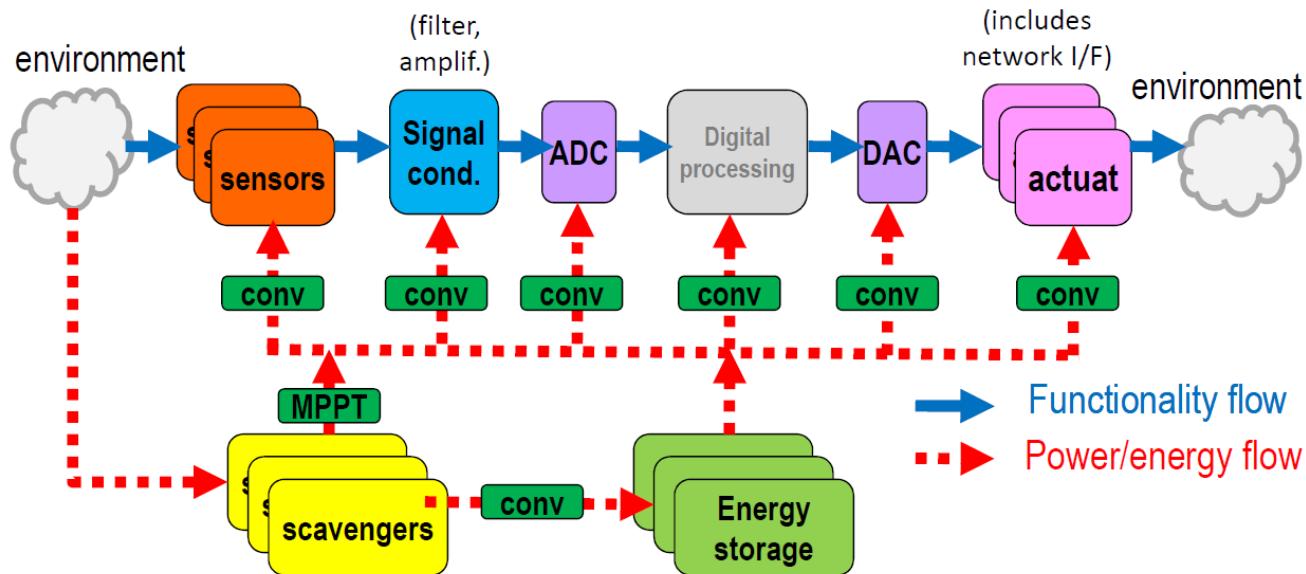


# **Lab 3**

## **Energy storage, generation and conversion**

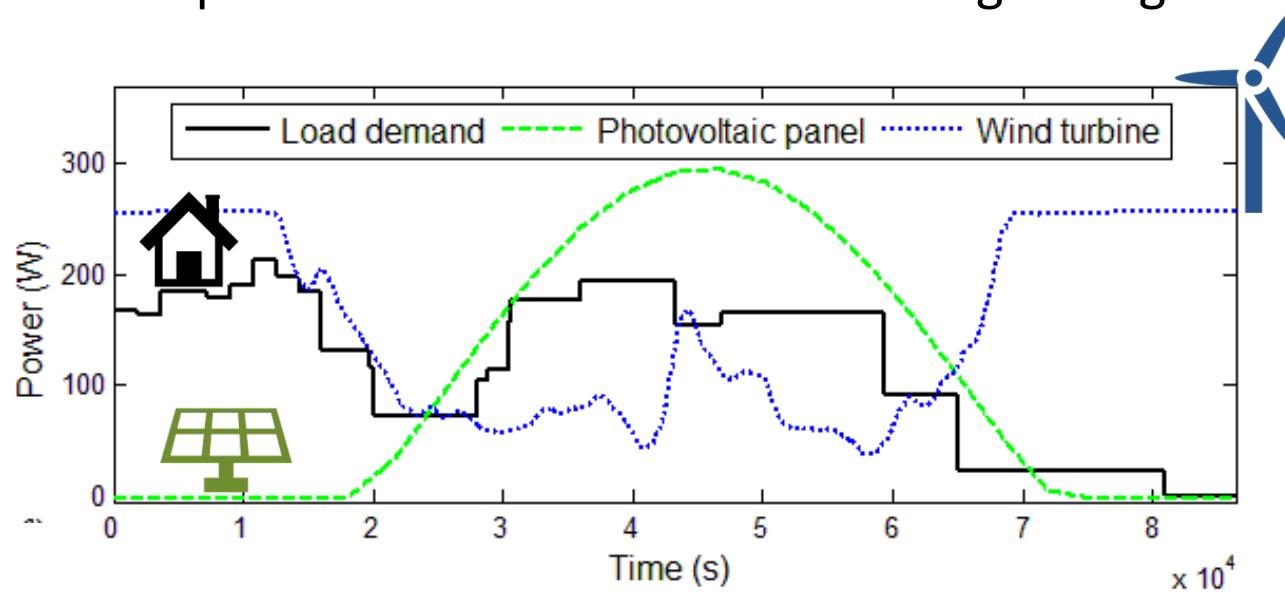
# Energy storage, generation and conversion

- Focus: **power perspective of the system**
  - Energy storage: appropriate size to sustain loads?
  - Generation: how much provided by power sources?
  - Any loss due to conversions?



# Energy storage, generation and conversion

- Crucial to model and simulate the overall system to validate and estimate the behavior of single components and of the overall system beforehand
  - On any scale of system!
  - Even more crucial when autonomousness must be guaranteed
    - Ensure operation of the IoT device for long enough...

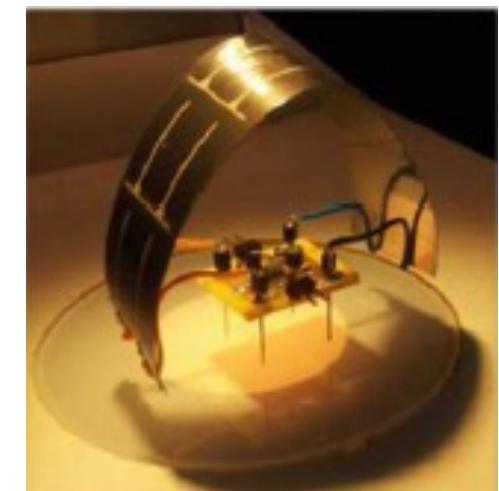


# Energy storage, generation and conversion

- Crucial to model and simulate the overall system to validate and estimate the behavior of single components and of the overall system beforehand
  - On any scale of system!
  - Even more crucial when autonomousness must be guaranteed
    - Ensure operation of the IoT device for long enough...
  - Need to create models for the components
    - And to launch simulations to trace quantities

# Objective and organization

- Goal of this lab is to simulate an IoT device made of:
  - 4 sensors
  - Memory and control unit (MCU)
  - A module to transmit data over ZigBee (RF Radio)
  - A battery with a DC-DC converter
  - A thin-film photovoltaic module operated at the MPP with a DC-DC converter



# Objective and organization

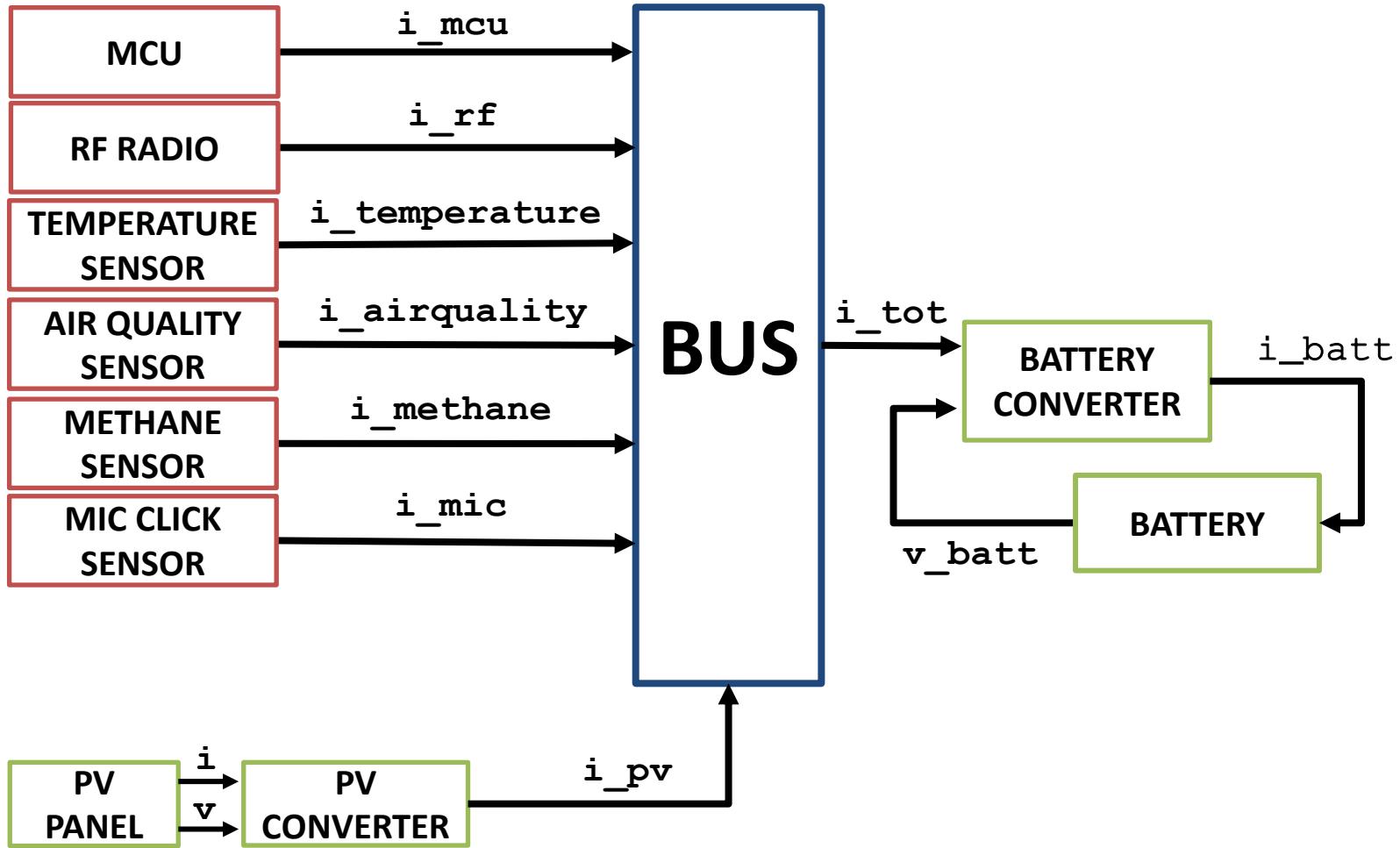
- Goal of this lab is to simulate an IoT device:
  - Implemented in **SystemC/SystemC-AMS**
    - Standard-de-facto for energy simulation
    - Predefined simulation skeleton
    - To be populated with models for the components
  - Populate the system, analyse and optimize its behavior

# Objective and organization

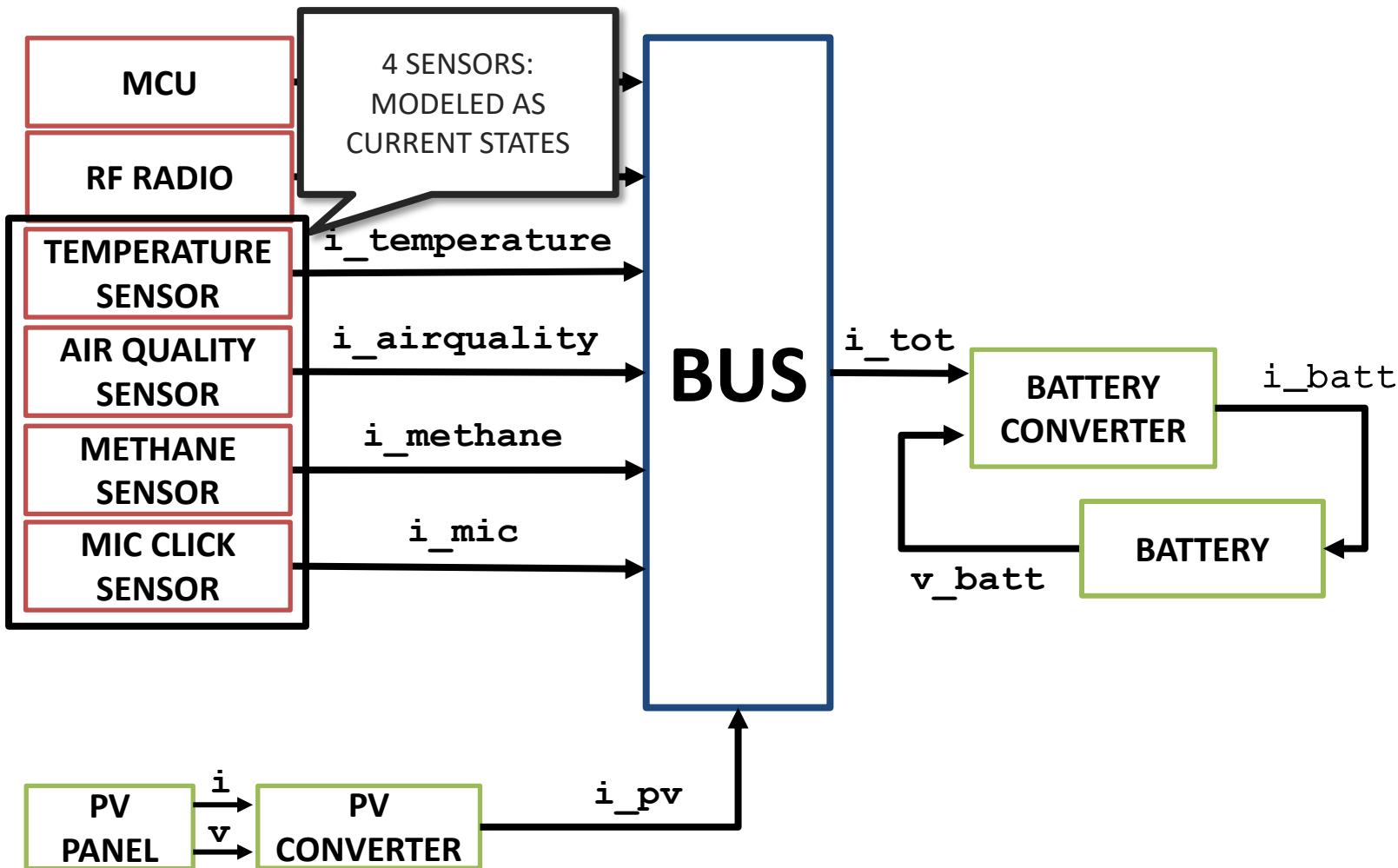
- Organization
  - 1 assignment to deliver
  - 3 days
    - *Jan 09th*
    - *Jan 15th*
    - *Jan 16th*
  - Required software: C++ Compiler, SystemC, SystemC-AMS
  - Additional useful software: Python Interpreter, Matlab

