Computer Architectures and Arduino

Embedded Real Time Systems
Prof. Davide Brugali
Università degli Studi di Bergamo
Feedback control loop

Sensor → Embedded computer → Actuator
Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt
Microcontroller-Based Embedded Computer

ROM or Flash to hold program
Flash ~256 KB

ARM Core
~ 100 MHz

~100 KB

Disk drives, audio, analog in, analog out, parallel (digital) I/O, etc.

Non-volatile memory, volatile memory, clock/calendars, analog in/analogue out
Non-volatile memory, clock/calendars, digital sensors, analog in/analogue out

Network interfaces
Host computers

I/O

Memory

Bus interface

Counter timers

RAM

Serial port (UART)

Digital I/O

CPU

SPI

ROM

Analog in (ADC)

Sensors

Switches, buttons, motors, on/off controls, digital interfacing

Host computer

Simple networks, other computers, Infrared communications (IrDA)

Temperature, light, acceleration, vibration, humidity, pressure, magnetic field
Large Embedded Systems

- Embedded doesn’t mean small

- Large embedded systems such as network switches and routers don’t use microcontrollers
  - CPU is typically in 100’s of MHz
  - 500MB-1 GB RAM
  - 250-500 MB Flash
  - Power consumption could be an issue

- They are embedded systems like their smaller counterparts in the sense that they perform a single well-defined set of tasks
The Arduino Development Board

A printed circuit board designed to facilitate work with a particular microcontroller.
Arduino motor controller

www.TheEngineeringProjects.com
Arduino programming
// Name of sketch
// Brief Description
// Date:

void setup()
{
    // put your setup code here, to run once:
}

void loop()
{
    // put your main code here, to run repeatedly:
}
Using Arduino

• Write your sketch

• Press Compile button

• Press Upload button to transfer the program to the Arduino board

• The program starts right after
Arduino input / output

GND
```
int pinLED = 8;
int pinBUTTON = 12;
int buttonState;

void setup() {
    Serial.begin(9600);
    pinMode(pinBUTTON, INPUT);
    pinMode(pinLED, OUTPUT);
}

void loop() {
    buttonState = digitalRead(pinBUTTON);

    if(buttonState == HIGH)
        digitalWrite(pinLED, HIGH);
    else
        digitalWrite(pinLED, LOW);

    delay(500);
}
```
```cpp
int buttonState;

void loop() {
  buttonState = digitalRead(pinBUTTON);
  if(buttonState == HIGH)
    digitalWrite(pinLED, HIGH);
  else
    digitalWrite(pinLED, LOW);
  delay(500);
}
```
Arduino Timing

• *delay*(ms)
  – Pauses for a few milliseconds

• *delayMicroseconds*(us)
  – Pauses for a few microseconds
Periodic tasks: Blinking LEDs

- Make blinking three different LEDs, each one at a different frequency

  - **Led1**: every 3s
  - **Led2**: every 7s
  - **Led3**: every 11s
Blinking leds – version 1

```c
int led1 = 11; int led2 = 12; int led3 = 13; // pin numbers
int count = 0;

void setup() {
    pinMode(led1, OUTPUT);
    pinMode(led2, OUTPUT);
    pinMode(led3, OUTPUT);
}

void loop() {
    if(count%3 == 0)
        digitalWrite(led1);
    if(count%7 == 0)
        digitalWrite(led2);
    if(count%11 == 0)
        digitalWrite(led3);
    count++;
    if(count == 3*7*11)
        count = 0;
    delay(1000);
}
```
void loop() {
    c = 0;
    if(count%3 == 0) {
        digitalWrite(led1, HIGH);
        delay(100); c += 100;
        digitalWrite(led1, LOW);
    }
    if(count%7 == 0) {
        digitalWrite(led2, HIGH);
        delay(200); c += 200;
        digitalWrite(led2, LOW);
    }
    if(count%11 == 0) {
        digitalWrite(led3, HIGH);
        delay(300); c += 300;
        digitalWrite(led3, LOW);
    }
    count++;
    if(count == 3*7*11) {
        count = 0;
    }
    delay(1000-c);
}
Blinking leds – version 2

