Programming Examples (i) 2022
L&R Ing. CL2bm1 CPU Board

L&R Ingeniería – Rev. 1c 12-06-22 - Rafael Oliva

1. INTRODUCTION

The CL2bm1 semi-industrial board module from L&R Ing. (Figure 1) is an Atmel – AVR (now Microchip) based unit, which is usually fitted with the ATMega1284P [ref00] microcontroller @ 14.7456 MHz. It can be programmed in C using the Codevision AVR compiler [ref01], the C/C++ Avr-gcc compiler [ref02] using IDEs such as Microchip Studio [ref03], Microsoft Visual Studio Code with PlatformIO plug-in [ref04] [ref05], or the classical Arduino IDE[ref06]. The latest boards are provided with a pre-loaded, modified Optiboot [ref07] bootloader which works through the COM0 port (if native RS232 is available) or via a standard RS232-USB adapter. In-circuit programming is also possible through a standard AVR 6-pin connector and AVRISP Mkii [ref08] or compatible USB programmers. The board is not fully Arduino-compatible in pinout but a great number of functions and libraries work on it "out of the box". A TCXO DS32kHz chip keeps an accurate timing coupled with a standard PCF8563 Real Time Clock IC. A low-dropout LM2937-5 regulator supplies power for the circuit from an external source between 7 and 25 Vdc. Consumption is typically 0.05 A @ 12.8 V. A complete schematic diagram can be downloaded from [ref21] https://github.com/LyRIng/CL2bm1/tree/master/schem

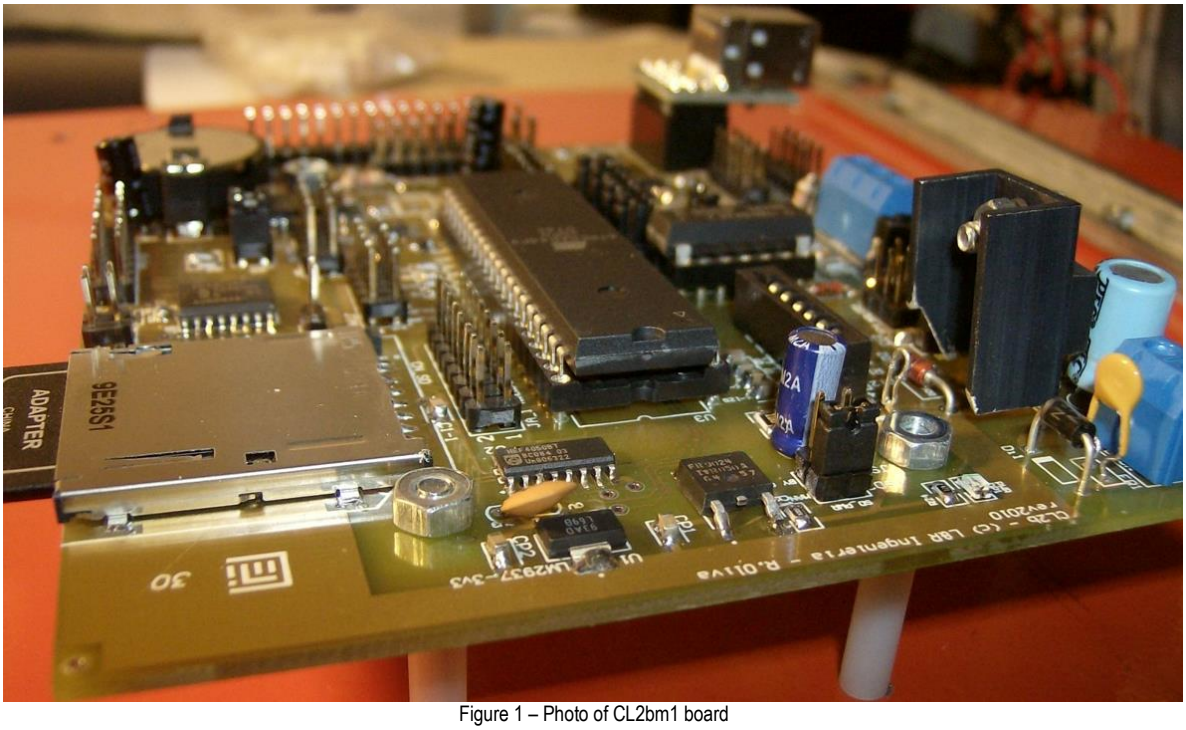


Figure 1 – Photo of CL2bm1 board

2. BLOCK DIAGRAM AND PROGRAMMING MODEL

2.a Block Diagram: L&R Ing. CL2bm1 boards were designed from 2009 to 2010, and have been reliably in use in many applications mainly for data logging and other control functions [ref09, ref10], usually teamed with the peripheral M4/E [ref11] board and with the METEO board/module [ref12]. The block diagram of the board can be seen in Figure 2.

CL2_b Proto (AtMega32/644/1284) Rev5F-2010

(C) R.Oliva - L&R Ing. 2009/10

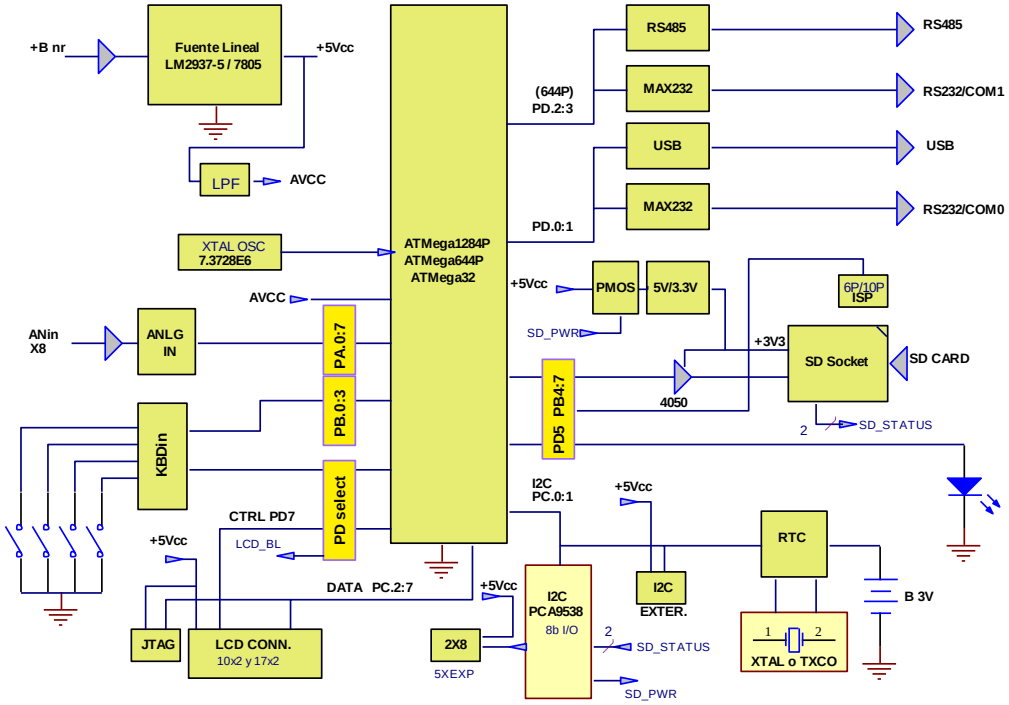


Figure 2 – Block diagram of the CL2bm1 board

Newer boards are fitted with the ATMega1284P (formerly Atmel / now Microchip) controller, with 128 KB of Flash, 16 KB of RAM, 4 KB of EEPROM and two serial ports, running at 14.7456 MHz from an external crystal oscillator. Their main components are:
a) Power supply with LDO regulators for +5 y 3.3V, up to 16V input (8-12V recommended).
b) Analog inputs for onboard 10-bit ADC
c) 4-contact membrane keyboard interface
d) Character LCD interface.
e) I2C expansion port and PCF8563 Real Time Clock with DS32kHz TCXO.
f) Industrial SD card interface
g) Serial RS232, RS485 and socketed USB module (FT232).
h) I2C PCA9538 I/O port

I2C bus communications are used to communicate with auxiliary boards based on PSoC-1 family of microcontrollers. The most common board is the already mentioned M4/E. This board also offers an I/O interface for the PCA9538 device.

2.b Programming model The ATMega1284P controller on CL2bm1 can be programmed in C or C++ using compatible cross-compilers. The examples in Section 3 will use CodevisionAVR from HPInfoTech [ref01], only available for Windows 7-10. Future examples will use VSC+PIO [ref04] [ref05], and the classical Arduino IDE[ref06], all available also on Ubuntu to show slight differences. Programs are compiled on a standard PC and downloaded to the board via:
a) The pre-loaded, modified Optiboot [ref07] bootloader mentioned in Section 1, which works through the COM0 port.
b) A 6 pin ISP connector, using the low cost USB ATAVRISP2 interface from Atmel. Field upgrades and EEPROM saving are straightforward. New or different bootloaders can also be programmed with this method.

The CL2bm1 programming interface is shown on Figure 3. The 8 ADC channels are general purpose 10-bit resolution using the internal AVR SAR analog to digital converter [ref13]. If greater resolution is required, 3 13-bit ADC inputs are available on the M4/E

expansion boards. The serial COM0 port can be routed to the FT232/USB module or towards the half DS14C232 interface (see schematic [ref21]), COM1 can be routed to the other DS14C232 half or to an RS485 (DS3695) interface with optional hardware Driver Enable circuit. Future examples will show how this interface works.

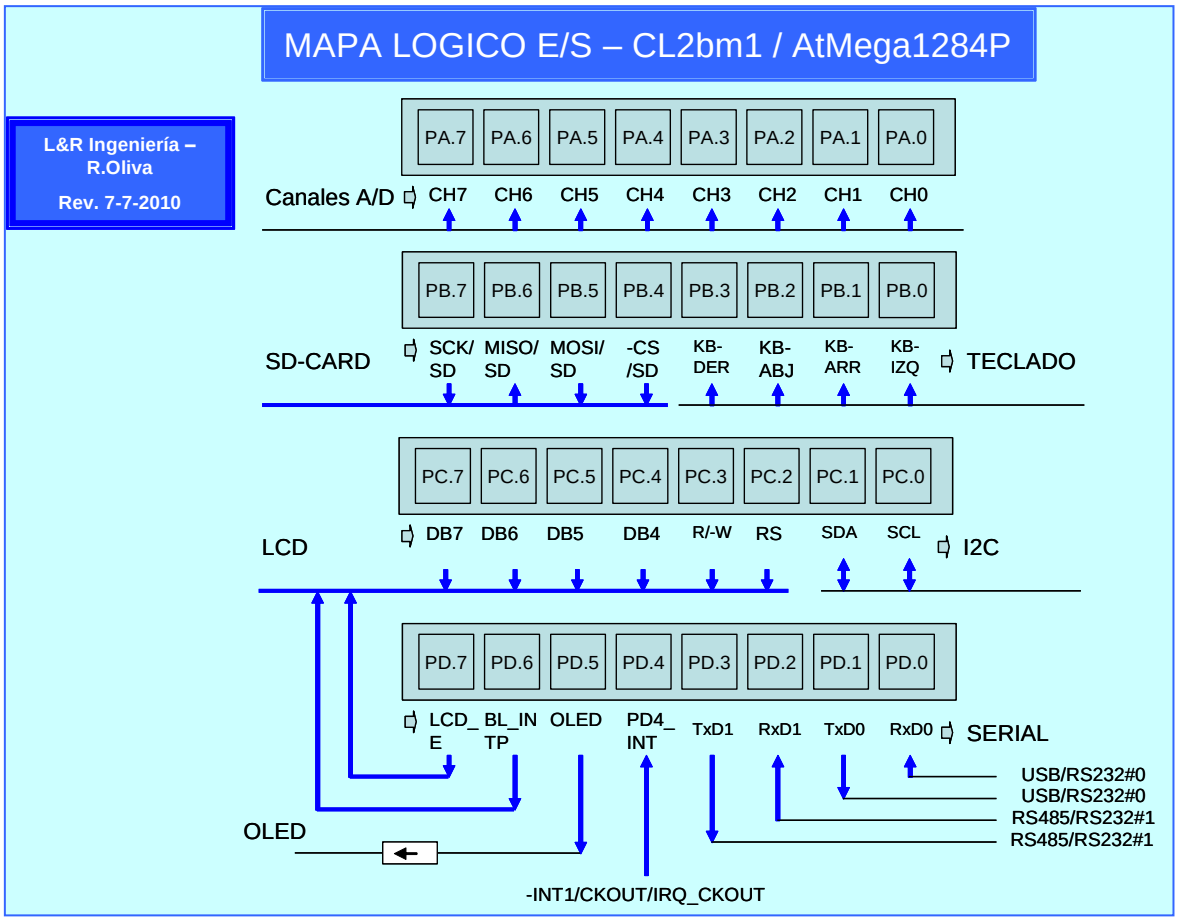


Figure 3 – Programming model of the CL2bm1 board for ATMega1284P

See APPENDIX I for typical Jumper Settings on CL2 board. In the following examples we will use:

Example1: (/blinkled) OLED output, which as seen is mapped to PD.5 port as output, blinks LED by using simple blocking delays. This example uses only the drivers/bsp routines (BSP stands for Board Support Package, usually low level hardware access specific to each board)
Example2: (/serialanalog) In Example1, adds use of the COM0 - RXD0/TXD0 serial port as a terminal at 19200,N,8,1 mapped to PD0./PD.1 and from there to the RS232#0 adapter using the /drivers/uart0 version (see subdirectory /doc in this driver for documentation). A low cost - general purpose RS232 serial to USB (e.g. Manhattan) converter is supposed to be available. An on-board FT232 to USB adapter can be installed, but is rather expensive. This example also uses the /drivers/adc/adctimer1 (see subdirectory /doc in this driver for documentation).
Example3: (/lcdserialanalog) It builds on Example 2 and adds the alphanumeric LCD interface shown in Figure 3. This example uses the low level /drivers/lcd/lcd4x20 routines, but configured for a simple 2x16 LCD. It also uses a higher level /modules/display set of routines, which “isolates” the display requirements from the specific display hardware.

The examples seek to strictly follow the In-House C Coding Rules (IHCR) as stated in :
[ref22] https://github.com/LyRIng/InHouseCCodingRules

3. EXAMPLES

These examples are prepared to be used with the **Codevision AVR IDE/Compiler** [ref01]. The program compiler is not free but relatively low-cost, very compact and efficient. A size-limited free version can be downloaded from: http://www.hpinfotech.ro/cvavr-download.html. The examples can be downloaded directly as .zip or by cloning the repo at: https://github.com/LyRIng/CL2bm1

If Microchip Studio [ref03] is installed (a much bigger program), a demo version plug-in of the Codevision AVR can be downloaded into Microchip Studio → Tools→Extensions and Updates (Online) as in Figure 4, to use the examples with no changes:

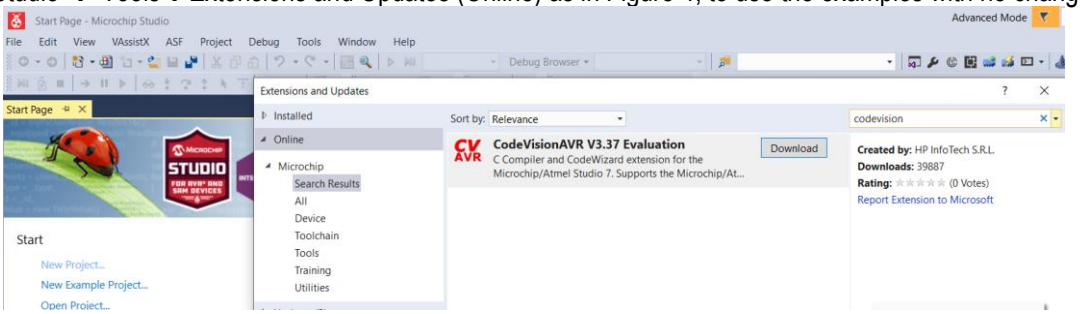


Figure 4 – CodevisionAVR as extension of Microchip Studio

3.a Example 1 / blinkled: This simple program simply blinks the OLED general purpose on-board LED. This example uses only the drivers/bsp routines (BSP stands for Board Support Package, usually low level hardware access specific to each board), to turn on or off the OLED output, which is mapped to PD.5 port as output, and uses simple blocking delays.
The directory structure is similar for all examples and is shown in Figure 5. The /test/blinkled1 directory has the project files generated via the Codevision / IDE compiler (*.prj is like a visual version of Makefile). If an automated test tool is used the testing files (e.g.Ceedling) will also be included in /test. Project-generated files are excluded from the repo with a specific .gitignore, but are regenerated when the example is rebuilt on your machine. The /drivers directory holds the specific low level routines (here only /bsp) and the /src/main.c and /inc/main.h files hold the user program.

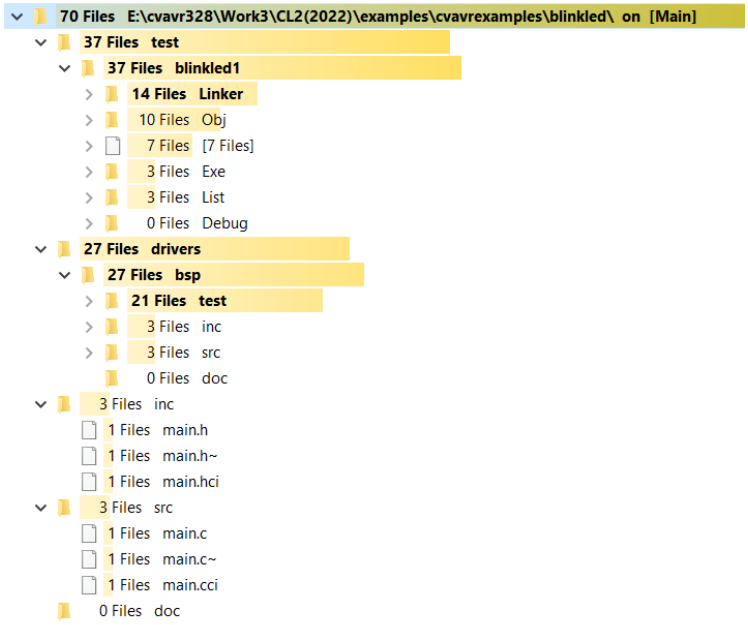


Figure 5 – Example 1 directory structure

3.a.1 Example 1 listing: This program is quite elementary but can be used to ensure that all board jumper settings (APPENDIX I) and interfaces are correct and that the compiler is working as expected.

```
/*****************************************************************************
**
** to use Doxygen
**
  * @brief  main() Blinkled function - Turns OLED on/off Version 1.2 5.03.22
  * @param  None
  * @retval none
  *
**
**/

void main(void)
{
      // CL2 board configuration
      // in /drivers/bsp
      CL2Board_init(CL2BOARD_SIMPLE);

    // Simple OLED blink with blocking delays
    for(;;){
        // in /drivers/BSP
        CL2Board_setOLED();
        CL2Board_delay_ms(ONE_SEC_DELAY_MS);
        CL2Board_clearOLED();
        CL2Board_delay_ms(HALF_SEC_DELAY_MS);
    }

}
```

Listing Example 1

3.a.2 Using CodeVision AVR: Open the IDE and navigate in the downloaded repository on your local machine to /examples/blinkled/test/blinkled1, and open the blinkled1.prj project File, as in Figure 6.

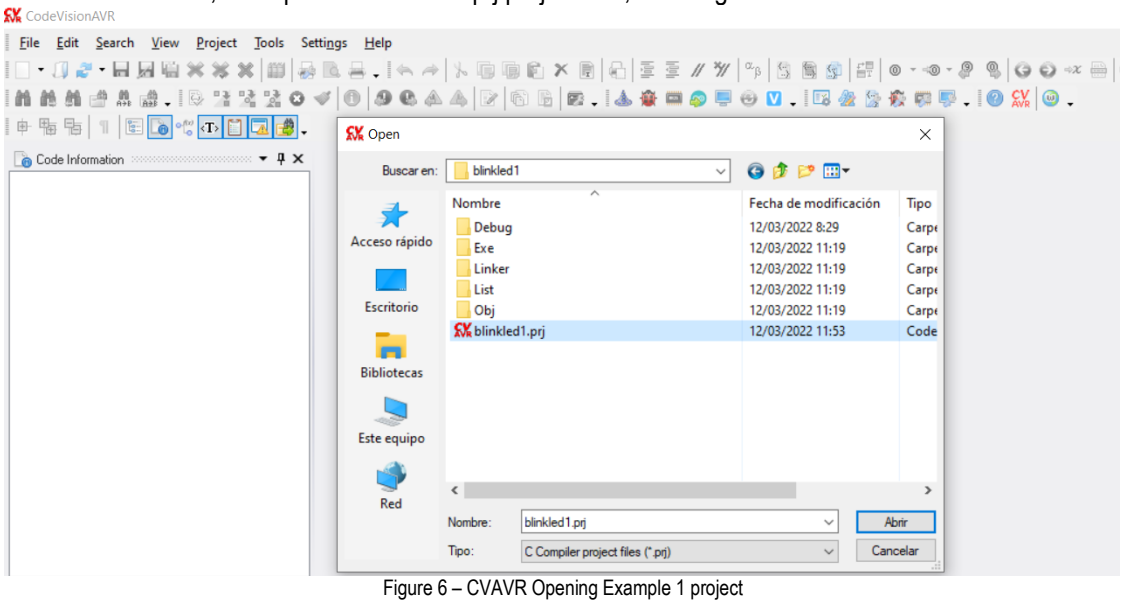


Figure 6 – CVAVR Opening Example 1 project

With File→ Open the contents of the source files can be reviewed, see Figure 7.

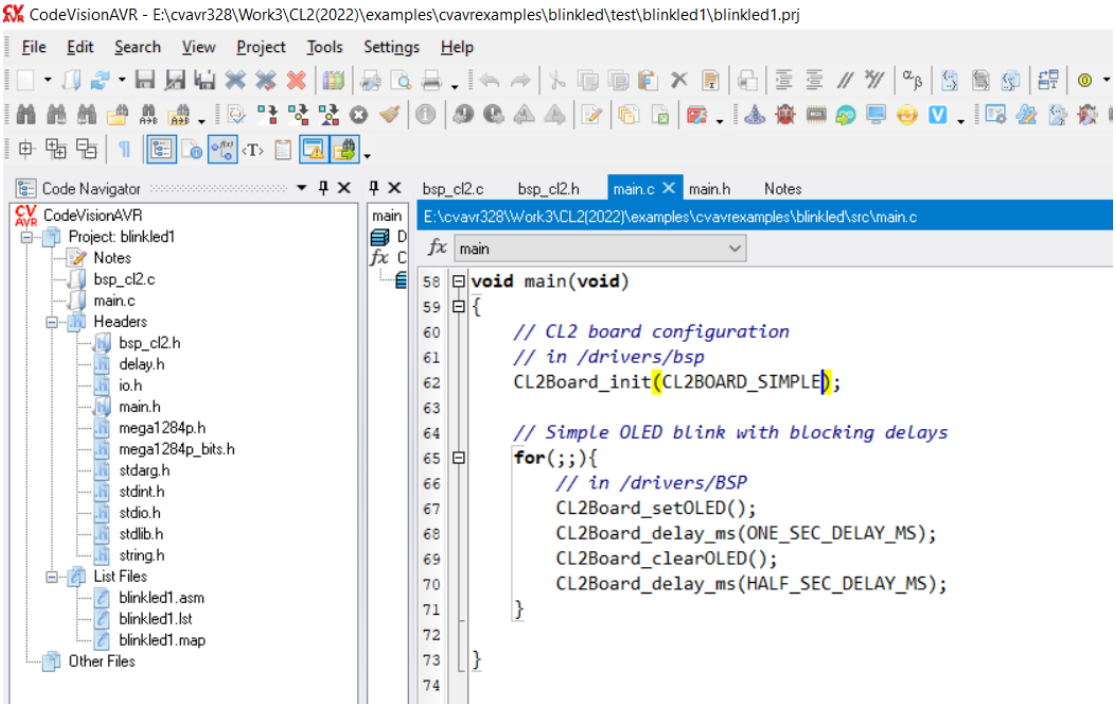


Figure 7 – Navigating the Example 1 project

Navigate in the main menu to **Project→Configure**, which works as a visual Makefile, the settings tell the compiler to use the .c files listed there under **Files**, and the compiler options under **C Compiler** to use the CL2bm1 board as in Figure 8.

