MSP430 Workshop

MSP430 MCU Teaching ROM Upgrade - MSP430 Microcontrollers Essentials - 2nd edition

Texas Instruments Incorporated
University of Beira Interior (PT)

Pedro Dinis Gaspar, António Espírito Santo, Bruno Ribeiro
University of Beira Interior, Electromechanical Engineering Department
dinis@ubi.pt, aes@ubi.pt, bruno@ubi.pt

OUTLINE

MSP430 Teaching ROM Tour

Example demo: F5529 Experimenter board

Example demo: eZ430-Chronos kit

>> Contents

WORLDWIDE PREMIÈRE

MSP430 Teaching ROM
2nd Edition

Chapter 4
MSP430F5529 Experimenter Board
Introduction

Texas Instruments Incorporated
University of Beira Interior (PT)

Pedro Dinis Gaspar, António Espírito Santo, Bruno Ribeiro
University of Beira Interior, Electromechanical Engineering Department
dinis@ubi.pt, aes@ubi.pt, bruno@ubi.pt

>> Contents

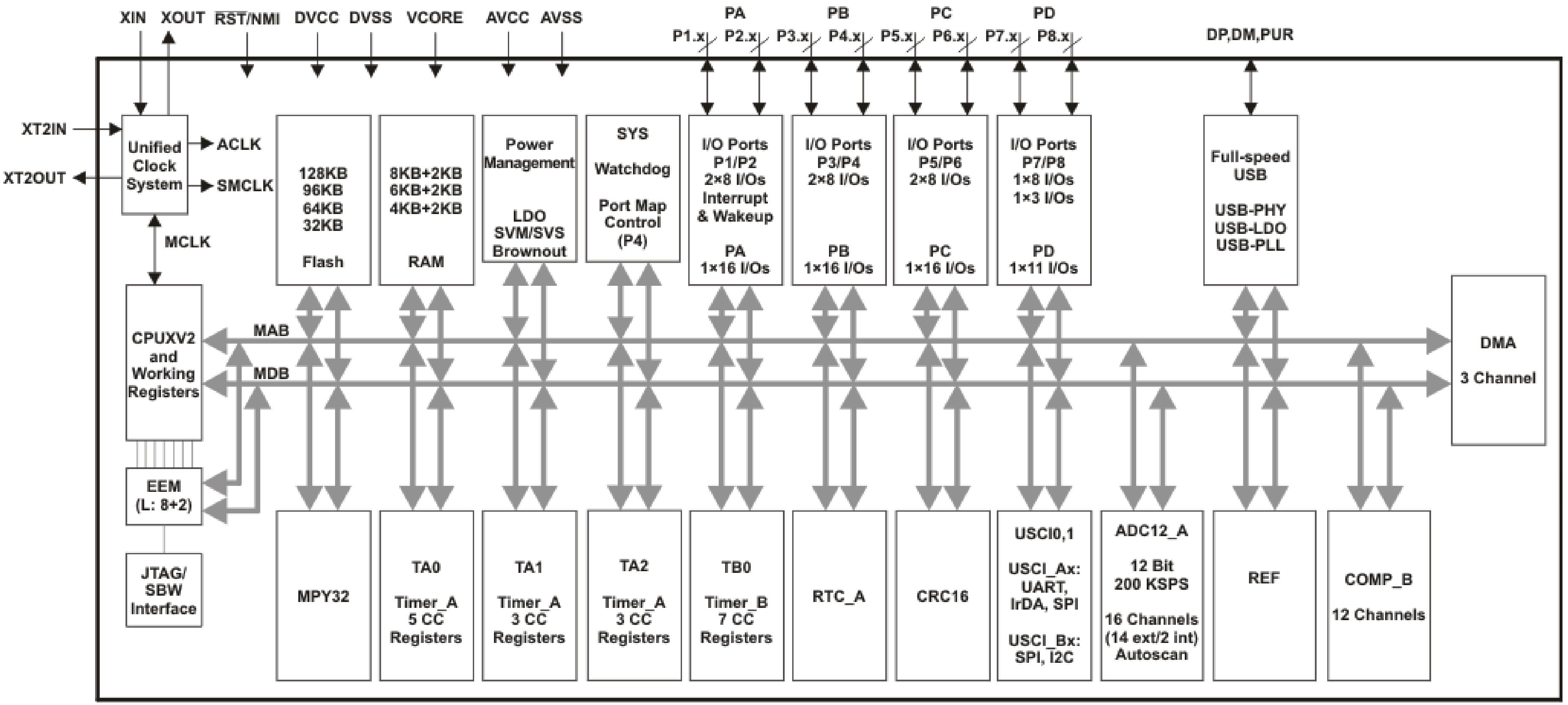
Contents

MSP430F5XX Family Overview

- The family of microcontrollers from TI has several low-power devices that include different peripherals for various applications.
- The architecture combined with the low power modes of operation makes this family very suitable for low-powered battery applications.
- Detailed information concerning the MSP430F5529 can be found here.
- The datasheet can be consulted to give more specific data.
- For information on the family the user guide should be consulted.

MSP430F5XX Family Overview

- The processing unit fitted in the MSP-EXP430F5529 has an architecture as shown below:

