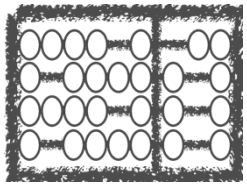
INF-741: Embedded systems programming for IoT

Prof. Edson Borin



Institute of Computing

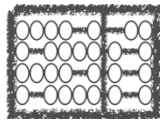
University of Campinas

Agenda

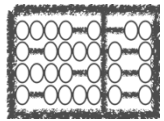
Memory Mapped I/O in C

GPIO

Lab Activities



Memory Mapped I/O in C



Memory Mapped I/O (MMIO) in C

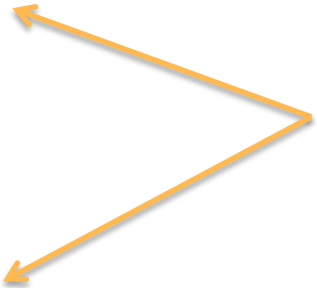
Recap: Memory-mapped I/O on ARM

Input:

```
ldr r0, =0x53FA0008
ldr r1, [r0]
```

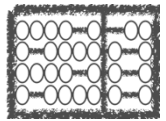
Output:

```
ldr r0, =0x53FA0000
str r1, [r0]
```



These addresses are mapped onto peripherals.

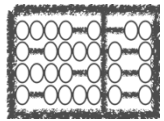
Programmer selects the addresses!



Memory Mapped I/O (MMIO) in C

C Code

```
// Make pointer point to I/O device
int* p = (int*) 0x53FA0008
// Read from I/O device mapped on this addr.
int  v = *p;
```

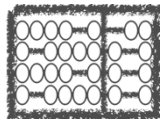


Memory Mapped I/O (MMIO) in C

C Code

```
// Make pointer point to I/O device
int* p = (int*) 0x53FA0008
// Read from I/O device mapped on this addr.
int  v = *p;
```

Cast to prevent compiler warnings!

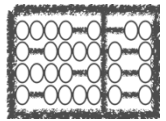


Memory Mapped I/O (MMIO) in C

C Code

```
// Make pointer point to I/O device
int* p = (int*) 0x53FA0008
// Read from I/O device mapped on this addr.
int  v = *p;
```

WARNING: Compiler thinks this pointer is pointing to memory and may assume the place it points to never changes unless the code stores to this position.

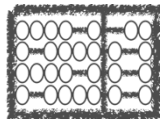


Memory Mapped I/O (MMIO) in C

C Code

```
// Make pointer point to I/O device
volatile int* p = (int*) 0x53FA0008
// Read from I/O device mapped on this addr.
int  v = *p;
```

You must inform the compiler that the value stored on the place this pointer points to may change, i.e., it is volatile!



Memory Mapped I/O (MMIO) in C

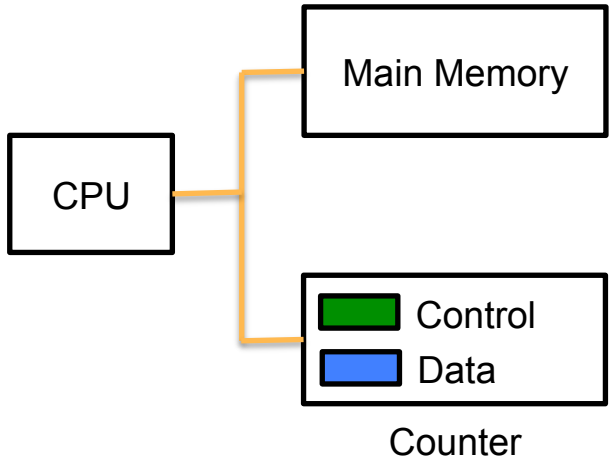
Example: Configuring and reading data from a peripheral

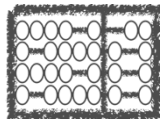
Peripheral = Counter

Increments counter every 100ms.

Registers:

- Control register (8 bits)
  - Configures the counter
- Data register (8 bits)
  - Returns the counter data!





Memory Mapped I/O (MMIO) in C

Example: Configuring and reading data from a peripheral

Peripheral = Counter

Control Register:

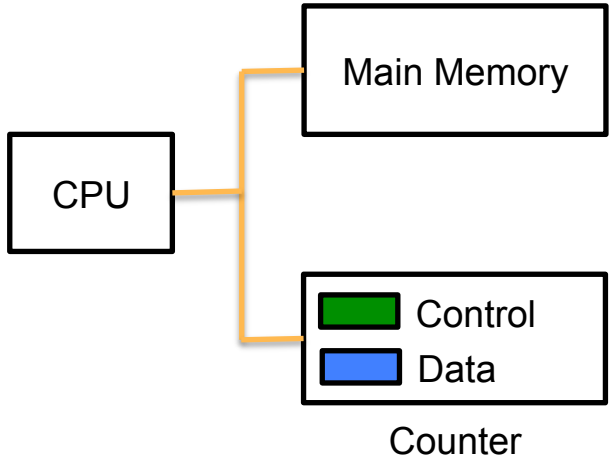
bit	7	6	5	4	3	2	1	0
	r	p	c	x	x	x	x	x

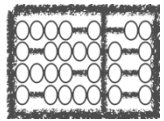
r = reset

p = pause

c = continue

Writing 1 to r resets the counter



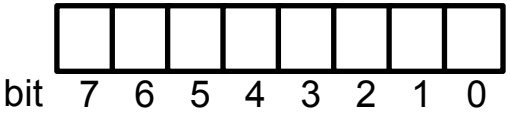


Memory Mapped I/O (MMIO) in C

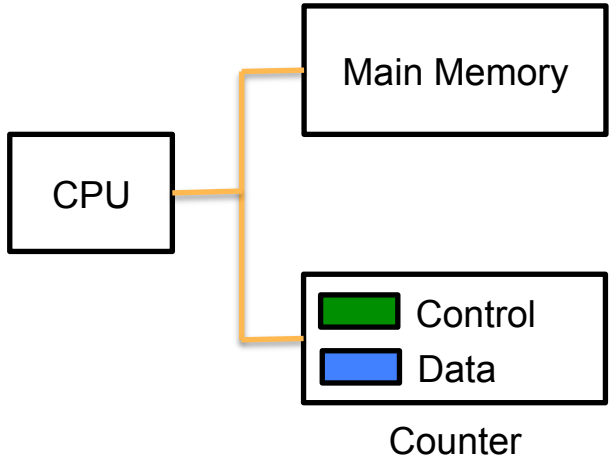
Example: Configuring and reading data from a peripheral

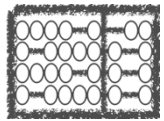
Peripheral = Counter

Data Register



Stores the counter value.