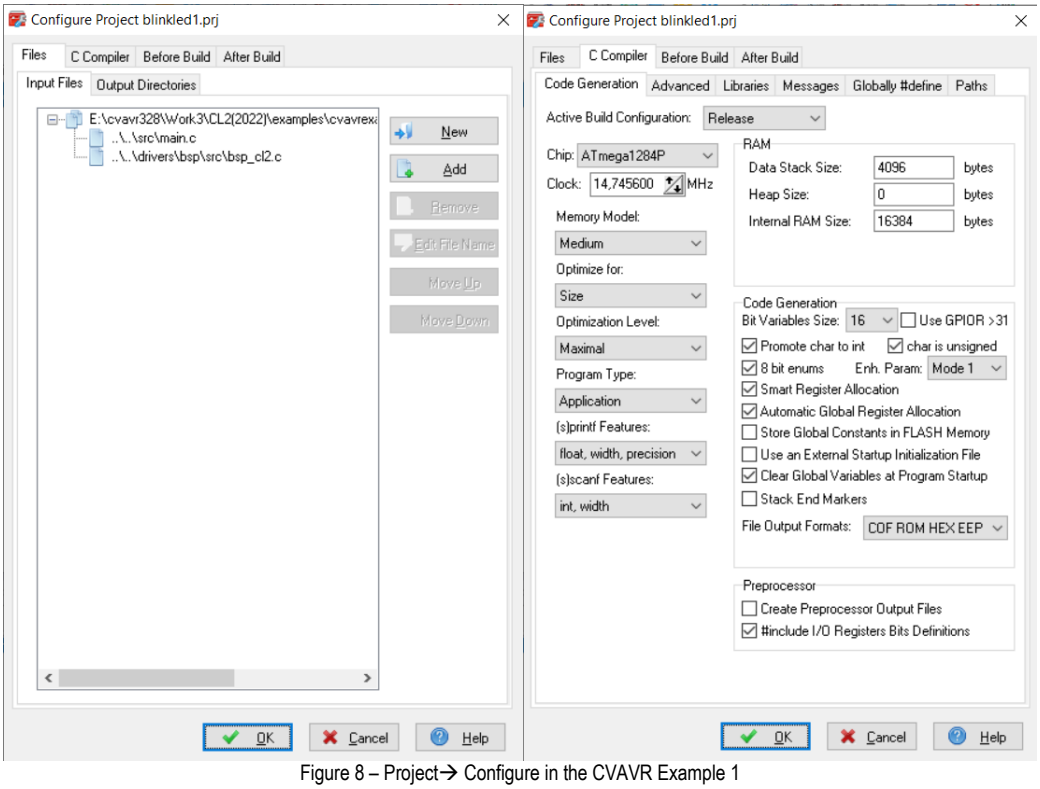


Figure 8 – Project→ Configure in the CVAVR Example 1

3.a.3 Building the /blinkled1.prj: Navigate again to the **Project** menu item. Here you can compile the code, with →**Compile** (or F9 shortcut) or directly **Build** (Shift-F9) the project (Figure 9a), which will compile and link the code, regenerating the output directories and files shown in Figure 5 under /test/blinkled1 directory. The IDE produces the outputs shown in Figure 9b in the F9 or Shift-F9 cases if no errors are found.

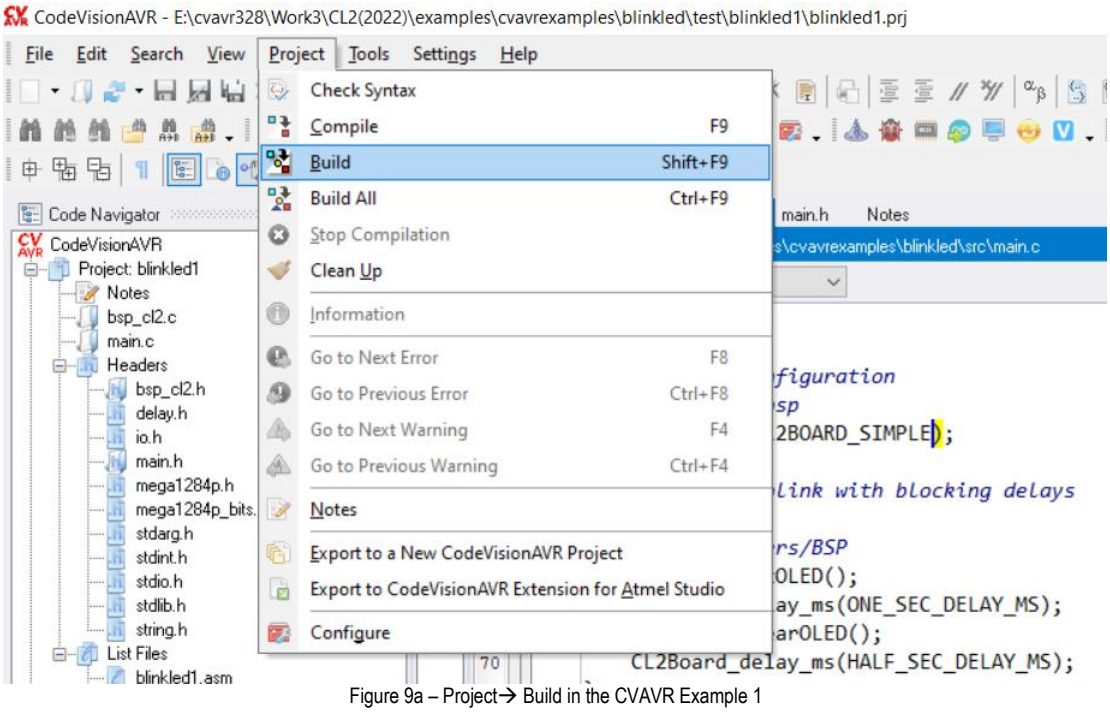


Figure 9a – Project→ Build in the CVAVR Example 1

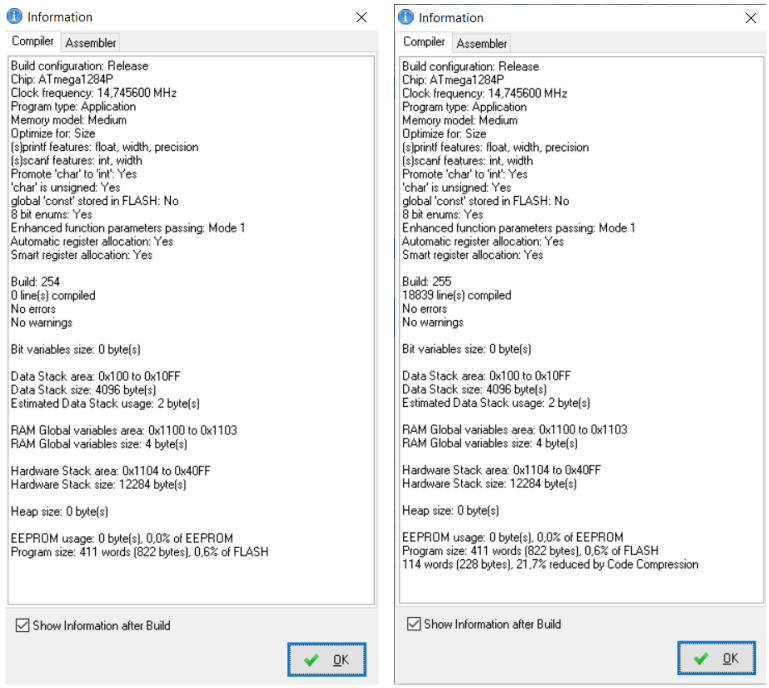


Figure 9b – Results of Project→ Compile (left) and Project→ Build (right) in the CVAVR Example 1

If **Build** proceeds correctly, the usable **blinkled1.hex** file to be downloaded into the controller flash memory will be in the /test/blinkled1/Exe as in Figure 10.

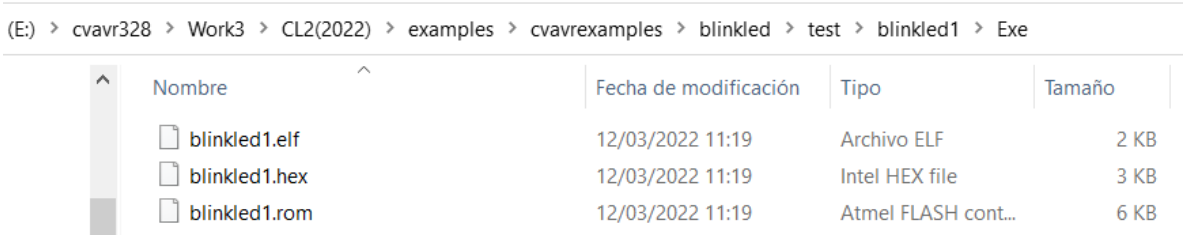


Figure 10 – Successful Build generates the populated /Exe directory as shown.

3.a.4 Testing on CL2bm1 board: Your CL2bm1 will probably have a preinstalled Optiboot Bootloader, which can be used together with the open source **AVRdude** tool [ref14] to write the flash microcontroller memory. To simplify testing this tool has been included in the repo under the /SendThruOptiboot2022 directory, which should appear as in Figure 11. The **blinkled1.hex** should be copied into this directory for testing.

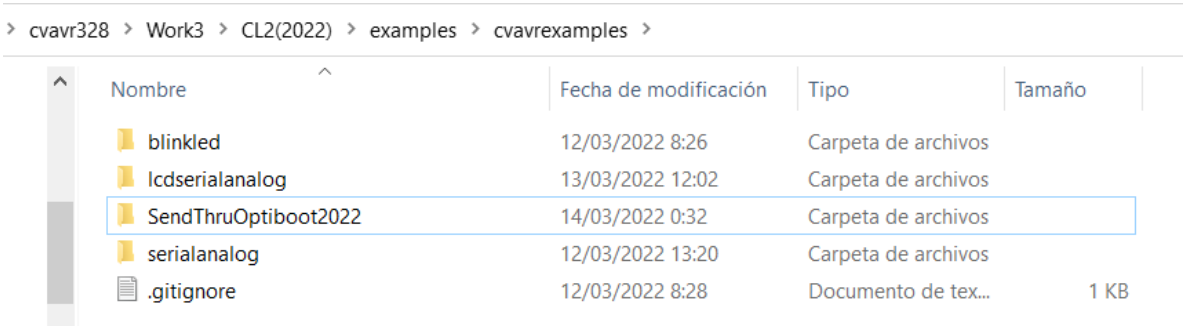


Figure 11a – /SendThruOptiboot2022 directory

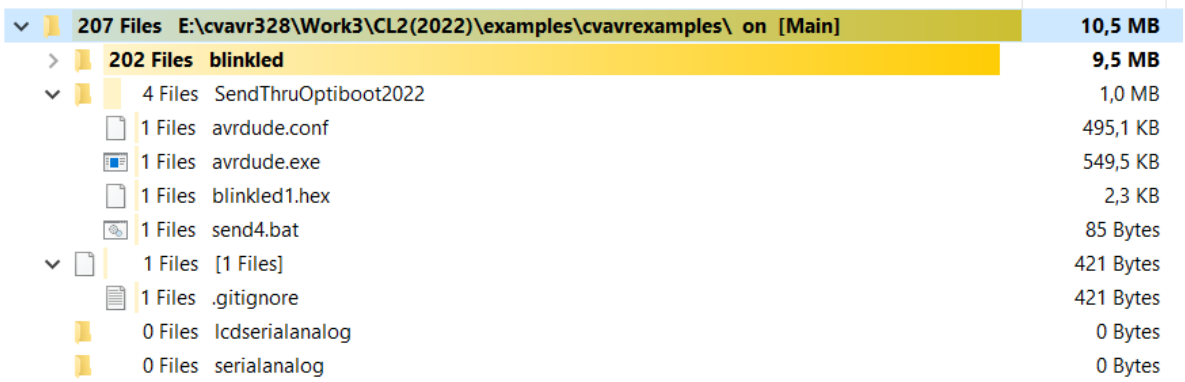


Figure 11b – Contents of /SendThruOptiboot2022 directory after blinkled1.hex is copied in it.