# **SystemC Simulator**

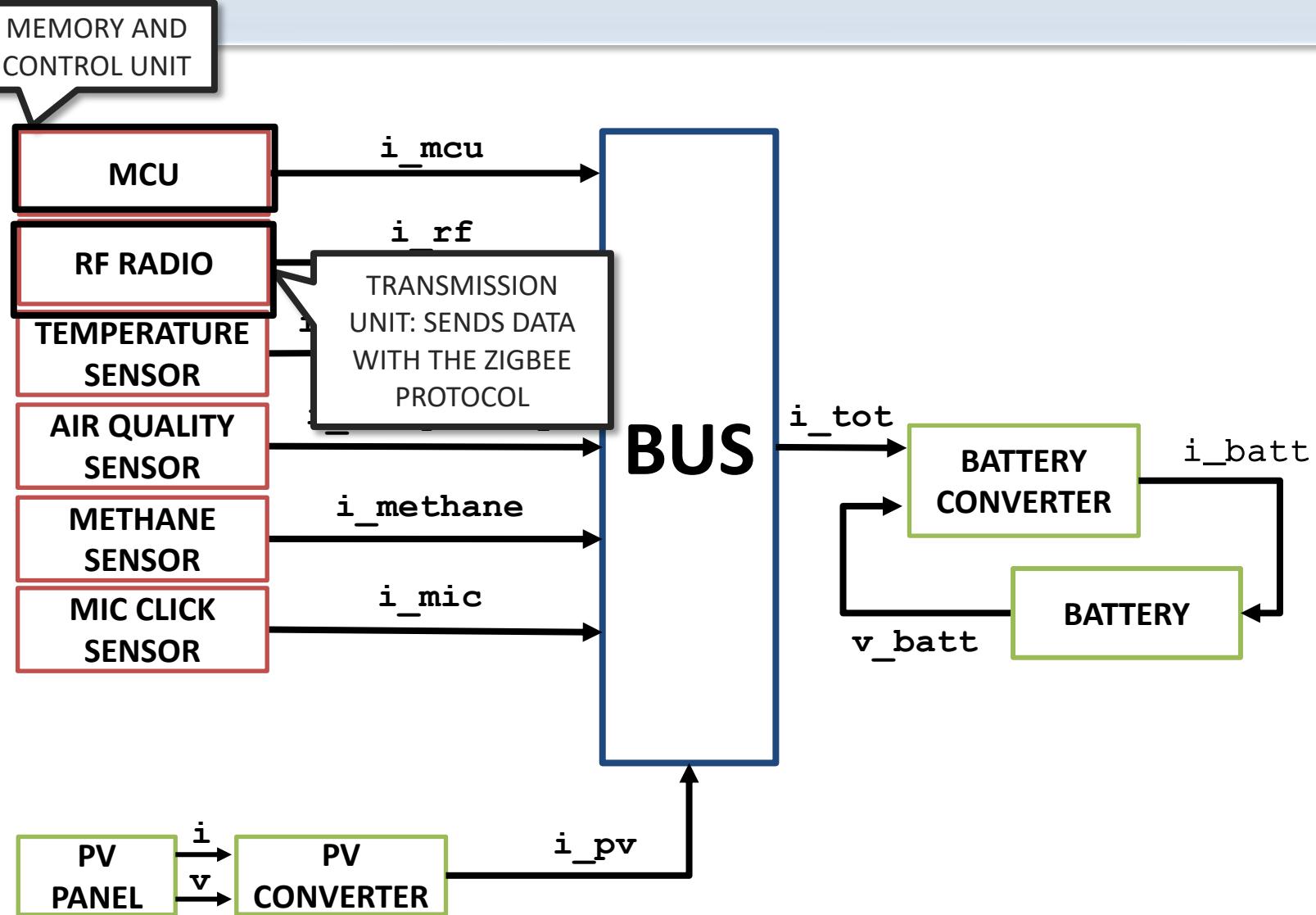
# Simulator Overview



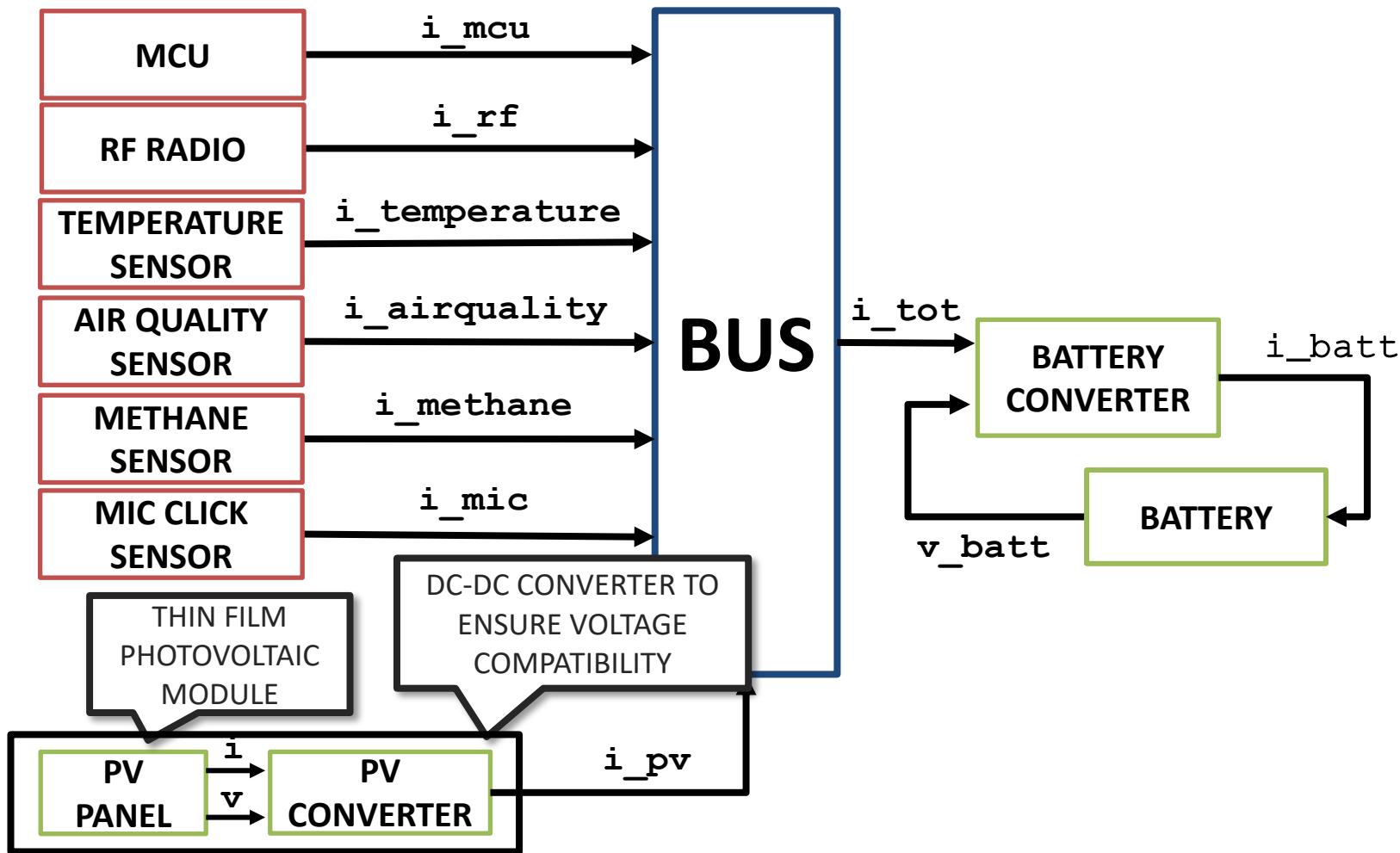
# Simulator Overview



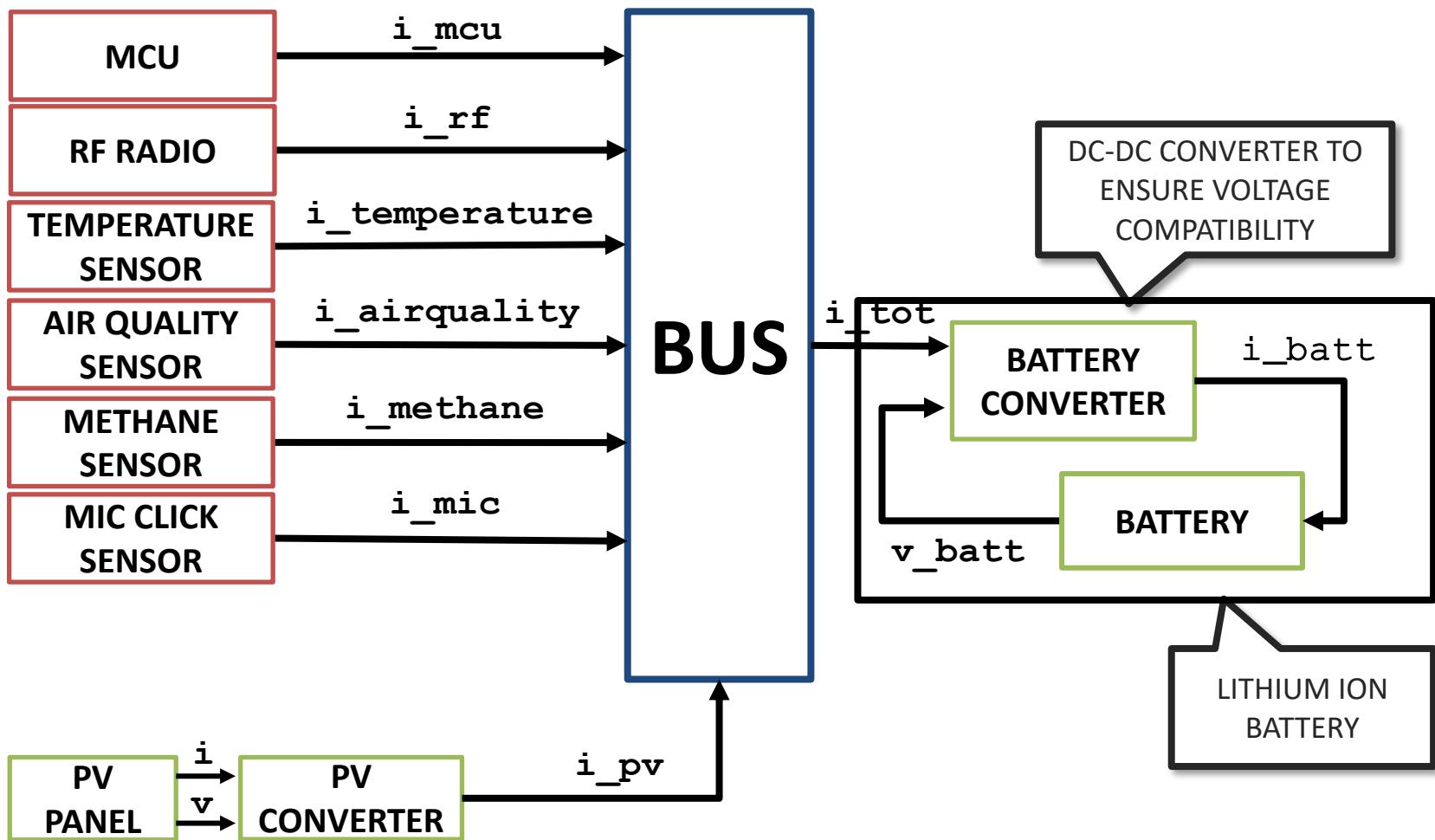
# Simulator Overview



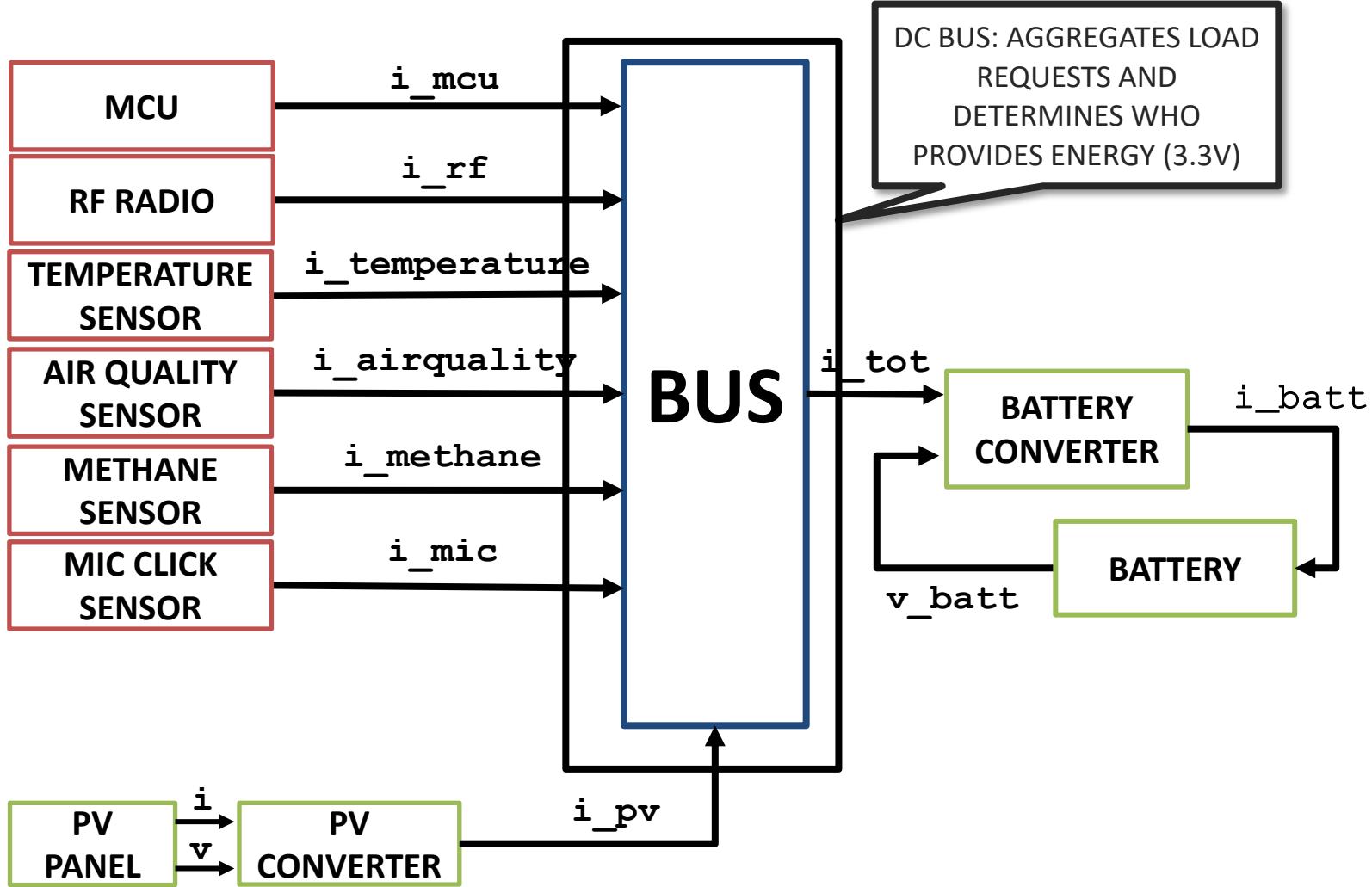
# Simulator Overview



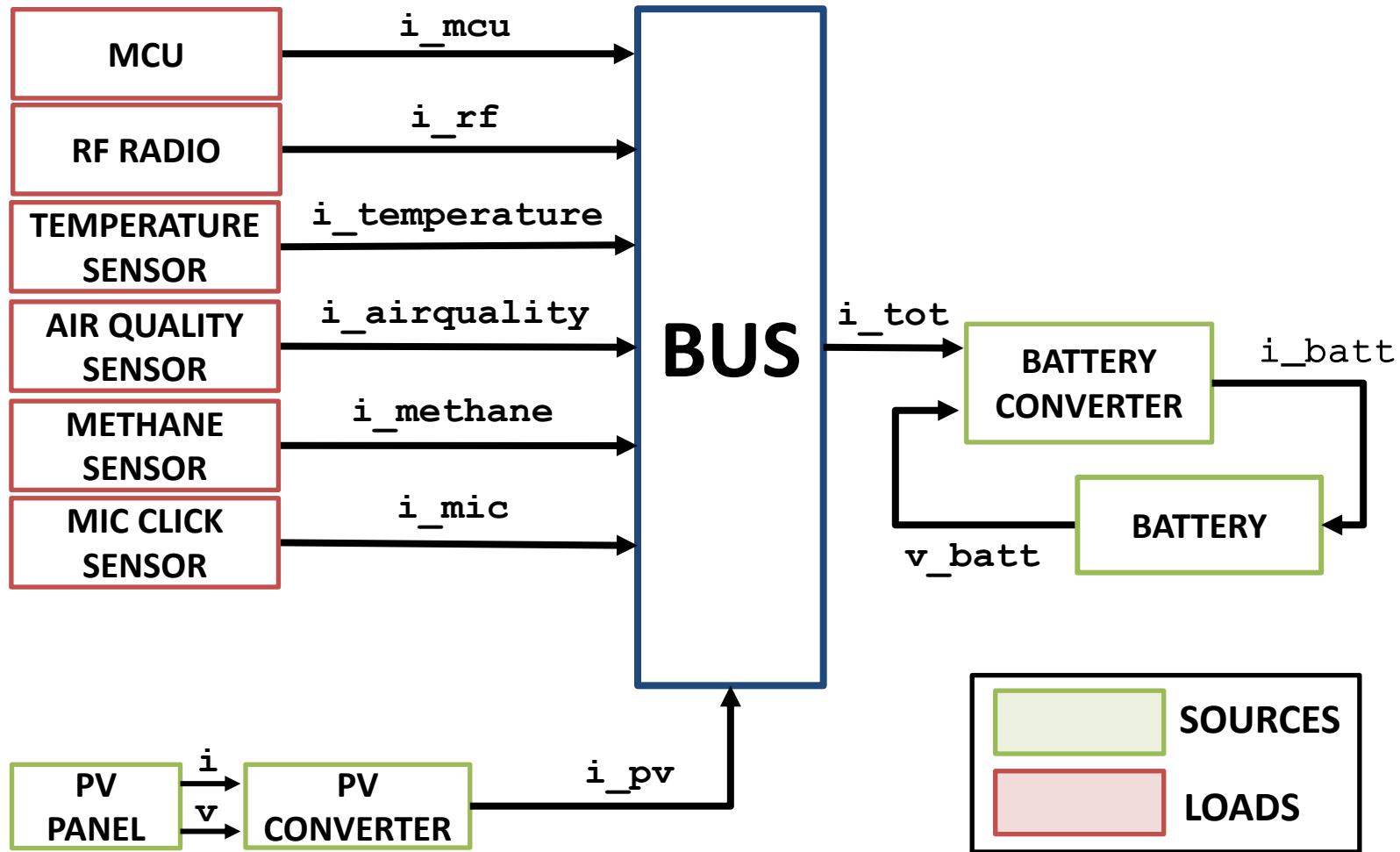
# Simulator Overview



# Simulator Overview



# Simulator Overview

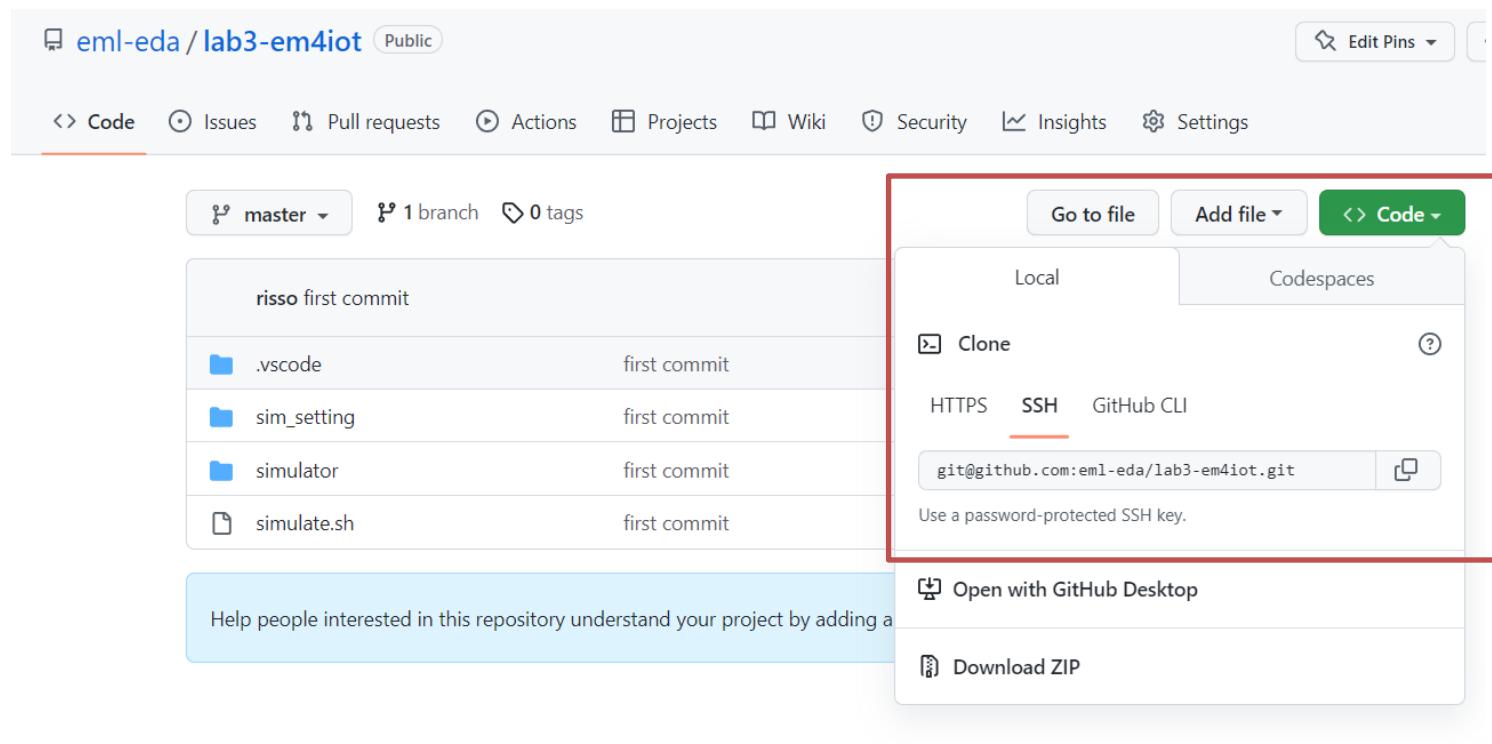


# **Lab 3 – Part 0**

## **Simulator Setup**

# Download Simulator

- Clone the github repository at:  
<https://github.com/eml-eda/lab3-em4iot>



# Install SystemC

- Requirements: g++, make
- Download the `systemc-2.3.3.tar.gz` from course website.
- Expand archive.
- Follow instructions contained in the INSTALL file.
- Export **SYSTEMC\_HOME** env variable pointing to your SystemC directory:
  - `$ export SYSTEMC_HOME="path"`
  - Suggest to put this command in your `.bashrc`

# Install SystemC-AMS

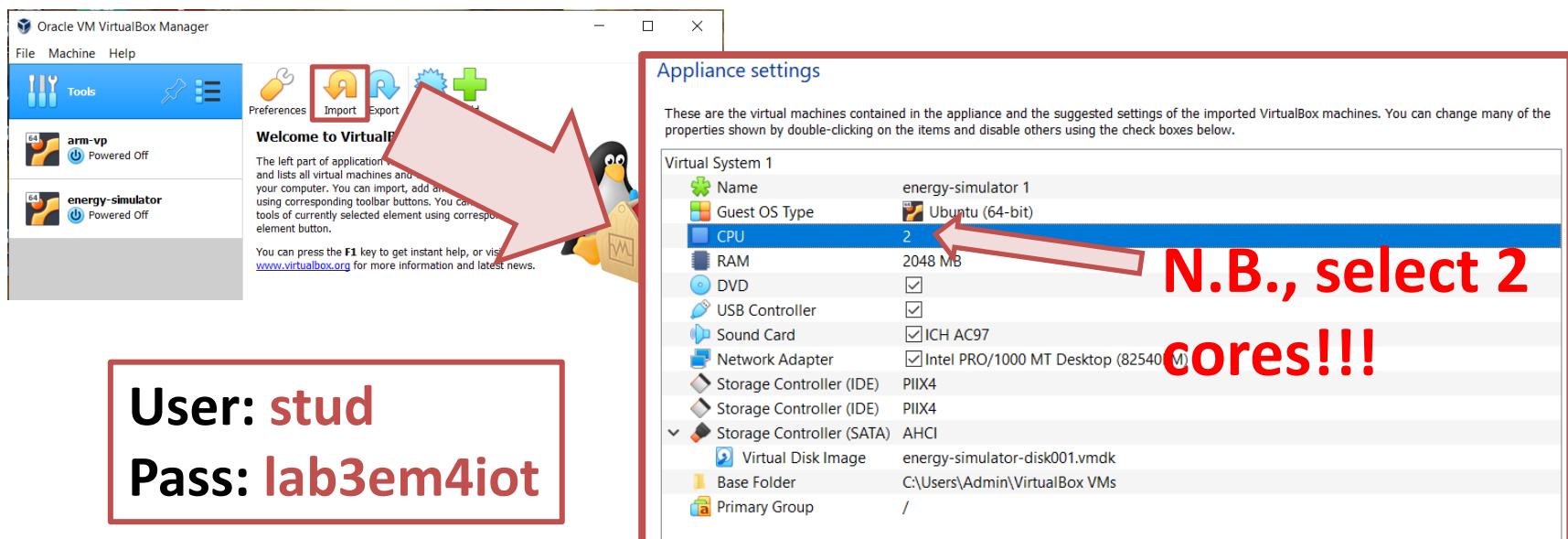
- Requirements: g++, make
- Download the `systemc-ams-2.3.tar.gz` from course website.
- Expand archive.
- Follow instructions contained in the `INSTALL` file.
- Export `AMS_HOME` env variable pointing to your SystemC-AMS directory:
  - `$ export AMS_HOME="path"`
  - Suggest to put this command in your `.bashrc`

# Python

- Requirements: Python3.8>

# Alternative: Virtual Machine

- In alternative, you can use the provided Virtual Machine image `energy-simulator.ova`.
    - N.B., **MUST BE USED WITH VIRTUALBOX**



- How to setup shared folder in virtualbox: <https://unix.stackexchange.com/a/156469>
  - How to change keyboard language: `sudo dpkg-reconfigure keyboard-configuration`
