5. Embedded C for AVR

Embedded C – 8 bits

AVR Architecture

ATMEL → Bought by MICROCHIP

- AVR Architecture designed specifically for C code
- Key for efficiency is 32 fast register (one clk cycle to access)
- Arithmetic and logical instructions work on these registers

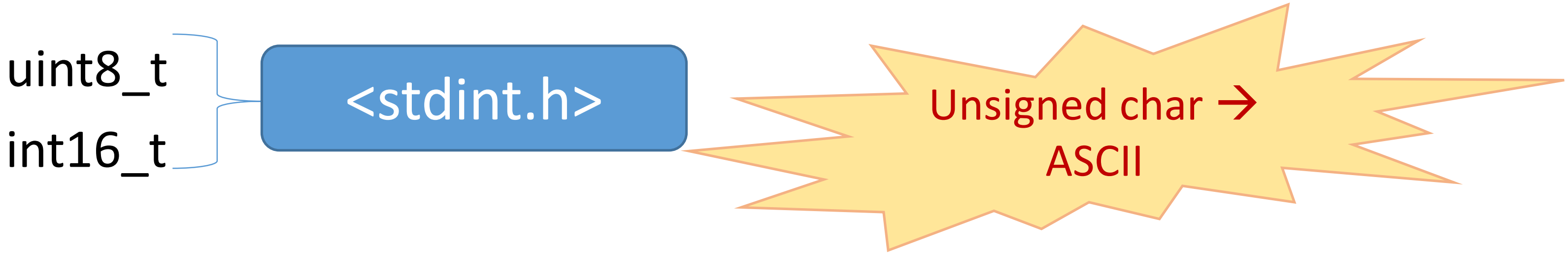
Size

Speed

Readability

Embedded C – 8 bits (Datatypes)

- NOT Standard C data types: int, unsigned long, float, char, and so on.



uint8_t

int16_t

<stdint.h>

Unsigned char → ASCII

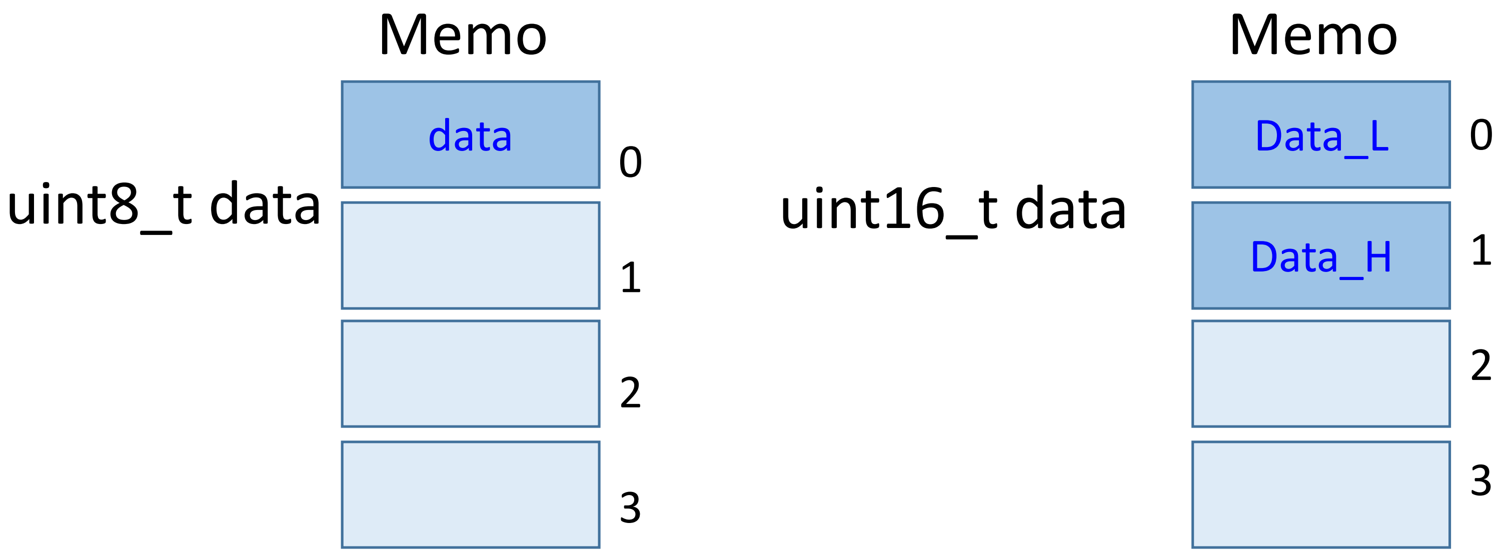
To avoid ambiguity of variable sizes on embedded systems.

