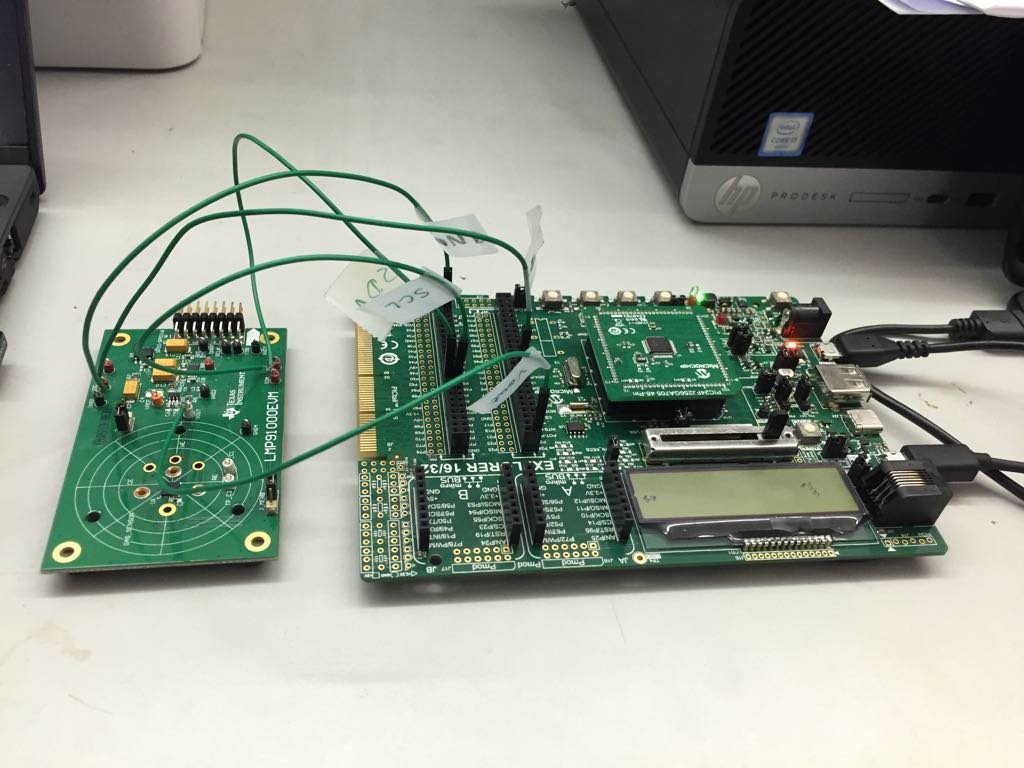
**Sensors Interface for IoT**

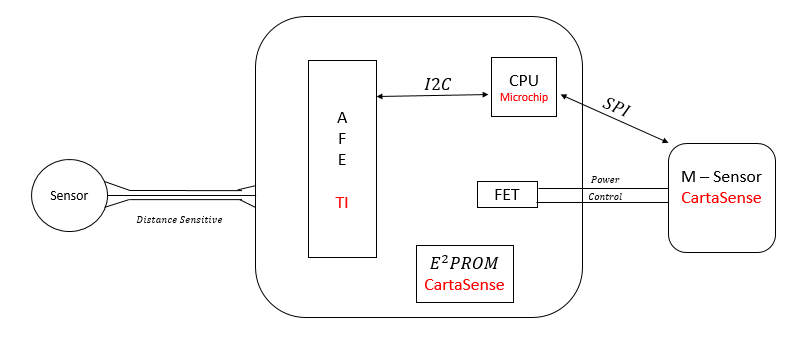
**Project number 17-1-1-1438**

**User’s Guide**

*By*



**Hardware**



The system consists of 3 main components -

1. The Sensor –

Has 3 connections:

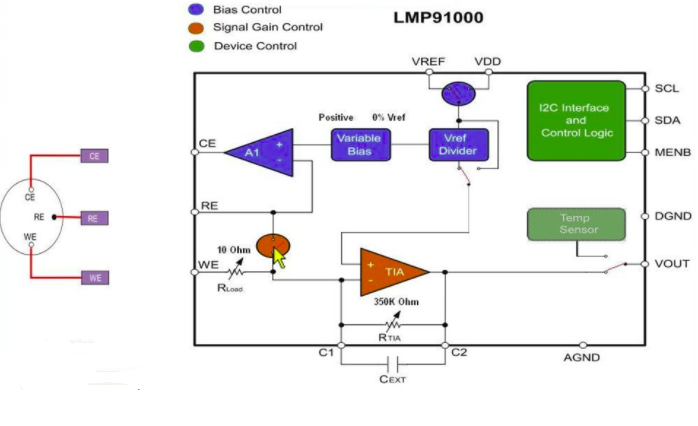
* CE (counter electrode)
* WE (working electrode)
* RE (reference electrode)

1. The AFE –

* We used the Texas Instruments’ LMP91000EVM board as the Analog Front End in the system.
* In operation, current is driven into the CE connection and the AFE’s circuitry monitors the voltage at the RE point. A closed-loop control circuit keeps this voltage constant, which in turn changes the return current present at the WE. The resulting return current at the WE connection can be converted to a voltage via a transimpedance amplifier (TIA).
* The AFE is configured through messages from the CPU. The formula that describes the dependency between the A1 amplifier’s positive input and the is –

The configuration we used –

* + TIA (Transimpedance amplifier) –
  + REF – the Variable bias is configured to be 6% of the (positive) and the is 20% of the .
  + MODE – configured so the AFE works with 3 inputs



1. The CPU –

* We used the CPU from , and the development was done over the .
* We programmed the CPU with IDE called “MPLAB X IDE”, using the MCC library, and compiling over the MPLAB XC Compiler.