  - Remember to shutdown the vm using: `shutdown -h now`

# SystemC-AMS Primer

- User Guide: [LINK](#).
- Each module is described by two files:

## .h file

```
1 #include <systemc-ams.h>
2
3 #include "config.h"
4
5
6 SCA_TDF_MODULE(methane_sensor)
7 {
8     sca_tdf::sca_out<double> i; // Consumed current
9
10 SCA_CTOR(methane_sensor): i("i"), cnt(0) {}
11
12 void set_attributes();
13 void initialize();
14 void processing();
15
16 private:
17     int cnt;
18 }
```

Include library

Define module as a child class of  
**SCA\_TDF\_MODULE**

Interface: input and output ports (somehow similar to HDL)

Module's methods,  
always to be declared

Constructor: give names to ports and initializes other class private internal variables

Other internal variables

# SystemC-AMS Primer (cont'd)

- User Guide: [LINK](#).
- Each module is described by two files:

## .cpp file

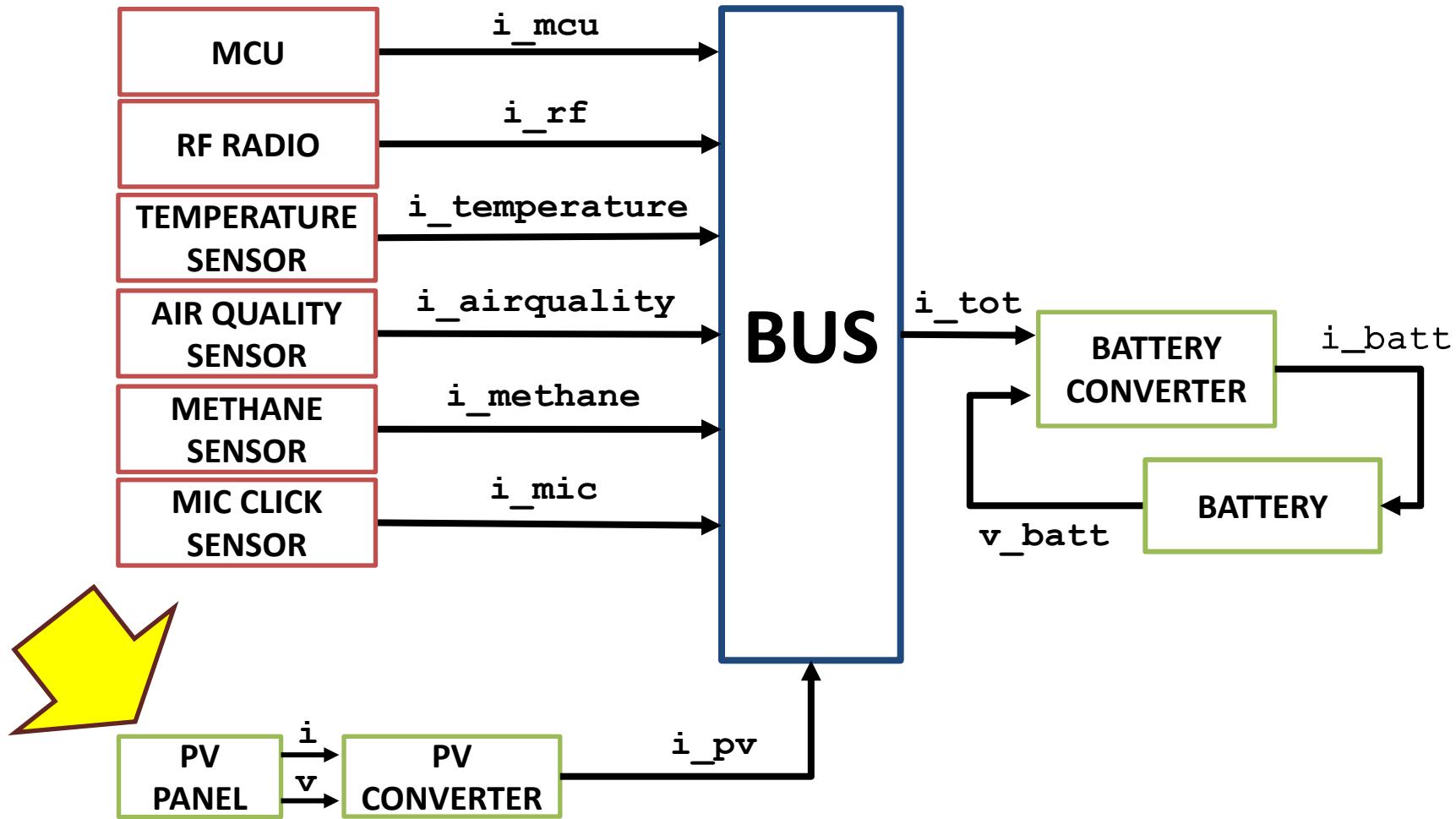
```
1 #include "methane_sensor.h"           ← Include .h file
2
3
4 void methane_sensor::set_attributes() ← Define set_attributes
5 {                                     method.
6     i.set_timestep(SIM_STEP, sc_core::SC_SEC);
7 }
8
9 void methane_sensor::initialize() {}   ← Define initialize method.
10
11 void methane_sensor::processing() {}   ← Define processing method.
12 {
13     if(cnt >= METHANE_SENSOR_T_ACT &&
14         cnt < METHANE_SENSOR_T_ACT + METHANE_SENSOR_T_ON)
15     {
16         i.write(METHANE_SENSOR_I_ON);
17         cnt = (cnt + 1) % PERIOD;
18     }
19     else
20     {
21         i.write(METHANE_SENSOR_I_IDLE);
22         cnt = (cnt + 1) % PERIOD;
23     }
24 }
```

Define **processing** method.  
Is the core of the simulation  
and defines the behavior of  
the module.