- Typedef → to define types and to create composite data types using structures.

```
typedef struct {
uint_8 x;
uint_8 y;
} OrderedPair;
```

- **Use the smallest possible type to get the job done**
- **Use an unsigned type whenever possible**

Embedded C – 8 bits (Datatypes)



One access

Two access

Micro 8-bits

Embedded C – 8 bits (Boolean)

- C has no Boolean data type
- Boolean variables shall be declared as type bool.

(From Barr Group 2018)

#include <stdbool.h>

C99

Note the difference between Boolean operators &&, || and bitwise logical operators &, |

```
if ( k && m)   //Test if k and m both TRUE (non-zero
               // values)
if ( k & m)    //Compute bitwise AND between m and n,
               //then test whether the result is non-
               // zero (TRUE)
```

Embedded C – 8 bits (Standard and Tips)

- Coding following **Barr Group Embedded C** coding standard
- MIRSA standard → Automotive

"Atmel AVR4027: Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers"

https://ww1.microchip.com/downloads/en/AppNotes/doc8453.pdf

- Use local variables whenever possible.
- Use the smallest applicable data type. Use unsigned if applicable.
- A static function is easier to optimize.
- Much more…

Embedded C – 8 bits (Standard and Tips)

“AVR035: Efficient C Coding for AVR”

http://ww1.microchip.com/downloads/en/Appnotes/doc1497.pdf

Efficient Use of Variables.

- Variables declared inside a function → local variables → Register
  - **Static** to be preserved.
  - Preferably assigned to a register
  - Variable kept in the same register until end of the function
- Variables declared outside a function → global variables → SRAM
  - Loaded from the SRAM into the working registers before they are accessed. (More than one clock cycle)
  - **Static** → accessed only in the file in which they are defined
  - **Volatile** → it can be accessed out of the main program (ISR, Peripheral,..)

Embedded C – 8 bits (Standard and Tips)

“AVR035: Efficient C Coding for AVR”

- *Functions*
  - **Static** → is invisible outside of the file in which it is declared
  - Inline → If a static function is called only once in the file, the function will be optimized automatically by the compiler as an inline function

- *Control flow*
  - **"if-else",** always put the most probable conditions in the first place. Time is saved for most cases.
  - Better Using "**switch-case**" , the compiler usually generates lookup tables with index and jump to the correct place directly.

Embedded C – 8 bits (Compiler)