A simple way to use Optiboot bootloader and avrdude together, is to use the provided **send4.bat** from the standard Windows command prompt. The PC serial port in this batch file is set up to COM4, but can be set up to another COM as informed once the adapter is connected to the host machine. The single-line **send4.bat** (Listing 1) can be edited for example with free Notepad++ and its contents are:

```
avrdude -C avrdude.conf -v -patmega1284p -carduino -PCOM4 -b115200 -D -Uflash:w:%1:i
```

Listing 1

3.a.4.1 Connecting CL2bm1 board: Hook up a standard serial to USB Adapter cable (Figure 12-left) to the provided cable Figure 12-right, highlighted. Connect the CL2bm1 CN-COM1 male header (see Appendix 1) to the IDC female socket on the provided cable.

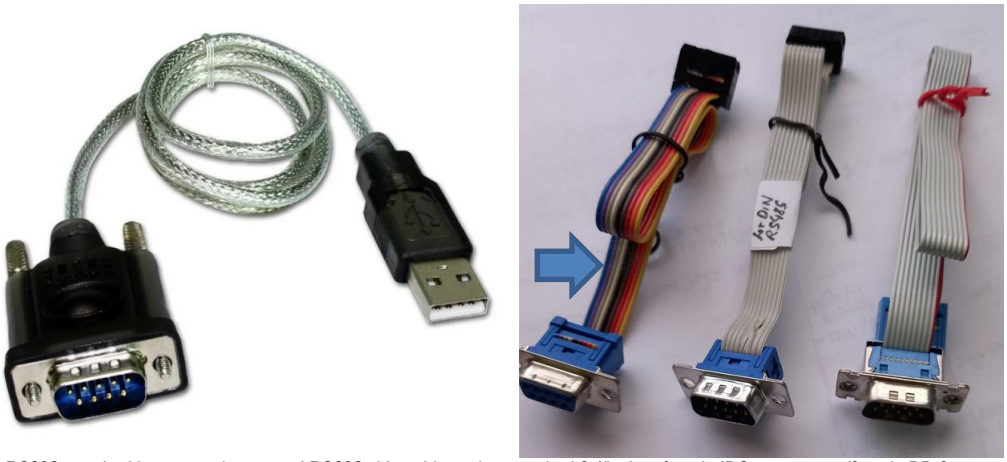


Figure 12 – USB to Serial RS232 standard low-cost adapter, and RS232 side cables using standard 0.1" crimp female IDC connectors (female DB-9 connector for CL2Bm1, left)

3.a.4.2 Program CPU on CL2bm1 board: In Windows Explorer, on this directory, click on the directory window and type "cmd". This should open a terminal in this directory (Figure 13a,b,c):

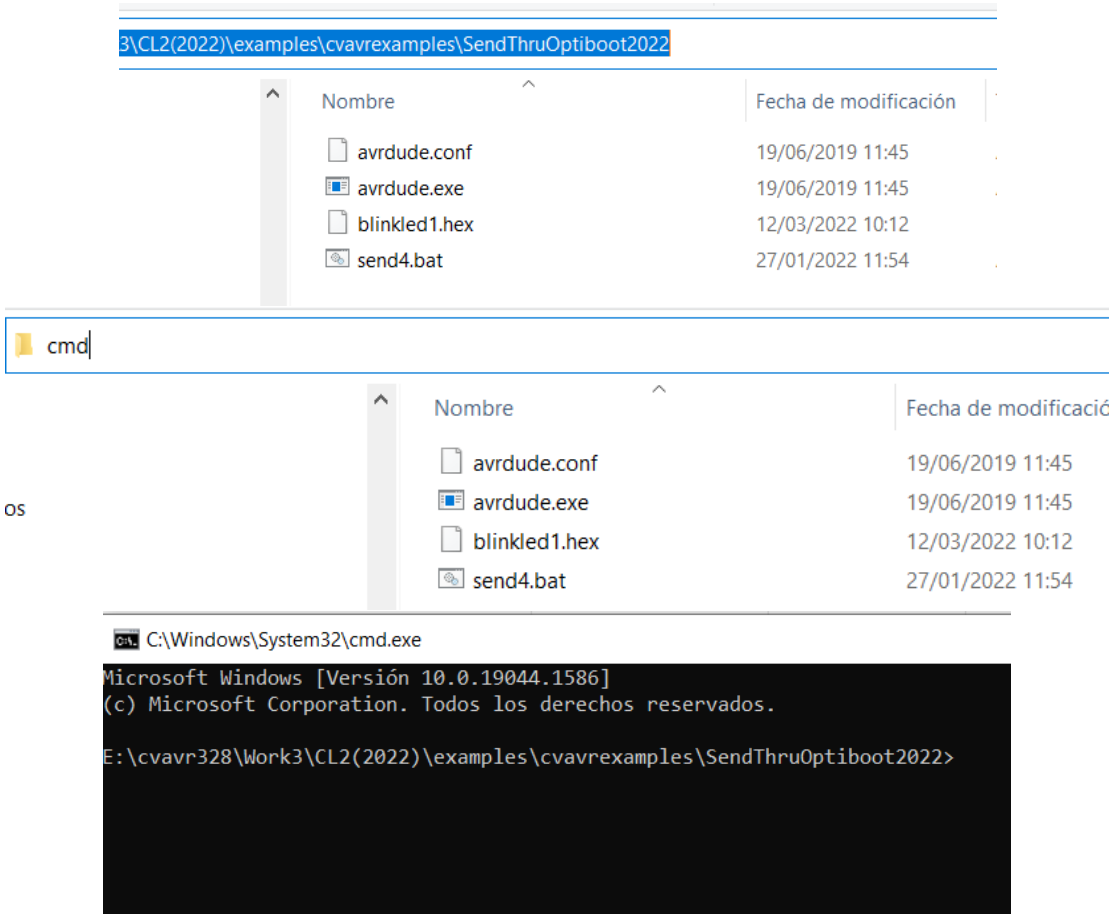


Figure 13a,b,c – Opening command prompt in Testing directory

As seen in previously shown Listing 1, the send4.bat requires COM4 to be setup, usually through a USB to Serial converter. If another port X is available, simple edit the send4.bat with Notepad++ and change the PCOM4 to PCOMX. Then simply type:

```
send4 blinkled1.hex (+ Enter)
```

Listing 2

After Enter, immediately turn **on** the CL2 board power. The Optiboot firmware should recognize that **avrdude** is trying to send your executable file **blinkled1.hex** to the ATMega1284P and should proceed as shown in Figure 14 (since the program is very small, it will go very fast):

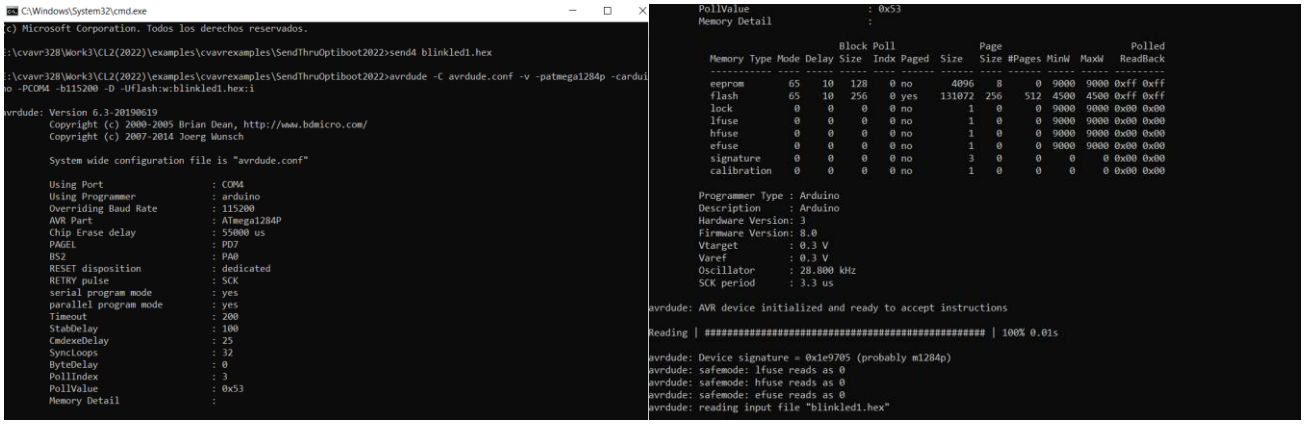


Figure 14a,b,c – Programming sequence of send4 command in Listing 2

The reason why this “works” is that the bootloader program (**Optiboot** here) resides in a special “boot” or startup region of Flash memory within the ATMega1284P that is executed on startup. Optiboot opens the COM0 port and expects a series of characters to recognize a programmer software (here **avrdude**) on the host machine. If the sequence is correct, a handshake protocol is carried out to program the internal “user” region of flash memory, then control is transferred to the user program, and Optiboot “releases” the COM0 port. In future documents the use of the ISP interface will be shown.

In Figure 15 the sequence of LED turning on/off is shown, and a link to a short video is shown.

.

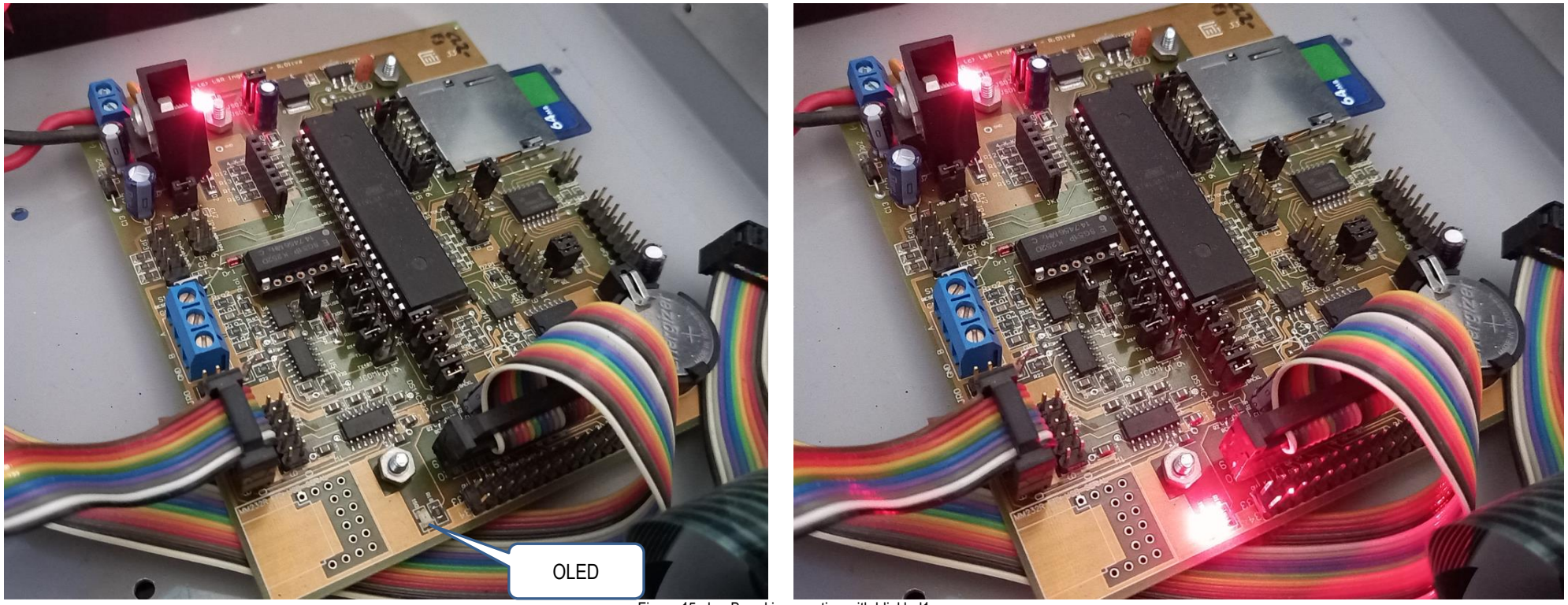


Figure 15a,b – Board in operation with blinkled1

Operation video:
https://drive.google.com/file/d/1k-SO-dKsKXhmRcZrhoVKRapqvFmy7bzG/view?usp=sharing

3.b Example 2 / serialanalog: This example builds on /blinkled example, adding communication via the same COM0 serial port with a terminal program, and reading the 8 on-board ADC channels. The directory structure is similar to Example 1, and is shown in Figure 16. The /test/serialanalog1 directory has the project files generated via the Codevision / IDE compiler. The /drivers directory holds the specific low level routines (here adc/adctimer1 and /uart0 is added to /bsp, /uart1 is not used but required by /bsp) and the /src/main.c and /inc/main.h files hold the user program.