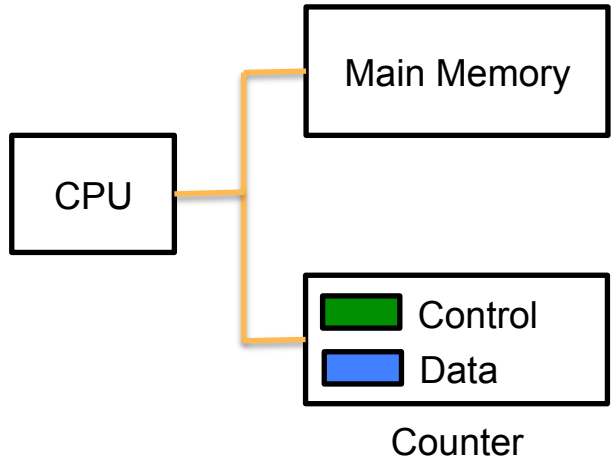
Memory Mapped I/O (MMIO) in C

Example: Configuring and reading data from a peripheral

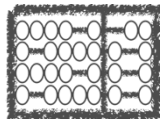
Assuming the control register is mapped to address 0x056

Write a function that resets the counter.

Hint: Your function must write 1 on bit 7 of the control register.



	r	p	c	x	x	x	x	x
bit	7	6	5	4	3	2	1	0



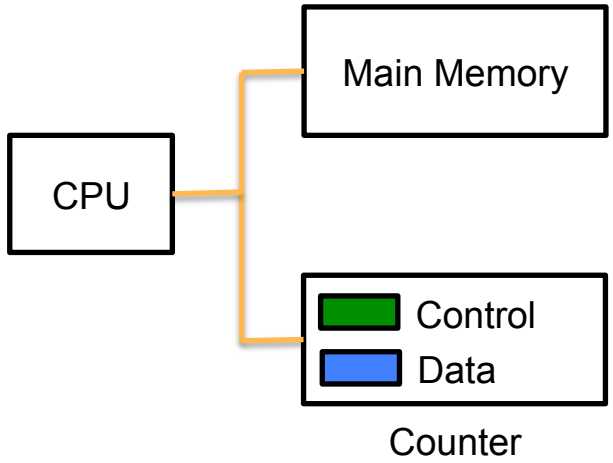
Memory Mapped I/O (MMIO) in C

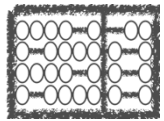
Example: Configuring and reading data from a peripheral

Assuming the control register is mapped to address 0x056

Write a function that resets the counter.

```
void reset_counter()
{
  volatile char* c = (char*) 0x056;
  *c = (1 << 7);
}
```





Memory Mapped I/O (MMIO) in C

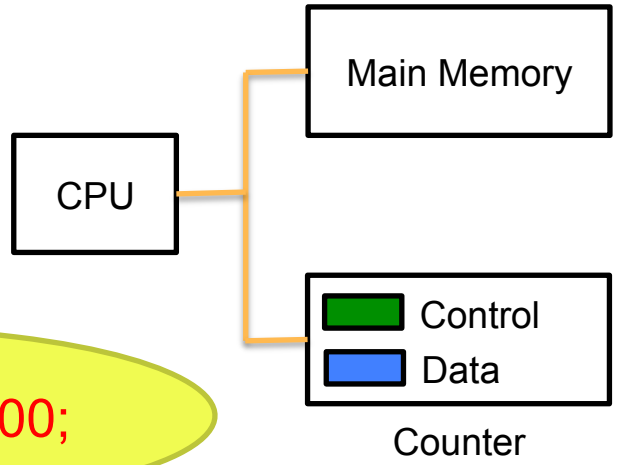
Example: Configuring and reading data from a peripheral

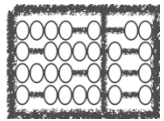
Assuming the control register is mapped to address 0x056

Write a function that resets the counter.

```
void reset_counter()
{
  volatile char* c = (char*) 0x056;
  *c = (1 << 7);
}
```

*c = 0b10000000;





Memory Mapped I/O (MMIO) in C

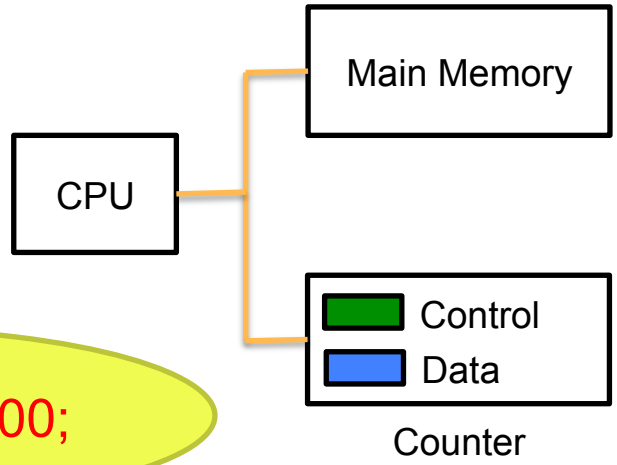
Example: Configuring and reading data from a peripheral

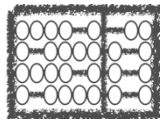
Assuming the control register is mapped to address 0x056

Function to **pause** the counter.

```
void pause_counter()
{
  volatile char* c = (char*) 0x056;
  *c = (1 << 6);
}
```

*c = 0b01000000;





