Design with Microprocessors

Year III Computer Science
1-std Semester

Lecture 4: Input/output and interrupts for Arduino systems
Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

Development IDE: C language programming

Resources / references:

Arduino official site: http://arduino.cc/

Tutorials, examples, components:
  http://www.tehnorama.ro/arduino/
  http://www.robofun.ro/

Books:
## Development boards

<table>
<thead>
<tr>
<th></th>
<th>UNO</th>
<th>Mega</th>
<th>Leonardo</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clock freq.</strong></td>
<td>16Mhz</td>
<td>16 Mhz</td>
<td>16 Mhz</td>
<td>84 Mhz</td>
</tr>
<tr>
<td><strong>Microcontroller</strong></td>
<td>ATmega328</td>
<td>ATmega2560</td>
<td>ATmega32U4</td>
<td>AT91SAM3X8E</td>
</tr>
<tr>
<td><strong>No. Digital I/O pins</strong></td>
<td>11</td>
<td>52</td>
<td>11</td>
<td>52</td>
</tr>
<tr>
<td><strong>No. Analog I/O pins</strong></td>
<td>6</td>
<td>16</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td><strong>No. PWM pins</strong></td>
<td>6</td>
<td>14</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td><strong>No. Serial Ports</strong></td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Flash size</strong></td>
<td>32Kb</td>
<td>256Kb</td>
<td>32Kb</td>
<td>512Kb</td>
</tr>
<tr>
<td><strong>Internal voltage</strong></td>
<td>5V</td>
<td>5V</td>
<td>5V</td>
<td>3.3V</td>
</tr>
<tr>
<td><strong>Mouse &amp; KB emulation</strong></td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Main advantage over Cerebot boards: **power supply and programming / debugging over USB cable** (standard USB A-B printer cable)
Arduino UNO (rev. 3)

Digital I/O pins are connected to the µC pins/ports

Analogue pines are inputs for the 10 bit ADC of the µC.

• 3V3 & 5V pins are outputs !!! (supplying a regulated voltage)
• IOREF (out) – ref. voltage for shields
• AREF (in) – external ref. voltage for the ADC

Some pins have special functions:

• Serial comm: 0 (RX) and 1 (TX) – used for board programming via USB - avoid them !!!
• External Interrupts: 2 (INT0) and 3 (INT1)
• PWM (8 bit): 3, 5, 6, 9, 10, 11.
• SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).
• TWI: A4 or SDA pin and A5 or SCL pin
Pin mapping for the Atmega8, 168, and 328 is identical.
Arduino MEGA (rev. 3)

Special function pins:
- External Interrupts: 2 (INT 0), 3 (INT 1), 18 (INT 5), 19 (INT 4), 20 (INT 3), and 21 (INT 2)
- PWM: 2 to 13 and 44 to 46
- LED: 13

Special function (serial comm):
- Serial: 0 (RX) and 1 (TX);
  Serial 1: 19 (RX) and 18 (TX);
  Serial 2: 17 (RX) and 16 (TX);
  Serial 3: 15 (RX) and 14 (TX)
- SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS)
- TWI: 20 (SDA) and 21 (SCL)
Arduino MEGA (rev. 3)

Pin mapping for the Atmega2560
http://arduino.cc/en/Hacking/PinMapping2560
Input / output for Arduino systems

The programming environment performs the PIN correspondence / mapping
The programming logic is PIN-number oriented

Digital I/O
• can be programed as an input or output, using `pinMode()`
• can be accessed as an input/output with `digitalWrite()`, and `digitalRead()` functions
• operate at 5 volts / can provide or receive a maximum of 40 mA
• an internal pull-up resistor (disconnected by default) of 20-50 kOhms
• some pins can have specialized functions ....

Analogue I/O
• read an analogue value (input): `analogRead()`
• generate a PWM signal (output): `analogWrite()` - PWM
• configure the reference voltage used for analog input - `analogReference()`

Input / output for Arduino systems

Example 1:
- Input: BTN connected to a digital pin
- Use of an explicit/external “pull down” resistor (BTN not pressed: input – “0”)
- Output: on-board LED (shared with pin 13)
Input / output for Arduino systems

Example 1:

```c
const int ledPin = 13; // choose the pin for the LED
const int inputPin = 2; // choose the input pin (for a pushbutton)

void setup() {
    pinMode(ledPin, OUTPUT); // declare LED as output
    pinMode(inputPin, INPUT); // declare pushbutton as input
}

void loop(){
    int val = digitalRead(inputPin); // read input value
    if (val == HIGH) // check if the input is HIGH
    {
        digitalWrite(ledPin, HIGH); // turn LED on if switch is pressed
    }
    else
    {
        digitalWrite(ledPin, LOW); // turn LED off
    }
}
```

Alternative code:

```c
void loop()
{
    digitalWrite(ledPin, digitalRead(inputPin));
}
```
Example 2:

- Input from a switch (no external resistor)
- Internal ‘Pull-Up’ logic of each pin is used (enabled)

```cpp
const int ledPin = 13;  // output pin for the LED
const int inputPin = 2; // input pin for the switch

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(inputPin, INPUT);
  digitalWrite(inputPin, HIGH); // turn on internal pull-up on the inputPin
}

void loop() {
  int val = digitalRead(inputPin); // read input value
  if (val == HIGH) // check if the input is HIGH
  {
    digitalWrite(ledPin, HIGH); // turn LED OFF
  }
  else
  {
    digitalWrite(ledPin, LOW); // turn LED ON
  }
}
```
Example 3: reading of unstable data (i.e. buttons without internal de-bounce logic [http://www.robofun.ro/bricks/buton-mare-brick](http://www.robofun.ro/bricks/buton-mare-brick))

- A mechanical contact can oscillate (*bounce*) between close and open state several times until reaches a stable state
- A µC is fast enough to ‘sense’ these oscillations – perceives them as repeated on/off states
- The oscillations can be compensated/filtered out by software

**Note:** the Digilent Pmod BTN has an intrinsic/hardware de-bounce logic implemented – no other software logic should be implemented !!!
Input / output for Arduino systems

Example 3: software de-bounce

- **Filtering principle**: check the input for several times until a stable level is encountered
- **Outcome**: bouncing period is ignored; input value is validated only when is stable

```c
const int inputPin = 2;  // Time delay (ms) after which input signal should be stable
const int ledPin = 13;
const int debounceDelay = 10;

void setup()
{
    pinMode(inputPin, INPUT);
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    if (debounce(inputPin))
    {
        digitalWrite(ledPin, HIGH);
    }
}
```

```c
boolean debounce(int pin)
{
    boolean state;
    boolean previousState;
    previousState = digitalRead(pin);  // store switch state
    for(int counter=0; counter < debounceDelay; counter++)
    {
        delay(1);  // wait for 1 millisecond
        state = digitalRead(pin);  // read the pin
        if( state != previousState )
        {
            counter = 0;  // reset the counter if the state changes
            previousState = state;  // and save the current state
        }
    }  // here when the switch state has been stable longer than the debounce period
    return state;
}
```
Example 4: keyboard

• Pressing one key: contact between the corresponding row and the column

• Default Row status = ‘1’ (pull-up logic enabled)

• If a key is pressed:
  • If corresponding Column = ‘0’ ⇒ corresponding Row = ‘0’
  • Else (Column = ‘1’) ⇒ nothing happens (Row = ‘1’ – pull-up logic)

**Keyboard scanning principle**

• Each column is activated in a sequential order (set to ‘0’). Read the status of the rows

• Columns pins - configured as outputs. Row pins - configured as inputs

<table>
<thead>
<tr>
<th>Arduino pin</th>
<th>Keypad connector</th>
<th>Keypad row/column</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>Row 1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>Row 2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>Column 2</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Column 0</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>Row 3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>Row 0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>Column 1</td>
</tr>
</tbody>
</table>
Input / output for Arduino systems

Example 4: keyboard

```c
const int numRows = 4;    // number of rows in the keypad
const int numCols = 3;    // number of columns
const int debounceTime = 20; // number of milliseconds for switch to be stable

// this array determines the pins used for rows and columns
const int row Pins[numRows] = { 7, 2, 3, 6 }; // Rows 0 through 3
const int col Pins[numCols] = { 5, 8, 4 }; // Columns 0 through 2

// keymap defines the character returned when the corresponding key is pressed
const char keymap[numRows][numCols] = {
    { '1', '2', '3' },
    { '4', '5', '6' },
    { '7', '8', '9' },
    { '*', '0', '#' }
};

void setup()
{
    Serial.begin(9600);
    for (int row = 0; row < numRows; row++)
    {
        pinMode(row Pins[row], INPUT);    // Set row pins as input
        digitalWrite(row Pins[row], HIGH); // turn on Pull-ups
    }
    for (int column = 0; column < numCols; column++)
    {
        pinMode(col Pins[column], OUTPUT);    // Set column pins as outputs
        digitalWrite(col Pins[column], HIGH); // for writing
    }
}
```
Example 4: keyboard

```c
void loop()
{
    char key = getKey();
    if( key != 0) { // if the character is not 0 then
        // it’s a valid key press
        Serial.print("Got key ");
        Serial.println(key);
    }
}

// returns with the key pressed, or 0 if no key is pressed
char getKey()
{
    char key = 0; // 0 indicates no key pressed

    for(int column = 0; column < numCols; column++)
    {
        digitalWrite(colPins[column],LOW); // Activate the current column.
        for(int row = 0; row < numRows; row++) // Scan all rows for
            // a key press.
        {
            if(digitalRead(rowPins[row]) == LOW) // Is a key pressed?
            {
                delay(debounceTime); // debounce
                while(digitalRead(rowPins[row]) == LOW)
                    // wait for key to be released
                    key = keymap[row][column]; // Remember which key
                                          // was pressed.
            }
        }
        digitalWrite(colPins[column],HIGH); // De-activate the current column.
    }
    return key; // returns the key pressed or 0 if none
}
```
Example 4: keyboard

```c
void loop()
{
    char key = getKey();
    if( key != 0 ) { // if the character is not 0 then
        // it's a valid key press
        Serial.print("Got key ");
        Serial.println(key);
    }
}

// returns with the key pressed, or 0 if no key is pressed
char getKey()
{
    char key = 0; // 0 indicates no key pressed

    for(int column = 0; column < numCols; column++)
    {
        digitalWrite(colPins[column],LOW); // Activate the current column.
        for(int row = 0; row < numRows; row++) // Scan all rows for
            // a key press.
        {
            if(digitalRead(rowPins[row]) == LOW) // Is a key pressed?
            {
                delay(debounceTime); // debounce
                while(digitalRead(rowPins[row]) == LOW) // wait for key to be released
                {
                    delay(debounceTime);
                }
                key = keymap[row][column]; // Remember which key
                // was pressed.
            }
        }
        digitalWrite(colPins[column],HIGH); // De-activate the current column.
    }
    return key; // returns the key pressed or 0 if none
```
External Interrupts for Arduino systems