# **Lab 3 – Part 1**

## **Model of the photovoltaic module**

# Simulator overview



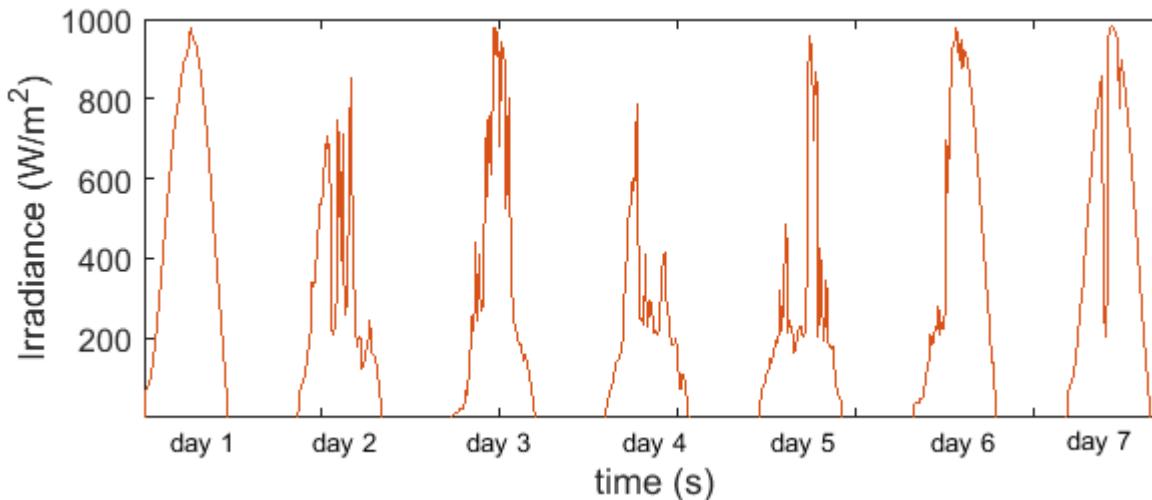
# Photovoltaic module

- Gmonth
  - Can be inspected in MATLAB
    - To load it, type:
      - `load ('gmonths.mat')`
    - Or you can refer to `gmonths.txt` inside simulator (more info later)
    - Values of irradiance over time
      - Input of the photovoltaic cell
      - Two columns matrix:
        - First column: time (in seconds)
        - Second column: irradiance value (in W/m<sup>2</sup>)
        - Each line corresponds to a new value of irradiance at a given time

Editor - panel.m		
Gmonth		
1590x2 double		
	1	2
1	0	0
2	900	67.6853
3	1800	77.7214
4	2700	87.3263
5	3600	100.5830
6	4500	124.0773
7	5400	155.2077
8	6300	191.0537
9	7200	233.1320
10	8100	278.4866
11	9000	324.2709
12	9900	370.6427
13	10800	422.2269
14	11700	469.9433
15	12600	514.6853
16	13500	563.9495
17	14400	608.4416
18	15300	653.1028
19	16200	692.3697
20	17100	732.9693
21	18000	759.4004

# Photovoltaic module

- Gmonth
  - Irradiance trace for about 3 months
    - Irradiance varies depending on weather (sunny/rainy/cloudy)
    - E.g.: figure  
`plot (Gmonth (:, 1), Gmonth (:, 2))`



Editor - panel.m		
Gmonth		
1590x2 double		
1	0	0
2	900	67.6853
3	1800	77.7214
4	2700	87.3263
5	3600	100.5830
6	4500	124.0773
7	5400	155.2077
8	6300	191.0537
9	7200	233.1320
10	8100	278.4866
11	9000	324.2709
12	9900	370.6427
13	10800	422.2269
14	11700	469.9433
15	12600	514.6853
16	13500	563.9495
17	14400	608.4416
18	15300	653.1028
19	16200	692.3697
20	17100	732.9693
21	18000	759.4004

# Photovoltaic module

- Photovoltaic module
  - Two lookup tables (**LUT**)
    - Voltage and current at the MPP given irradiance in input
    - Must populate the lookup tables!

`inc/pv_panel.h`

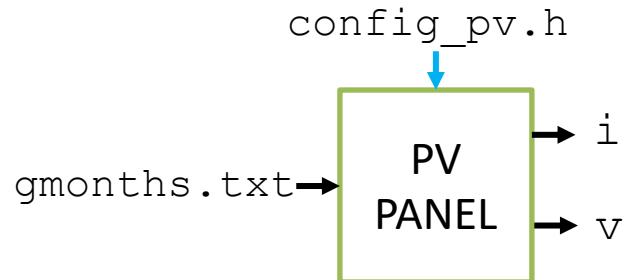
```
#include "config.h"
#include "config_pv.h" →
#include "lut.h"

SCA_TDF_MODULE(pv_panel)
{
    sca_tdf::sca_out<double> i; // Output current at mpp
    sca_tdf::sca_out<double> v; // Output voltage at mpp

    SCA_CTOR(pv_panel): i("i"), v("v") {}

    void set_attributes();
    void initialize();
    void processing();

private:
    int cnt; // when cnt % TRACE_PERIOD == 0 a new measure of irradiance
            // is read from the file
    ifstream top; // file from which irradiance values are retrieved
    double g_top;
    LUT lut_i = LUT(G, I_MPP, SIZE_PV);
    LUT lut_v = LUT(G, V_MPP, SIZE_PV); ←
};
```



`inc/config_pv.h`

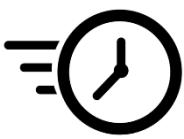
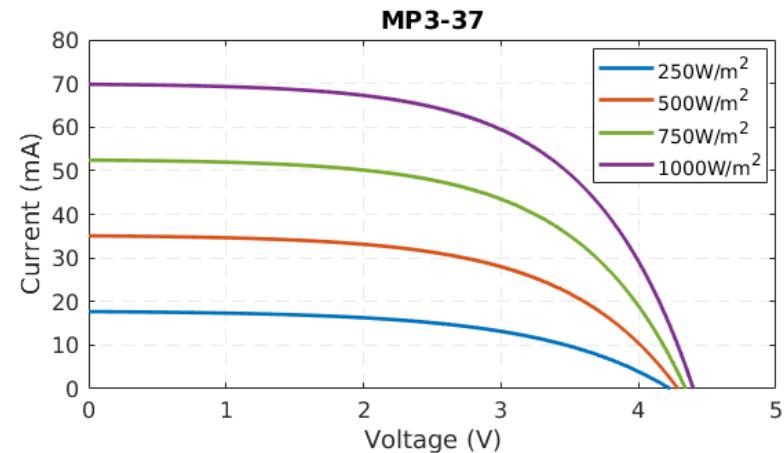
```
#define TRACE_PERIOD 900
#define SIZE_PV 4
static const double G[SIZE_PV] = {TO-BE-POPULATED};
static const double I_MPP[SIZE_PV] = {TO-BE-POPULATED};
static const double V_MPP[SIZE_PV] = {TO-BE-POPULATED};
```



**TO-BE-POPULATED**

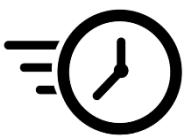
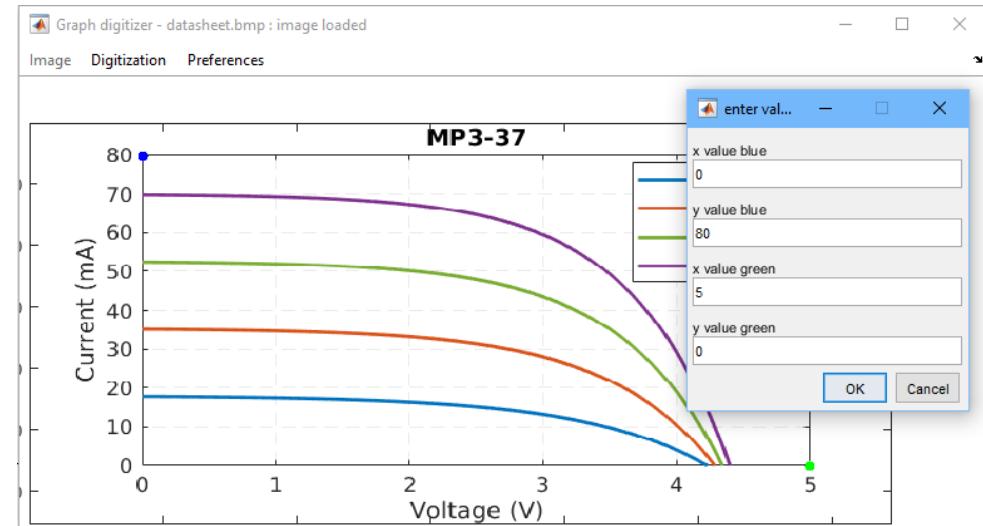
# Photovoltaic module

- Open the PVdatasheet.pdf file
- Provides a graph of current versus voltage given different irradiance values
- **Your Task:**
  - Save the image
  - For each curve:
    1. Digitize the curve
      - Can use the digitizer tool
    2. Determine the voltage and current at the MPP for that irradiance value
    3. Use the sample for the lookup table



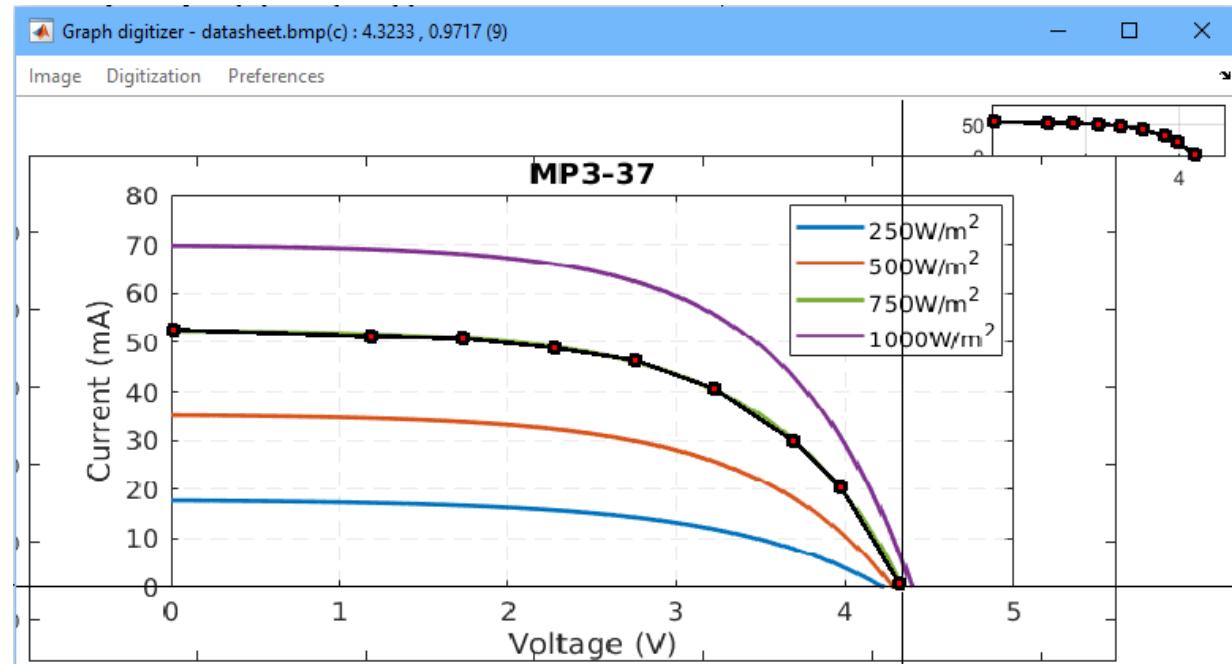
# Photovoltaic module

- *Suggestion: use MATLAB digitizer tool*
- How to use the tool:
  1. Load the image
  2. Calibrate the image
    - Select two points and give corresponding x and y values
    - Conversion factor 1
    - 4 decimals precision



# Photovoltaic module

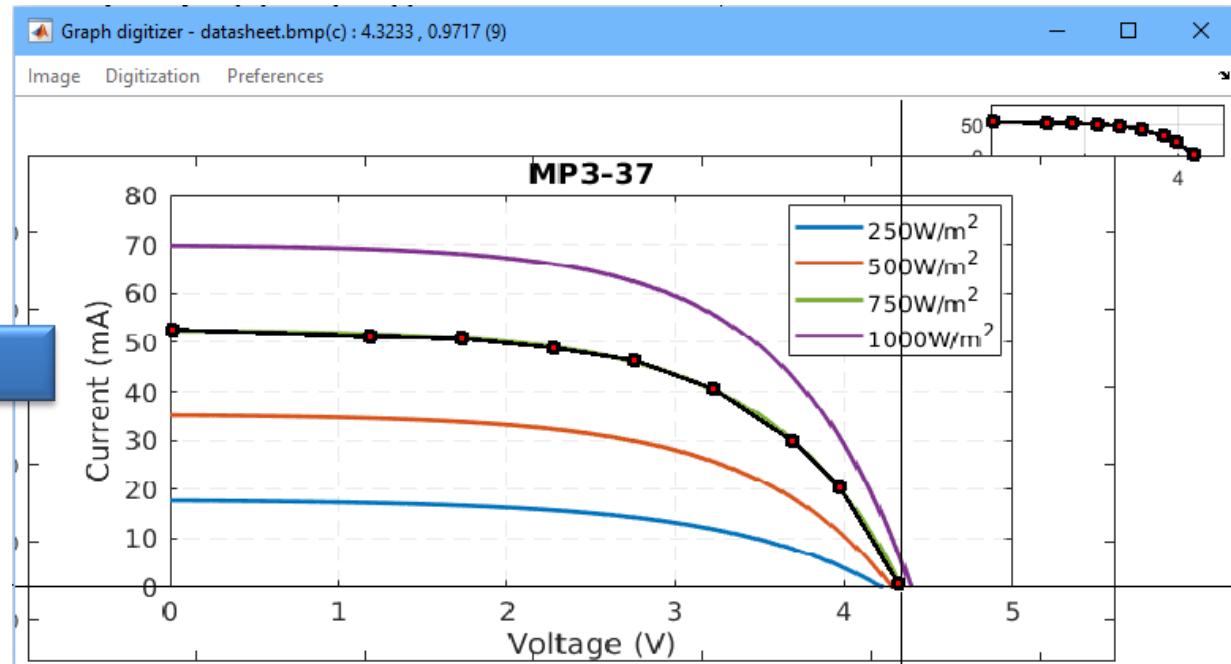
- How to use the digitizer tool:
  3. Select samples of one curve
    - Mouse left button = new sample
    - Mouse middle button = stop inserting samples