**Board Assembling**

The jumper’s connecting between the and the –

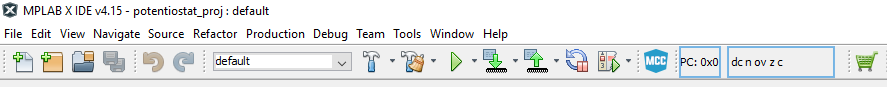
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

After connecting the jumpers, we need to connect the computer to the input of the via a USB cable in order to load the code to the board and supply power from the computer. Moreover, we need to connect the computer to the serial input, also via a USB cable, in order to receive the data samples from the board in the computer’s terminal (more about that part in the last section).

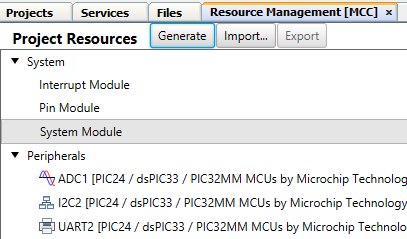
**The code**

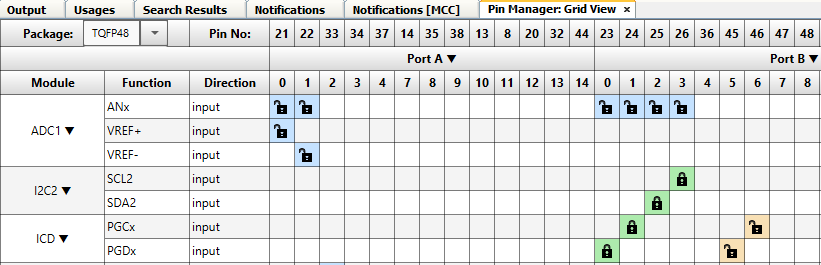
1. MCC –

The project was developed over Microchip’s IDE, the MPLAB X IDE. In order to use peripherals, we used the built in MCC –

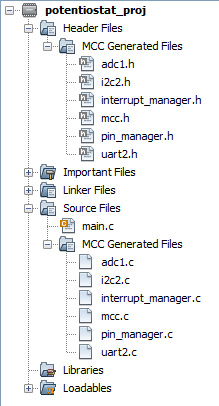


In this section we can choose a peripheral, configure the correct pins of the CPU and generate an API (.c and .h files):





After generating the code, the files can be found under the “MCC\_Generated\_Files” folder:

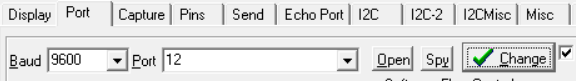
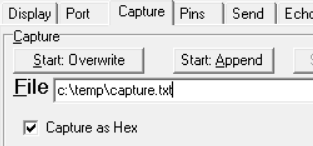


The mcc.c file holds all the pin configurations of the project. When adding a peripheral it’s important to check it doesn’t override the configuration in the mcc.c fiie.

1. The code flow –
2. The Terminal –

We are using **RealTerm** (<https://realterm.sourceforge.io/>) to listen to the UART port.

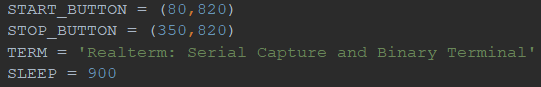
It is necessary to configure RealTerm as following:

* Display: Display as Hex(space).  
  
* Port: Set Baud to 9600, and Port to relevant USB port (can be found via **Device Manager** – press Win+R 🡪 type “devmgmt.msc” and click OK) and click “Change”.  
  
* Capture: Choose your output path and check the box “Capture as Hex”.  
  

Once configured correctly, you should be able to see the incoming UART data on the screen. We are now ready to run the Python script.

1. Python script –

Open **realterm.py** and examine the following parameters:

* SLEEP: Number of seconds to run the experiments (we used 15 minutes)
* TERM: The title of the RealTerm window (varies from PC to PC)
* START\_BUTTON, STOP\_BUTTON: The coordinates of the two buttons. If you are using RealTerm in fullscreen (recommended) do not touch those!  
  

You can now run the file.

1. Plotting the samples (MATLAB) –

The plotting script requires MATLAB with Signal Processing Toolbox, hence using TAU’s remoteapp version is recommended.

We have generated three scripts – **adc\_reads.m** (plot data from board), **read\_mpt.m** (plot data from the biological machine), **compare.m** (to compare the two plots automatically).

* adc\_reads.m  
  You should only touch *filename* (path to .txt file generated by RealTerm) and *exp\_time* (duration of experiment in seconds).  
    
  **NOTICE:** Since the RealTerm capture is not necessarily synchronized with the measures (i.e. we might start capturing in the middle of a transmission since one measure consists of 4 bytes), we need to go to *capture.txt* and trim the file’s first and last few bytes. Note that every 4 bytes are split by the byte ‘20’, therefore ’20’ should be the last byte in the file, and the fifth from the start. Not trimming the file would raise an exception in MATLAB!
* read\_mpt.m  
  You should only touch *fname* (path to .txt/.mpt file generated by the biological machine).  
  