➢Bit size I/O

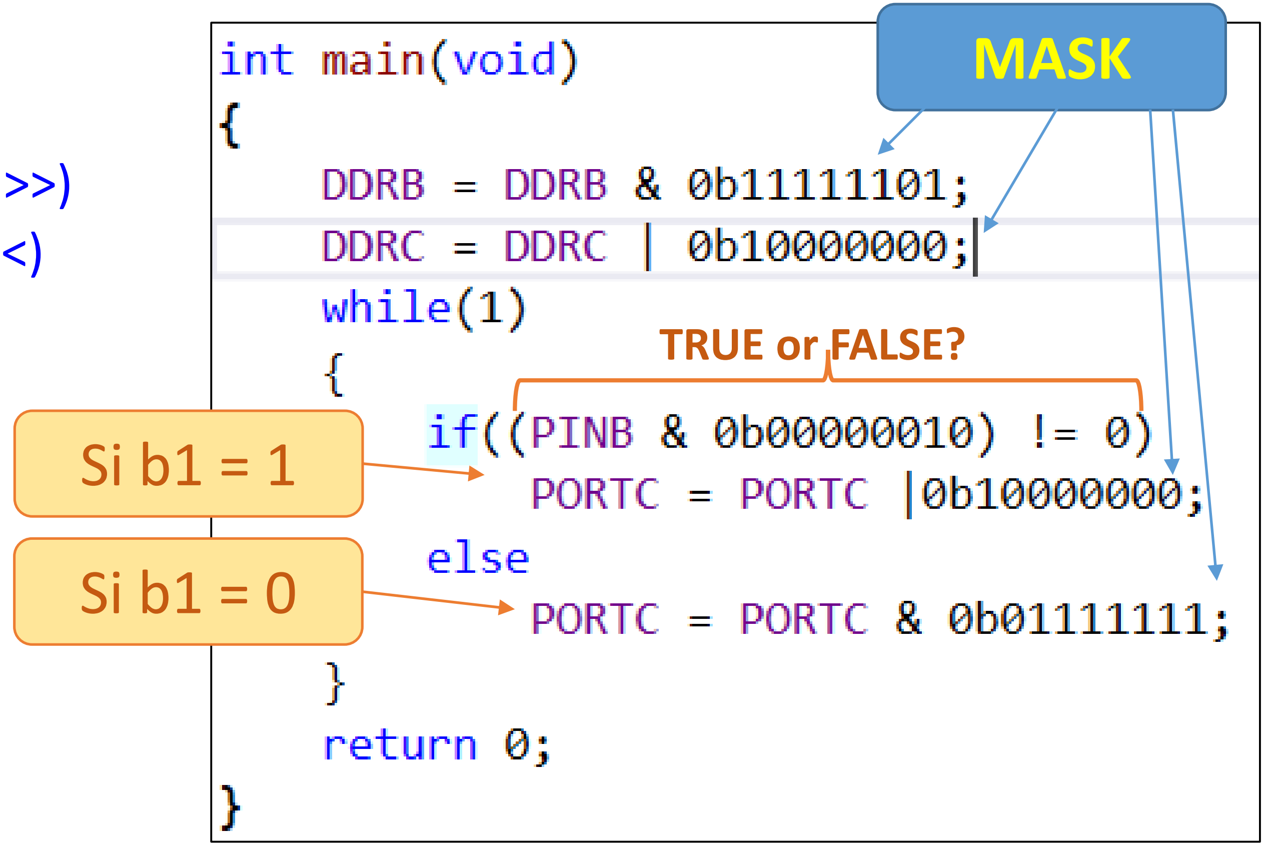
- Some MCU are bit accesible but some compilers do not support this feature.
- But C has the ability to to perform bit manipulation.

```
PORTB.1 = 1        // Only supported by some compilers
```

- To write **portable code** to be compiled on differents compilers, better use AND/OR wise operations to access a single bit of a byte → masking

Embedded C – 8 bits (C bit-wise operators)

- You are used to use C Logical operators → AND (&&), OR(||),….
- Bit-wise operators →widely used in embedded systems
    - AND (&)
    - OR (|)
    - XOR (^)
    - Inverter (~)
    - Shift-right (>>)
    - Shift-left (<<)



```
int main(void)
{
    DDRB = DDRB & 0b11111101;
    DDRC = DDRC | 0b10000000;
    while(1)
    {
        if((PINB & 0b00000010) != 0)
            PORTC = PORTC |0b10000000;
        else
            PORTC = PORTC & 0b01111111;
    }
    return 0;
}
```

Embedded C – 8 bits (C bit-wise operators)

- Bit-wise operators → widely used in embedded systems
  - Shift-right (>>)
  - Shift-left (<<)

To leave the generation of ones and zeros to the compiler and improve the clarity of the code and avoid errors, shift operators are preferred.