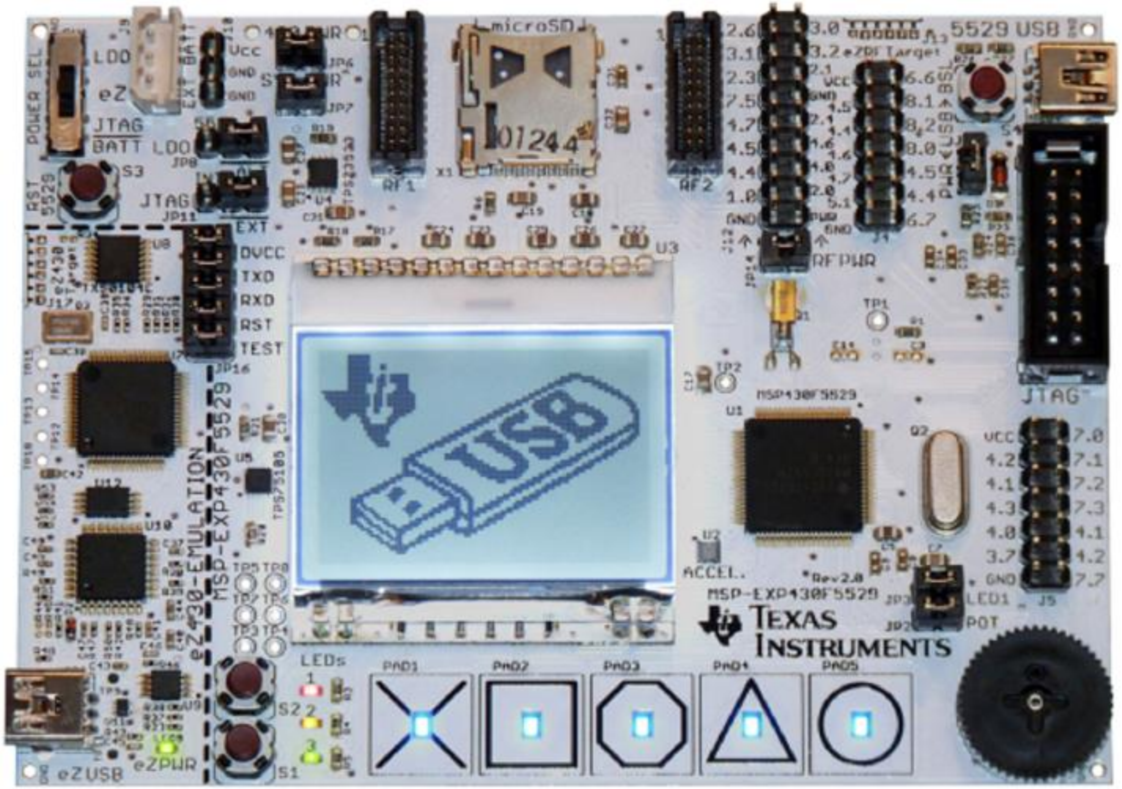


Note: Memory sizes, available peripherals, and ports may differ, depending on the device.

>> Contents

MSP-EXP430F5529 Experimenter Board Contents

- **What you can expect from the MSP-EXP430F5529 Experimenter Board**
  - A laboratory set included in this chapter has been developed using the MSP-EXP430F5529 Experimenter Board.



MSP-EXP430F5529 Experimenter Board Contents

- **In addition to the MCU MSP430F5529, a number of features listed below are included in the board:**
  - LCD dot-matrix 102x64;
  - Interface for microSD memory card;
  - 3-axis accelerometer;
  - Five capacitive buttons and four pressure buttons;
  - Analog thumb-wheel;
  - Nine LED;
  - General interface to MCU pins;
  - Ability to program the MCU either via JTAG using an external FET, using the emulator or on-board;
  - Different power possibilities: JTAG, eZ430 emulator, LDO.

>> Contents

MSP-EXP430F5529 Experimenter Board Contents

- **MSP-EXP403F5529 hardware components**

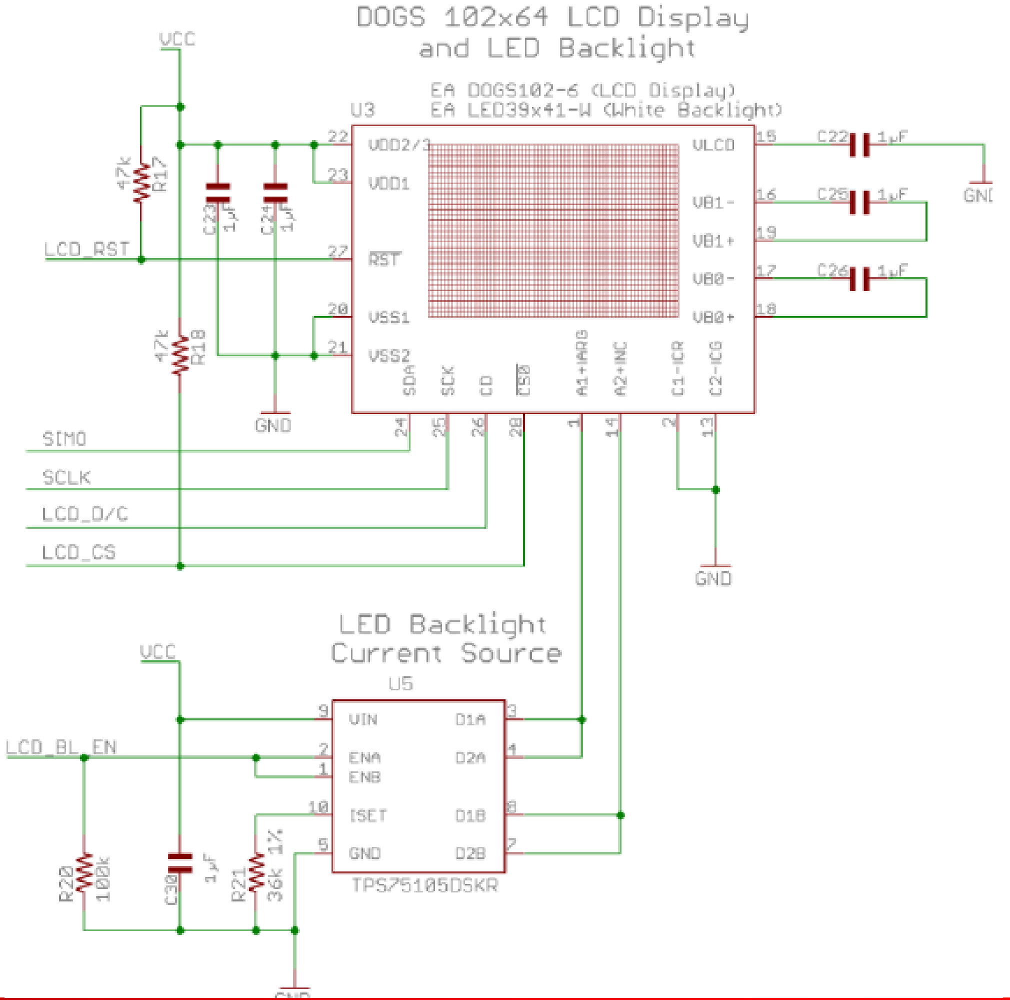
- **Dot-matrix LCD**
  - The user can interact with the MSP-EXP430F5529 Experimenter board through the 106x64 resolution LCD dot-matrix produced by Electronic Assembly, reference EA DOG102W-6.
  - The integrated back-light driver can be controlled by a PWM signal originating from the MCU via pin LCD_BL_EN (P7.6).
  - The interface with the MCU is made via an SPI connection using the SIMO (P4.1/PM_UCB1SDA), SCLK (P4.3/UCB!STE), LCD_D/C (P5.6), LCD_CS (P7.4) and LCD_RST (P5.7) pins.