Memory Mapped I/O (MMIO) in C

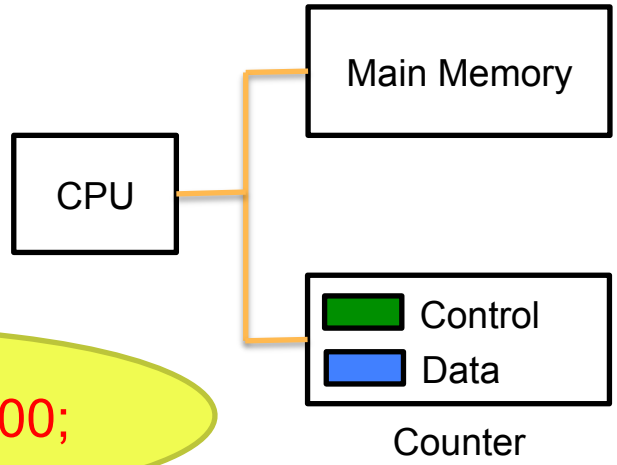
Example: Configuring and reading data from a peripheral

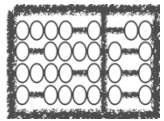
Assuming the control register is mapped to address 0x056

Function to **continue** the counter.

```
void continue_counter()
{
  volatile char* c = (char*) 0x056;
  *c = (1 << 5);
}
```

*c = 0b00100000;



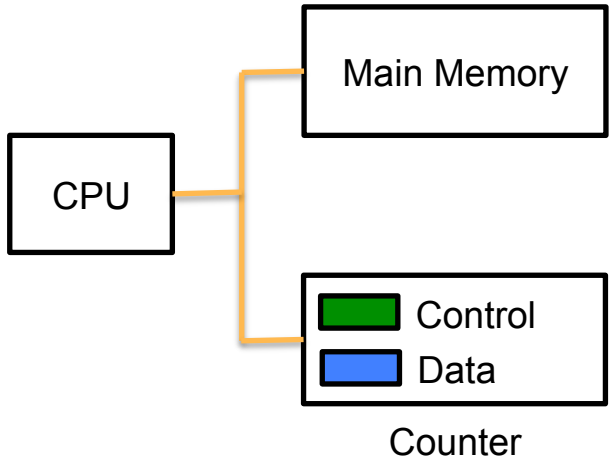


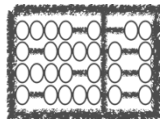
Memory Mapped I/O (MMIO) in C

Example: Configuring and reading data from a peripheral

Assuming the **data** register is mapped to address **0x057**

Write a function that reads the counter data





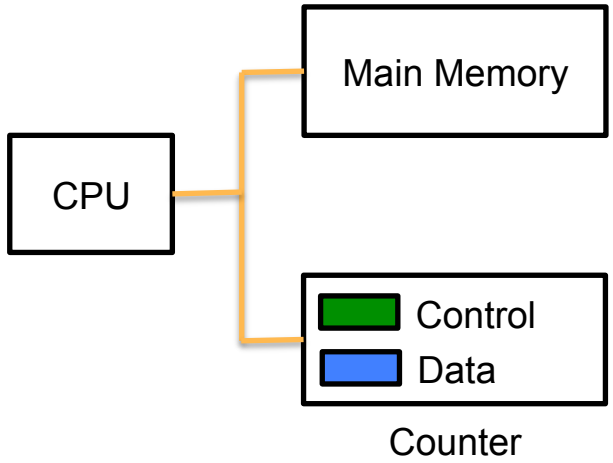
Memory Mapped I/O (MMIO) in C

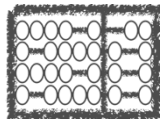
Example: Configuring and reading data from a peripheral

Assuming the **data** register is mapped to address **0x057**

Write a function that reads the counter data

```
char read_counter()
{
  volatile char* c = (char*) 0x057;
  return *c;
}
```



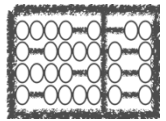


Agenda

Memory Mapped I/O in C

GPIO

Lab Activities

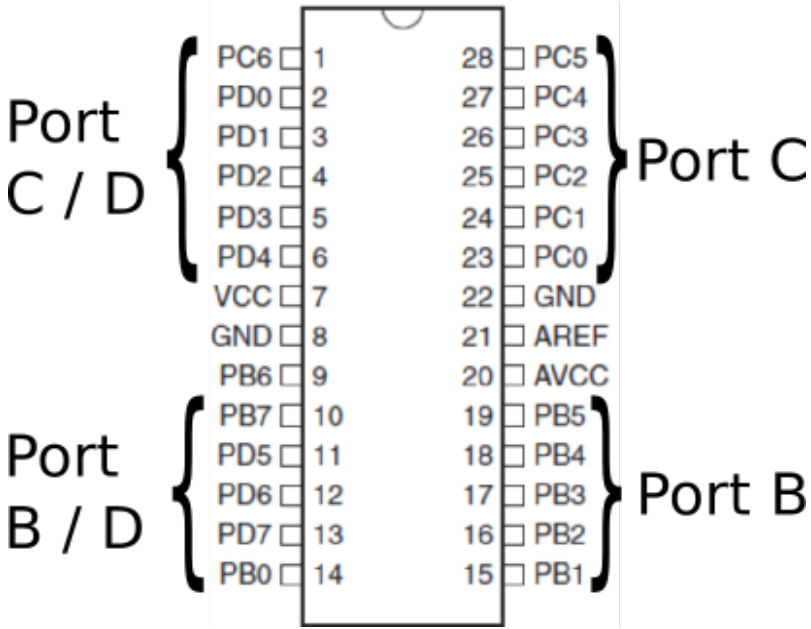


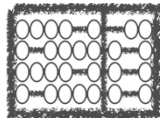
General-purpose input/output (GPIO)

GPIO is a interface to interact with external components, like sensors and actuators.