0b00000001 << 5 is the same as 1<<5

To write 0b11101111 → ~(1 <<5)

To write 0b00101000 → (1<<3) | (1 <<5)

Embedded C – 8 bits (C bit-wise operators)

```
#define PIN_LED       0
#define PIN_SENSOR    2

int main(void)
{
GPIO_1_MODE  |=  (1 << PIN_LED);    // As output
GPIO_2_MODE  &= ~(1 << PIN_SENSOR); // As input
GPIO_2_OUT   |= (1 << PIN_SENSOR);  // Enable pull-up

while (1){
    if ((GPIO_2_IN & (1 << PIN_SENSOR))!=0){// SENSOR ='1'
    //if (!(GPIO_D_IN & (1 << PIN_SENSOR))) // SENSOR ='0'
        GPIO_1_OUT |= (1<<PIN_LED);         // LED ON
    }
    else {
        GPIO_1_OUT &= (~(1<<PIN_LED));  // LED OFF
    }
}
```

Readable

Clarity

Author: Mirian Cifredo Chacón (Dpt. Ingeniería en Automática, Electrónica, Redes y Tecnología de Computadores)

Embedded C – 8 bits (AVR - Atmel Studio)

- AVR 8-bit GNU Toolchain
- avr-libc is the Standard C Library for AVR 8-bit GCC

Embedded C – 8 bits (Macros vs Regular functions)

https://embeddedinventor.com/c-macro-function-vs-regular-function-vs-inline-functions/

Macros for:

- Magic numbers
- Bitwise operations

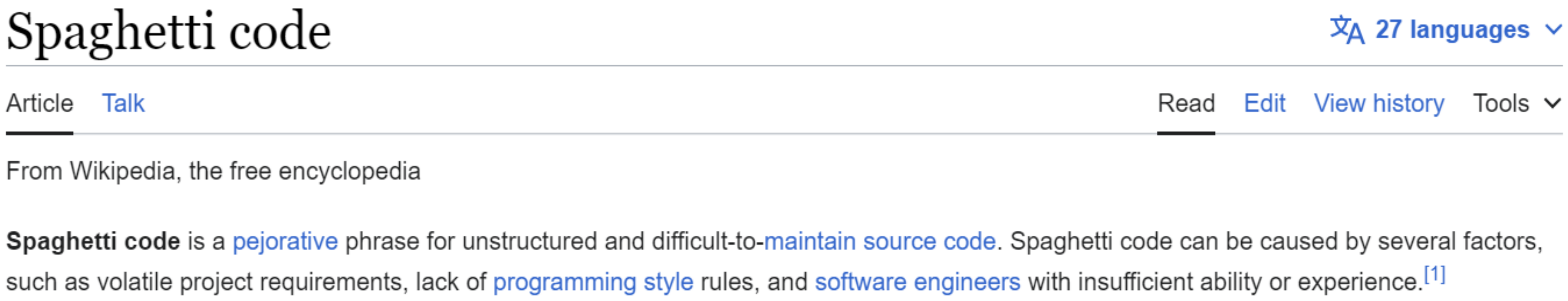
Embedded C – 8 bits (Defining útil macros)

Is it better to use macro or function in embedded C?

- **Macros are faster than functions** as they don't involve actual function call overhead.

Finite State Machine (FSM)

Embedded C



Spaghetti code

From Wikipedia, the free encyclopedia

Spaghetti code is a pejorative phrase for unstructured and difficult-to-maintain source code. Spaghetti code can be caused by several factors, such as volatile project requirements, lack of programming style rules, and software engineers with insufficient ability or experience.[1]

BARR group SOFTWARE EXPERTS

*A **state machine (FSM)** is any object that behaves different based on its history and current inputs. Many embedded systems consist of a collection of state machines at various levels of the electronics or software.*

https://barrgroup.com/blog/how-code-state-machine-c-or-c

Embedded C – Programming style: FSM

FSM → finite-state machine

- **Inputs**—any event that requires our system to generate an output or change its behaviour.
- **State transitions--** State transitions can only be triggered by an event.
- **Outputs**—actions that need to be taken by the system in each state.
- **States**—A state represents the expected behaviour of the system. What to do when an event occurs.