Figure 16 – Example 2 serialanalog directory structure

3.b.2 Example 2 listing: This example (Listing Example 2) adds some complexity to the previous program. It uses the /uart0 driver with its own reception/transmission buffers and interrupt-driven (ISR) routines, and also the /adc/adctimer1 with specific interrupts and the auxiliary Timer1 ISRs. The ADC reading is triggered by Timer 1 and readings are copied to a global `adc_FPSChannel` array. Finally the readings are sent to the terminal in the same elementary OLED blinking loop.

.
```
/*****************************************************************************
**
** to use Doxygen
**
  * @brief  main() serialanalog function **  Version 1.2 5.03.22
  * @param  None
  * @retval none
  *
**
**/

void main(void)
{
    // CL2 board configuration
    // in /drivers/bsp
    CL2Board_init(CL2BOARD_SIMPLE);

    // CL2 BSP Terminal @COM0 19200 configuration
```

```
    // uses function in /drivers/uart0
    CL2Board_uartConfig(UART0_TER, UART_BAUD19200);

    // ADC_Init standard configurable Parameter
    // Parameters for ADC_param in ADC_Init()
    // Select (A) FreeRunning or (B) Timer1OVF as trigger,
    // ADC clock is 115.2 kHz @ 14.7456 MHz Clk
    // (A)#define ADC_INIT_FR_115K2   0
    // (B)#define ADC_INIT_TMR1_115K2 1
    // REMEMBER to set correct Parameter..(B here)
    ADC_Init(ADC_INIT_TMR1_115K2);

    // ***********************************************
    // 10.2.2018 - Added - Timer1 Use - Renamed 2022
    CL2Board_Timer1_Initialize();

    // Start interrupts
    SEI();
    printf(CRLF);
    printf(MAIN_VER);
    printf(COM_MSG1);
    // Analog Read, printout + simple OLED blink with blocking delays
    for(;;){
        // in /drivers/BSP
        if (1 == adc_vector_completed){
            ADC_Copy_Vector();
        }
        sprintf (s,"\n\r ADC Read:  ");
        printf("%s",s);
        for(j_1=0; j_1<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_1++){
             sprintf (s,"Ch(%d):%04d ", j_1, adc_FPSChannel[j_1]);
             printf("%s",s);
             }
        CL2Board_setOLED();
        CL2Board_delay_ms(HALF_SEC_DELAY_MS);
        CL2Board_clearOLED();
        CL2Board_delay_ms(HALF_SEC_DELAY_MS);
    }

}
```

Listing Example 2

3.b.3 Opening, compiling and building /serialanalog with CodeVision AVR: Open the serialanalog1.prj within the downloaded repository example, in /serialanalog/test/serialanalog1, which should give a screen similar to that shown in Figure 17. The configuration is shown in Figure 18 and is similar to Example 1 but with the addition of the new driver files.

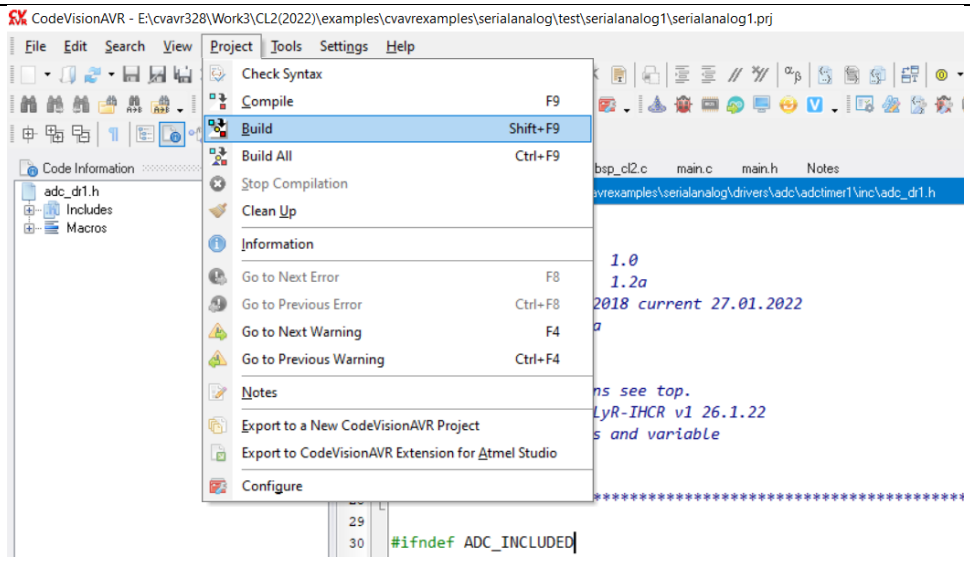


Figure 17 – Example 2 serialanalog project and building option

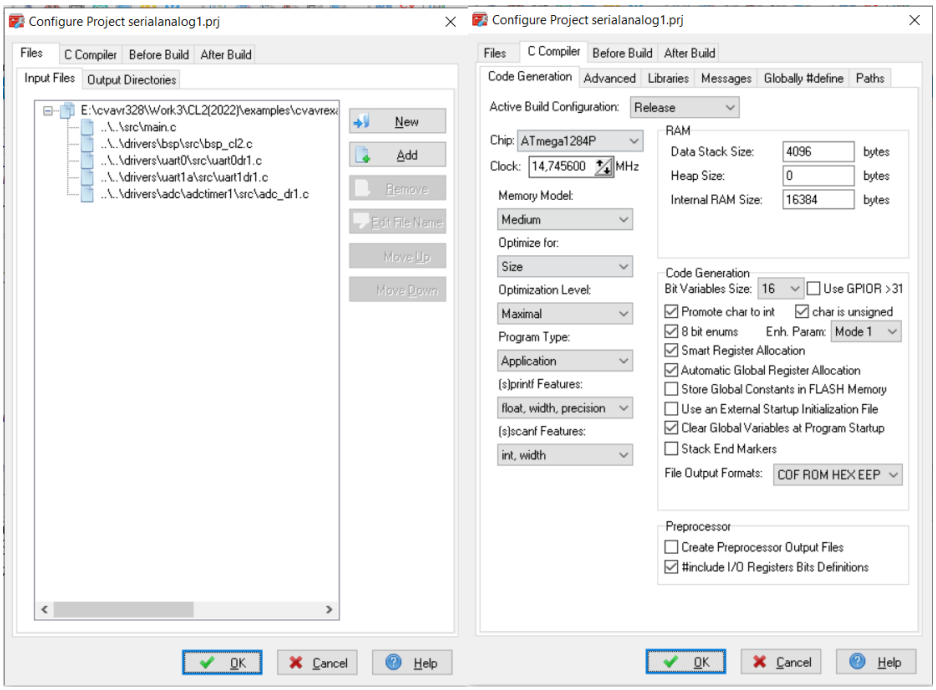
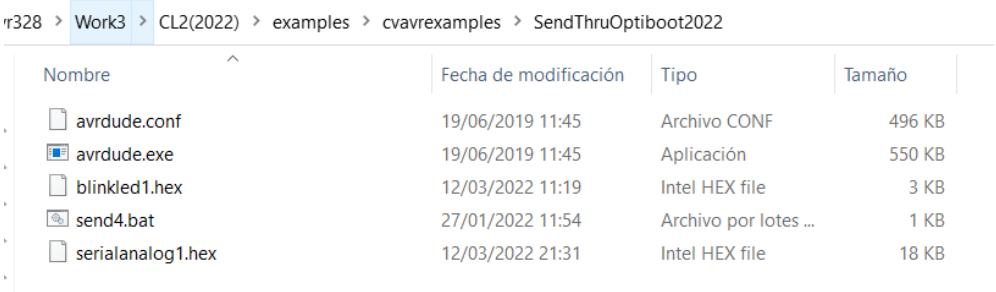


Figure 18 – Example 2 serialanalog project and building option

Compilation and building should proceed without error, as in Example 1. As before you should find serialanalog1.hex in the **/test/serialanalog/exe** directory, and copy it to **/SendThruOptiboot2022** directory, which should appear as follows:



3.b.4 Program CPU on CL2bm1 board: In Windows Explorer, on this directory, click on the directory window and type "cmd". This should open a terminal in this directory, as shown in Figures 13a,b,c. Now the command sequence should be:

```
send4 serialanalog1.hex (+ Enter)
```

Listing 3

After Enter, immediately turn **on** the CL2 board power. The Optiboot firmware should recognize that **avrdude** is trying to send your executable file **serialanalog1.hex** to the ATMega1284P and should proceed as shown in Figure 19.

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% -0.00s

avrdude: Device signature = 0x1e9705 (probably m1284p)
avrdude: safemode: lfuse reads as 0
avrdude: safemode: hfuse reads as 0
avrdude: safemode: efuse reads as 0
avrdude: reading input file "serialanalog1.hex"
avrdude: writing flash (6328 bytes):

Writing | ################################################## | 100% 0.94s

avrdude: 6328 bytes of flash written
avrdude: verifying flash memory against serialanalog1.hex:
avrdude: load data flash data from input file serialanalog1.hex:
avrdude: input file serialanalog1.hex contains 6328 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.67s

avrdude: verifying ...
avrdude: 6328 bytes of flash verified

avrdude: safemode: lfuse reads as 0
avrdude: safemode: hfuse reads as 0
avrdude: safemode: efuse reads as 0
avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.


E:\cvavr328\Work3\CL2(2022)\examples\cvavrexamples\SendThruOptiboot2022>
```

Figure 19 – Example 2 serialanalog programmed correctly on CL2bm1

3.b.5 Testing Serialanalog1 on CL2bm1 board, connected to terminal: Once the CL2bm1 is programmed and finished, **avrdude** releases the COM4 (or whichever is in use) in the Host machine, and CL2bm1 can be powered off.

3.b.5.1 Prepare a terminal program: We need to open a terminal program to see output from serialanalog1 Example 2. One of the possible programs is free TeraTerm [ref15], although many other can be used. Figure 20 shows the first connection with Teraterm. The use of the Serial option is necessary and here COM4 is connected to our USB-toRS232 adapter.

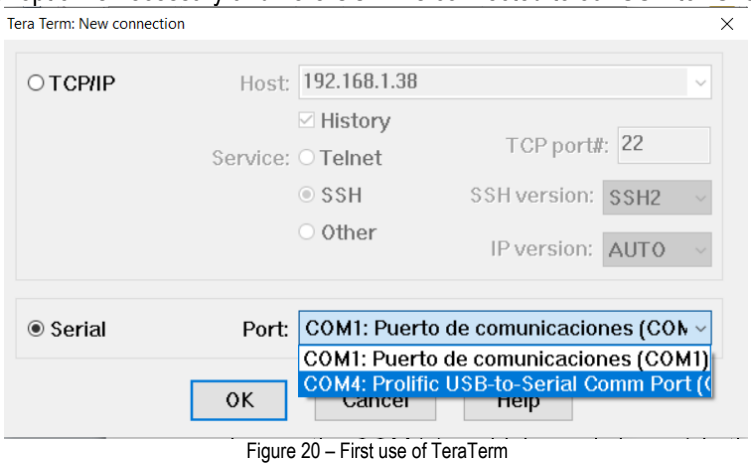


Figure 20 – First use of TeraTerm

Going back to Example 2 Listing on 3.a, we see one of the first instructions after initialization of the board is:

```
// CL2 BSP Terminal @COM0 19200 configuration
// uses function in /drivers/uart0
CL2Board_uartConfig(UART0_TER, UART_BAUD19200);
```

Listing 3

This function sets up the first serial port COM0 in the AVR controller at 19200 baud, 8 bits, no parity and 1 Stop bit. We need to set up our terminal on the host machine to work at the same parameters on COM4. In Teraterm, use Menu→Setup→Serial port .Here only choose 19200 (Figure 21), and then press New setting.

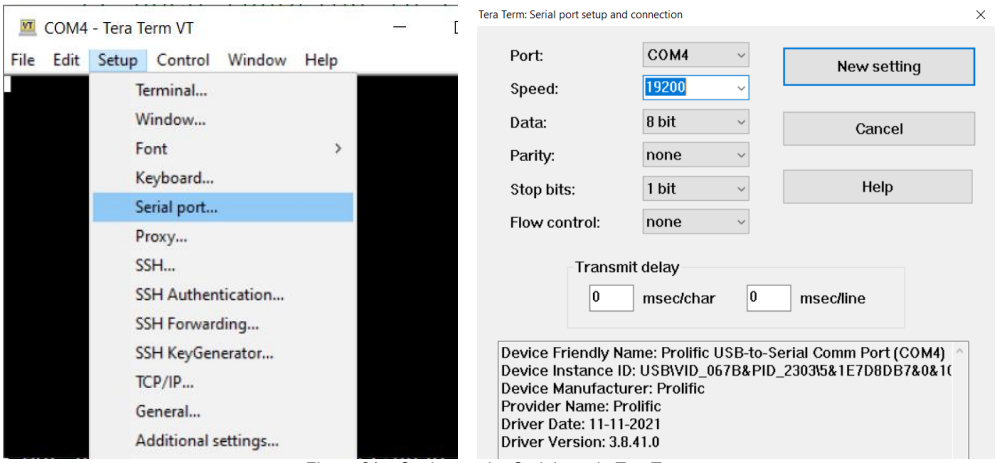


Figure 21 – Setting up the Serial port in TeraTerm

Then for convenience, this setup can be saved in a file using Menu→Setup→Save setup, and later retrieved with the similar Retrieve menu item.

3.b.5.2 Power on CL2bm1 connected to host machine with open terminal program: Finally, turn CL2bm1 power on again, and the terminal output should show something similar to Figure 22. At intervals coincident with LED Blinks, the sampled channels of the ADC inputs on **JA-1** (See Circled JA-1 header in Appendix I diagram) are read, since the ADC reference is connected to AVCC = +5Va on the board, the physical range is 0 to 5 V, referenced to ground, and the setup configures for single-ended ADC readings of Ch(0) to Ch(7). In this test, Ch(0) and Ch(7) where "jumpered" to ground, therefore their readings are 0.

```
SerialAnalog v1a 2022
Probando SerialAnalog v1a 11-03-2022

 ADC Read:  Ch(0):0000 Ch(1):0000 Ch(2):0000 Ch(3):0000 Ch(4):0000 Ch(5):0000 Ch(6):0000 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0039 Ch(2):0064 Ch(3):0089 Ch(4):0105 Ch(5):0121 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0039 Ch(2):0064 Ch(3):0089 Ch(4):0104 Ch(5):0120 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0038 Ch(2):0064 Ch(3):0088 Ch(4):0104 Ch(5):0120 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0038 Ch(2):0064 Ch(3):0088 Ch(4):0104 Ch(5):0119 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0037 Ch(2):0064 Ch(3):0087 Ch(4):0104 Ch(5):0119 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0037 Ch(2):0064 Ch(3):0087 Ch(4):0104 Ch(5):0119 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0087 Ch(4):0105 Ch(5):0119 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0118 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0035 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0035 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0035 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0086 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0036 Ch(2):0064 Ch(3):0087 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0037 Ch(2):0064 Ch(3):0087 Ch(4):0105 Ch(5):0119 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0037 Ch(2):0064 Ch(3):0087 Ch(4):0105 Ch(5):0120 Ch(6):0137 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0037 Ch(2):0064 Ch(3):0087 Ch(4):0104 Ch(5):0119 Ch(6):0136 Ch(7):0000
 ADC Read:  Ch(0):0000 Ch(1):0038 Ch(2):0064 Ch(3):0088 Ch(4):0105 Ch(5):0120 Ch(6):0136 Ch(7):0000
```

Figure 22 –TeraTerm output of serialanog1 using the above settings

Notice that the internal for() loop in Listing Example 2 instructs the unsigned integer values of `adc_FPSChannel[]` to be printed in sequence with left 0-padding, since the 10 bit ADC range will give values from 0 to 1023 (Listing 4). Codevision AVR also has a very complete floating point option for *sprintf()* and *printf()* routines. This will be shown in further examples.

```
for(j_1=0; j_1<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_1++){
        sprintf (s,"Ch(%d):%04d ", j_1, adc_FPSChannel[j_1]);
        printf("%s",s);
        }
```

Listing 4

3.b.5.3 Real connection of Analog inputs: As an example, Figure 23 shows a variable input test setup using a 10 kOhm variable resistor or potentiometer, connected to a clean analog +5Vdc source or AVCC, and to Ch(1) input. [**Caution** must be taken not to exceed 5 V in any input]. This setup will show Ch(1) readings the terminal program from 0 (0V) to 1023 (near 5V).

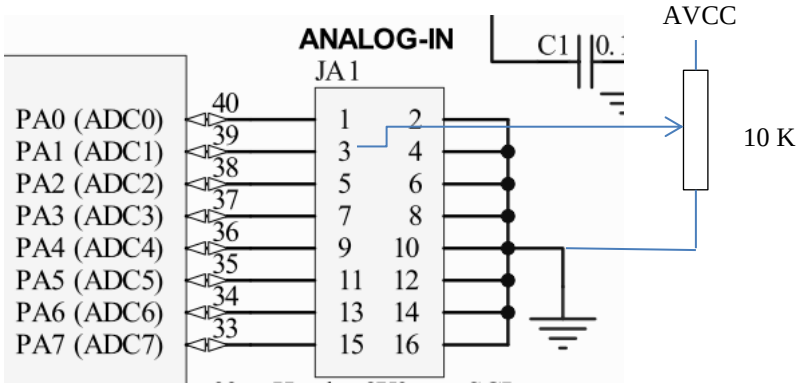


Figure 23 – Connection of a variable input on Ch(1) potentiometer – See page 2 of the CL2bm1 schematic in the repository.

For permanent connection, L&R Ing. offers analog signal conditioning boards such as the 4-channel M2, shown in Figure 24. This kind of board can be configured for specific input ranges or bought without input resistor components, which can be soldered by the end-user. Two of these would be necessary to cover the 8 available inputs of the CL2bm1 boards. In Figure 25, the schematic of the M2 board is shown, the first order RC filters and TS1854IDT [ref16] operational amplifier followers allow for slow-varying signals and are adequate for a large range of typical applications.

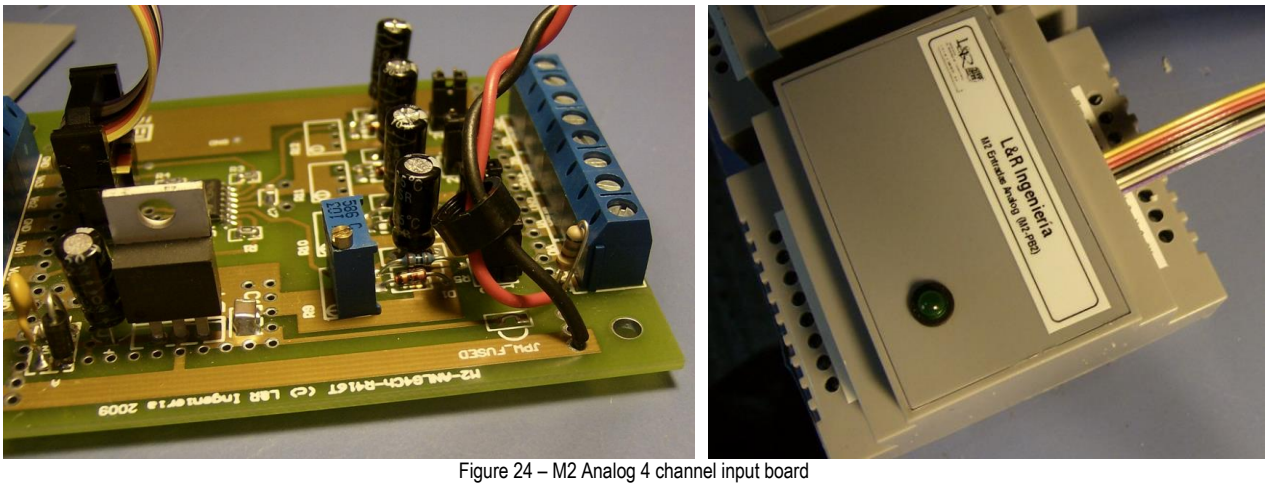


Figure 24 – M2 Analog 4 channel input board

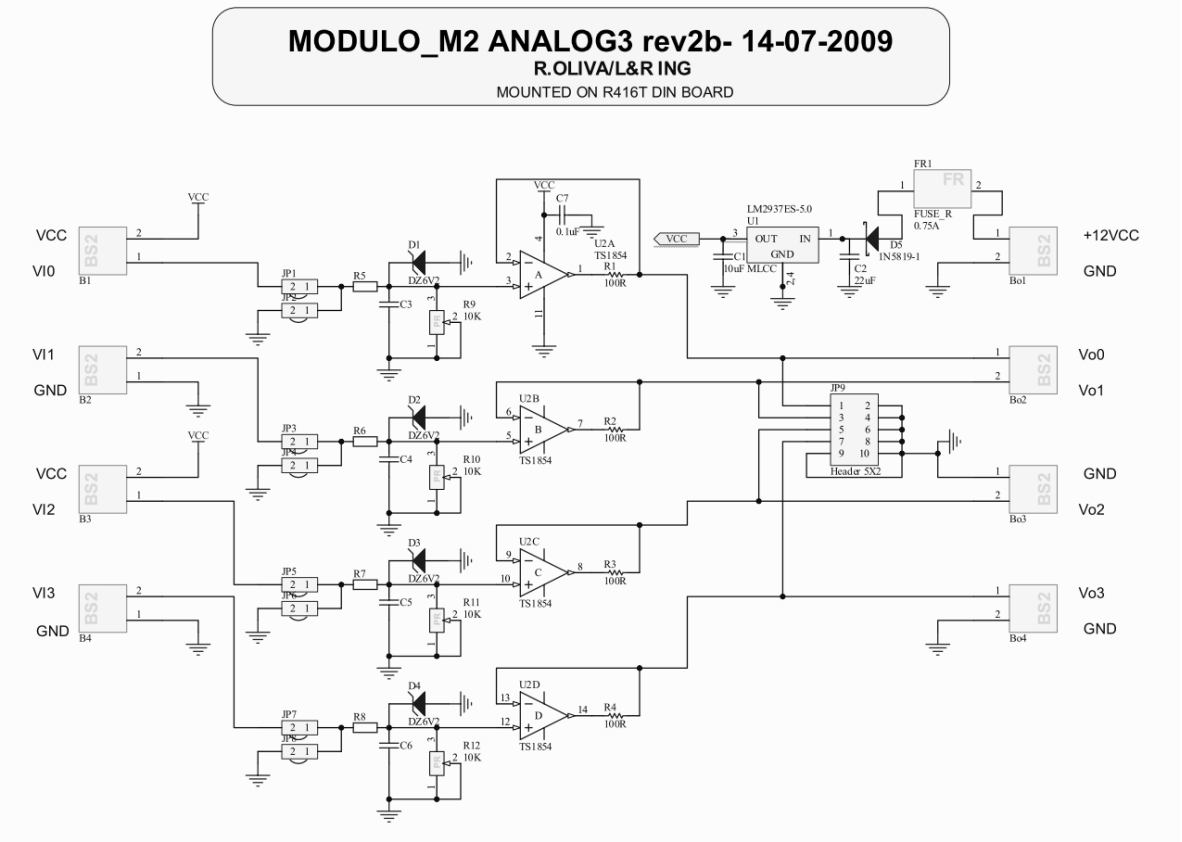


Figure 25 – L&Ring. M2 Analog board – 4 channel single-ended.

3.c Example 3 / lcdserialanalog: This example builds on the /serialanalog example, adding alphanumeric LCD display capabilities through the /lcd driver, and adding a higher level /module/display component. The directory structure is similar to Example 2, and is shown in Figure 26.

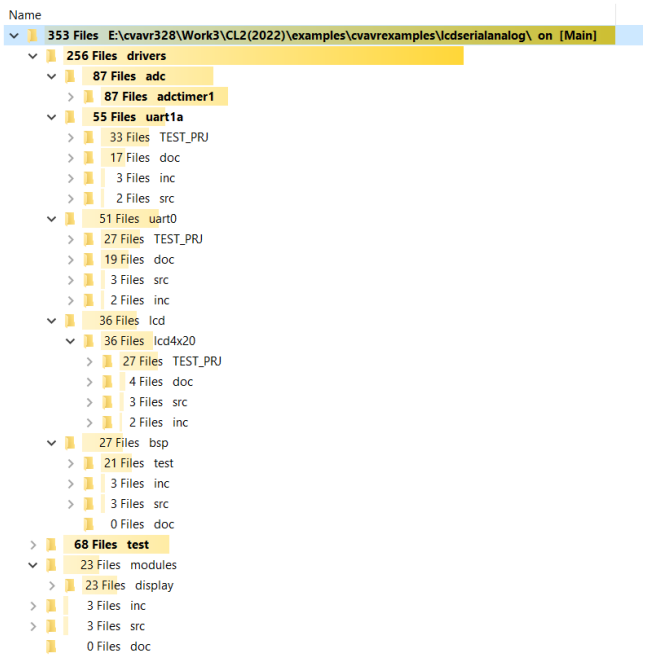


Figure 26 – Example 3 /lcdserialanalog directory structure

3.c.2 Example 3 listing: This example (Listing Example 3) adds an elementary LCD display capability to the previous program. It uses the alphanumeric LCD interface model shown in Figure 3, and the low level /drivers/lcd/lcd4x20 routines, but it is configured for a simple 2x16 LCD. It also uses a higher level /modules/display set of routines, which "isolates" the display requirements from the specific display hardware.