# Photovoltaic module

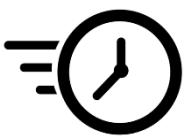
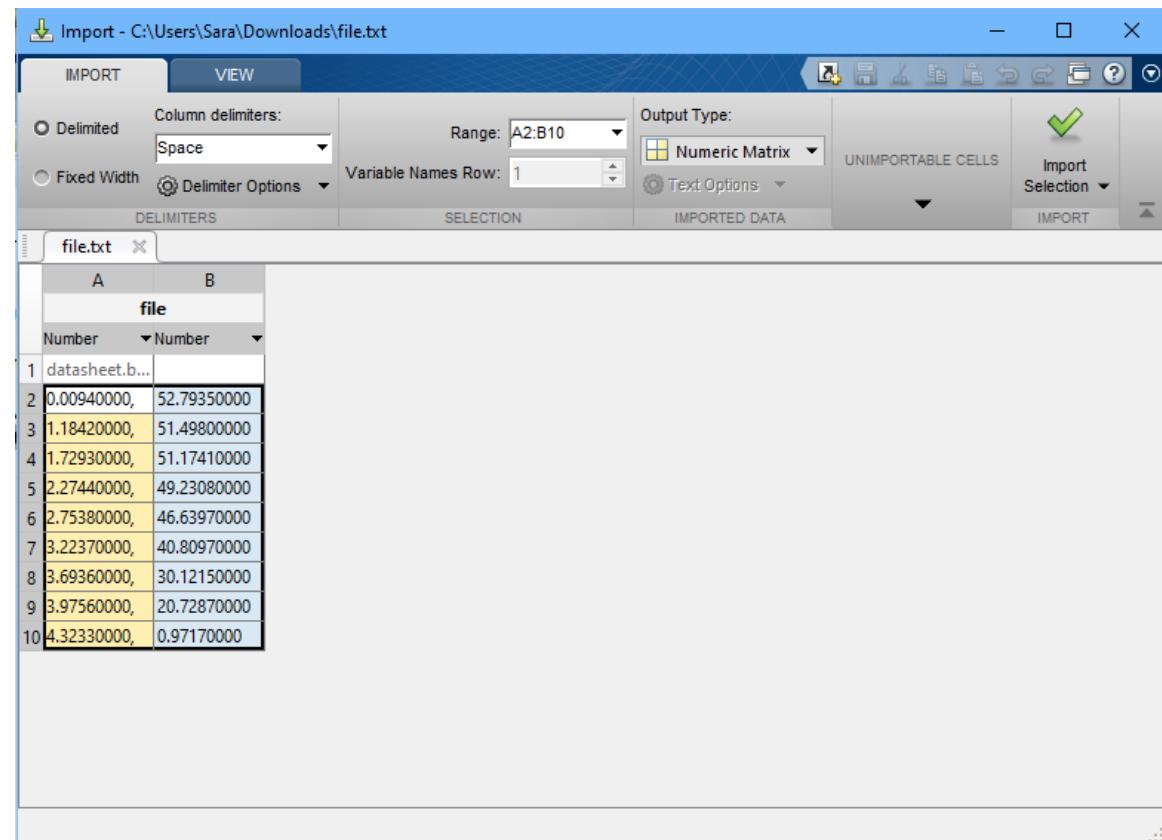
- How to use the digitizer tool:
  3. Select samples of one curve
    - Mouse left button = new sample
    - Mouse middle button = stop inserting samples
  4. Save to file

```
datasheet.txt
0.00940000, 52.79350000
1.18420000, 51.49800000
1.72930000, 51.17410000
2.27440000, 49.23080000
2.75380000, 46.63970000
3.22370000, 40.80970000
3.69360000, 30.12150000
3.97560000, 20.72870000
4.32330000, 0.97170000
```



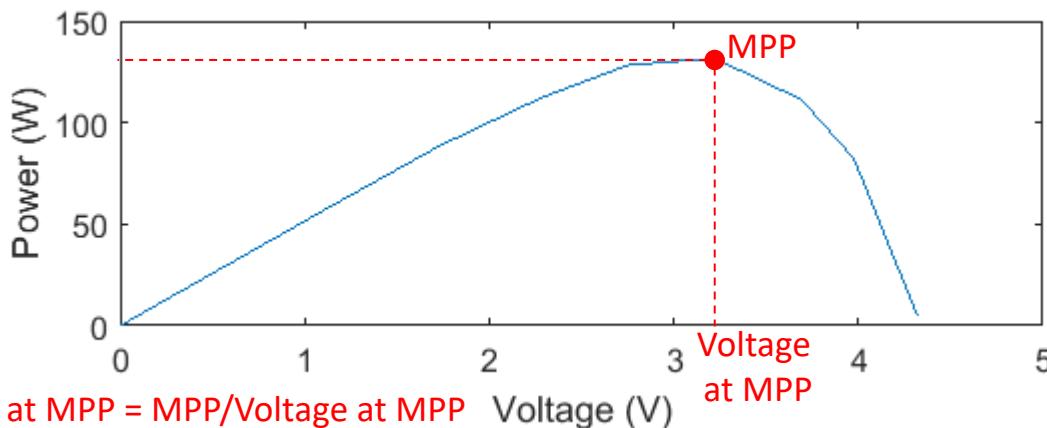
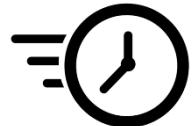
# Photovoltaic module

- Each file represents one curve
- Import the file in Matlab:
  - Home > Import data > Select the file
  - File as a matrix variable



# Photovoltaic module

- Each file represents one curve
  - Import the file in Matlab as variables
- To extrapolate MPP:
  - Build the corresponding power curve
  - Extrapolate V and I at the MPP
    - I.e., voltage and current corresponding to the maximum value of power for this curve



datasheet.txt

0.00940000, 52.79350000
1.18420000, 51.49800000
1.72930000, 51.17410000
2.27440000, 49.23080000
2.75380000, 46.63970000
3.22370000, 40.80970000
3.69360000, 30.12150000
3.97560000, 20.72870000
4.32330000, 0.97170000

# Photovoltaic module

- Populate the array in `inc/config_pv.h`:
  - Vector of values of G:
    - `G[SIZE_PV] = {250, 500, 750, 1000};`
  - Vector of corresponding values of current at the MPP
    - `I_MPP[SIZE_PV] = {I250, I500, I750, I1000};`
  - Vector of corresponding values of voltage at the MPP
    - `V_MPP[SIZE_PV] = {V250, V500, V750, V1000};`

N.B., Take care of units!! We  
want voltages in [V] and  
currents in [mA]



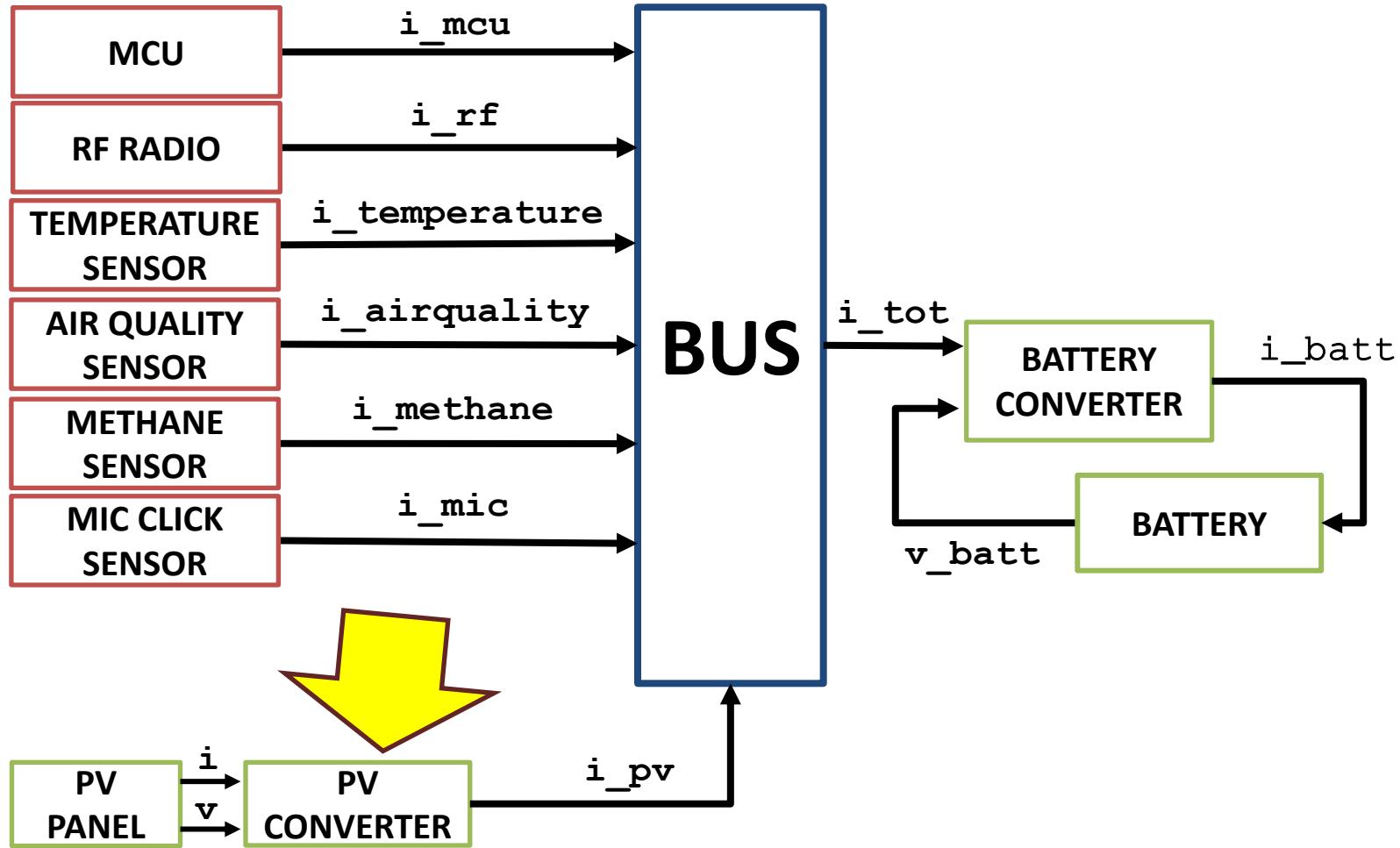
# Photovoltaic module

- Lookup table output:
  - If input irradiance is one of the sampled values, the lookup table outputs the corresponding value of voltage/current
  - Else, the lookup table interpolates between the available samples and estimate the value of voltage/current

# **Lab 3 – Part 2**

## **Model of DC-DC converters and battery**

# Simulator overview



# DC-DC converter of PV module

- DC-DC converter of the photovoltaic module
  - Contains a **LUT** of efficiency given input PV voltage
    - **Efficiency (eta) in [0, 100]**
  - Block implementation (in `src/converter_pv.cpp`):
    - $(V_{PV} \cdot I_{PV}) \cdot \text{eta} = (V_{BUS} \cdot I) \rightarrow I = (V_{PV} \cdot I_{PV} \cdot \text{eta}) / V_{BUS}$

`inc/converter_pv.h`

```
#include "config.h"
#include "config_converter_pv.h"
#include "lut.h"

SCA_TDF_MODULE(converter_pv)
{
    sca_tdf::sca_in<double> i_in; // Current from PV panel
    sca_tdf::sca_in<double> v_in; // Voltage from PV panel
    sca_tdf::sca_out<double> i_out; // Current generated delivered to the BUS

    SCACTOR(converter_pv): i_in("i_in"),
                           v_in("v_in"),
                           i_out("i_out") {};

    void set_attributes();
    void initialize();
    void processing();

private:
    LUT lut_eta = LUT(V_CONV_PV, ETA_CONV_PV, SIZE_CONV_PV);
```

`config_converter_pv.h`



`inc/config_converter_pv.h`

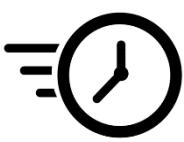
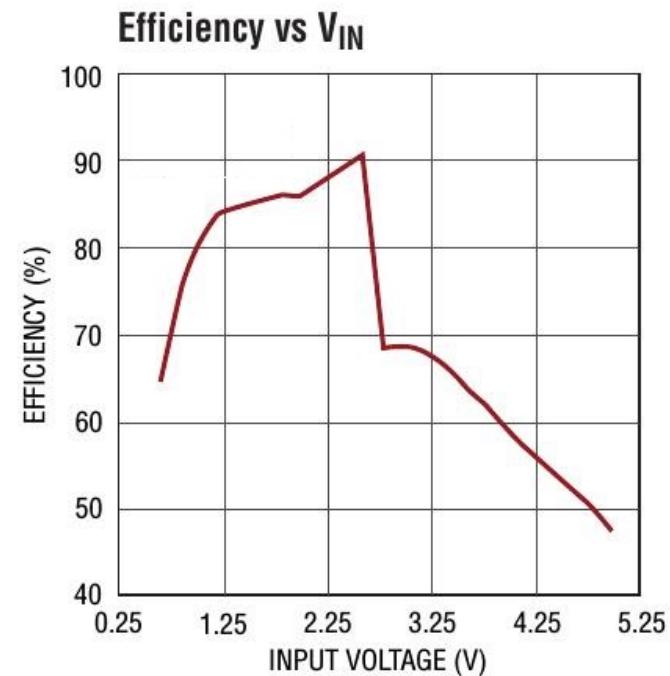
```
#define SIZE_CONV_PV DEPENDS-ON-NUM-DIGITIZED-SAMPLES
static const double V_CONV_PV[SIZE_CONV_PV] = {TO-BE-POPULATED};
static const double ETA_CONV_PV[SIZE_CONV_PV] = {TO-BE-POPULATED};
```



TO-BE-POPULATED

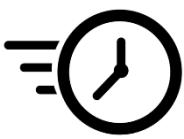
# DC-DC converter of PV module

- Must populate the lookup table of efficiency given input voltage of the photovoltaic module
  - Open the **PV\_DCDCConverter.pdf** file
    - Page 5: curve of efficiency w.r.t. input voltage
  - Use the samples of the curve to populate the lookup table

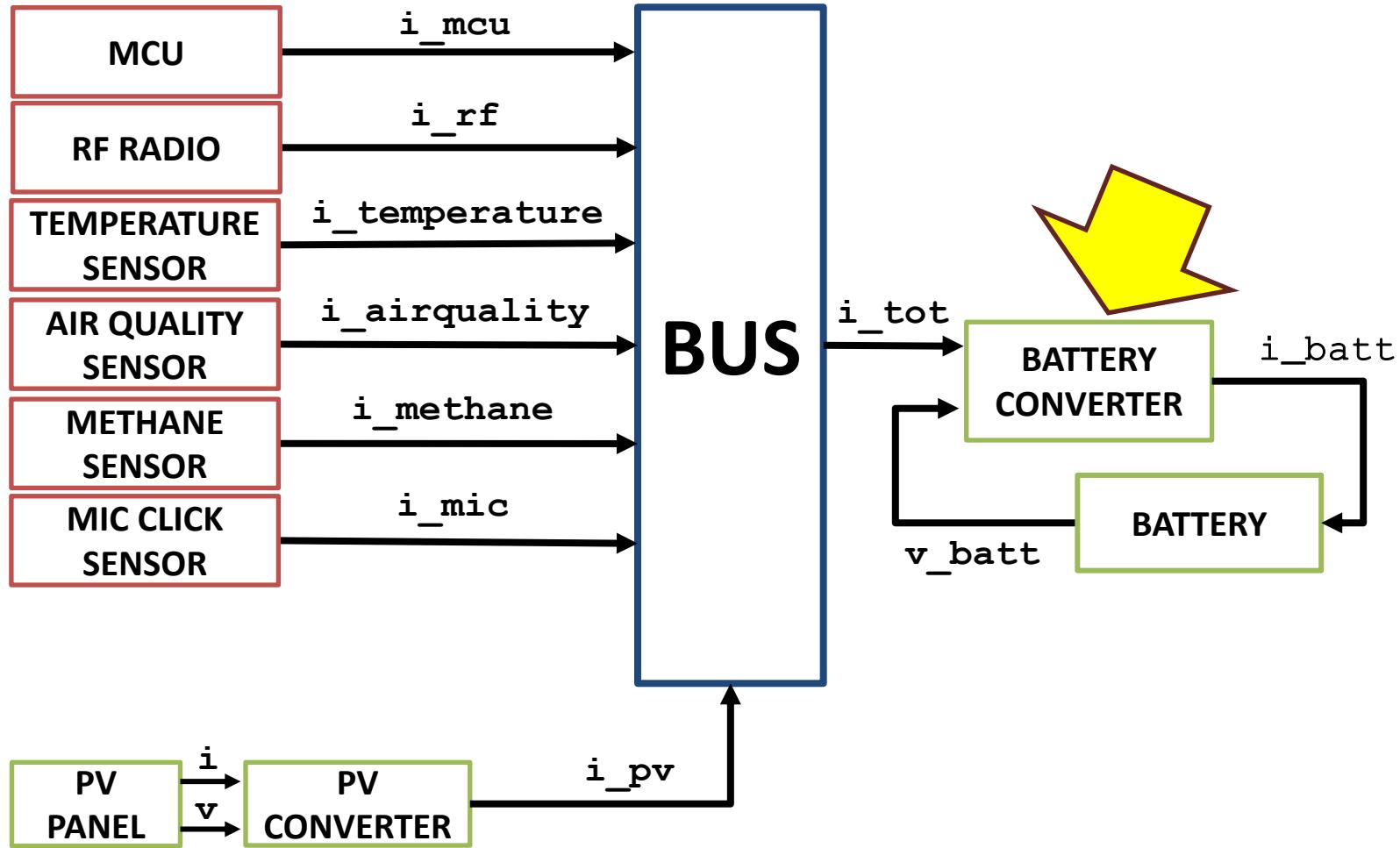


# DC-DC converter of PV module

- Your Task:
  1. Digitize the curve
  2. Export digitized samples
  3. Set the parameters in `inc/config_converter_pv.h`:
    - `#define SIZE_CONV_PV #NUM-SAMPLES`
    - `V_CONV_PV[SIZE_PV] = {S1, S2, . . . ,};`
    - `ETA_CONV_PV[SIZE_PV] = {S1, S2, . . . ,};`
    - More digitized samples = more accurate!!!



