Home Automation

WeMos Lolin32 and Nextion Enhanced set-up

1 Introduction

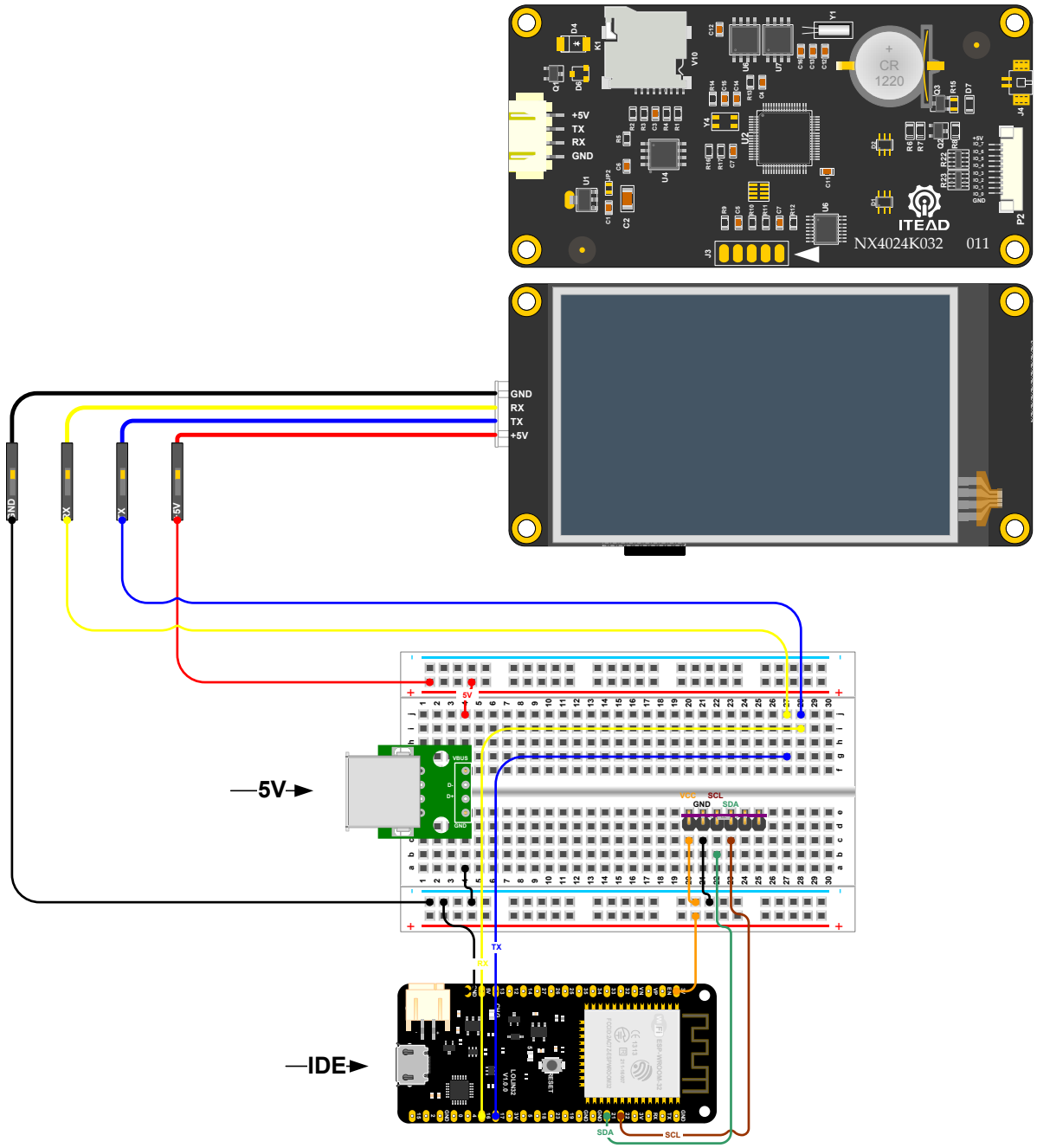
This document describes the different steps and library corrections to setup a prototype configuration of a microcontroller WeMos Lolin 32 (Exspresif 32 MCU) connected to a Nextion (Enhanced) display (HMI).
It highlights the necessary physical connections between a WeMos Lolin32 (ESP32), a Nextion 3.2" Enhanced and a BME 280 barometric sensor, which are the basic components of a Home Automation Weather or Thermostat Station.
Not all the Nextion functions are used in this scenario, so this document me be subject to enhancement in the future if more functions are used.
The functions currently covered by this project are:

- NexConfig
- NexHardware
- NexRtc
- NexPage
- NexPicture
- NexGauge
- NexButton
- NexText
- NexNumber
- NexTimer
- NexVariable
- NexDSButton

Home Automation

WeMos Lolin32 and Nextion Enhanced set-up

1.1 WeMos Lolin32, Nextion 3.2” enhanced and BME280



For sufficient power a separate power supply is used to forthe Nextion, while the WeMos is powered through the USB interface.