>> Contents

MSP-EXP430F5529 Experimenter Board Contents

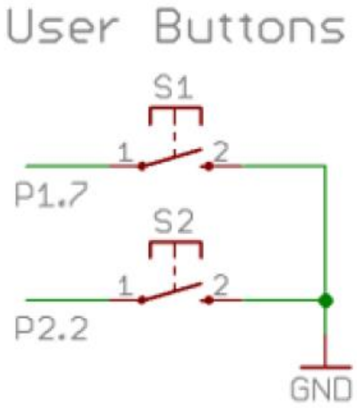
- **MSP-EXP403F5529 hardware components**

- **Dot-matrix LCD**
  - The data transfer is unidirectional.
  - Only data can be written to the device.
  - Mode 3 SPI protocol: (MSB first).
  - SCL up to 33MHz(3.3V)
  - Additional info here.



❑ Pushbuttons

- Two general purpose pushbuttons S1 (P1.7) and S2 (P2.2).
- No pull-up resistors (port internal resistances should be used).



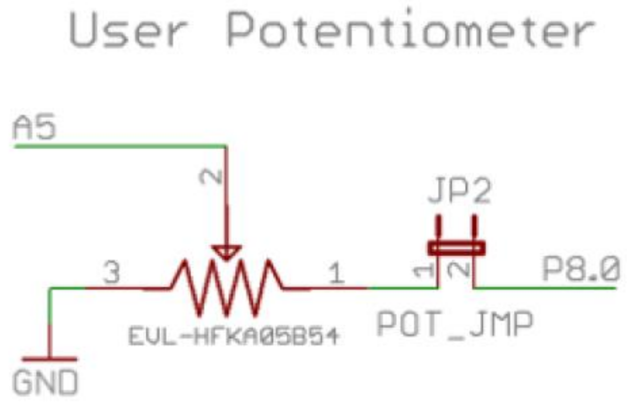
- Two additional buttons which are used for specific functions:
  - The button S3 allows resetting the MCU.
  - The button S4 triggers the BSL process through the USB port.

❑ Potentiometer wheel

- Another interface, quite versatile, is the potentiometer wheel.

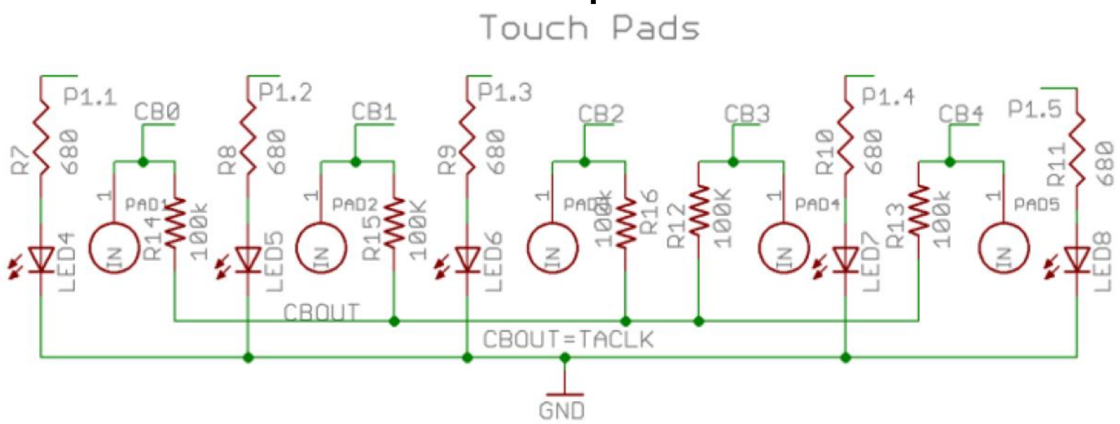
- This interface can be disconnected via jumper JP2.

- The position is determined by acquiring the voltage value at the terminals of the voltage divider through the channel A5 of the ADC.



❑ Capacitive touch Pads

- Capacitive interface: five capacitive buttons:

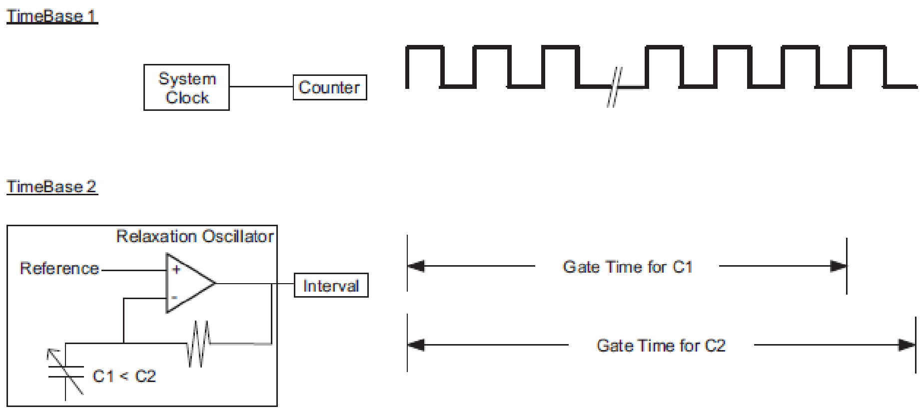


- In each of these buttons there is a LED, connected to ports P1.1 to P1.5, and which can be used by the user to indicate the operating state of the button or other functionality required.

- Each input CB0…CB4 is connected to an input of the comparator COMPB included in the MCU. Meanwhile, the CBOUT pin is connected to the comparator output.