STEPS:

1) what your system needs to do
2) State chart → What to do in each state (Table)
3) C programming → switch-case

Embedded C – FSM: Structure event-driven

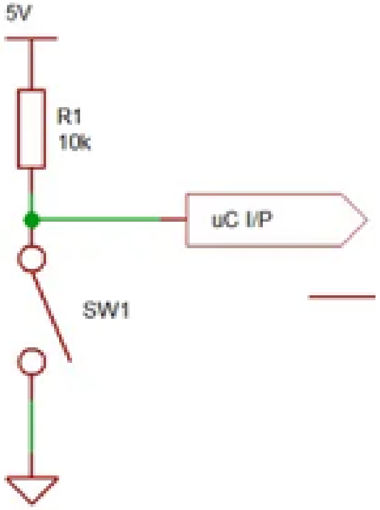
```
Superloop:

    1. Check the event/s
    2. Set transitions based upon the event
         (Switch - Nexstate)
    3. Set ouputs based on the current state
         (Switch - State)
    4. Update current state
         (State=Nexstate)
```

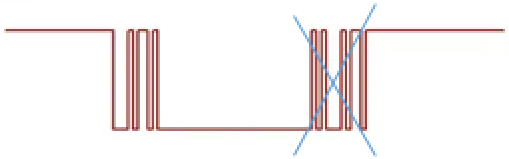
More safety!!!

Easier to add more states and outputs!!!

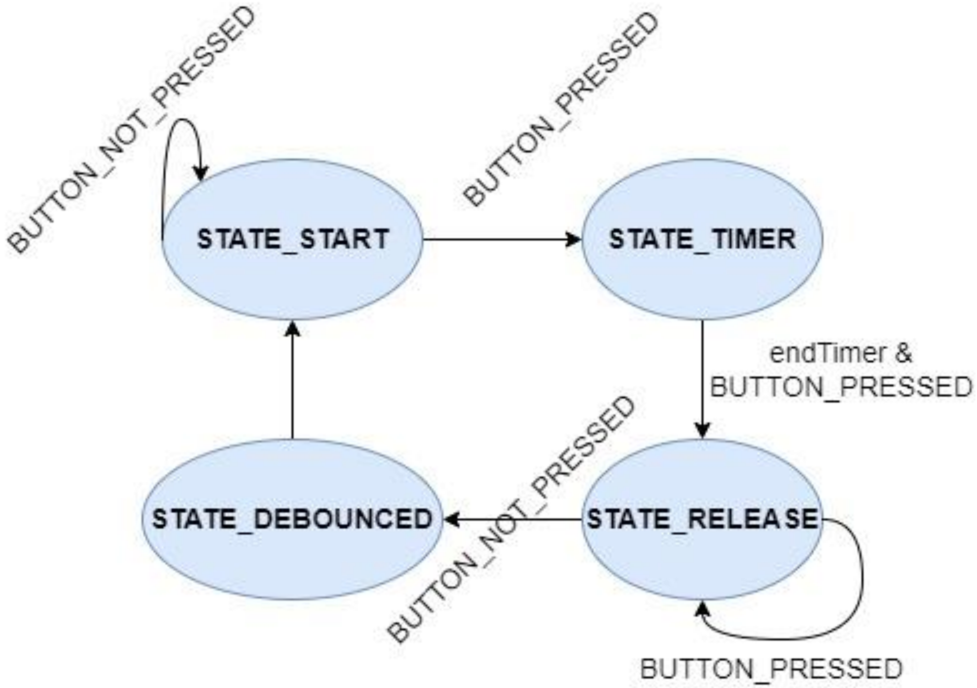
Embedded C – Programming style: FSM

FSM – “Toggle a LED with a debounced pushbutton” – **P02_Exercise 06**





Events:
- Push button
- Timer


STATE_START
STATE_TIMER
STATE_RELEASE
STATE_DEBOUNCED
BUTTON_NOT_PRESSED
BUTTON_PRESSED
endTimer &
BUTTON_PRESSED
BUTTON_NOT_PRESSED
BUTTON_PRESSED