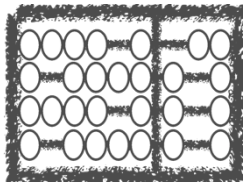
INF-741: Embedded systems programming for IoT

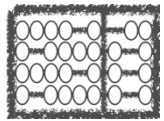
Prof. Edson Borin



Institute of Computing

University of Campinas

Embedded Software Architecture and Debugging



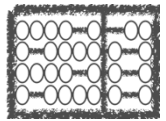
Agenda

Embedded Software Architecture

Embedded Software Debugging

Arduino Serial

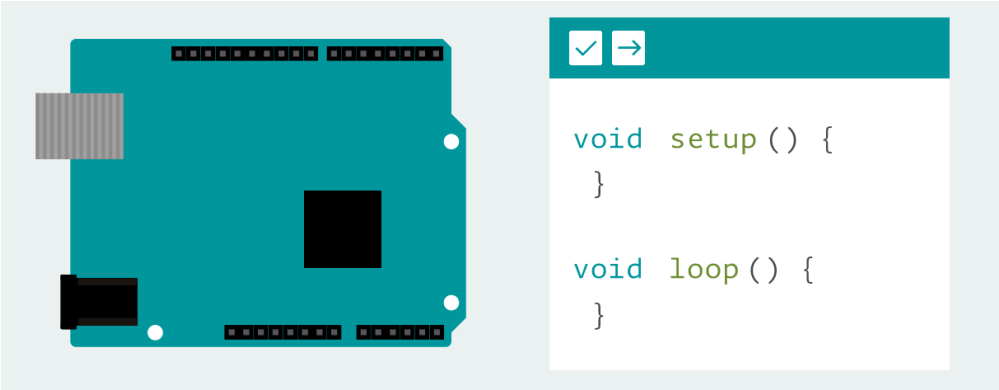
Lab Activities

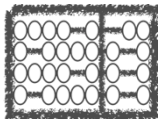


Embedded Software Architecture

Simple Control Loop

- Software is composed by an infinite loop.
- Loop may call subroutines to manage parts of the hardware or the software.
- Ex: Arduino setup/loop.

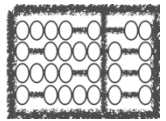




Embedded Software Architecture

Interrupt Controlled System

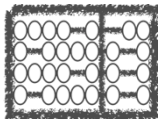
- System predominantly controlled by interrupts.
- Tasks are triggered by events, which are generated by interrupts.
  - Ex: The general purpose timer could be programmed to generate interrupts in a predefined frequency.
- The system may handle events on ISRs and run a single task in a main loop. The task in the main loop is usually not very sensitive to unexpected delays.
- Alternatively, the ISRs may add longer tasks to a queue structure and let the main loop execute these tasks.
- Ex: Arduino setup/loop + Interrupt Service Routines



Embedded Software Architecture

Cooperative Multitasking

- Programmer defines a series of tasks that are executed by the system.
- Tasks can yield control so that the system may run other tasks.
  - Control is usually yielded via API calls: “pause”, “wait”, “yield”, “stop”, “nop”, ...
- System relies on tasks giving up time to other tasks.
  - Yield may have to be carefully placed on tasks to ensure a smooth running system.
  - A poorly designed program may cause a task to consume all CPU time for itself (e.g. performing long calculations or I/O via busy waiting). It may cause the whole system to hang.
- Ex: Arduino runtime system for ESP8266.

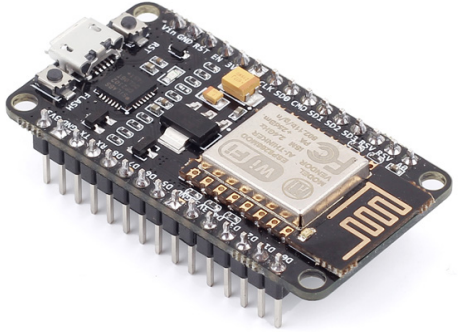


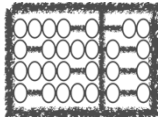
Embedded Software Architecture

Cooperative Multitasking – Arduino Runtime System for ESP8266

- Programming IDE similar to Arduino – Looks like a setup/loop system.
- May look like setup/loop, but it is a cooperative multitasking system.

```
112 static void loop_wrapper() {
113     static bool setup_done = false;
114     preloop_update_frequency();
115     if(!setup_done) {
116         setup();
117 #ifdef DEBUG_ESP_PORT
118         DEBUG_ESP_PORT.setDebugOutput(true);
119 #endif
120         setup_done = true;
121     }
122     loop();
123     run_scheduled_functions();
124     esp_schedule();
125 }
```