Led 1

Led 2

Led 3
Blinking leds – version 3

Led 1: HIGH

Led 2: LOW, LOW

Led 3: LOW
void loop() {
    c1 = 0; c2 = 0; c3 = 0; d1 = 0; d2 = 0; d3 = 0;
    if(count%3 == 0) {
        c1 = 100;
        digitalWrite(led1, HIGH);
    }
    if(count%7 == 0) {
        c2 = 200;
        digitalWrite(led2, HIGH);
    }
    if(count%11 == 0) {
        c3 = 300;
        digitalWrite(led3, HIGH);
    }
    if(count%3 == 0) { d1 = c1;
        delay(d1);
        digitalWrite(led1, LOW);
    }
    if(count%7 == 0) { d2 = c2-d1;
        delay(d2);
        digitalWrite(led2, LOW);
    }
    if(count%11 == 0) { d3 = c3 - d2 - d1;
        delay(d3);
        digitalWrite(led3, LOW);
    }
    count++;
    if(count == 3*7*11)
        count = 0;
    delay(1000-c1-c2-c3);
}
Arduino Interrupts
Control Flow in Absence of Intermits

• Processors do only one thing:
  • From startup to shutdown, a CPU simply reads and executes (interprets) a sequence of instructions, one at a time
  • This sequence is the CPU’s control flow
Interrupt-Driven I/O

- External hardware alerts the processor that input is ready
- Processor suspends what it is doing
- Processor invokes an interrupt service routine (ISR)
- ISR interacts with the application concurrently

```c
interrupt_routine() {
    ...
}
```

```
<startup>
inst_1

inst_2

...

inst_n

<shutdown>
```
Interrupt-Driven I/O

- External hardware alerts the processor that input is ready
- Processor suspends what it is doing
- Processor invokes an interrupt service routine (ISR)
- ISR interacts with the application concurrently
Using Interrupts

• On an Arduino UNO board, two pins can be used as interrupts: pins 2 and 3.
• The interrupt is enabled through the following line:
  • attachInterrupt(interrupt, function, mode)
    • attachInterrupt(0, doEncoder, FALLING);
Interrupt example

```c
int pin_led = 12;
int pin_button = 2;  // interrupt 1

volatile int state = LOW;

void setup()
{
    pinMode(led, OUTPUT);
    attachInterrupt(1, blink, CHANGE);
}

void loop()
{
    // ... do something
}

void blink()
{
    if(state == LOW) state = HIGH;
    else state = LOW;
    digitalWrite(pin_led, state);
}
```
Timer functions (timer.h library)

- **int every(long period, callback)**
  - Run the 'callback' every 'period' milliseconds. Returns the ID of the timer event.

- **int every(long period, callback, int repeatCount)**
  - Run the 'callback' every 'period' milliseconds for a total of 'repeatCount' times. Returns the ID of the timer event.

- **int after(long duration, callback)**
  - Run the 'callback' once after 'period' milliseconds. Returns the ID of the timer event.
Timer functions (timer.h library)

• int oscillate(int pin, long period, int startingValue)
  • Toggle the state of the digital output 'pin' every 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.

• int oscillate(int pin, long period, int startingValue, int repeatCount)
  • Toggle the state of the digital output 'pin' every 'period' milliseconds 'repeatCount' times. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.
Timer functions (Timer.h library)

- **int pulse(int pin, long period, int startingValue)**
  - Toggle the state of the digital output 'pin' just once after 'period' milliseconds. The pin's starting value is specified in 'startingValue', which should be HIGH or LOW. Returns the ID of the timer event.

- **int stop(int id)**
  - Stop the timer event running. Returns the ID of the timer event.