# Simulator overview



# DC-DC converter of battery

- DC-DC converter of the battery

inc/converter\_battery.h

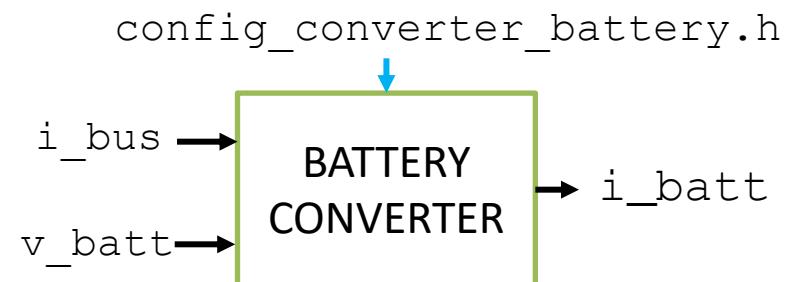
```
#include "config.h"
#include "config_converter_battery.h"
#include "lut.h"

SCA_TDF_MODULE(converter_battery)
{
    sca_tdf::sca_in<double> i_bus; // Current requested/delivered to battery conv
    sca_tdf::sca_in<double> v_batt; // Battery voltage
    sca_tdf::sca_out<double> i_batt; // Battery current

    SCACTOR(converter_battery): i_bus("i_bus"),
                                v_batt("v_batt"),
                                i_batt("i_batt") {};

    void set_attributes();
    void initialize();
    void processing();

    private:
        LUT lut_eta = LUT(I_CONV_BATT, ETA_CONV_BATT, SIZE_CONV_BATT);
};
```



inc/config\_converter\_battery.h

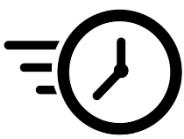
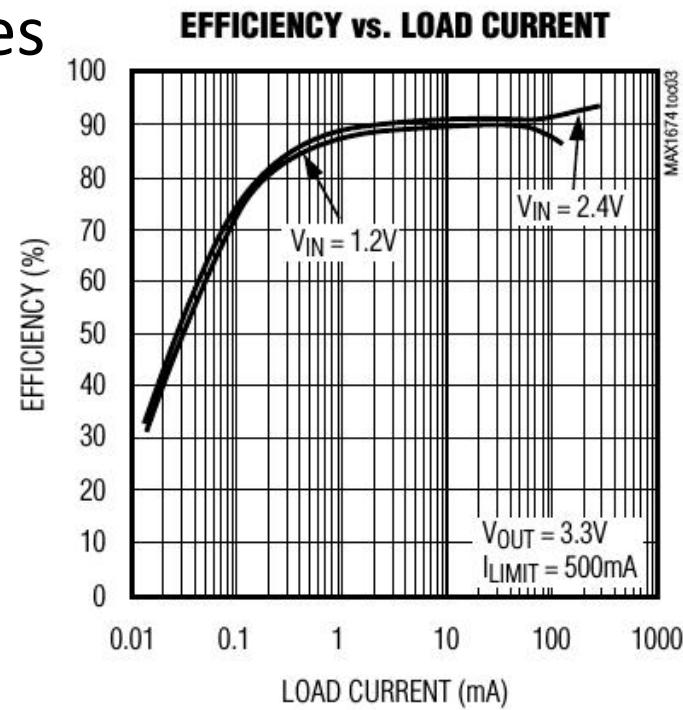
```
#define SIZE_CONV_BATT DEPENDS-ON-NUM-DIGITIZED-SAMPLES
static const double I_CONV_BATT[SIZE_CONV_BATT] = {TO-BE-POPULATED};
static const double ETA_CONV_BATT[SIZE_CONV_BATT] = {TO-BE-POPULATED};
```



TO-BE-POPULATED

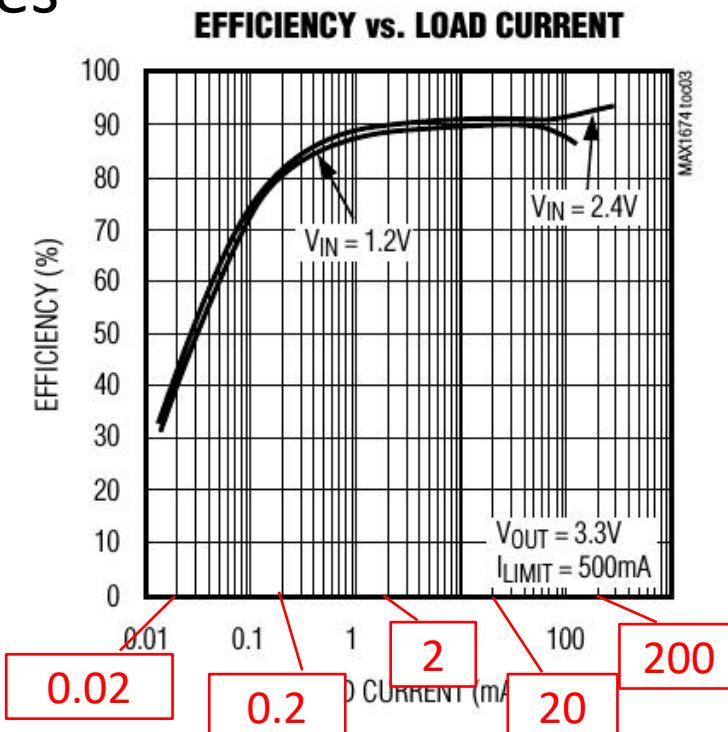
# DC-DC converter of battery

- Open the **BATT\_DCDCconv.pdf** file
  - Page 4: curve of efficiency given input current with voltage to the bus = 3.3V
- Your Task:
  1. Digitize one of the two curves
    - We avoid dependency from battery voltage
  2. Export the samples

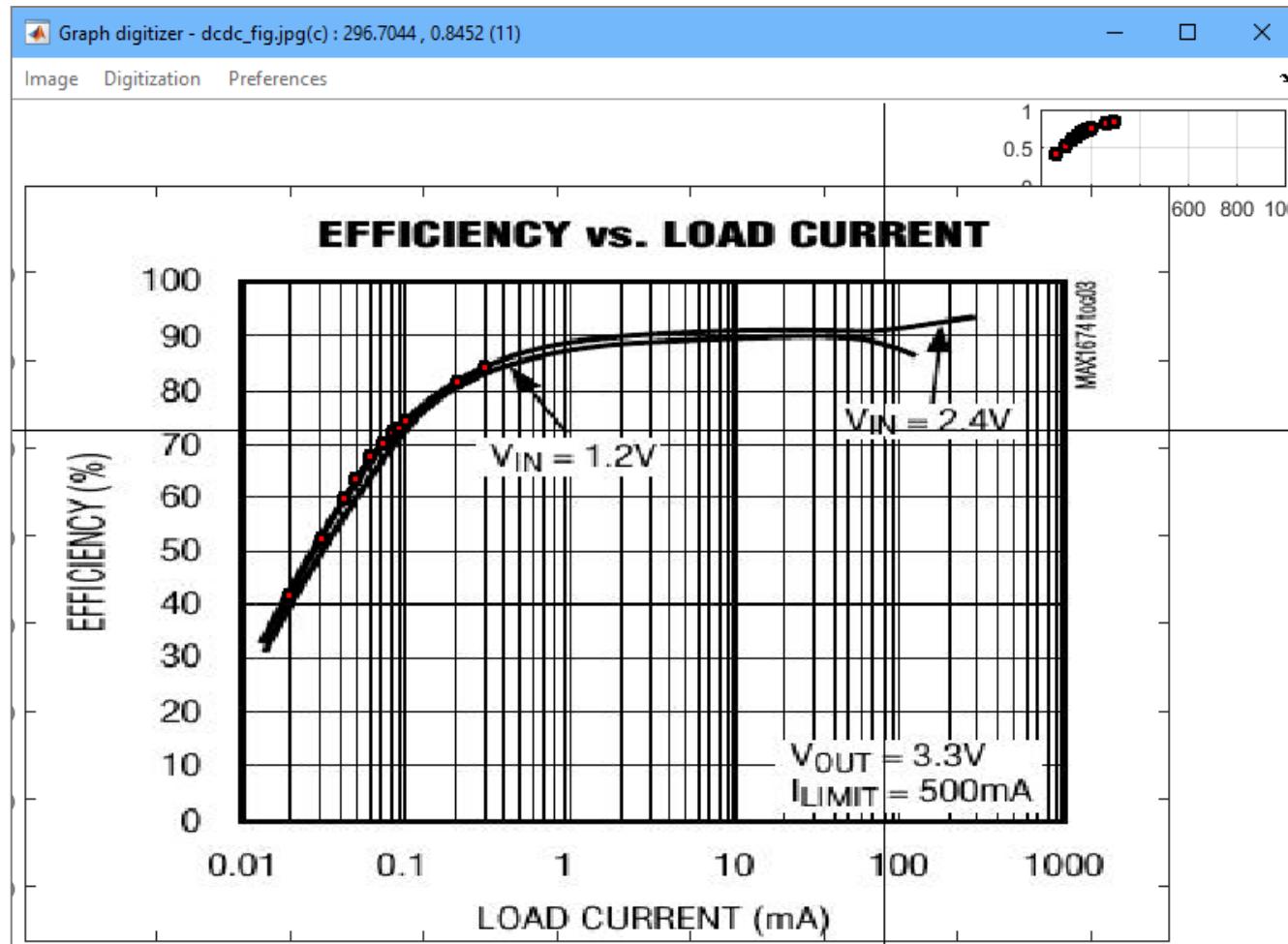


# DC-DC converter of battery

- Open the **BATT\_DCDCconv.pdf** file
  - Page 4: curve of efficiency given input current with voltage to the bus = 3.3V
- Your Task:
  1. Digitize one of the two curves
    - We avoid dependency from battery voltage
    - **ISSUE: x axis is in logarithmic scale**
      - Sample points corresponding to the vertical lines
      - Can replace the x coordinate of samples with known values
  2. Export the samples

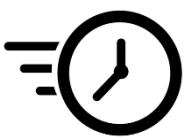


# DC-DC converter of battery

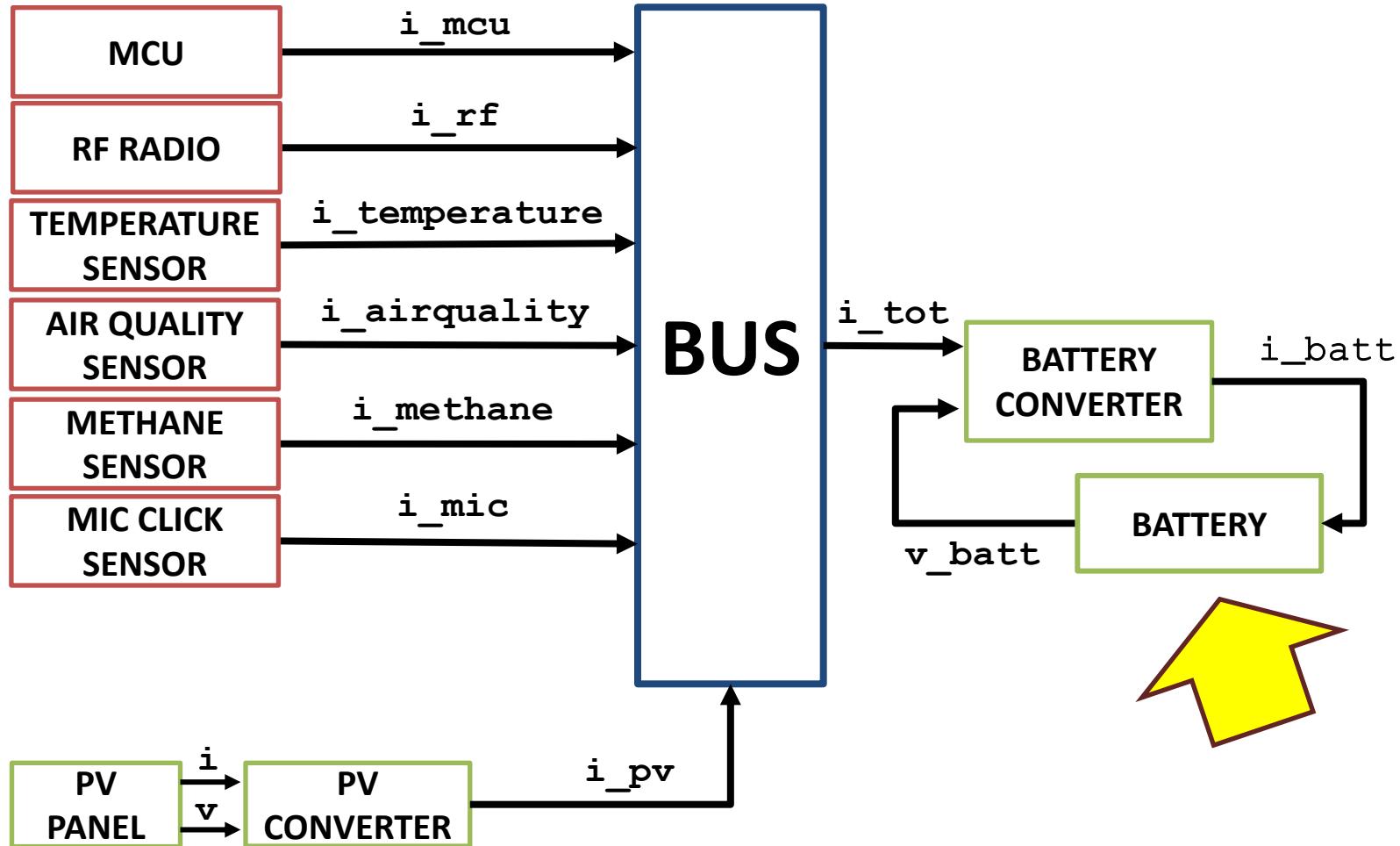


# DC-DC converter of battery

- Your Task:
  3. Set the parameters in  
`inc/config_converter_battery.h` (as before).