❑ Capacitive touch Pads

- The relaxation oscillator is implemented by COMPB:



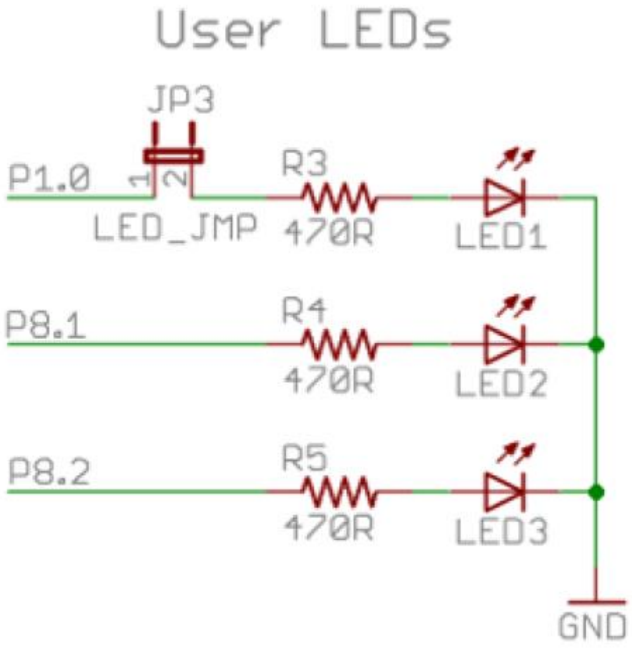
- Gate time variable with the period of the relaxation oscillator.

- Timer_A1 used to establish the number of oscillations (gate time).

- The capacity is represented by the number of software cycles counted during the gate time.

LEDs

- In addition to the aforementioned LED, the user still has four LEDs for general use.
- These LEDs are connected to pins:

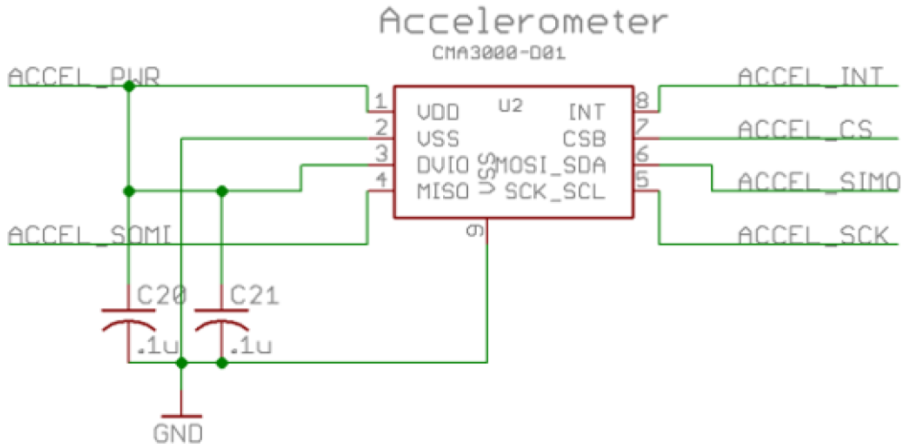


- Note that the P1.0 pin can be interrupted through jumper JP3.

MSP-EXP430F5529 Experimenter Board Contents

- **Three-axis accelerometer CMA3000-D01**
  - The digital accelerometer included on the MSP-EXP430F5529 Experimenter Board is capable of measuring acceleration in 3-axes.
  - Additional information concerning the internal operation of this device can be found here.
  - The CMA3000-D01 has SPI and I2C digital interfaces and is optimized for systems with tight power requirements.
  - It is a small device powered between 1.7V to 3.6V.

- **Three-axis accelerometer CMA3000-D01**
  - Measure mode: 70/50/11 μA (sampling frequencies of 400/100/40 Hz respectively).
  - Motion detection mode: 7 μA (sampling frequency of 10 Hz).
  - Two measuring ranges: ± 2g or ±8g (8-bit resolution).
  - Connection of the accelerometer to the MCU:



- **MSP430_USB_Developers_Package**
  - One of the features of the F55xx family is the USB connectivity.
  - This feature will be explored during Laboratory 4.3.
  - To facilitate the development of applications that include a USB connection, the following API is available:
  - MSP430_USB_Developers_Package_3_1_0_0
  - This software package has additional application examples that the user can study.
  - Reading the application note Starting a USB Design is also recommended.

- **MSP430_USB_Developers_Package**
  - Additionally, an interesting document also describes the possibility of updating the MCU firmware via USB is the USB Field Firmware Updates on MSP430™ MCUs.
  - These documents can be accessed through the MSP430ware or alternatively by installing the package available here.

Hardware Abstraction Layer for the MSP-EXP430F5529

- **The development of applications for the MSP-EXP430F5529 Experimenter Board can be achieved more quickly if the Hardware Abstraction Layer (HAL) is used.**

- **This software package is composed of the following modules:**
  - HAL_AppUart: UART management;
  - HAL_Board: basic configuration of the Experimenter Board;
  - HAL_Buttons: buttons S1 and S2 management functions;
  - HAL_Cma3000: accelerometer management functions;
  - HAL_Dogs102x6; LCD management functions;
  - HAL_Menu: user menu interface management functions;
  - HAL_SDCard: SD card management functions;
  - HAL_Wheel: wheel management functions.

>> Contents

Project Preparation for Labs 4.1, 4.2 and 4.3

- Laboratories Lab4.1, Lab4.2 and Lab4.3 are developed in the same project.
- The structure used allows partitioning the laboratories.
- The main file Lab4a.c prepares the MSP-EXP43F5529 Experimenter Board to develop the laboratories.
- The project has been developed based on the example provided by TI with the libraries already compiled.
- This solution allows running the applications developed in the laboratories on the limited version of CCSv5.1.
- However, it is necessary to note that HAL already uses resources of the processor.

>> Contents

Chapter 4
MSP430F5529 Experimenter Board
Lab4.4: USB Interface with MatLab

Texas Instruments Incorporated
University of Beira Interior (PT)

Pedro Dinis Gaspar, António Espírito Santo, Bruno Ribeiro
University of Beira Interior, Electromechanical Engineering Department
dinis@ubi.pt, aes@ubi.pt, bruno@ubi.pt

>> Contents

Contents

LAB4.4: USB INTERFACE WITH MATLAB

❑ Introduction to the MSP430 USB API stack

- The Application Programming Interface (API) USB stack for the MSP430 allows the rapid development of a USB connection between the MSP430 and a host.

- This API supports three classes of USB devices:
  - Communication Device Class (CDC);
  - Human Interface Device Class (HID);
  - Mass Storage Class (MSC).