```
/*****************************************************************************
**
** to use Doxygen
**
  * @brief  main() lcdserialanalog function **  Version 1.2 5.03.22
  * @param  None
  * @retval none
  *
**
**/

void main(void)
{
    // CL2 board configuration
    // in /drivers/bsp
    CL2Board_init(CL2BOARD_SIMPLE);

    // CL2 BSP Terminal @COM0 19200 configuration
    // uses function in /drivers/uart0
    CL2Board_uartConfig(UART0_TER, UART_BAUD19200);

    // ADC_Init standard configurable Parameter
    // Parameters for ADC_param in ADC_Init()
    // Select (A) FreeRunning or (B) Timer1OVF as trigger,
    // ADC clock is 115.2 kHz @ 14.7456 MHz Clk
    // (A)#define ADC_INIT_FR_115K2   0
    // (B)#define ADC_INIT_TMR1_115K2 1
    // REMEMBER to set correct Parameter..(B here)
    ADC_Init(ADC_INIT_TMR1_115K2);

    // ***********************************************
    // 10.2.2018 - Added - Timer1 Use - Renamed 2022
    CL2Board_Timer1_Initialize();

    // Start interrupts
    SEI();

      // Initial messages
    printf(CRLF);
    printf(MAIN_VER);
    printf(COM_MSG1);

    // Initial Display Messages
    Display_Init(LCD2X16);
    Display_Screen(DISPLAY_WELCOME);

    // Analog Read, printout + simple OLED blink with blocking delays
    for(;;){
        // in /drivers/BSP
        if (adc_vector_completed == 1){
```

```
            ADC_Copy_Vector();
        }
        sprintf (s,"\n\r ADC Read:  ");
        printf("%s",s);
        for(j_1=0; j_1<(LAST_ADC_INPUT-FIRST_ADC_INPUT+1); j_1++){
             sprintf (s,"Ch(%d):%04d ", j_1, adc_FPSChannel[j_1]);
             printf("%s",s);
             }
        sprintf (s,"ADC Ch1:%04d    ",adc_FPSChannel[1]);
        Display_Line(1,s);
        CL2Board_setOLED();
        CL2Board_delay_ms(HALF_SEC_DELAY_MS);
        CL2Board_clearOLED();
        CL2Board_delay_ms(HALF_SEC_DELAY_MS);
    }

}
```

Listing Example 3

3.c.2 Example 3 hardware connection: The cabling of the display follows ACM1602 [ref17] but there are multiple manufacturers offering the same type of display (Figure 27). This display has an LED backlight which can be switched On or Off via the A+ and K- pins and for the CL2bm1 board through a driver transistor Q2 connected to the PD.6 pin. There are two connectors for the display on CL2bm1, the CN_LCD1 uses a 4-bit interface for the most common pinout.

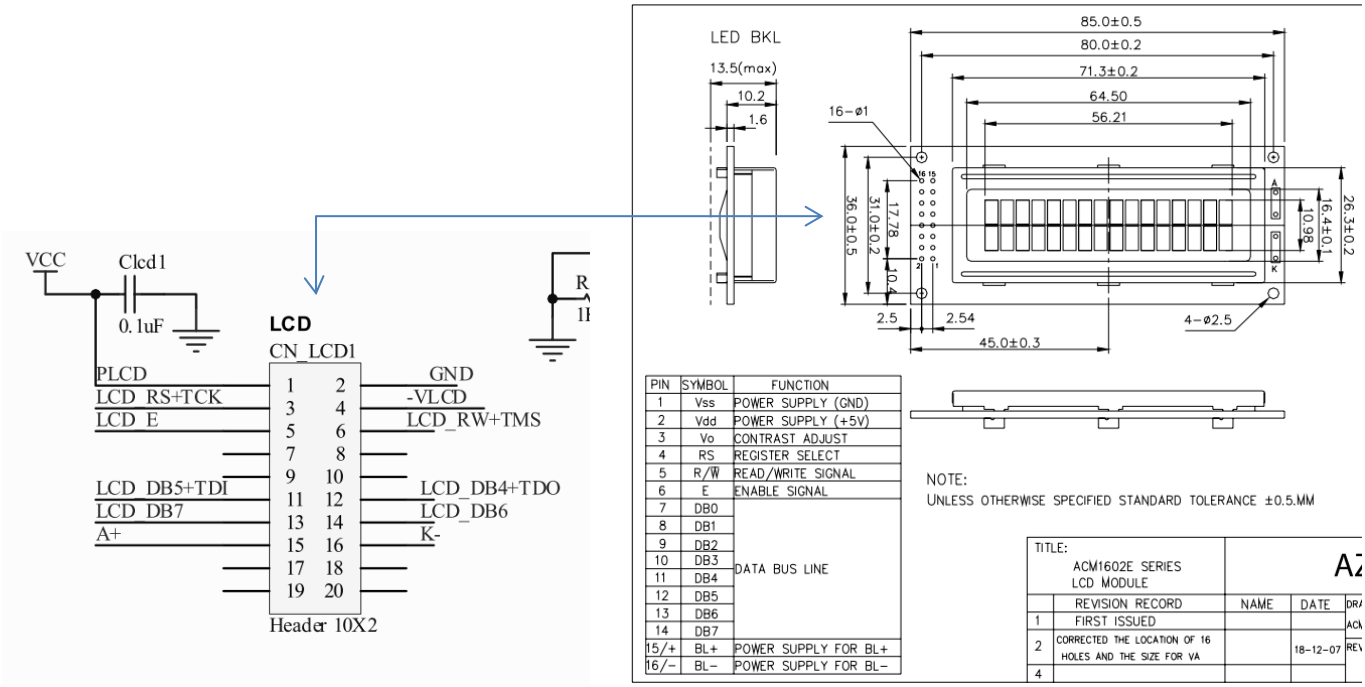


Figure 27 – Connection of Conventional LCD, here ACM1602 type to CN_LCD1 – See page 2 of the CL2bm1 schematic in the repository.

In Figure 28, the practical cabling of an ACM1602-type display to the CL2bm1 CN_LCD1 connector is shown. The cabling can be done with a provided cable or using colored or gray ribbon cable and two standard 2 x 10 IDC-female crimp connectors. Notice that the four lower pins (17 to 20) of the 2x10 CN_LCD1 are unused. The driver programs the output to use the lower DB4-DB7 four bit lines of the display data bus. Care must be taken with the cable placement but, as can be seen the direct connection of the cable is compatible with these displays. **Always connect components with power turned off**.

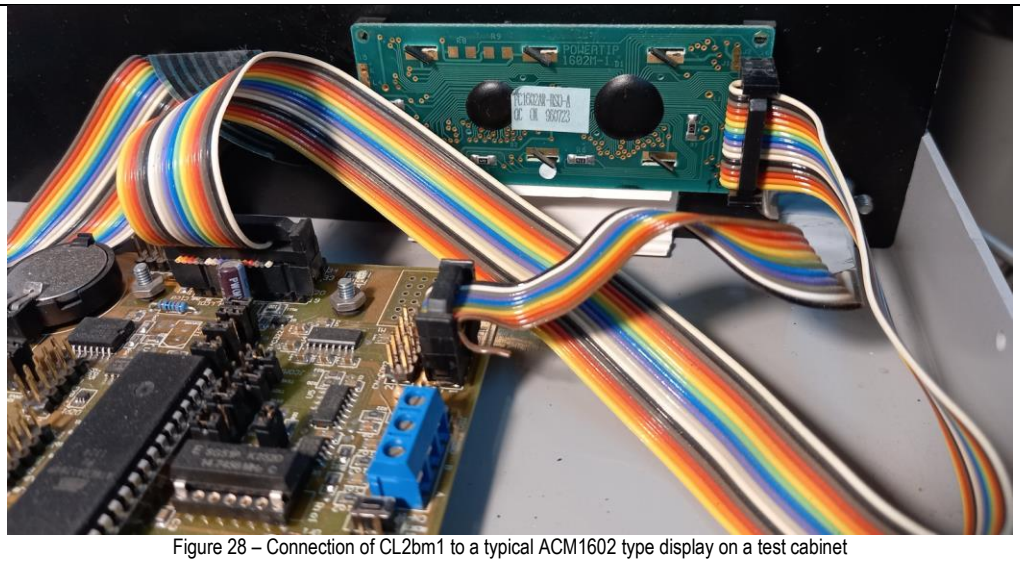


Figure 28 – Connection of CL2bm1 to a typical ACM1602 type display on a test cabinet

3.c.3 Build and Program CPU on CL2bm1 board: Building of the lcdserialanalog1.prj proceeds in a similar way as seen in section 3.b.3 for serialanalog1.prj, so none of that section is repeated here. The resulting lcdserialanalog1.hex file should be copied to the Optiboot testing directory as before, shown in Figure 29.

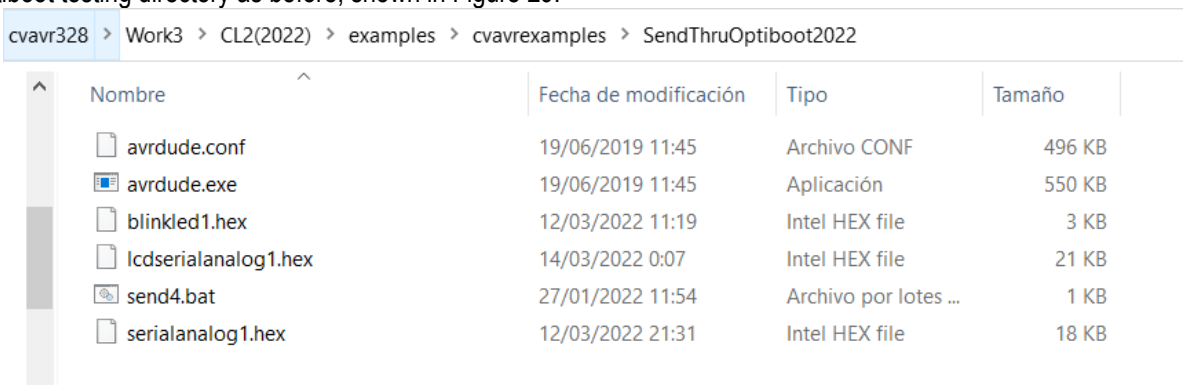


Figure 29 – lcdserialanalog.hex copied to the Optiboot testing directory.

For programming, in Windows Explorer, on the directory, click on the directory window and type “cmd”. This should open a terminal in this directory, as already shown in Figures 13a,b,c. Now the command sequence should be as in Listing 5, output on Figure 30:

`send4 lcdserialanalog1.hex` (+ Enter)
Listing 5