Embedded Software Architecture

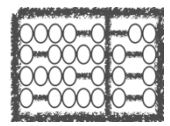
Cooperative Multitasking – Arduino Runtime System for ESP8266

- Programming IDE similar to Arduino – Looks like a setup/loop system.
- May look like setup/loop, but it is a cooperative multitasking system.

```
112  static void loop_wrapper() {
120          setup_done = true;
121      }
122      loop();
123      run_scheduled_functions();
124      esp_schedule();
125  }
```

Remember that there is a lot of code that needs to run on the chip besides the sketch when WiFi is connected. WiFi and TCP/IP libraries get a chance to handle any pending events each time the `loop()` function completes, OR when `delay` is called. If you have a loop somewhere in your sketch that takes a lot of time (>50ms) without calling `delay`, you might consider adding a call to `delay` function to keep the WiFi stack running smoothly.

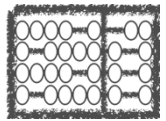
There is also a `yield()` function which is equivalent to `delay(0)`. The `delayMicroseconds` function, on the other hand, does not yield to other tasks, so using it for delays more than 20 milliseconds is not recommended.



Embedded Software Architecture

Preemptive Multitasking

- Switching controlled by system and does not rely on tasks yielding.
- System may switch between tasks (or threads) based on a timer (connected to an interrupt), for example.
  - Low-level code that switches between tasks may be considered an operating system kernel.
- Concurrent execution introduces several other challenges (re-entrant code, data/memory consistency, synchronization, etc...)
  - Typically a preemptive multitasking system contains features to handle tackle these challenges, such as semaphores.
- Usually organized as Real Time Operating Systems (RTOS)



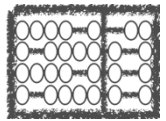
Agenda

Embedded Software Architecture

Embedded Software Debugging

Arduino Serial

Lab Activities



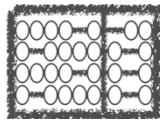
Embedded Software Debugging

Embedded systems usually lack operating systems, keyboards, monitors, ...

- hard to debug the system using desktop debugging techniques.

LED: turning on/off or even blinking LEDs may be used to monitor information and debug the system.

- Turning LED on/off may require changing the code, which may interfere with the bug.



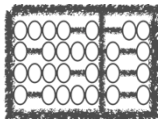
Embedded Software Debugging

printf: similar to using a LED, however, the amount of information that can be extracted per execution is larger.

- May require a more sophisticated output device (Serial, Monitor, etc...)
- Also requires changing the code, which may interfere with the bug.

simulation: Instead of executing the software on the hardware platform, the developer may run it using a Instruction Set Simulator (ISS), which emulates the hardware behavior.

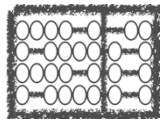
- The ISS is executed on a desktop and contains extensive support for debugging the embedded software – similar to debugging desktop applications.



Embedded Software Debugging

JTAG standard: standard to access (read/write) sub-blocks of integrated circuits.

- Used as a transport mechanism to inspect and set state during execution – Can be integrated with a GUI.



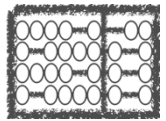
Agenda

Embedded Software Architecture

Embedded Software Debugging

Arduino Serial

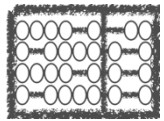
Lab Activities



Arduino Serial

Serial is used for communication between the Arduino board and another device (eg. a computer or another Arduino).

All Arduino boards have at least one serial port (aka UART or USART)

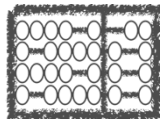


Arduino Serial - Setup

The serial interface must be initialized by defining the speed of communication in bits per second (Baud Rate).