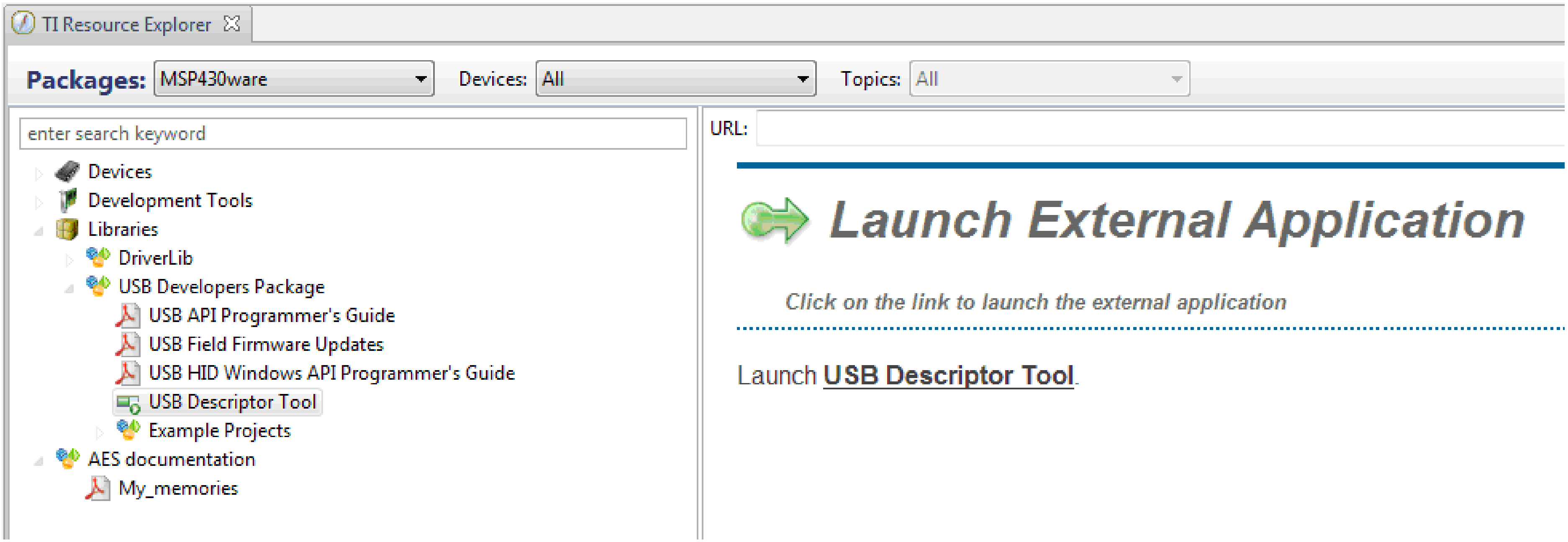
- All the communication protocol is handled automatically by the API.

- The interface between the user's application and the API is very simple.

LAB4.4: USB INTERFACE WITH MATLAB

❑ MSP430 USB Descriptor Tool

- The application can be launched via the MSP430ware:



LAB4.4: USB INTERFACE WITH MATLAB

- **Example #1: Lab4.4a - cdcSendDataWaitTilDone (BYTE* databuf, WORD size, BYTE intfNum, U LONG ultimeout)**

  - This function manages the transmission of data located in dataBuf, of dimension size, via the USB interface called intfNum.

  - The function returns only after submitting all data.

  - The 32-bit value ultimout defines the number of times that the function USBCDC_intfStatus() is executed, in order to determine if the data transmission is complete.

  - If this parameter is zero, it will not use the timeout functionality.

- **Example #1: Lab4.4a - cdcSendDataWaitTilDone (BYTE* databuf, WORD size, BYTE intfNum, U LONG ultimeout)**

  - The function returns the following parameters.

Returns	*0:* the call succeeded; all data has been sent *1:* the call timed out, either because the host is unavailable or a COM port with an active application on the host wasn't opened. 2: the bus is unavailable.

  - Two consecutive send operations can be performed using the function cdcSendDataWaitTilDone().

  - The result of the transmission is tested in case of problems in sending the packet.

  - In this case, the transmission procedure is canceled with the function USBCDC_abortSend() and the application terminates the current context.

LAB4.4: USB INTERFACE WITH MATLAB

- **Example #1: Lab4.4a - cdcSendDataWaitTilDone (BYTE* databuf, WORD size, BYTE intfNum, U LONG ultimeout)**

  - The example of application of cdcSendDataWaitTilDone() is used in project Lab4.4a.

  - The application send data by it generated.

  - The application in MatLab makes the graphical presentation of data received through the USB port.

LAB4.4: USB INTERFACE WITH MATLAB

- **Example #2: Lab4.4b - cdcSendDataInBackgound (BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout)**

  - This function manages the sending of data located in dataBuf of dimension size, via the USB interface called intfNum.

  - The execution of the function returns before the process is complete.

  - The function returns the following parameters.

Returns	*0:* the call succeeded; all data has been sent *1:* the call timed out, either because the host is unavailable or a COM port with an active application on the host wasn't opened. 2: the bus is unavailable.

- **Example #2: Lab4.4b - cdcSendDataInBackgound (BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout)**

  - This function manages the sending of data located in dataBuf of dimension size, via the USB interface called intfNum.

  - The execution of the function returns before the process is complete.

  - The function returns the following parameters.

Returns	*0:* the call succeeded; all data has been sent *1:* the call timed out, either because the host is unavailable or a COM port with an active application on the host wasn't opened. 2: the bus is unavailable.

- **Example #2: Lab4.4b - cdcSendDataInBackgound (BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout)**

  - Within the main loop, two consecutive transmissions are performed using the function cdcSendDataInBackgound() using two different buffers:
    - dataBuffer1,
    - dataBuffer2.

  - The result of the transmission is tested in case of problems in sending the packet.

  - The transmission procedure is canceled with function USBCDC_abortSend() and the application terminates the current context.

  - The example of application of cdcSendDataInBackgound() is used in the project Lab4.4b.

