to interrupt program execution and switch to an Interrupt Service Routine (ISR)
- Interrupts can usually be remapped and masked
- Example
  - After completing the transmission of a byte, an UART could trigger an interrupt
  - The ISR can then initiate the sending of a next byte
Interrupts

- **Interrupt Vector**
  - An array of pointers, indirect pointers or single instructions that leads to the ISRs
  - May reside in ROM (predefined) or in RAM (programmable)

- Interrupts triggered by external devices such as timers and sensors are known as **external interrupts**

- Interrupts triggered by internal events such as arithmetic exceptions are know as **exceptions**
Case Study: H8/3292

- **H8/3292 interrupt table**
  - Stored at \(0x0000 - 0x0049\) (ROM in the RCX)
  - RCX ROM vectors redirect interrupts to the on-chip RAM interrupt table
  - Decreasing priority

- **on-chip RAM interrupt table**
  - Stored at \(0xfd80 - 0xfdbf\)
  - Pointers to user-defined handlers

- **Masking**
  - Globally (except NMI) CCR bit 7
  - Individually through the off-chip register field

<table>
<thead>
<tr>
<th>Vector</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reset</td>
</tr>
<tr>
<td>1 - 2</td>
<td>reserved</td>
</tr>
<tr>
<td>3</td>
<td>NMI</td>
</tr>
<tr>
<td>4 - 6</td>
<td>IRQs</td>
</tr>
<tr>
<td>7 - 11</td>
<td>reserved</td>
</tr>
<tr>
<td>12 - 18</td>
<td>16-bit timer</td>
</tr>
<tr>
<td>19 - 21</td>
<td>8-bit timer 0</td>
</tr>
<tr>
<td>22 - 24</td>
<td>8-bit timer 1</td>
</tr>
<tr>
<td>25 - 26</td>
<td>reserved</td>
</tr>
<tr>
<td>27 - 30</td>
<td>serial</td>
</tr>
<tr>
<td>31 - 34</td>
<td>reserved</td>
</tr>
<tr>
<td>35</td>
<td>ADI</td>
</tr>
<tr>
<td>36</td>
<td>WOVF</td>
</tr>
</tbody>
</table>
Interrupt Dispatching
Interrupt Dispatching

Diagram:
- CPU H8/300
- H8/3292
- ROM
- on-chip RAM
- on-chip Register Field
- off-chip Register Field
- dispatcher
- interrupted code
- RAM
Interrupt Dispatching

CPU H8/300

H8/3292

dispatcher

ROM

RAM

off-chip Register Field

on-chip Register Field

interrupted code

dispatcher

pointer

handler
Interrupt Dispatching

CPU
H8/300

H8/3292

dispatcher
ROM

on-chip
RAM

on-chip
Register Field

RAM

handler

interrupted code

off-chip Register Field
Interrupt Dispatching

- CPU H8/300
- H8/3292
- ROM
- dispatcher
- on-chip Register Field
- on-chip RAM
- running code
- off-chip Register Field
- RAM
LEGO RCX Interrupt Handling

- **H8 dispatching**
  ```c
  push pc
  push ccr
  ccr[7]=1 /* int disable */
  ```

- **H8/300 Handler (ROM)**
  ```c
  void h8_handler(void) {
      push r6
      mov RCX_Int_Table[n], r6
      jsr @r6
      pop r6
      rte
  }
  ```

- **RCX Handler**
  ```c
  void rcx_handler(void) {
      /* push registers */
      /* handle interrupt */
      /* pop registers */
  }
  ```

- **RCX Interrupt table**
  ```c
typedef void (RCX_Handler)(void);
RCX_Handler ** RCX_Int_Table = (RCX_Handler **)0xfd80;
RCX_Int_Table[n] = &rcx_handler;
  ```
Case Study: AVR

- AT90S interrupt vector
  - Stored in the first 14 words of program memory
  - Usually jumps to ISRs
  - Decreasing priority

- Masking
  - I bit in SREG (Global Interrupt Enable)
  - GIMSK (IRQ0 and IRQ1)
  - TIMSK (Timers)
  - UCR (UART)
  - SPCR (SPI)

<table>
<thead>
<tr>
<th>Vector</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>1</td>
<td>IRQ0 (external)</td>
</tr>
<tr>
<td>2</td>
<td>IRQ1 (external)</td>
</tr>
<tr>
<td>3..7</td>
<td>timer1 events</td>
</tr>
<tr>
<td>8</td>
<td>timer0 overflow</td>
</tr>
<tr>
<td>9</td>
<td>SPI Tx complete</td>
</tr>
<tr>
<td>10..12</td>
<td>UART events</td>
</tr>
<tr>
<td>13</td>
<td>Analog comparator</td>
</tr>
</tbody>
</table>
Interrupt Dispatching

AT90S8515

AVR CPU

Program Flash

Register File

I/O Ctrl

RAM

Running Code
Interrupt Dispatching

AT90S8515

Dispatcher

Program Flash

Interrupted Code

AVR CPU

Register File

I/O Ctrl

RAM
Interrupt Dispatching

AT90S8515

Dispatcher

Program Flash

Interrupted Code

Handler

AVR CPU

Register File

I/O Ctrl

RAM
Interrupt Dispatching

AT90S8515

AVR CPU

Program Flash

Running Code

Handler

Register File

I/O Ctrl

RAM
Case Study: AVR

- After an interrupt is triggered, interrupts are globally disabled
  - Subsequent interrupt requests are flagged and executed in order of priority after the first ISR returns
  - The `reti` instruction reenables interrupts
  - Users may use cascading interrupts by reenabling interrupts in the ISR
  - External Interrupts (IRQs) are only flagged as long as the IRQ signal is active
Case Study: AVR (handler)

interrupts:

rjmp reset ; reset
reti ;
reti ;
reti ;
reti ;
reti ;
reti ;
reti ;
rjmp timer ; timer 0
    overflow
reti ;
reti ;
reti ;
reti ;
reti ;
reti ;

reset:

ldi r20, 0x02 ; reset handler
out 0x3e, r20
ldi r20, 0x5f
out 0x3d, r20
sei
main:
rjmp main ; application
timer:
    inc r0 ; timer overflow
        handler
reti
Case Study: AVR (handler programming)

```c
#define IRQ0 __vector_1
#define SIGNAL __attribute__((signal))

int main (){  
    while(1);  
    return 0;  
}

void IRQ0(void) SIGNAL;

void IRQ0(void)  
{  
    PORTB = ~PORTB;  
}
```