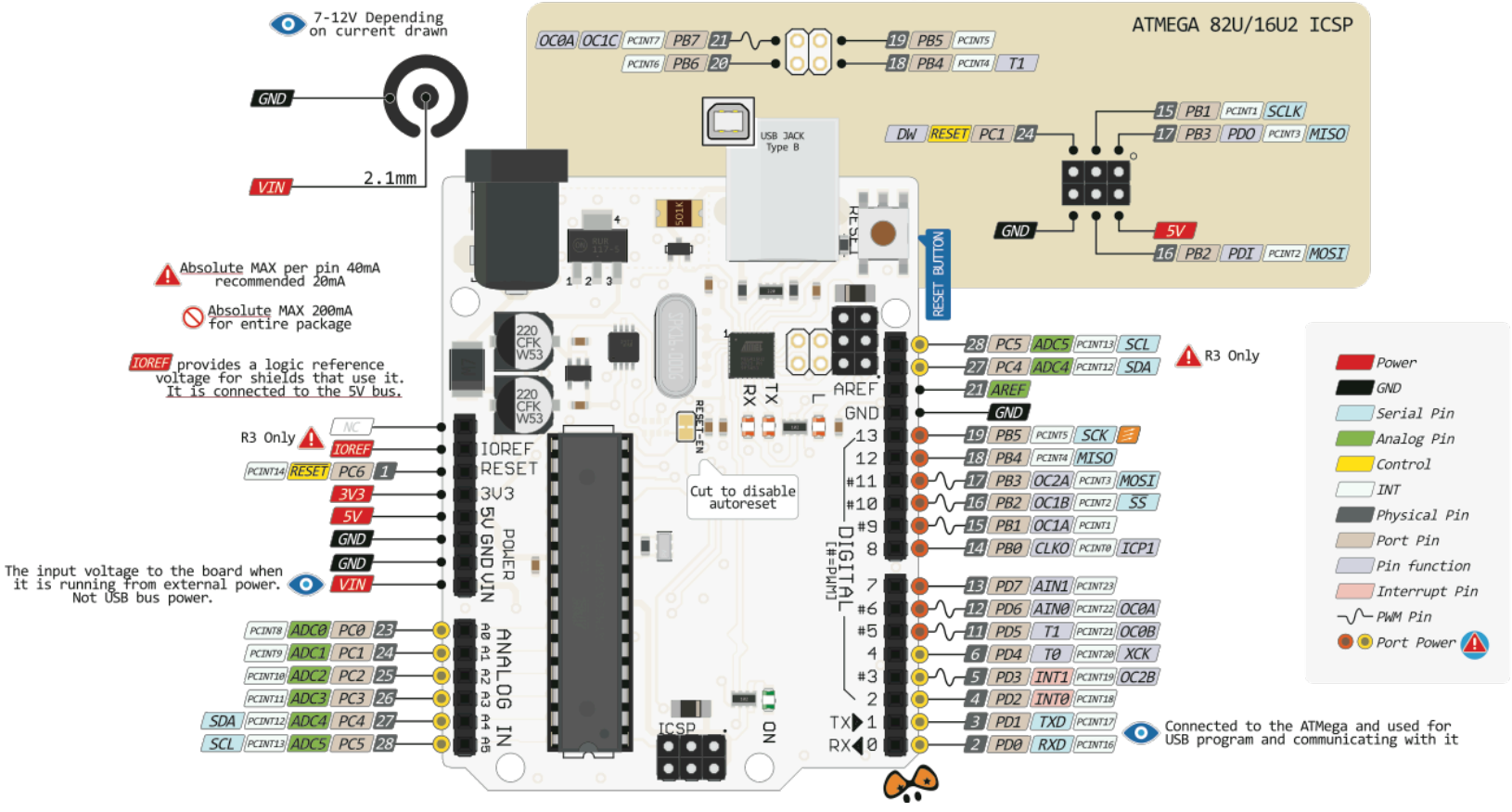
The ATMega328 microcontroller has 2 8-bit I/O ports (B, D) and 1 7-bit I/O port (C).

Every port has three registers associated with it in order to control a particular pin

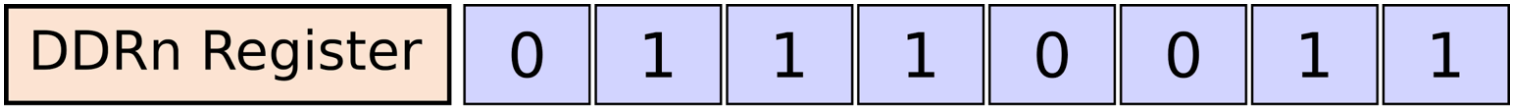




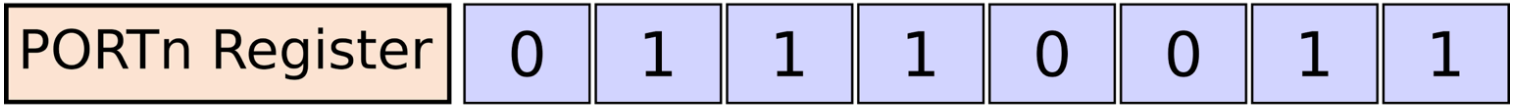
General-purpose input/output (GPIO)