LAB4.4: USB INTERFACE WITH MATLAB

- **Example #2: Lab4.4b - cdcSendDataInBackgound (BYTE* dataBuf, WORD size, BYTE intfNum, ULONG ulTimeout)**

```
while(1)
{
    switch(USB_connectionState())
    {
       case ST_ENUM_ACTIVE:
          {
             .
             .
             .
             dataBufferX = temporaryBuffer1;
             if(cdcsendDataInBackground(dataBufferX,100,1,0)
             {
               USBCDC_abortSend(&x,1);
               break;
             }
             .
             .
             dataBufferY = temporaryBuffer2;
             if(cdcSendDataInBackground(dataBufferY,100,1,0)
             {
               USBCDC_abortSend(&x,1);
               break;
             }
             .
             .
             .
          }
    }
}
```

Second operation may be open; don't edit dataBufferY

First operation may be open; don't edit dataBufferX

LAB4.4: USB INTERFACE WITH MATLAB

Data reception operations

- The data reception operation through a CDC interface has different implications inherent to the sending process.

- In the transmission process, the speed is set by the availability of the bus and the host.

- In a receiving data process, the application does not know:
  - How much data will be sent;
  - The time in which they will be sent.

- The API allows the implementation of some techniques that adapt to the data reception in different conditions to cope with these requirements.

❑ Data reception operations

- Techniques adapted to different data reception types:

	Size is Known?	Expected Size	Description
cdcReceiveDataInBuffer() **hidReceiveDataInBuffer()**	No	Small	Receive operations aren't opened until data is already waiting in the USB endpoint buffers. Therefore, the operations are immediately completed. The application must always be available to respond to *handleDataReceived()*.
Continuously-Open Receive	No	Large	The application maintains an open, larger-than-needed receive operation at all times. When it wants to use the data, it simply looks inside the user buffer.
Fixed Receive	Yes	n/a	A receive operation is opened for the exact amount of data expected. When this amount is received, a *handleReceiveCompleted()* event is generated, and the application handles it.

- **Example #3: Lab4.4c - cdcReceiveDataInBuffer (BYTE* databuf, WORD size, BYTE intfNum)**

  - This function opens a data receive operation for the interface intfNum.
  - The data are placed in dataBuff. Once the number of bytes size received, the function ends.
  - This function is used when the event USBCDC_handleDataReceived() informs that there is data waiting to be collected in the USB buffer.
  - The function returns the following parameters.

Returns	The number of bytes received into *dataBuf*

- **Example #3: Lab4.4c - cdcReceiveDataInBuffer (BYTE* databuf, WORD size, BYTE intfNum)**

  - The application example of cdcReceiveDataInBuffer() is used in project Lab4.4.
  - The user must open a prompt line in Matlab to link with the USB device.
  - Then, different values are sent to the USB device using the function fwrite().
  - The user can test the transmission of data packets with different sizes.

```
>> fwrite (s,['test'],'async');
>> fwrite (s,[1 2 3 4 5 6],'async');
```

- **Example #4: Lab4.4d – Continuously-open Receive Operation**

  - A listening operation of the communication channel is kept open constantly in this example.

  - A single function call USBCDC_receiveData() places the data values that are received in a buffer larger than the size of the data frame.

  - All incoming data values are automatically sent to the user buffer.

  - When ready, the application can process the received data.

  - Following this operation, the application can restart the receiving data process.

- **Example #4: Lab4.4d – Continuously-open Receive Operation**

  - Within the main loop, the USB interface state is checked:
    - If the state is ST_ENUM_ACTIVE, the application checks for an active reception operation.
    - If this condition is FALSE, it opens a receive data operation, stating that data should be placed in the user buffer MEGA_Size.

  - Then, the application checks the number of data sent (bytesTX) and received (bytesRX) by the USB interface.

  - The received data processing is initiated if the number of bytes received is greater than the limit threshold.

  - The receive operation is inhibited by the function abortReceive() and the data in the reception buffer RXBuffer are processed by the function process_the_data().

- **Example #4: Lab4.4d – Continuously-open Receive Operation**

  - The application example using the Continuously-open technique is used in project Lab4.4d.
  - The user must open a connection with the USB device through the prompt line in Matlab.
  - Then, different data packets are sent to the USB device using function fwrite().
  - The user can try sending data packets generated by the function rand() with 10k elements size.

```
package = rand(1,10000);
fwrite (s,package,'async');
```

❑ Example #5: Lab4.4e – Fixed-size Receive Operation

- If the size of the data packet to receive is known, then a specific data reception operation can be opened.
- Once all the data values that are part of the package have been received, an event is generated that is serviced by the function USBCDC_handleReceiveCompleted().
- This performs the necessary package processing.
- After making the necessary settings, the application enables the event kUSB_receiveCompletedEvent.

❑ Example #5: Lab4.4e – Fixed-size Receive Operation

- Within the main cycle, interrupts are disabled and a data reception operation is open into command buffer, composed of 34 bytes if the USB interface state is ST_ENUM_ACTIVE:
  - The device is placed in LPM0 with interrupts enabled if:
    - The receive operation starts (ret == kUSBCDC_receiveStarted) or,
    - There was an error at the starting of operation (ret == kUSBCDC_intfBusyError).

- The event handler kUSB_receiveCompletedEvent enables the reception flag bReceiveCompleted_event and returns the value TRUE to activate the device.

- With the CPU active, if the number of bytes that make up the command has been received, this value is processed by the function execute_the_command().

LAB4.4: USB INTERFACE WITH MATLAB

❑ Example #5: Lab4.4e – Fixed-size Receive Operation

- A new receive operation is opened if the USB interface is available.
- The receive operation is aborted if the bus is not available.
- The application example of the Continuously-open technique is used in project Lab4.4e.
- The user must open a connection with the USB device through the prompt line of MatLab.
- Then, different data packets are sent to the USB device using the function fwrite().
- The user can test the transmission of data packets with 34 elements size.

