

LamaPLC: Arduino code collection

Frequently used code collection

TON

```
// var def block
unsigned long startMillis, currentMillis; // current and start time
const unsigned long period = 2000; // the value is a number of
milliseconds

// code block
currentMillis = millis(); // get the current "time"
(actualy the number of milliseconds since the program started)
if (currentMillis - startMillis >= period) // test whether the period
has elapsed
{
  startMillis = currentMillis; // time update
  // after time has elapsed the call
  // ...
}
```

check serial monitor

```
// code in setup()
Serial.begin(9600);

while (!Serial) {
  ; // wait for serial port to connect. Needed for native USB port only
}
Serial.println("Serial monitor OK");
```

Converting code collection

Float to char[]

```
// lib init (optional)
#include "avr/dtostrf.h"
// converting
float a;
```

```
char sendValue[10];
dtostrf (a, 10, 8, sendValue); // float_value, min_width,
num_digits_after_decimal, where_to_store_string
```

Int to char[]

```
int a;
char sendValue[10];
itoa (i,sendValue,10); // int value, char * str, int base
// base: Numerical base used to represent the value as a string, between 2
and 36, where 10 means decimal base, 16 hexadecimal, 8 octal, and 2 binary
```

String to char[]

```
char place[20];
strcpy(place, "Home");
```

Modbus code collection

Convert 2 registers to float

call: [float] = modbus_2regs_2_float([1.register], [2.register])

```
float modbus_2regs_2_float(uint16_t a, uint16_t b) {
    uint32_t combined = ((uint32_t)a << 16) | b;
    float f;
    memcpy(&f, &combined, sizeof f);
    return f;
}
```

call: [float] = modbus_2regs_2_float([first register])

```
float modbus_2regs_2_float(uint16_t addr) {
    uint16_t a = node.getResponseBuffer(addr);
    uint16_t b = node.getResponseBuffer(addr+1);
    uint32_t combined = ((uint32_t)a << 16) | b;
    float f;
    memcpy(&f, &combined, sizeof f);
    return f;
}
```

Convert float 2 registers

def: int16_t regs[2];
call: *modbus_set_float*([float], regs);
back: register1: regs[1], register2: [regs[2]]

```
void modbus_set_float(float f, uint16_t *dest)
{
    uint32_t i = 0;

    memcpy(&i, &f, sizeof(uint32_t));
    dest[0] = (uint16_t)i;
    dest[1] = (uint16_t)(i >> 16);
}
```

Source: https://docs.ros.org/en/melodic/api/libmodbus/html/modbus-data_8c.html

Convert 2 registers to int32

```
uint32_t make_32bit_word(uint16_t hi_word, uint16_t lo_word)
{
    uint16_t words_16bit[2] = {hi_word, lo_word};
    uint32_t word_32bit = 0;

    memcpy(&word_32bit, words_16bit, 4);

    return word_32bit;
}
```

int32 convert direct to float by division by a thousand:

```
float make_float_32bit_word(uint16_t hi_word, uint16_t lo_word)
{
    uint16_t words_16bit[2] = {hi_word, lo_word};
    uint32_t word_32bit = 0;
    float back;

    memcpy(&word_32bit, words_16bit, 4);
    back = word_32bit / 1000.0;
    return back;
}
```

call: [float] = *make_float_32bit_word*([first reg])

```
float make_float_32bit_word(uint16_t addr)
{
```

```

uint16_t words_16bit[2];
words_16bit[1] = node.getResponseBuffer(addr);
words_16bit[0] = node.getResponseBuffer(addr+1);
uint32_t word_32bit = 0;
float back;

memcpy(&word_32bit, words_16bit, 4);
back = word_32bit / 1000.0;
return back;
}

```

Float to / from one register

Note: The maximum value of the int16 type is 32767, so in this format only the value range 0 .. 327.67 can be stored with 2 decimal places!

```

// float 2 reg
holdingRegs[REG_HUM] = (int16_t)(Humidity * 100);
// back one reg to float
float Humidity = holdingRegs[REG_HUM] / 100.0;

```

Modbus #define conversions

```

#define MODBUS_GET_HIGH_BYTE(data) (((data) >> 8) & 0xFF)
#define MODBUS_GET_LOW_BYTE(data) ((data) & 0xFF)
#define MODBUS_GET_INT64_FROM_INT16(tab_int16, index) \
\
    (((int64_t) tab_int16[(index)] << 48) | ((int64_t) tab_int16[(index) + \
1] << 32) | \
    ((int64_t) tab_int16[(index) + 2] << 16) | (int64_t) tab_int16[(index) \
+ 3])
#define MODBUS_GET_INT32_FROM_INT16(tab_int16, index) \
    (((int32_t) tab_int16[(index)] << 16) | (int32_t) tab_int16[(index) + \
1])
#define MODBUS_GET_INT16_FROM_INT8(tab_int8, index) \
    (((int16_t) tab_int8[(index)] << 8) | (int16_t) tab_int8[(index) + 1])
#define MODBUS_SET_INT16_TO_INT8(tab_int8, index, value) \
    do { \
        ((int8_t *) (tab_int8))[(index)] = (int8_t) ((value) >> 8); \
        ((int8_t *) (tab_int8))[(index) + 1] = (int8_t) (value); \
    } while (0)
#define MODBUS_SET_INT32_TO_INT16(tab_int16, index, value) \
    do { \
        ((int16_t *) (tab_int16))[(index)] = (int16_t) ((value) >> 16); \
        ((int16_t *) (tab_int16))[(index) + 1] = (int16_t) (value); \
    } while (0)
#define MODBUS_SET_INT64_TO_INT16(tab_int16, index, value)