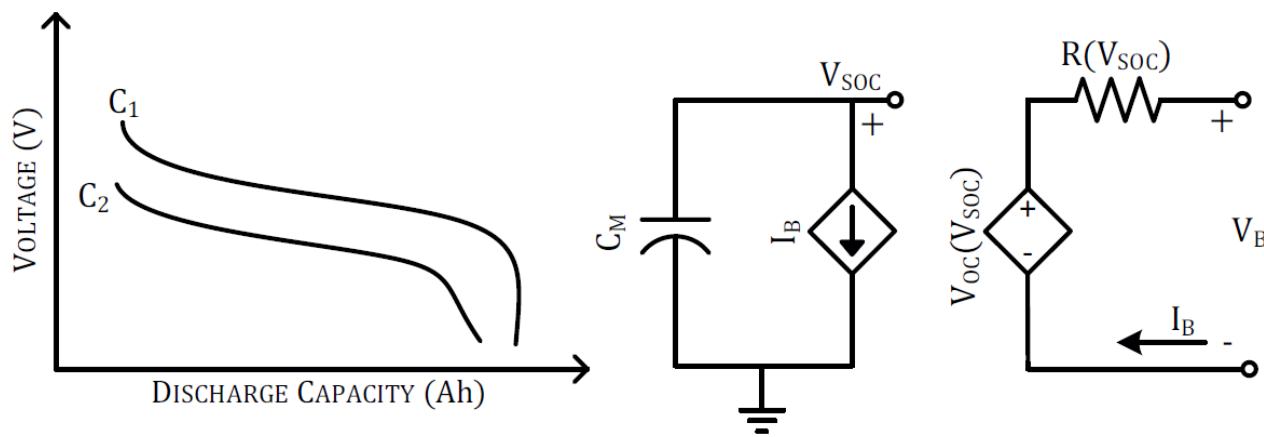


# Simulator overview



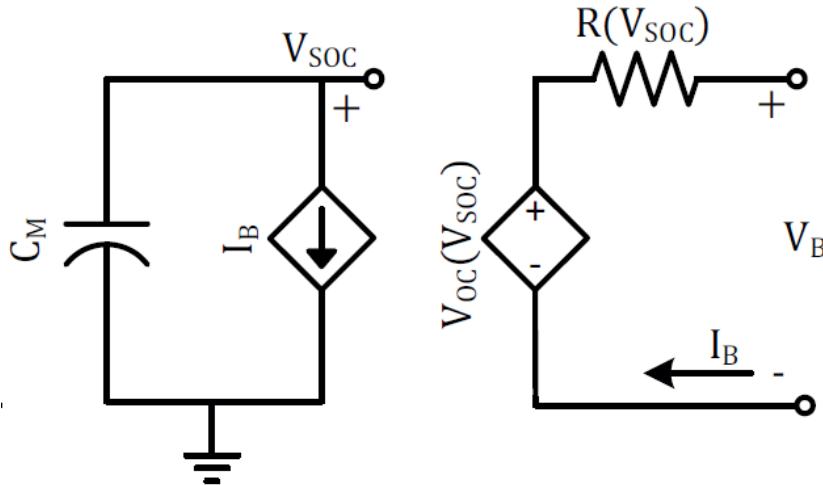
# Battery model

- Model: a circuit model
  - Configuration and settings extracted solely from data provided in the datasheet
  - Model made of two branches:
    - Left: models battery lifetime and the SOC
    - Right: models battery dynamics



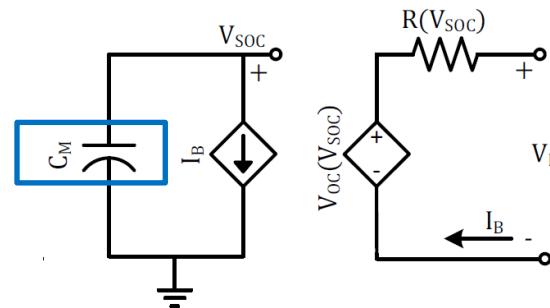
# Battery model

- Left:
  - $C_M$  available usable capacity
  - $I_B$  discharge current
  - $V_{SOC}$  state of charge
- Right:
  - $V_{OC}$  models the dependency between battery voltage vs. the SOC
  - $I_B$  discharge current
  - $R$  series resistance modeling voltage drop due to Ohmic losses (i.e., the internal resistance of the battery)
  - $V_B$  is battery voltage



# Battery model

- We must reconstruct the values for the single elements of the circuit from the datasheet
- Capacity
  - 3200mAh (3.2 Ah)
  - Timestep is 1s
    - Multiply per 3600 (seconds in 1 hour)
  - Current in mA

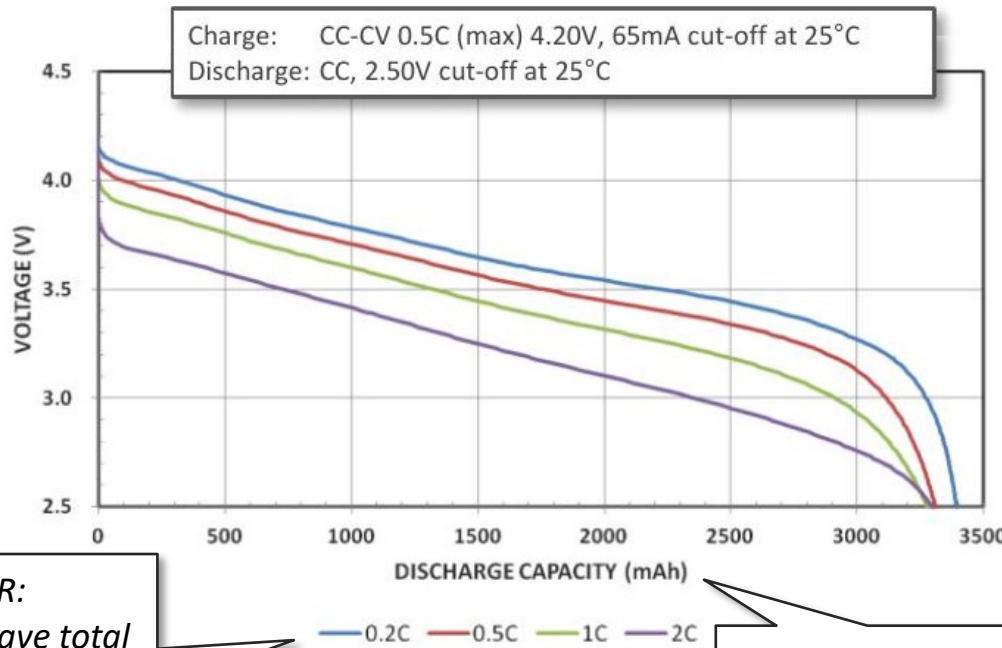


## Specifications

Rated capacity <sup>(1)</sup>	Min. 3200mAh
Capacity <sup>(2)</sup>	Min. 3250mAh Typ. 3350mAh
Nominal voltage	3.6V
Charging	CC-CV, Std. 1625mA, 4.20V, 4.0 hrs
Weight (max.)	48.5 g
Temperature	Charge*: 0 to +45°C Discharge: -20 to +60°C Storage: -20 to +50°C
Energy density <sup>(3)</sup>	Volumetric: 676 Wh/l Gravimetric: 243 Wh/kg

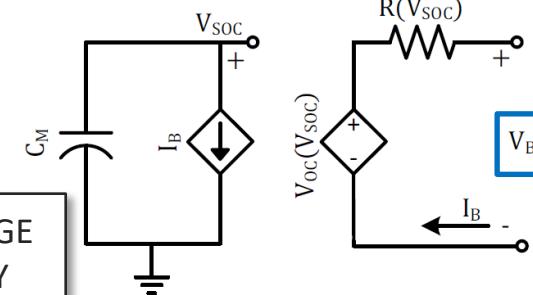
# Battery model

- $V_B$  battery voltage
  - Function of the load current and of the SOC



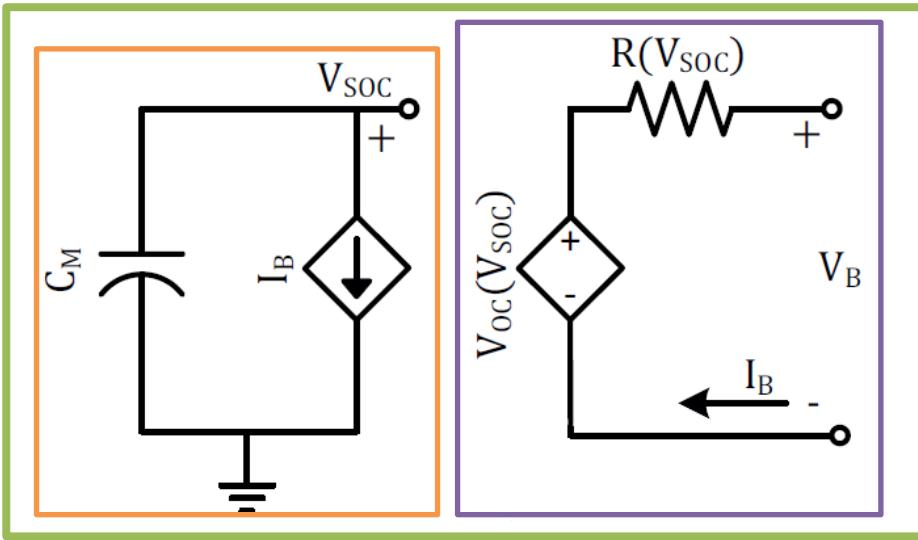
REMINDER:  
 $1C$  = current to have total  
discharge in 1 hour

≈ STATE OF CHARGE  
OF THE BATTERY



# SystemC implementation

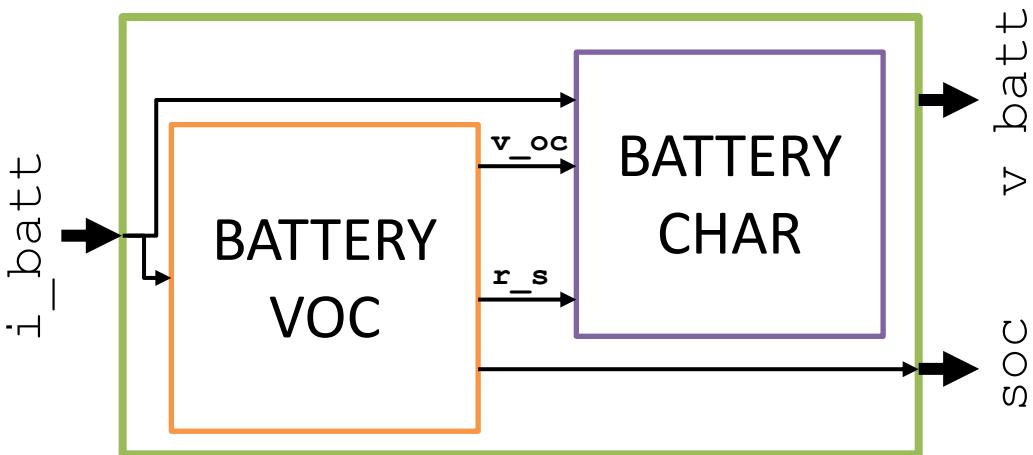
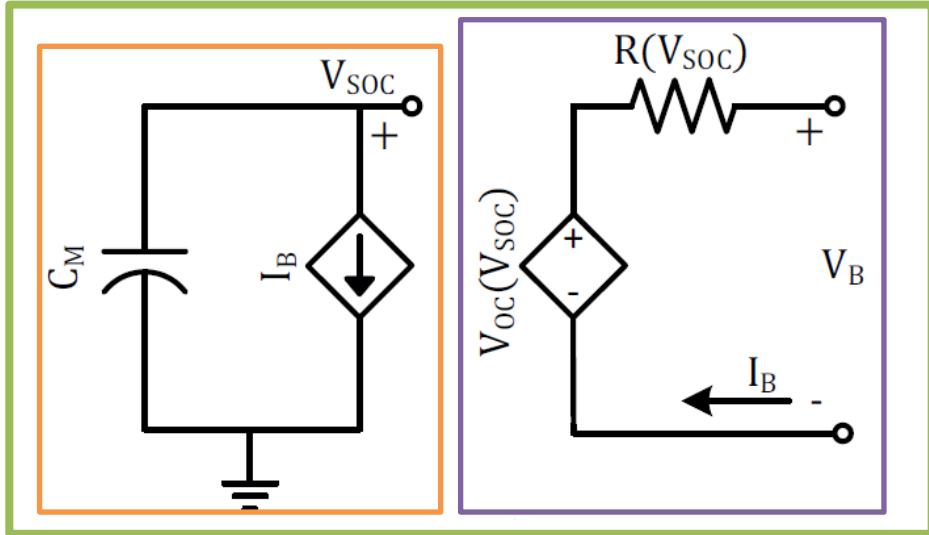
- Battery is defined in inc/battery.h
  - Includes two submodules:
    - inc/battery\_voc.h → Implement left part of circuit
    - inc/battery\_char.h → Implement right part of circuit



inc/battery.h

```
#include "battery_char.h"  
#include "battery_voc.h"  
  
SC_MODULE(battery)  
{  
    // Interface and internal components declaration  
    sca_tdf::sca_in<double> i_batt; // Battery current  
    sca_tdf::sca_out<double> v_batt; // Battery voltage  
    sca_tdf::sca_out<double> soc; // Battery SOC  
  
    // Connecting signals  
    sca_tdf::sca_signal<double> v_oc, r_s;  
  
    // Instantiation of battery components  
    battery_voc* voc_module; ←  
    battery_char* char_module; ←  
  
    SC_CTOR(battery): i_batt("i_batt"),  
                      v_batt("v_batt"),  
                      soc["soc"] ← You, last week • ne  
    {  
        voc_module = new battery_voc("voc"); ←  
        char_module = new battery_char("batt"); ←  
  
        voc_module->i(i_batt);  
        voc_module->v_oc(v_oc);  
        voc_module->r_s(r_s);  
        voc_module->soc(soc);  
  
        char_module->r_s(r_s);  
        char_module->i(i_batt);  
        char_module->v_oc(v_oc);  
        char_module->v(v_batt);  
    };  
};
```

# SystemC implementation (cont'd)

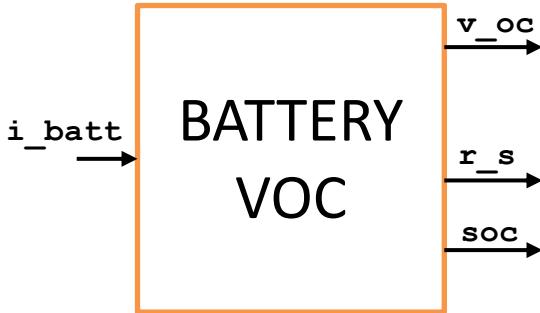
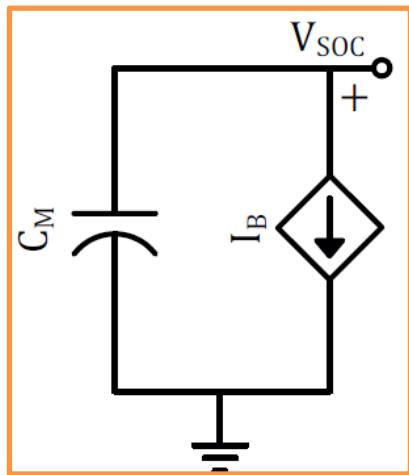


inc/battery.h

```
#include "battery_char.h"  
#include "battery_voc.h"  
  
SC_MODULE(battery)  
{  
    // Interface and internal components declaration  
    sca_tdf::sca_in<double> i_batt; // Battery current  
    sca_tdf::sca_out<double> v_batt; // Battery voltage  
    sca_tdf::sca_out<double> soc; // Battery SOC  
  
    // Connecting signals  
    sca_tdf::sca_signal<double> v_oc, r_s;  
  
    // Instantiation of battery components  
    battery_voc* voc_module; // ←  
    battery_char* char_module; // ←  
  
    SC_CTOR(battery): i_batt("i_batt"),  
                      v_batt("v_batt"),  
                      soc("soc")  
    {  
        voc_module = new battery_voc("voc"); // ←  
        char_module = new battery_char("batt"); // ←  
  
        voc_module->i(i_batt);  
        voc_module->v_oc(v_oc);  
        voc_module->r_s(r_s);  
        voc_module->soc(soc);  
  
        char_module->r_s(r_s);  
        char_module->i(i_batt);  
        char_module->v_oc(v_oc);  
        char_module->v_batt(v_batt);  
    };
```