Chapter 5
eZ430-Chronos Development Kit
LAB5.3: Pressure Sensor/Altitude measurement

Texas Instruments Incorporated
University of Beira Interior (PT)

Pedro Dinis Gaspar, António Espírito Santo, Bruno Ribeiro
University of Beira Interior, Electromechanical Engineering Department
dinis@ubi.pt, aes@ubi.pt, bruno@ubi.pt

>> Contents

Contents

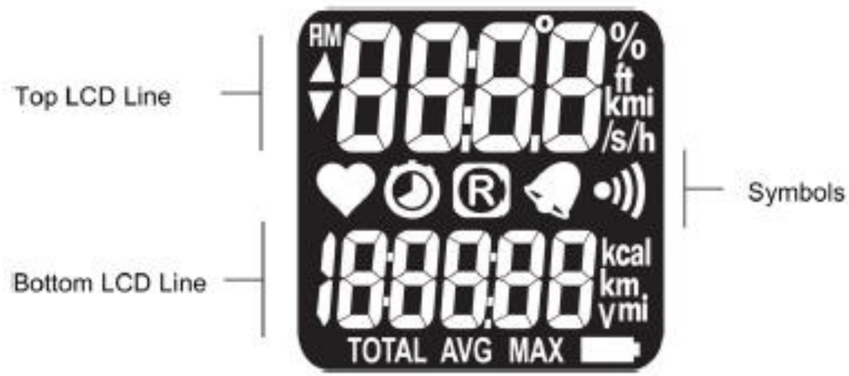
eZ430-Chronos Hardware Resources

UBI

❑ eZ430-Chronos watch

- The core of the eZ430-Chronos Watch is the CC430F6137 MCU, which includes a radio type CC1110 running on the sub-1-GHz band.

A)



B)

- The microcontroller controls the LCD and the temperature sensor.

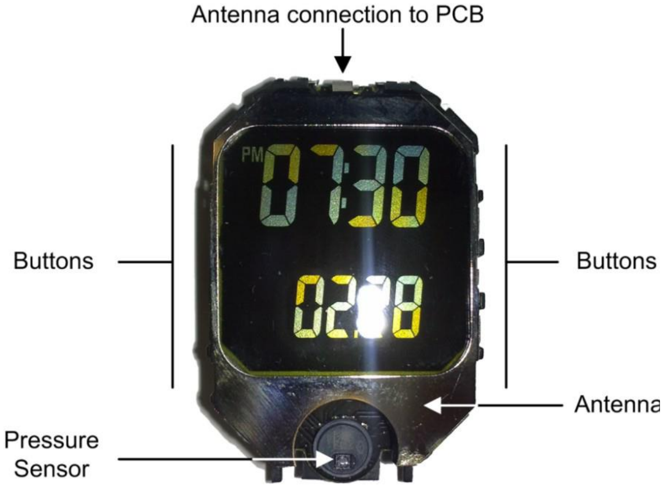
- The measurement of acceleration and pressure are the responsibility of specific integrated circuits.

eZ430-Chronos Hardware Resources

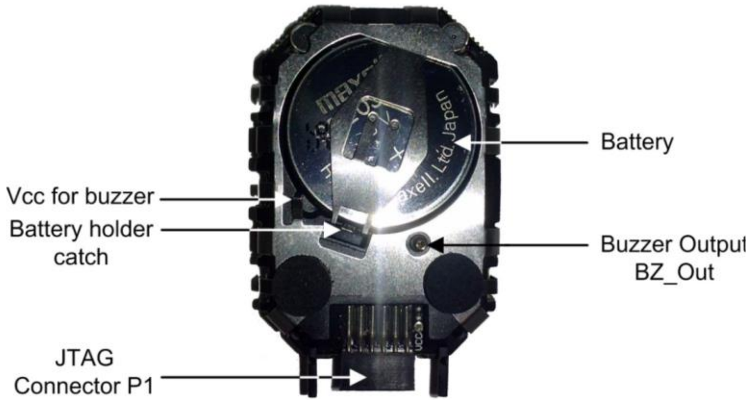
eZ430-Chronos watch

- The eZ430-Chronos Watch is a commercial product. To use the Chronos watch as a development tool it is necessary to disassemble it.

- The tools, accessories and instructions are available as part of the kit.



A)

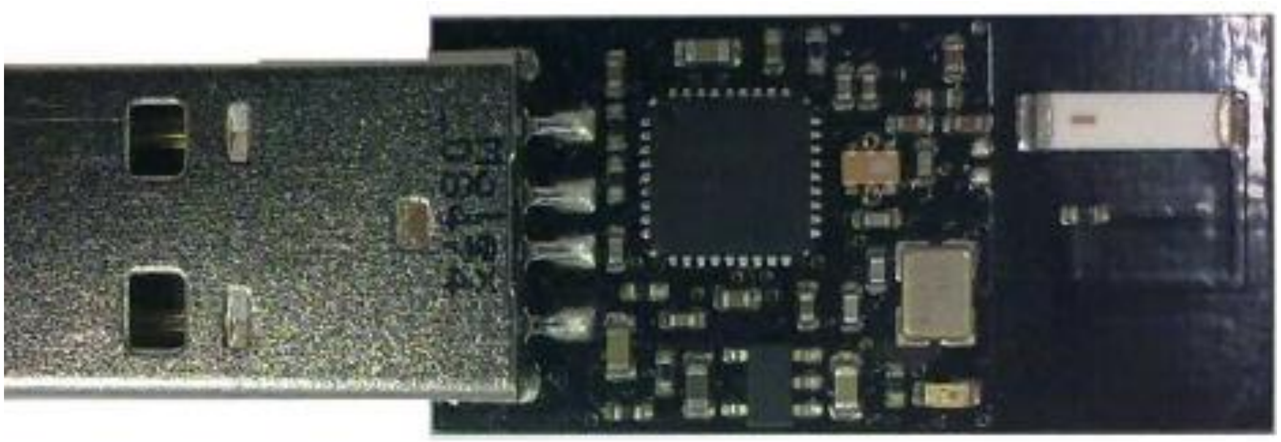


B)

eZ430-Chronos RF Access Point

- The eZ430-Chronos RF Access Point allows communication between the PC and eZ430-Chronos Watch for data transfer, information synchronization and computer applications control that run on the PC.

- This interface is based on CC1111F32 (Sub-1GHz System-on-Chip with MCU).