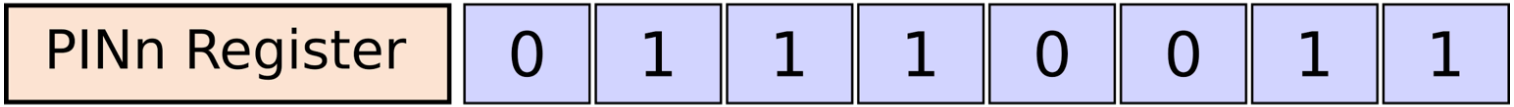
GPIO Registers



DDRn Configure a port as INPUT or OUTPUT

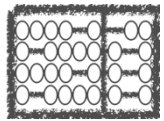


PORTn set the pin's value as LOW or HIGH



PINn stores the port input status

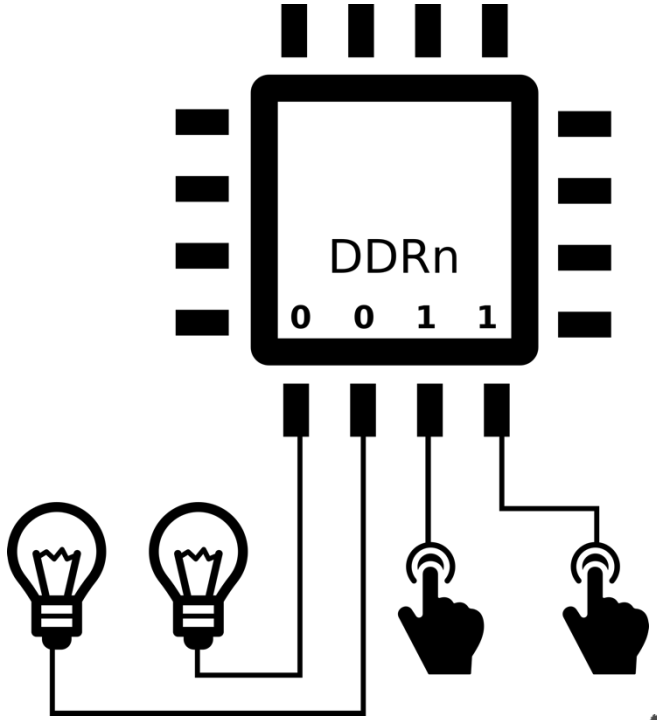
n- Indicates the port name i.e. A, B, C & D

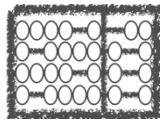


DDRn – Data Direction Register

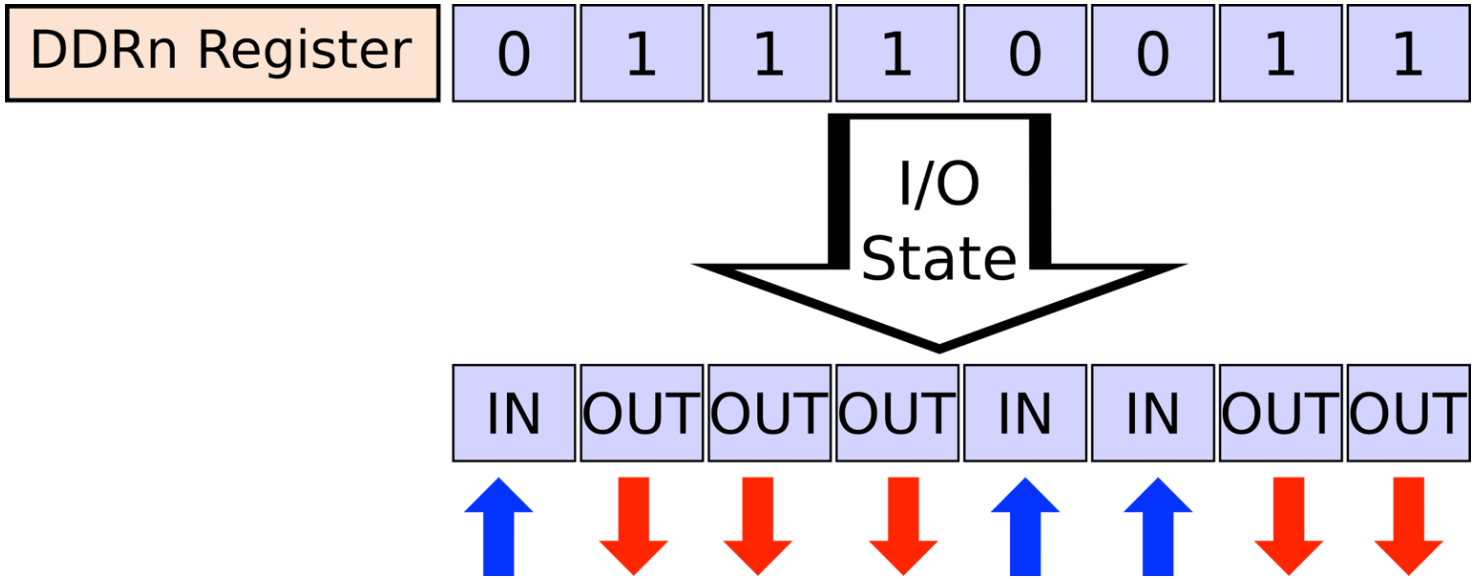
Before using the pins you must define wether it will be used as input or output. This is defined by the Data Direction Register (DDRn).

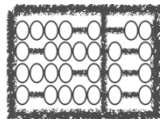
If you plug a LED on that pin, it must be set as an output pin, but if you plug a push button, it must be set as an input pin.





DDRn – Data Direction Register



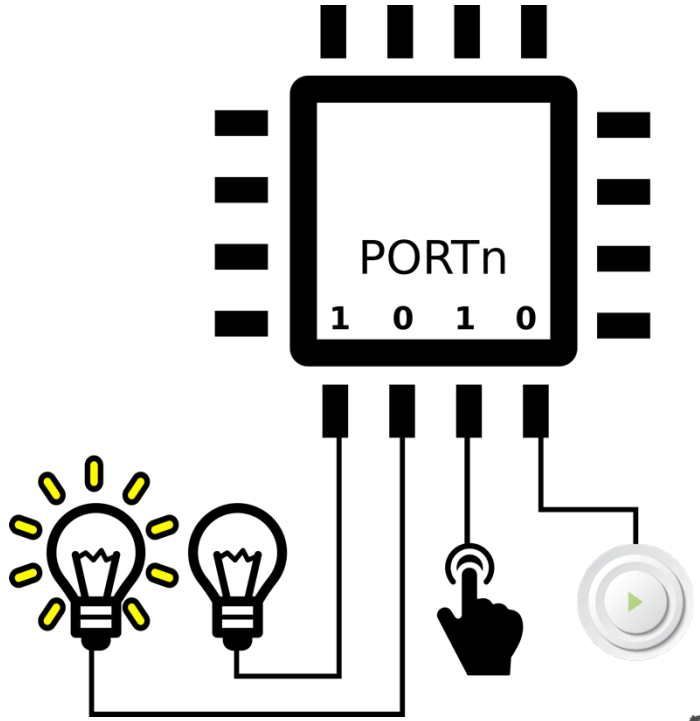


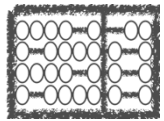
PORTn – Port Output data Register

The Data Register (PORTn) is used to set the output pin value.