```

```
\
do {
\
    ((int16_t *) (tab_int16))[(index)] = (int16_t) ((value) >> 48);
\
    ((int16_t *) (tab_int16))[(index) + 1] = (int16_t) ((value) >> 32);
\
    ((int16_t *) (tab_int16))[(index) + 2] = (int16_t) ((value) >> 16);
\
    ((int16_t *) (tab_int16))[(index) + 3] = (int16_t) (value);
\
} while (0)
```

Modbus messages

```
bool getResultMsg(uint8_t result)
{
    String tmpstr2;

    switch (result) {
        case node.ku8MBSuccess:
            return true;
            break;
        case node.ku8MBIllegalFunction:
            tmpstr2 = "Illegal Function";
            break;
        case node.ku8MBIllegalDataAddress:
            tmpstr2 = "Illegal Data Address";
            break;
        case node.ku8MBIllegalDataValue:
            tmpstr2 = "Illegal Data Value";
            break;
        case node.ku8MBSlaveDeviceFailure:
            tmpstr2 = "Slave Device Failure";
            break;
        case node.ku8MBInvalidSlaveID:
            tmpstr2 = "Invalid Slave ID";
            break;
        case node.ku8MBInvalidFunction:
            tmpstr2 = "Invalid Function";
            break;
        case node.ku8MBResponseTimedOut:
            tmpstr2 = "Response Timed Out";
            break;
        case node.ku8MBInvalidCRC:
            tmpstr2 = "Invalid CRC";
            break;
        default:
            tmpstr2 = "Unknown error: " + String(result);
    }
}
```

```

    break;
}
Serial.println(tmpstr2);
return false;
}

```

General code collection

MAC random generator

Generate random MAC using pseudo random generator, bytes 0, 1 and 2 are static (MAC_START), bytes 3, 4 and 5 are generated randomly

```

void generateMac() {
  // Marsaglia algorithm from https://github.com/RobTillaart/randomHelpers
  seed1 = 36969L * (seed1 & 65535L) + (seed1 >> 16);
  seed2 = 18000L * (seed2 & 65535L) + (seed2 >> 16);
  uint32_t randomBuffer = (seed1 << 16) + seed2; /* 32-bit random */
  memcpy(data.mac, MAC_START, 3); // set first 3 bytes
  for (byte i = 0; i < 3; i++) {
    data.mac[i + 3] = randomBuffer & 0xFF; // random last 3 bytes
    randomBuffer >>= 8;
  }
}

```

Arduino example codes topics on lamaPLC

Page	Date	Tags
• lamaPLC project: Arduino - OLED SH1106 with AHT20/BMP280 Sensor	2026/04/23 21:51	bmp280 , aht20 , temperature , humidity , pressure , sensor , arduino , oled , sh1106 , arduino code
• lamaPLC project: Arduino - Vibration sensors	2026/04/15 17:21	vibration , sensor , piezoelectric , mems , eddy-current , electrodynamic , gxfm0459 , ldtm-028k , arduino , arduino code
• lamaPLC project: Digitales Potentiometer Board Moduls	2026/04/11 18:29	sensor , module , arduino code , renesas , x9c series , x9c102 , x9c103 , x9c104 , x9c503 , xdcp , digitally controlled potentiometer
• lamaPLC project: Sension SCD CO² measurement module	2026/04/15 19:34	scd30 , scd40 , scd41 , iaq , ndir , sensor , i2c , arduino code
• LamaPLC: Arduino code collection	2026/04/23 21:51	arduino code , arduino , mac generator , ton , modbus messages , float to char , int to char , string to char , 2 registers to float , float 2 registers , 2 registers to int32 , float to from one register , modbus define conversions
• lamaPLC: MPU-6050 (HW-123, GY-521) 6-axis MotionTracking device	2026/04/23 21:52	mpu-6050 , hw-123 , gy-521 , 6-axis motiontracking , dmp , temperature , sensor , mems , arduino code , arduino , accelerometer , gyroscope , tilt

[arduino code](#), [arduino](#), [mac generator](#), [TON](#), [modbus messages](#), [Float to char](#), [Int to char](#), [String to char](#), [2 registers to float](#), [float 2 registers](#), [2 registers to int32](#), [Float to from one register](#), [Modbus define conversions](#)

This page has been accessed for: Today: 2, Until now: 103

From:

<https://lamaplc.com/> - **lamaPLC**

Permanent link:

https://lamaplc.com/doku.php?id=arduino:code_collection

Last update: **2026/04/21 20:48**