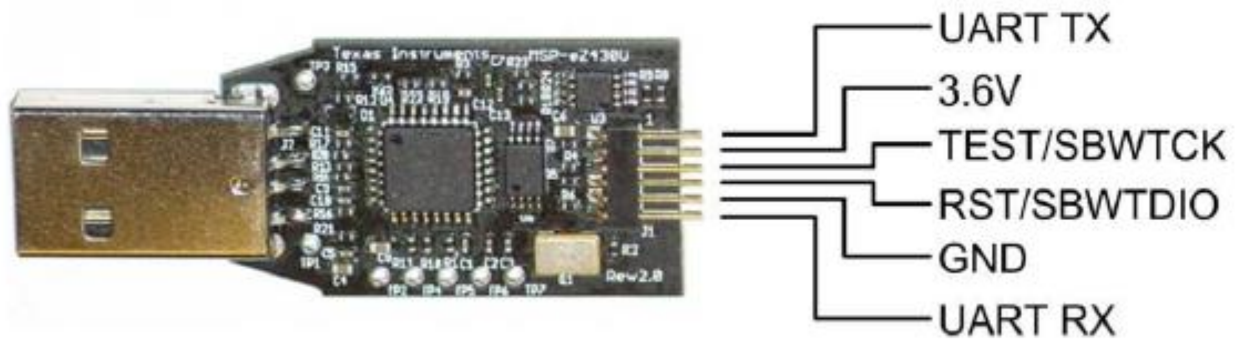


eZ430-Chronos Hardware Resources

eZ430-Chronos Debug Interface

- The eZ430-Chronos kit provides a debugging and programming interface.

- The programming interface includes a "back channel MSP430 application UART" that can be used simultaneously for debugging operations.



eZ430-Chronos Hardware Resources

eZ430-Chronos sensors

- The eZ430-Chronos kit has two sensors:
  - VTI SCP1000:
    - An absolute pressure sensor, which can detect atmospheric pressure from 30 to 120 kPa.
    - The pressure data is internally calibrated and temperature compensated.
  - VTI CMA3000:
    - 3-axis accelerometer.
    - For more information see the documents included in the kit, or consult the manufacturer's respective web pages.

LAB5.3: Pressure Sensor/Altitude measurement

Abstract

- This laboratory uses the absolute pressure sensor from VTI Technologies - reference SCP1000, incorporated into the eZ430-Chronos watch.
- The software package that comes in the eZ430-Chronos kit has the driver and application module for this sensor.
- Thus, the code is rearranged to accomplish the laboratory objectives.
- The graphical representation on the PC is done through an application developed in Processing.

LAB5.3: Pressure Sensor/Altitude measurement

❑ Solution Proposal and Analysis

- The remote application measures the absolute pressure and uses line 2 of the eZ430-Chronos display/menu.

- When the application starts, it shows the display/menu the “date” function on line 1.

- It is necessary to press the (#) button to enter in the “remote altitude” function.

- The connection with Access Point and the data transmission are initiated after pressing the "DOWN" button.

LAB5.3: Pressure Sensor/Altitude measurement

❑ Experiment Instructions

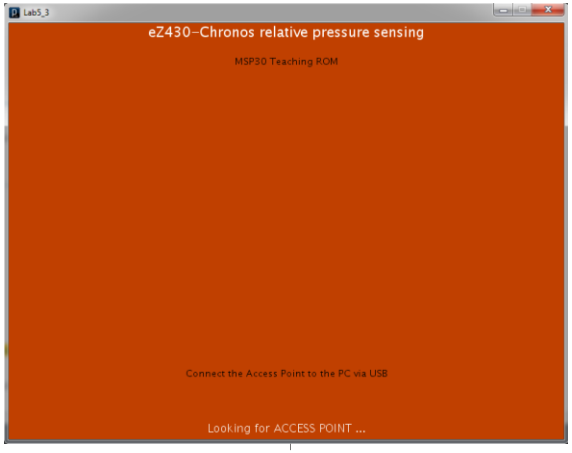
- 1 Import the CCS project named lab53
- 2 Change the SimpliciTI end device ID
  - The new ID is on the table
  - Select the main file named “main_MSP430_ROM.c”
  - Go to line 716 and change the ID
- 3 Rebuild the project according to the correct region
- 4 Download the project in to the chronos watch

LAB5.3: Pressure Sensor/Altitude measurement

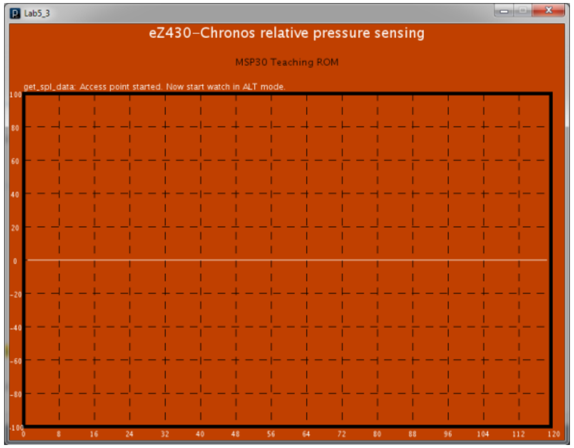
❑ Experiment Instructions

- 5 Start the Processing utility on the PC (project: lab5_3)
- 6 Select the measurement function by pressing the button # on the chronos
- **Caution – Every attendant must only do the following steps one by one and when requested**
- 7 Press the key "s" to start the access point
- 8 Press the "Down" button to start simpliciTI application.
- Note: The Zero can be adjusted by pressing the "c" key

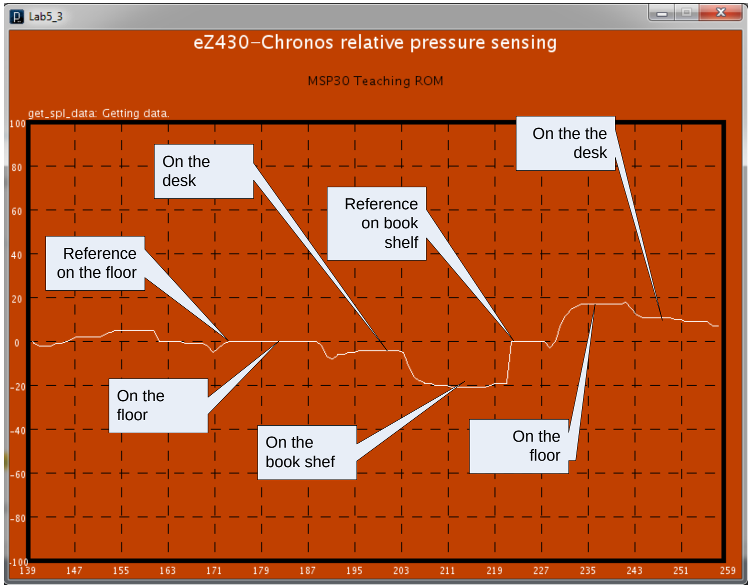
❑ Interface with the Processing application



a)



b)



c)