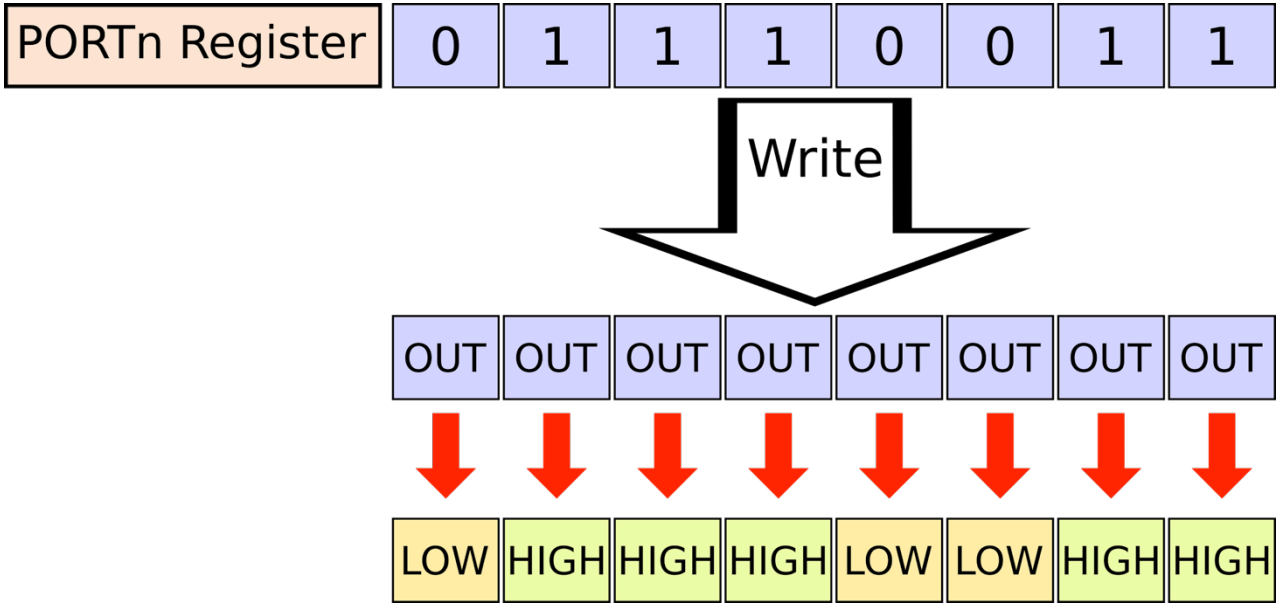
For example, to active a LED plugged in pin 0 of port B pin you should write the value 1 in the first bit of port B.

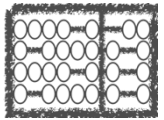
To keep the state of other pins, this register can be read, modified and then rewritten.





PORTn – Port Output data Register

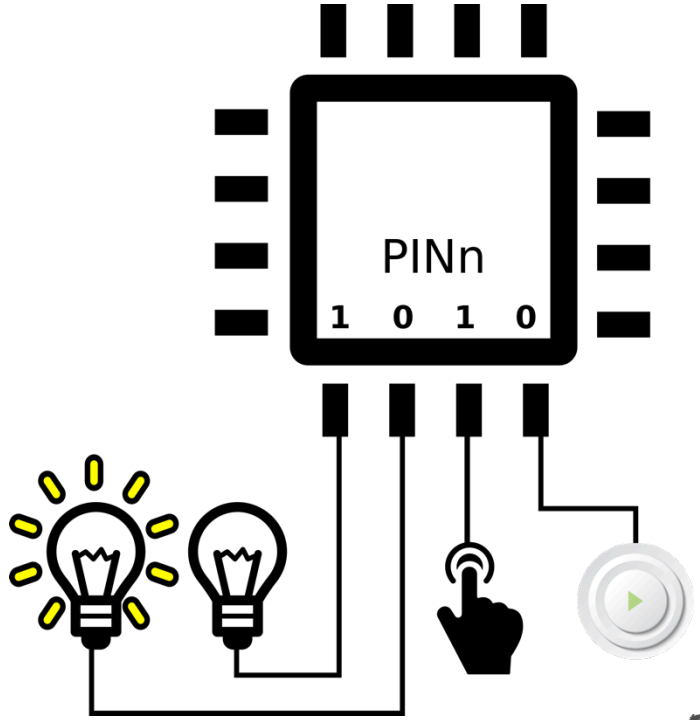


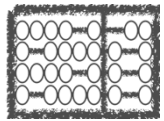


PINn – Port Input Register

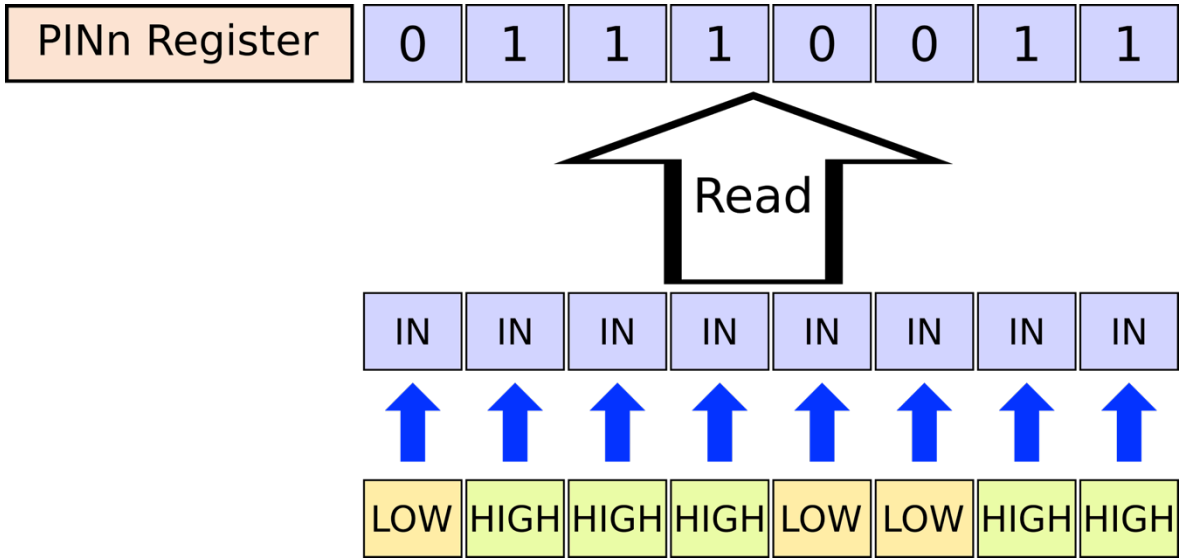
The Port Input Register (PINn) is used to read values from input pins.