- **int update()**
  - Must be called from 'loop'. This will service all the events associated with the timer.
#include "Timer.h"

Timer t;

int ledEvent;

void setup() {
    Serial.begin(9600);

    int tickEvent = t.every(2000, doSomething);

    pinMode(13, OUTPUT);
    ledEvent = t.oscillate(13, 50, HIGH);
    Serial.print("LED event started id=");
    Serial.println(ledEvent);

    int afterEvent = t.after(10000, doAfter);
    Serial.print("After event started id=");
    Serial.println(afterEvent);
}

void loop() {
    t.update();
}

void doSomething() {
    Serial.print("2 second tick: millis()=");
    Serial.println(millis());
}

void doAfter() {
    Serial.println("stop the led event");
    t.stop(ledEvent);
    t.oscillate(13, 500, HIGH, 5);
}
Arduino Robot
Robot BART

- Motors with Encoders
- SONAR sensors
- Line color sensor
Motor shield MD25
Motor shield MD25

- Processor
- PWM
- Motor
- H-BRIDGE circuit
- Encoder

 Connections:
- Processor to Motor
- Encoder to Processor
- Current flow from Processor to Motor
- 1 bit tick from Processor to Encoder
PWM motor control

- The speed of a DC motor is directly proportional to the supply voltage, so if we reduce the supply voltage from 12 Volts to 6 Volts, the motor will run at half the speed.
- How can this be achieved when the battery is fixed at 12 Volts? The speed controller works by varying the average voltage sent to the motor. It could do this by simply adjusting the voltage sent to the motor, but this is quite inefficient to do.
- A better way is to switch the motor's supply on and off very quickly. If the switching is fast enough, the motor doesn't notice it, it only notices the average effect.
Duty cycle

desiredDutyCycle = 40%

setDuty();
resetDuty();
Line Finder
Line following
Line following
void loop() {
    dx = digitalRead(rightPin);  sx = digitalRead(leftPin);

    if (dx == WHITE && sx == WHITE) {
        robot.setSpeed_L(20);
        robot.setSpeed_R(20);
    }
    else if (dx == BLACK && sx == WHITE) {
        robot.setSpeed_L(20);
        robot.setSpeed_R(-12);
    }
    else if (dx == WHITE && sx == BLACK) {
        robot.setSpeed_L(-12);
        robot.setSpeed_R(20);
    }
    else {
        ?
    }
    delay(200);
}
Line following - delay
Ultrasonic sensor

- The “ping” sound pulse is generated when the pingPin level goes HIGH for two microseconds.
- The sensor will then generate a pulse that terminates when the sound returns.
- The width of the pulse is proportional to the distance the sound traveled.
- The speed of sound is 340 meters per second, which is 29 microseconds per centimeter.
- The formula for the distance of the round trip is: \( \text{RoundTrip} = \frac{\text{microseconds}}{29} \)
Ultrasonic sensor

Sender/Receiver

Object

reflected wave

original wave

distance $r$
Sonar Example

```cpp
#include "Ultrasonic.h"

Ultrasonic left_sonar(3, 2);  // left sonar connected to digital pin 3 and 2
Ultrasonic right_sonar(5, 4); // right sonar connected to digital pin 5 and 4

int d_dx, d_sx; // distances measured by the sonar sensors

void setup()   {
    Serial.begin(9600);
}

void loop()   {
    d_sx=left_sonar.Ranging(CM);
    Serial.print("LX: ");        Serial.print(d_sx);

    d_dx=right_sonar.Ranging(CM);
    Serial.print("  RX: ");      Serial.println(d_dx);

delay(1000);
```
Obstacle Avoidance

Segui linea

Aggira ostacolo

Obstacle

Segui linea

Aggira ostacolo

Aggira ostacolo

Aggira ostacolo

Segui linea
State machine

- Line following
  - No obstacle
  - Obstacle

- Obstacle avoidance
  - Obstacle
  - No obstacle
State machine

```c
int    status = 1;       bool   obstacle = false;
void   loop()  {
    d_sx=left_sonar.Ranging(CM);
    d_dx=right_sonar.Ranging(CM);
    switch(status) {
        case 1 : // line following
            if(  d_sx > 0   ||   d_dx > 0) : // trovato ostacolo
                obstacle = true ;
                status = 2;
            else
                lineFollowing(Ldx, Lsx);
            break;
        case 2 : // obstacle avoidance
            if(   ! obstacle)
                status = 1;
            else
                obstacle= obstacleAvoidance(d_sx, d_dx);
            break;
    }
    delay(200);
}
```