Common arduino baud rates are: 300, 600, 1200, 2400, 4800, **9600**, 14400, 19200, 28800, 38400, 57600, and **115200**.

```
void setup()
{
    // opens serial port setting data rate to 9600 bps
    Serial.begin(9600);
}
void loop() { }
```



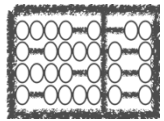
Arduino Serial – Printing data on the terminal

To prints data to the serial port as human-readable ASCII text use the function Serial.print( ).

Serial.println( ) works like Serial.print( ) but inserts the return character after the text.

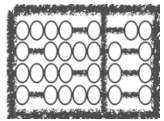
The optional second parameter formats numbers.

- Indicating the base: BIN (binary), OCT (octal), DEC (decimal), HEX (hexadecimal).
- For floating point numbers, this parameter specifies the number of decimal places to use.



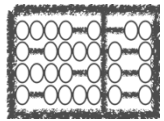
Arduino Serial – Printing data on the terminal

```
Serial.print(78);                    // prints "78"
Serial.print(1.23456);               // prints "1.23"
Serial.print('N');                   // prints "N"
Serial.print("Hello world.");        // prints "Hello world."

Serial.print(78, BIN);               // prints "1001110"
Serial.print(78, OCT);               // prints "116"
Serial.print(78, DEC);               // prints "78"
Serial.print(78, HEX);               // prints "4E"
Serial.print(1.23456, 0);            // prints "1"
Serial.print(1.23456, 2);            // prints "1.23"
Serial.print(1.23456, 4);            // prints "1.2346"

Serial.println();                    // prints a return character
Serial.print("\n");                  // prints a return character
Serial.print("\t");                  // prints a tabulation character
```



Arduino Serial – Sending raw data to the terminal

Use the function Serial.write() to send the sequence of bytes without formatting.



Arduino Serial – Reading data from the terminal

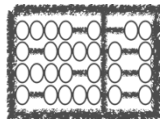
To read a data you first need to know if there is data to be read.

The function Serial.available() returns the number of bytes available for reading from the serial port.

Serial.read() returns the first byte of incoming serial data available (or -1 if no data is available)

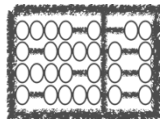
Serial.readString() reads characters from the serial buffer into a string. The function terminates if it times out.

Serial.setTimeout() sets the maximum milliseconds to wait for serial data.



Arduino Serial – Reading data from the terminal

```
// for incoming serial data
int incomingByte = 0;

void setup() {
    // opens serial port setting data rate to 9600 bps
    Serial.begin(9600);
}

void loop() {
    // send data only when you receive data:
    if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
     }
}
```