The WeMos Lolin32 is connected to the Nextion UART port using the 2nd Serial Interface. This UART2 is configured for the WeMos Lolin (Arduino IDE pins) on pins 16 for RX and 17 for TX.

TX pin of the WeMos Lolin shall be mapped to RX pin of the Nextion and vice versa.

Barometric (temperature, humidity and pressure) measurements are collected using a BME280 sensor.

BME280 is powered by a 3v3 power from the WeMos Lolin32.

BME280 SDA pin is connected to the SCL pin of the WeMos pin 22 while the SCL pin is connected to the WeMos SDA pin 21.

Arduino will use the Nextion library (see https://github.com/itead/ITEADLIB_Arduino_Nextion) that should be installed for the Arduino IDE.

1.2 UART configuration

By default the Nextion library uses the *SoftawerSerial* library ensuring connection for Nextion updates and IDE programming / debugging. However Software serial is not actually supported by the ESP32 devices so the ITEADLIB library requires modifications to support the *HardwareSerial* that allowing usage of different WeMos UARTs.

*Note: Both debugging (dbSerial) and Nextion Serial (Serial2()) interfaces are activated on **nextInit()**.*

See 1.2.4 NexHardware on page 5.

Several authors and articles describe these aspects

- Andrea Spiess (http://www.sensorsiot.org)

  https://www.youtube.com/watch?v=FSRx8h8iBnk&index=5&list=PL3XBzmAj53RlXES3erPR4F0Zk08Hx1YdA

- and

  https://github.com/espressif/arduino-esp32/issues/407

- and

  https://www.youtube.com/watch?v=GwShqW39jIE

1.2.1 Script configuration

To activate the Serial interface, the firmware script should include the statement below:

```
HardwareSerial Serial2(2);
```

1.2.2 HarwareSerial.h

One possibility is to modify the ESP32 *HardwareSerial.h* file **however this file is to be modified each time a new version of the ESP32 library is downloaded which may be cumbersome.**

See 1.2.3 NexConfig.h & NexUpload.cpp on page 4 for a better approach

This file is typically accessed from the directory:
<root>/Arduino/Sketch/Hardware/espressif/esp32/cores/esp32/HardwareSerial.h

And should be modified by adding at the bottom

```
extern HardwareSerial Serial1;
extern HardwareSerial Serial2;
```

```
<…>
extern HardwareSerial Serial;
extern HardwareSerial Serial1;
extern HardwareSerial Serial2;
#endif
```

Practically:

- Serial1 refers to UART 1 Serial1(1)
- Serial 2 refers to UART 2 Serial2(2)

Serial 1 (1) should not be used because connected to flash memory

For more info see more in the *HardwareSerial.h* file

```
HardwareSerial Serial(0);
HardwareSerial::HardwareSerial(int uart_nr) : _uart_nr(uart_nr), _uart(NULL) {}

void HardwareSerial::begin(unsigned long baud, uint32_t config, int8_t rxPin,
int8_t txPin, bool invert)

{

    if(0 > _uart_nr || _uart_nr > 2) {
        log_e("Serial number is invalid, please use 0, 1 or 2");
        return;
    }
    if(_uart) {
        end();
    }
    if(_uart_nr == 0 && rxPin < 0 && txPin < 0) {
        rxPin = 3;
        txPin = 1;
    }
    if(_uart_nr == 1 && rxPin < 0 && txPin < 0) {
        rxPin = 9;
        txPin = 10;
    }
    if(_uart_nr == 2 && rxPin < 0 && txPin < 0) {
        rxPin = 16;
        txPin = 17;
    }
    _uart = uartBegin(_uart_nr, baud, config, rxPin, txPin, 256, invert);
}
```

1.2.3 NexConfig.h & NexUpload.cpp

A better approach to connect a WeMos Lolin32 serial interface with Nextion is to modify the Nextion library (which looks to change less frequently) files *NexConfig.h* and *NexUpload.h*.

This modification allows selecting *HardwareSerial* configuration for a WeMos Lolin32 and by default for other MCU (ATMEL AVR) *SoftwareSerial*.

NexConfig.h

```
* Define dbSerial for the output of debug messages.
 */
#define dbSerial Serial
/**
 * Define nexSerial for communicate with Nextion touch panel.
 */
#define nexSerial Serial2

#if defined(ARDUINO_LOLIN32)
 #include <HardwareSerial.h>
 extern HardwareSerial Serial2;
#else
 #include <SoftwareSerial.h>
 extern SoftwareSerial Serial2;
#endif
```

NexUpload.cpp

```
#include "NexUpload.h"

#if defined(ARDUINO_LOLIN32)
 #include <HardwareSerial.h>
#else
 #include <SoftwareSerial.h>
#endif
```

1.2.4 NexHardware

1.2.4.1 Debugging Serial interface

The actual Nextion NexHardware library file enforces same speed (9600 bps) for both *dbSerial* (debugging) and *nexSerial* (Nextion interface).