Used for detecting external events (without polling with digitalRead() )

<table>
<thead>
<tr>
<th>Board</th>
<th>int.0</th>
<th>int.1</th>
<th>int.2</th>
<th>int.3</th>
<th>int.4</th>
<th>int.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uno, Ethernet</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mega2560</td>
<td>2</td>
<td>3</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>Leonardo</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

To use an interrupt an ISR (Interrupt Service routine should be attached:

attachInterrupt(interrupt, ISR, mode)

Parameters

interrupt: the number of the interrupt (int)

ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing.

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values:

- LOW to trigger the interrupt whenever the pin is low,
- CHANGE to trigger the interrupt whenever the pin changes value
- RISING to trigger when the pin goes from low to high,
- FALLING for when the pin goes from high to low.

To de-attach an interrupt: detachInterrupt(interrupt_no

To deactivate all interrupts use: noInterrupts(). To reactivate all interrupts use": interrupts(). Interrupts are activated by default! Deactivation should be done only temporary!
External Interrupts for Arduino systems

Example 5: toggles the status of the on-board LED when a BTN is pressed using interrupts

```c
int outputLED = 13;
volatile int stateLED = LOW; //current status of the LED

// the setup routine runs once when you press reset:
void setup() {
    pinMode(outputLED, OUTPUT);
    attachInterrupt(0, isr0, RISING); // init ISR0 (pin 2) and behavior
}

void isr0() {
    stateLED = !stateLED; //on ISR toggle the LED status
    if (stateLED == 1)
        digitalWrite(outputLED, HIGH);
    else
        digitalWrite(outputLED, LOW);
}

// the loop routine runs over and over again forever:
void loop() {
}
```

- All variables that are used in a ISR should be declared as “volatile”. The compiler will know that these variable can be modified at any time and shall not optimize the code by attaching them to registers, but instead will store them always into the RAM
- Only one ISR can run at a time (others are deactivated)
Example 6: measuring the pulse width of a digital signal (i.e. signal received from an IR remote controller – the pulse width is ‘translated’ as a ‘0’ or a ‘1’)

```cpp
const int irReceiverPin = 2;       // pin the receiver is connected to
const int numberOfEntries = 64;    // set this number to any convenient value

volatile unsigned long microseconds; // No. of microseconds elapsed from program start
volatile byte index = 0;            // Position in the transitions string
volatile unsigned long results[numberofEntries]; // Transitions string (contains the duration of the transitions) - output

void setup()
{
    pinMode(irReceiverPin, INPUT);
    Serial.begin(9600);
    attachInterrupt(0, analyze, CHANGE); // encoder pin on interrupt 0 (pin 2);
    results[0]=0;
}

void loop()
{
    if(index >= numberOfEntries) // Check if the max. no. of transitions is reached
    {
        Serial.println("Durations in Microseconds are:");
        for( byte i=0; i < numberOfEntries; i++)
        {
            Serial.println(results[i]);
        }
        index = 0; // start analyzing again
    }
    delay(1000);
}
Example 6: measuring the pulse width of a digital signal (i.e. signal received from an IR remote controller – the pulse width is ‘translated’ as a ‘0’ or a ‘1’)

```c
void analyze() // ISR
{
    if(index < numberOfEntries ) // If not end of transitions string
    {
        if(index > 0) // If not first transition
        {
            results[index] = micros() - microseconds;
        }
    }
    index = index + 1; // Advance the index
    microseconds = micros(); // Mem. current time (reference for the next transition)
}
```

• **micros()** – returns the number of microseconds elapsed from program start
• **[millis()]** - returns the number of microseconds elapsed from program start

**Homework:** Write the code for example 5 using the polling technique (no interrupts)

**References:**
Arduino documentation: [http://arduino.cc/](http://arduino.cc/)