Serial Monitor tool

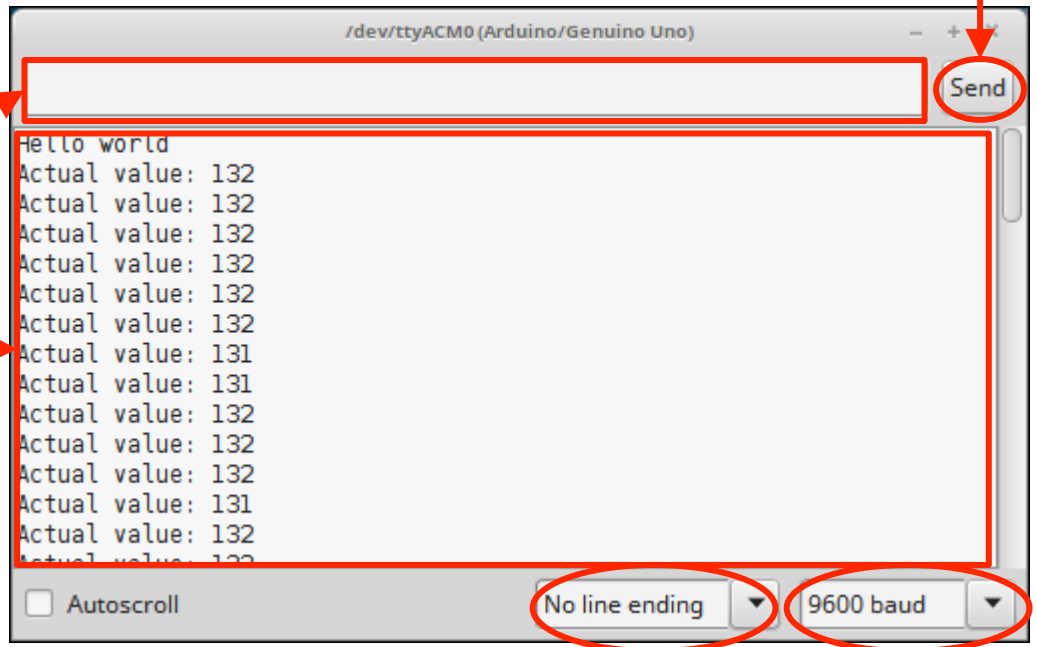
Send button

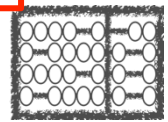
Text to send through Serial

Received text

Type of return character

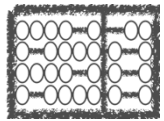
Baud rate selector





Debug function

```
void wait_to_continue() {
  // Print your variables here
  Serial.println("Send c to continue.");
  while(1) {
    if (Serial.available() > 0) {
      if (Serial.read() == 'c')
      return;
    }
  }
}
```



Software serial

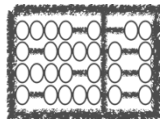
Arduino UNO has just one physical serial interface.

It’s used for communication between the board and the computer.

You may use two GPIO pins to emulate a serial interface using software.

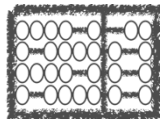
There is some limitations:

- Simultaneous data flows are not supported.
- Some pins and board restrictions.
- Read more in Arduino Software Serial reference.



Software serial example

```
#include <SoftwareSerial.h>

#define rxPin 10
#define txPin 11

// set up a new serial port
SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin);

void setup()  {
  // define pin modes for tx, rx:
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);

  // set the data rate for the SoftwareSerial port
  mySerial.begin(9600);
}

void loop() { }
```



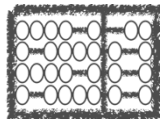
Agenda

Embedded Software Architecture

Embedded Software Debugging

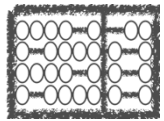
Arduino Serial

Lab Activities



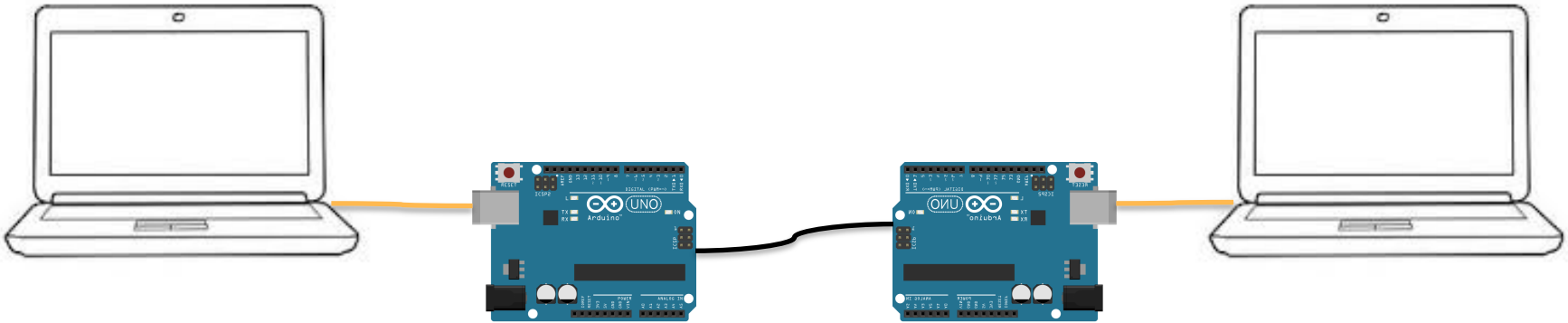
Lab 7: Encode terminal messages using morse code

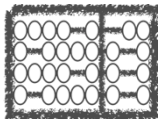
Modify your morse code sketch so that it encodes messages read from the terminal.



Lab 8: Arduino Chat

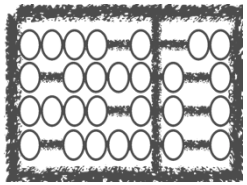
Use the software serial to connect two arduinos so that they echo messages from one PC terminal to another.





INF-741: Embedded systems programming for IoT

Prof. Edson Borin



Institute of Computing

University of Campinas