```
avrdude: reading input file "lcdserialanalog1.hex"
avrdude: writing flash (7420 bytes):

Writing | ################################################## | 100% 1.08s

avrdude: 7420 bytes of flash written
avrdude: verifying flash memory against lcdserialanalog1.hex:
avrdude: load data flash data from input file lcdserialanalog1.hex:
avrdude: input file lcdserialanalog1.hex contains 7420 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.78s

avrdude: verifying ...
avrdude: 7420 bytes of flash verified
```

Figure 30 – avrdude programming lcdserialanalog.hex

3.c.4 Testing Example 3 on CL2bm1 board: This example is similar to Example2, with the additional requirement of cabling the display. After the board is programmed, it can be turned off and the TeraTerm terminal at 19200 baud set up. The only different lines in example 3 relate to the LCD display. On startup, these lines configure the display and send a Welcome message:

```
// Initial Display Messages
Display_Init(LCD2X16);
Display_Screen(DISPLAY_WELCOME);
```

On the main loop, after "printing" on the Terminal all channels, a single channel reading - Ch(1) is sent continuously to the display, updated with each OLED blink. The output should be similar to that in Figure 30:

```
sprintf (s,"ADC Ch1:%04d    ",adc_FPSChannel[1]);
Display_Line(1,s);
```

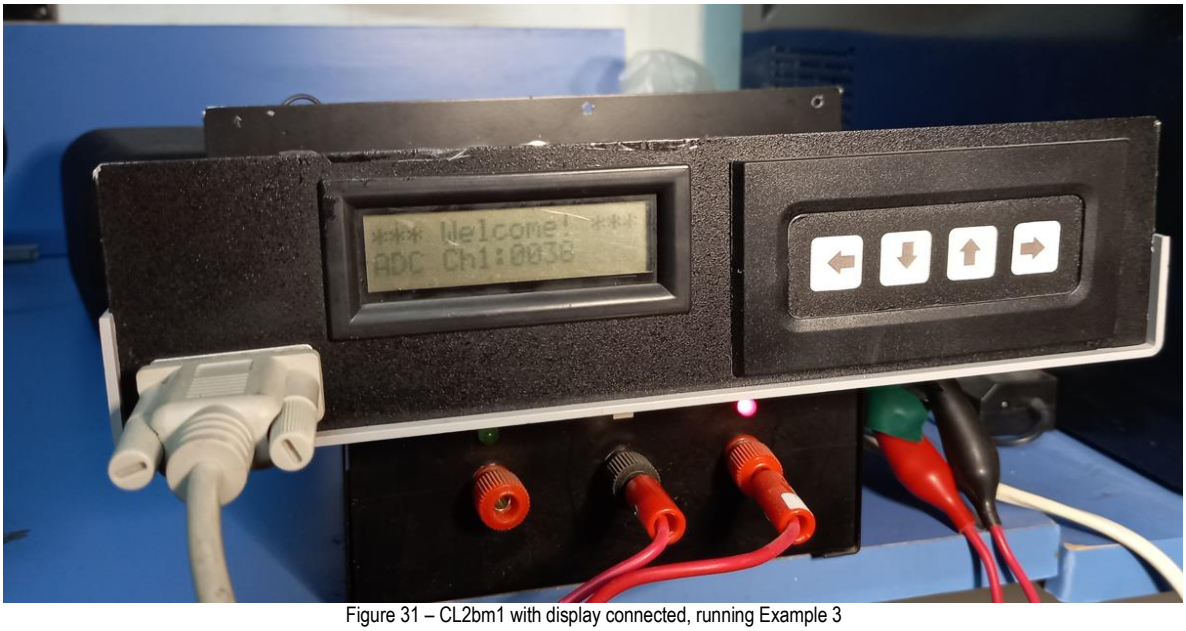


Figure 31 – CL2bm1 with display connected, running Example 3

4. CONCLUSIONS

A set of 3 example programs for the CL2bm1 board show the programming sequence using the Codevision CVAVR compiler and the Optiboot pre-installed bootloader. These examples in source code and documentation are available on the github repository for the CL2bm1 board: https://github.com/LyRIng/CL2bm1 Further references and additional applications can be found at [ref18]-[ref20].

5. REFERENCES

[ref00] ATMega1284P Microchip/AVR Microcontroller: https://www.microchip.com/en-us/product/ATmega1284P
[ref01] HP InfoTech / Codevision AVR C Compiler: http://www.hpinfotech.ro/cvavr-features.html
[ref02] AVR GCC compiler: from https://blog.zakkemble.net/avr-gcc-builds/
[ref03] Microchip Studio at https://www.microchip.com/en-us/tools-resources/develop/microchip-studio
[ref04] Microsoft Visual Studio Code https://code.visualstudio.com/
[ref05] Platform IO https://platformio.org/

[ref06] Arduino IDE https://www.arduino.cc/en/software
[ref07] MCUDude Optiboot https://github.com/MCUdude/optiboot_flash
[ref08] AVR-ISP Module: https://www.microchip.com/en-us/development-tool/ATAVRISP2
[ref09] Application 1 – SJ24 – See presentation: "Advances in greenhouse measurement and control systems using renewable energies in periurban locations" in https://en.energiasalternativas-unpa.net/publicaciones
[ref10] Application 2 – INTI https://www.lyringenieria.com.ar/power-curve-evaluation-system-for-small-wind-turbines-swts/
[ref11] M4E board https://github.com/LyRIng/PlacaM4-E
[ref12] METEO module https://github.com/LyRIng/METEO
[ref13] AVR Internal ADC – Driver documentation https://github.com/LyRIng/CL2bm1/examples-doc/ADC
[ref14] AVRDude open programming tool: https://www.nongnu.org/avrdude/
[ref15] Teraterm Terminal program, download: https://osdn.net/projects/ttssh2/releases/ [
[ref16] TS1854 IDT quad op-amp, SO-14 https://www.st.com/en/amplifiers-and-comparators/ts1854.html#sample-buy
[ref17] AmericanZettler ACM Displays – ACM1602 model: https://www.azdisplays.com/PDF/acm1602e.pdf
[ref18] CL2b board and interfaces: https://www.lyr-ing.com/Embedded/LyRAVR_CyEn.htm
[ref19] CL3 board: https://www.lyr-ing.com/Embedded/LyRCl3%2BM5E_En.htm
[ref20] UNPA-AEA Site: https://www.energiasalternativas-unpa.net/
[ref21] CL2b schematic https://github.com/LyRIng/CL2bm1/tree/master/schem
[ref22] IHCR / L&Ring. In House C Coding Rules: https://github.com/LyRIng/InHouseCCodingRules

Revision date: June 12th, 2022

APPENDIX 1 : CL2bm1 Typical Jumper Settings

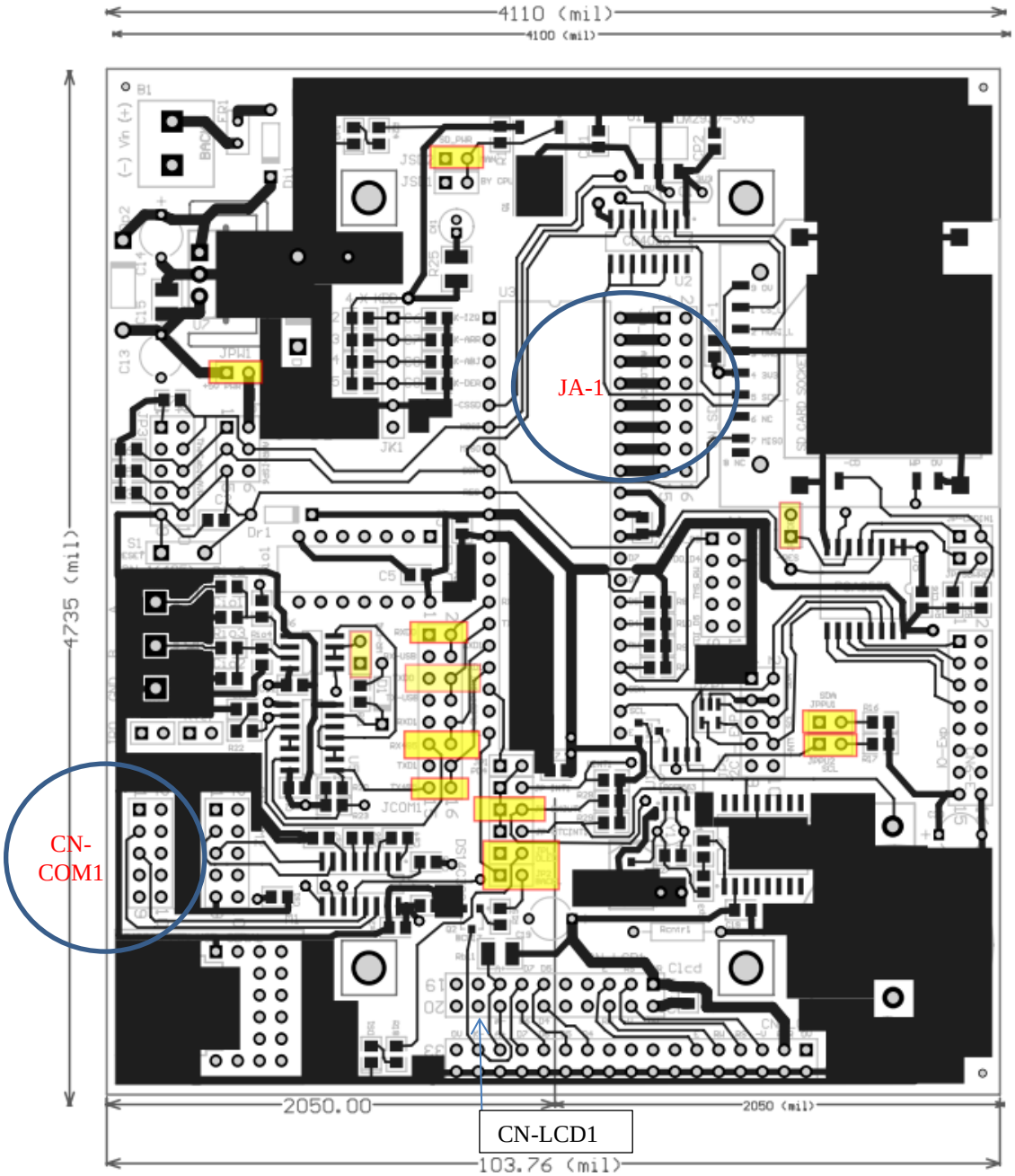


Figure A.1 - Jumper Settings (includes INT3 and rest)

ON – Jumpers highlighted in yellow - 13 jumpers total
- PWR +5V on (JPW1)
- Communications with JCOM1 - RS485 set to COM2 with JP-AutoEn , COM1 to RS232. (In text, sometimes COM0 refers to COM1, COM1 to COM2) – See Circled **CN-COM1** for programming thru Optiboot.
- I2C pullup enabled (JPPU1,2)
- SD Power on (JSD to manual)
- CKOut IRQ routed to PD.4 (JPCKOUT)
- OLED enabled (JP-OLED)
- BL (LCD Backlight) on (JP-BL).

Also see Circled **JA-1** for Analog Inputs referred to ground, 0 to 5V, and **CN-LCD1** for LCD example.