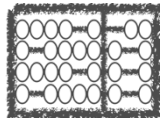
For example to read the current value of a button you read the PINn value for the related port then extract the desired bit using a bit mask.





PINn – Port Input Register



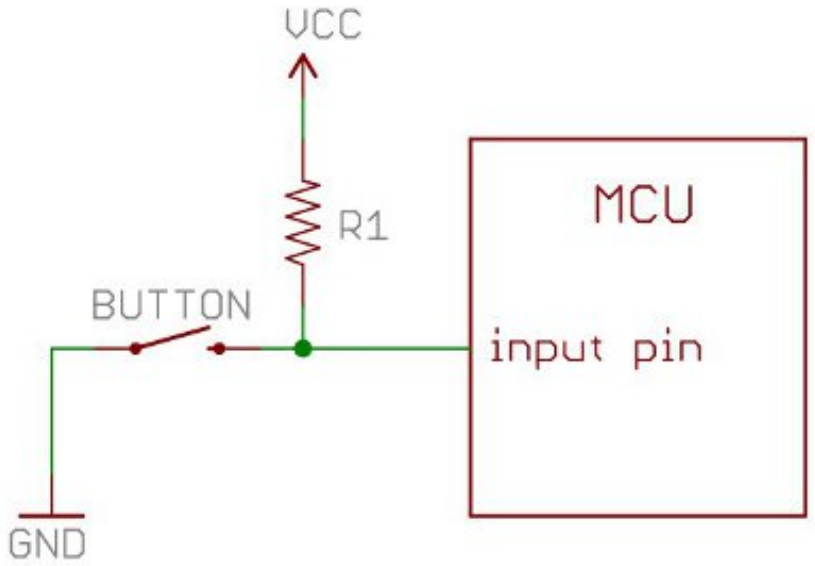


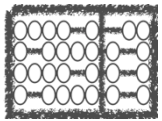
Pull-up / Pull-down

Pull-up circuit:

a) When the push button is not pressed the circuit is open and the input reads 1 (VCC).

b) When the push button is pressed the circuit is closed and the current flows to GND (R1 limits the current). In this case, the input reads 0 (GND).



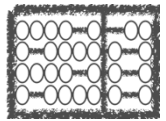


Using PORTn to set Pull-up/Pull-down

When the DDRn bit is set as input, the PORTn register is used to activate/ deactivate internal pull-up register.

The value 1 activate the internal pull-up and value 0 deactivate it.

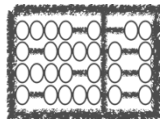
When the pull-up is enabled, the default state becomes 1, and turns to 0 just when you connect it to ground or pull-down.



Accessing ATMega328 GPIO registers

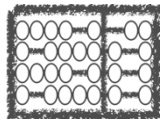
The registers are mapped in memory and can be manipulated through load/store instructions.

Full information can be found at Atmega328_Datasheet.



Accessing ATMega328 GPIO registers

Name	Address	Description
PORTB	0x25	Port B Data Register
DDRB	0x24	Port B Data Direction Register
PINB	0x23	Port B Input Pins Address
PORTC	0x28	Port C Data Register
DDRC	0x27	Port C Data Direction Register
PINC	0x26	Port C Input Pins Address
PORTD	0x2B	Port D Data Register
DDRD	0x2A	Port D Data Direction Register
PIND	0x29	Port D Input Pins Address



Setting bits

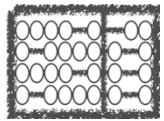
Control/Data registers may contain multiple bits (e.g. 8 bits), but sometimes we only want to change one of them!

Example: Write 1 to bit 2 of a control register mapped to address 0x25

```
volatile char* p = (char*) 0x25;
*p = *p | 0b00000010;
```

Same as:

```
volatile char* p = (char*) 0x25;
*p |= 0b00000010;
```



Setting bits

Control/Data registers m), but sometimes we only want to change one

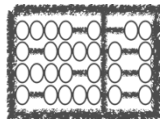
Write 1 is also known as Set!
Ex: set bit 2 of a control ...

Example: Write 1 to bit 2 of a control register mapped to address 0x25

```
volatile char* p = (char*) 0x25;
*p = *p | 0b00000010;
```

Same as:

```
volatile char* p = (char*) 0x25;
*p |= 0b00000010;
```



Setting bits

Write 0 is also known as Clear or Reset!
Ex: reset bit 2 of a control ...

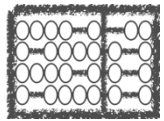
Control/Data registers m), but sometimes we only want to change one

Example: Write 0 to bit 2 of a control register mapped to address 0x25

```
volatile char* p = (char*) 0x25;
*p = *p & 0b11111101;
```

Same as:

```
volatile char* p = (char*) 0x25;
*p &= 0b11111101;
```



Testing bits

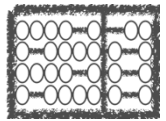
Control/Data registers may contain multiple bits (e.g. 8 bits), but sometimes we only want to test one of them!