An improvement introduced by Adrea Spiess allows changing the dbSerial speed. The NexHardware file shall be modified as follow:

NexHardware.h

```
bool nexInit(void);
```

To

```
bool nexInit(uint32_t baud=115200);
```

NexHardware.cpp

```
bool nexInit(void)
{
    bool ret1 = false;
    bool ret2 = false;

    dbSerialBegin(9600);
    nexSerial.begin(9600);
    sendCommand("");
    sendCommand("bkcmd=1");
    ret1 = recvRetCommandFinished();
    sendCommand("page 0");
    ret2 = recvRetCommandFinished();
    return ret1 && ret2;
}
```

To

```
bool nexInit(uint32_t baud)
{
    bool ret1 = false;
    bool ret2 = false;
    dbSerialBegin(baud);
    nexSerial.begin(9600);
    sendCommand("");
    sendCommand("bkcmd=1");
    ret1 = recvRetCommandFinished();
    sendCommand("page 0");
    ret2 = recvRetCommandFinished();
    return ret1 && ret2;
}
```

With this modification:

Default speed is set by using the Nextion Init function (nexInit ()—see *NexHardware.h* and *NexHardware.h.cpp*)

Serial0 (or Serial or *dbSerial)* is configured with a speed of 115200. The LCD UART connection is known as *nexSerial* and is by default set to **9600**.

However it is now possible to change the *dbSerialBegin* baud rate but specifying the speed option through the nexInit() function.

IMPORTANT NOTE: The script should use Serial and Serial2(2) without speed definition because already done by the nexInit() function.

Warning: Defining other Serial2 speed after the NexInit() function may results in some issues.

1.2.4.2 Nextion sendCommand delay

With Wemos Lolin32 (fast CPU, or MultiCore?!), successive Nextion commands may require some delay. Avoiding adding overhead delays in the code the *sendCommand* used by all Nextion is modified allowing adding a programmable delay.

Two new functions are implemented the *setTurnAroundDelay* and the *getTurnaroundDelay*. The default delay is 0;

The practical value with interface at 9600 bps is 55 ms.

The *sendCommand* is part of the *NexHardware* library witch is modified as follow:

NexHardware.h

Add the following

```
/** RVDB - Add new functions implementing sendCommand turnaround delay.
*/
void setTurnAroundDelay (uint32_t delay);   // Set the turnaround delay in ms.

void getTurnAroundDelay (uint32_t *number );// Get the turnaround delay in ms.
```

NexHardware.cpp

Add the following

```
/* Default SendCommand turnaround delay in ms.
*/
uint32_t _DELAY = 0;
/*
 * Set uint32_t delay.
 *
 * @param delay - set delay value in ms.
 *
 *
 */
void setTurnAroundDelay (uint32_t delay )
{
    _DELAY = delay;
}

/*
 * Get uint32_t delay.
 *
 * @param delay - save delay value in ms.
 *
 *
 */
void getTurnAroundDelay (uint32_t *number )
{
    *number = _DELAY;
```

```
}
```

Modify the following from:

```
/*
 * Send command to Nextion.
 *
 * @param cmd - the string of command.
 */
void sendCommand(const char* cmd)

{
    while (nexSerial.available())
    {
        nexSerial.read();
    }
    nexSerial.print(cmd);
    nexSerial.write(0xFF);
    nexSerial.write(0xFF);
    nexSerial.write(0xFF);
}
```

to

```
/*
 * Send command to Nextion.
 *
 * @param cmd - the string of command.
 */
void sendCommand(const char* cmd)
{
    while (nexSerial.available())
    {
        nexSerial.read();
    }

    nexSerial.print(cmd);
    nexSerial.write(0xFF);
    nexSerial.write(0xFF);
    nexSerial.write(0xFF);
    delay (_DELAY);
}
```

1.2.5 NexRTC libary

Using NexRtc library to update the RTC module, requires a correction:

The fist command should be corrected to address the RFC year register 0

NexRtc.cpp

```
        cmd += "0=";
        cmd += year;
        sendCommand(cmd.c_str());
        recvRetCommandFinished();
```

Should be

```
        cmd += "rtc0=";
        cmd += year;
        sendCommand(cmd.c_str());
        recvRetCommandFinished();
```

1.2.6 Nextion debugging (see NexConfig.h)

Debugging messages are associated to dbSerial; when enable in the NexConfig.h file (see DEBUG_SERIAL_ENABLE). If debugging is disable dbSerial (Serial0) is also disabled, so debugging option in the script will not be satisfied; in this case the Serial should be activated.

Typically by adding in the script set-up

```
#ifdef DEBUGGING
Serial.begin (115200);                                  // Necessary if not #define
DEBUG_SERIAL_ENABLE validate in #endif
```