Example: test whether bit 3 of a control register mapped to address 0x25 is 1

```
volatile char* p = (char*) 0x25;
if (*p & 0b00001000) { ... }
```

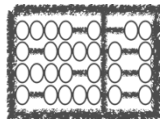
Same as:

```
volatile char* p = (char*) 0x25;
if (*p & (1<<3)) { ... }
```



Bitwise operators in C

Code Symbol	Logic Function
|	OR
&	AND
~	NOT
^	XOR
<<	Shift Left
>>	Shift Right

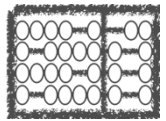


Setting / Reading bits of GPIO registers

```
volatile char* ddrB  = (char*) 0x024; // points ddrB variable to the DDRB register address
volatile char* pinB  = (char*) 0x023; // You should tell to compiler that
                                      // the value of the PINB may change at any time
volatile char* portB = (char*) 0x025;
bool read_value;

void setup() {
  *ddrB |=  (1 << 5);             // Makes pin 13 (PB5) of port B as output
  *ddrB &= ~(1 << 3);             // Makes pin 11 (PB3) of port B as input
  read_value  = *pinB & (1 << 3); // Stores the value read from pin 11 in read_value
  *portB |=  (1 << 5);            // Writes value 1 on pin 13 (PB5)
  *portB &= ~(1 << 5);            // Writes value 0 on pin 13 (PB5)
}

void loop() {}                    // Do nothing
```

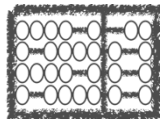


Agenda

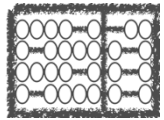
Memory Mapped I/O in C

GPIO

Lab Activities



Lab 3: Turn on the LED when the push button is pressed



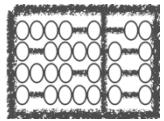
Lab 3: Turn on the LED when the push button is pressed

Create an Arduino sketch to turn on the built-in LED when the push button is pressed and turn it off when the button is released.

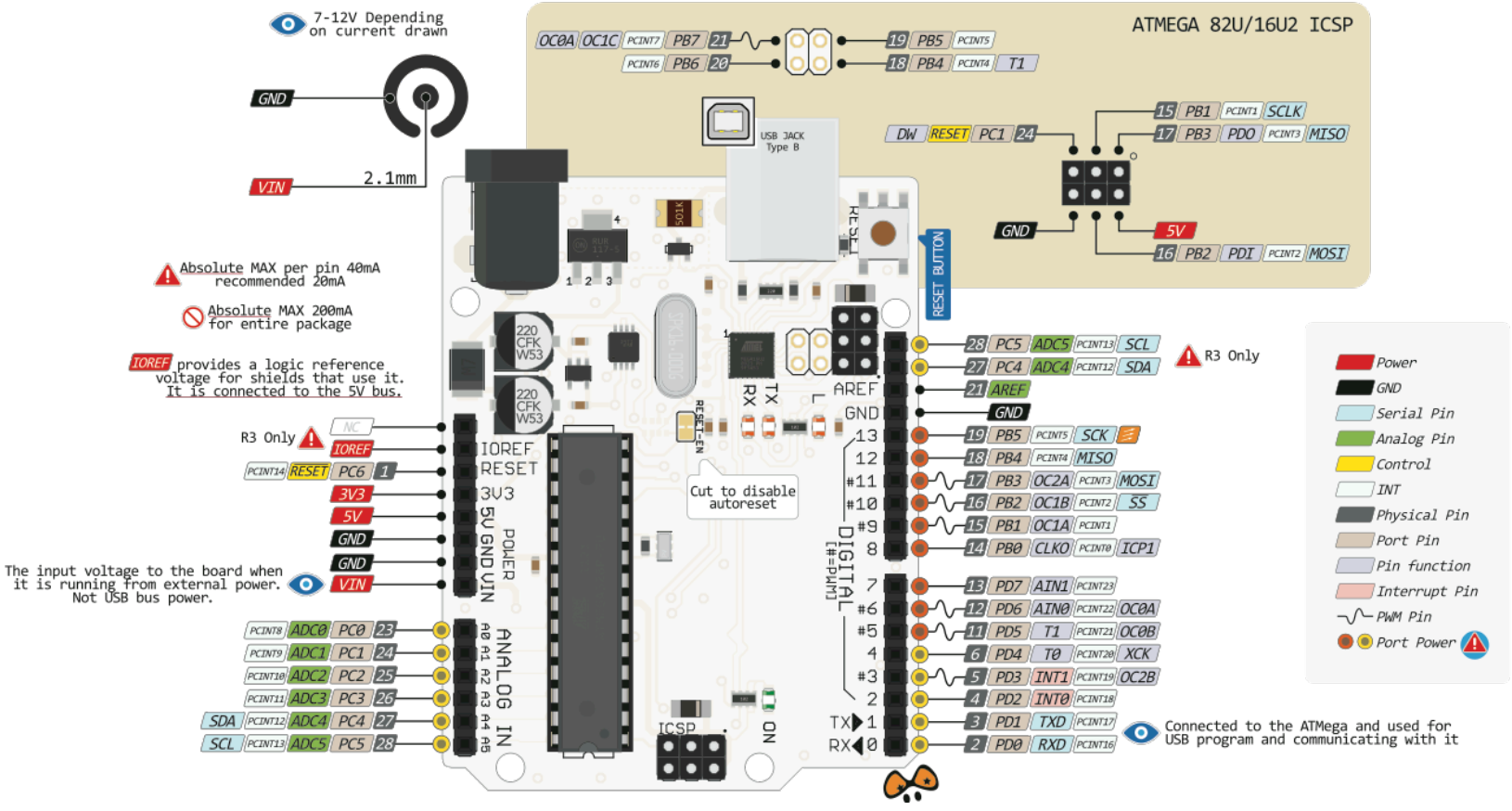
You may not use any Arduino function or AVR macro to control the GPIO. Instead, use load and store instructions to configure and use the GPIO registers.

The on-board LED is connected to PB5 (PIN 13 on the arduino board)

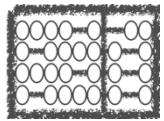
Connect the push button to PB0 (PIN 8 on the arduino board)



Lab 3: Turn on the LED when the push button is pressed



Lab 4: Trigger the Morse code using a push button

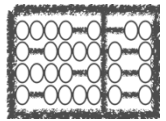


Lab 4: Trigger the Morse code using a push button

Modify your Morse Code sketch to trigger the message when the push button is pressed.

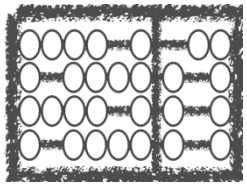
Once pressed send entire message.

Once the message is sent, wait for the user to press the push button again to send the message again.



INF-741: Embedded systems programming for IoT

Prof. Edson Borin



Institute of Computing

University of Campinas