# SystemC implementation (cont'd)

- **inc/battery\_voc.h** → Implement left part of circuit



**src/battery\_voc.cpp**

```
void battery_voc::processing()
{
    double tmpcurrent; // Battery current, if negative, the battery is charged
    tmpcurrent = i.read(); // Battery current, if negative, the battery is charged

    // Compute actual state-of-charge solving the integral:
    C_nom = TO-BE-FILLED;
    tmpsoc -= (((tmpcurrent + prev_i_batt) * SIM_STEP) /
               (2 * 3600 * C_nom)); // 3600 * Cnom, mAh to mA cause [sim step] = [s]
    prev_i_batt = tmpcurrent; // Update

    // Output the battery soc
    if(tmpsoc >= 1) // Not let the SOC overflow
    {
        soc.write(1);
        tmpsoc = 1;
    }
    else
    {
        soc.write(tmpsoc);
    }

    // SOC and battery Voc relationship
    v_oc.write(TO-BE-FILLED); // Place interpolated funct here

    // SOC and battery internal resistance relationship
    r_s.write(TO-BE-FILLED); // Place interpolated funct here

    // When the battery SOC decreases under 1%, the simulation stops.
    if(tmpsoc <= 0.01)
    {
        cout << "SOC is less than or equal to 1%:" << "@" << sc_time_stamp() << endl;
        sc_stop();
    }
}
```

**TO-BE-FILLED**

**TO-BE-FILLED**

**TO-BE-FILLED**

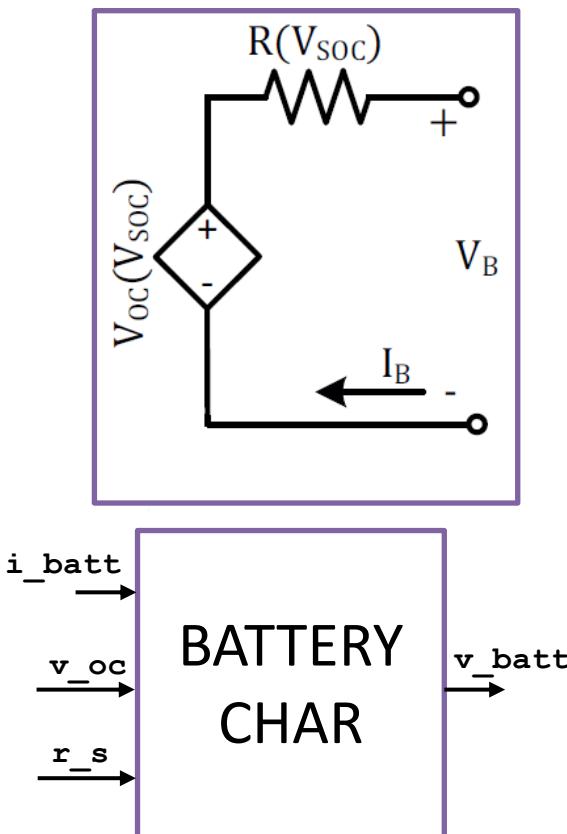
# SystemC implementation (cont'd)

- **inc/battery\_char.h** → Implement right part of circuit

**inc/battery\_char.h**

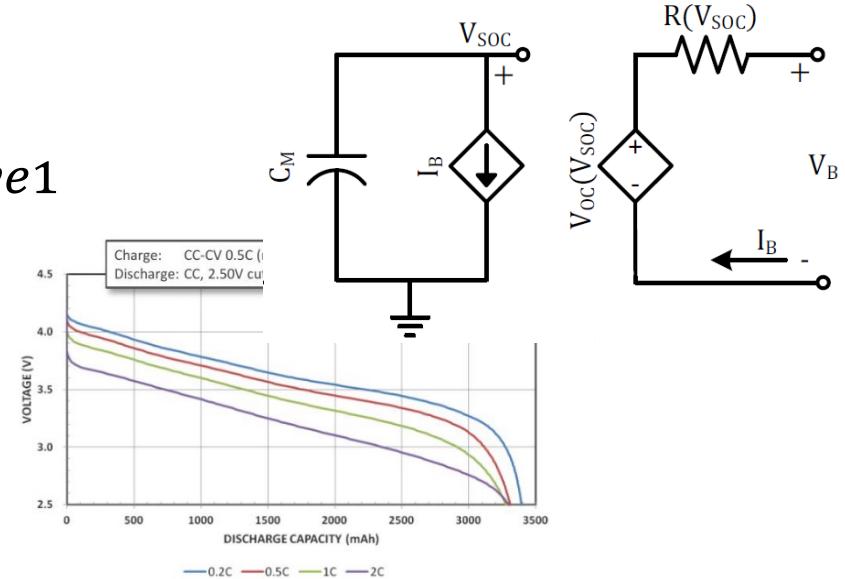
```
SC_CTOR(battery_char)
{
    // V_oc voltage instantiation
    V_oc = new sca_eln::sca_tdf::sca_vsource("V_oc");
    V_oc->inp(v_oc);
    V_oc->p(n1);
    V_oc->n(gnd);
    // Internal resistance instantiation
    R_s = new sca_eln::sca_tdf::sca_r("R_s");
    R_s->p(n1);
    R_s->n(n2);
    R_s->scale=1.0;
    R_s->inp(r_s);
    //Load current instantiation
    I_batt = new sca_eln::sca_tdf::sca_isource("I_batt");
    I_batt->inp(i);
    I_batt->p(n2);
    I_batt->n(gnd);
    //Output voltage of the battery
    V_batt = new sca_eln::sca_tdf::sca_vsink("V_batt");
    V_batt->p(n2);
    V_batt->n(gnd);
    V_batt->outp(v);
}
```

Implemented  
as a circuit  
netlist!



# Battery model

- Derive  $V_{OC}$  and  $R$  as a function of  $V$  (SOC)
  - Solving the equations associated with the right hand side branch of the circuit
  - $V_{OC} = V_{curve1} + R \cdot I_{curve1}$
  - $R = \frac{V_{curve2} - V_{curve1}}{I_{curve1} - I_{curve2}}$
  - Notation:
    - *curve1* and *curve2* refer to the operation with two different discharge currents, i.e., to two different discharge curves obtained with different discharge current values



# Battery model

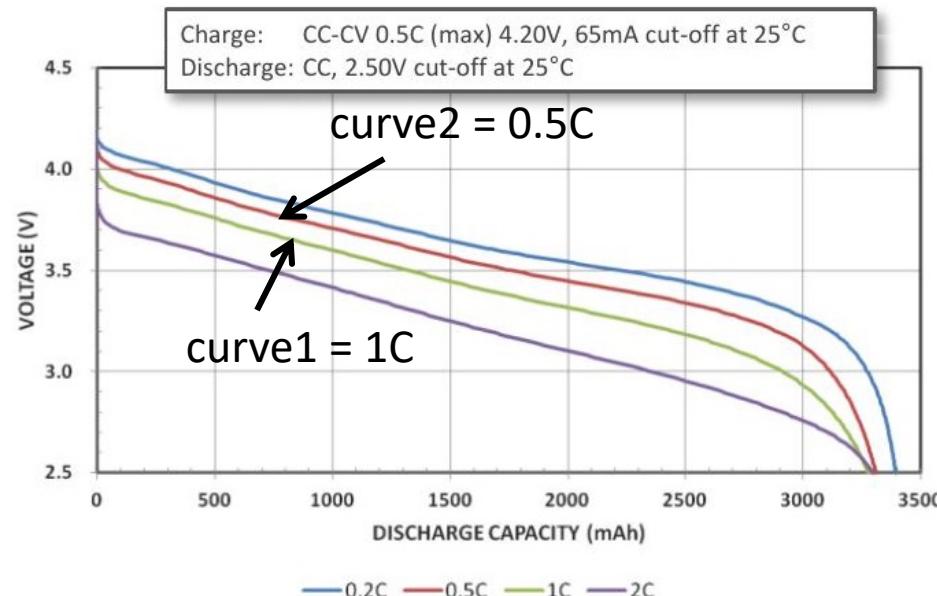
- In case you were wondering how we got here...
  - Equation solving the system:
    - $V_{OC} = R I + V$
  - Applied for both currents:
    - $\begin{cases} V_{OC} = RI_{curve1} + V_{curve1} \\ V_{OC} = RI_{curve2} + V_{curve2} \end{cases}$
  - $V_{OC} = RI_{curve1} + V_{curve1} = RI_{curve2} + V_{curve2}$
  - $R(I_{curve1} - I_{curve2}) = V_{curve2} - V_{curve1}$
  - $R = \frac{V_{curve2} - V_{curve1}}{I_{curve1} - I_{curve2}}$

# Battery model

- How do we populate the equations?

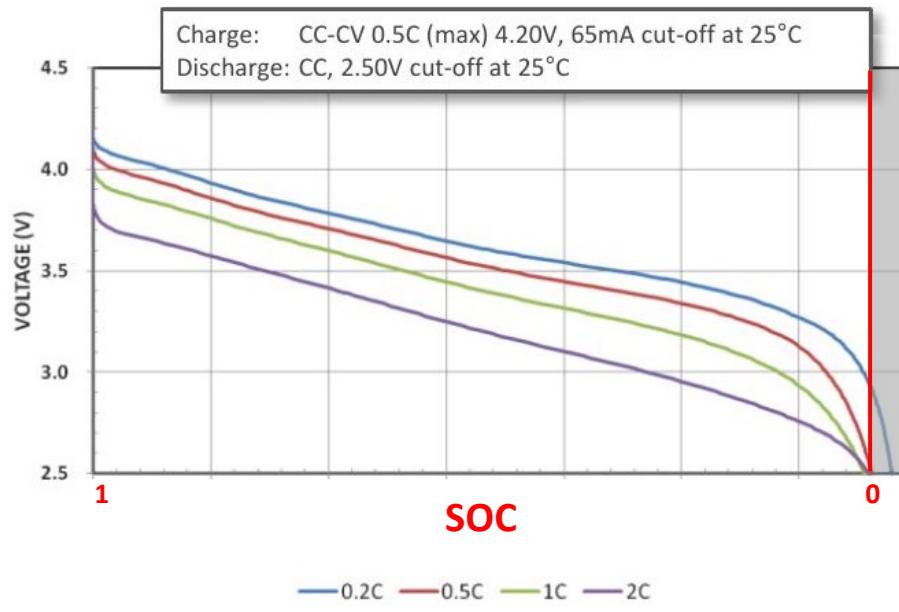
# Battery model

- Derive  $V_{OC}$  and  $R$  as a function of  $V$ 
  - $V_{OC} = V_{curve1} + R \cdot I_{curve1}$
  - $R = \frac{V_{curve2} - V_{curve1}}{I_{curve1} - I_{curve2}}$
  - Need two curves of  $V$  given the discharge capacity
    - See example →
    - Values of  $V$  given SOC and discharge current



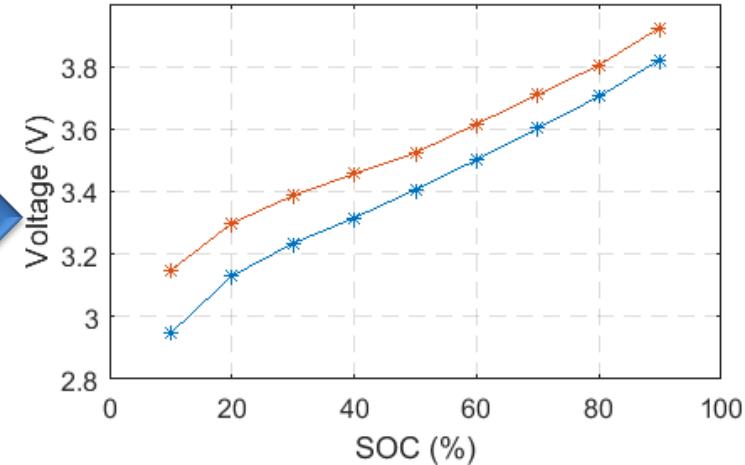
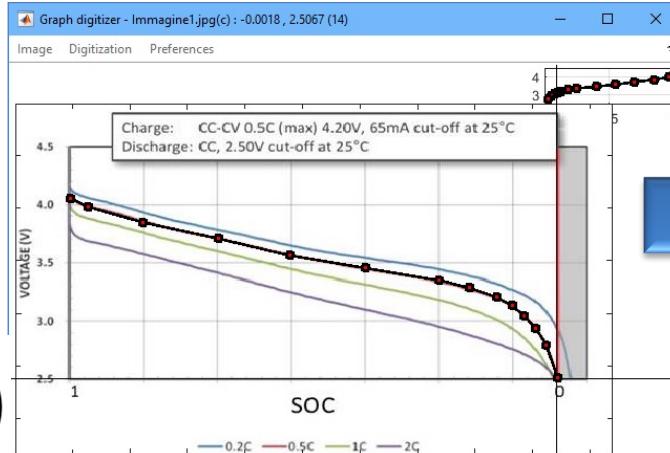
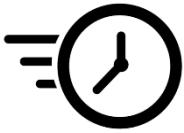
# Battery model

- Your Task:
  1. Download the datasheet figure (course website)
  2. Determine SOC axis
    - Fix the origin where the lower curve ends



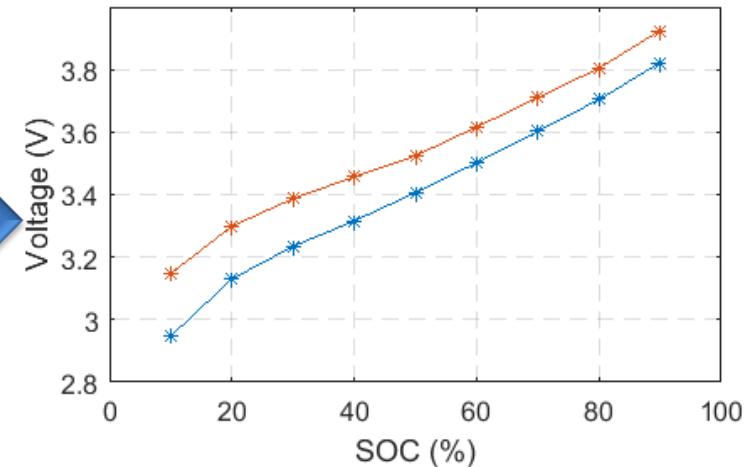
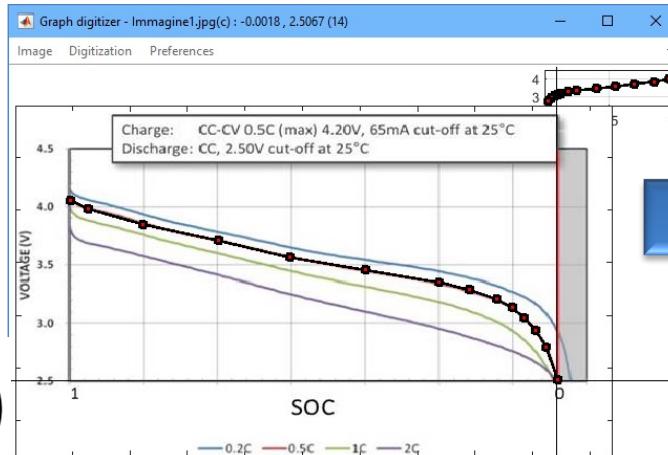
# Battery model

- Your Task:
  3. Use the digitizer to extract data from the figure
    - Extract values of two of the voltage vs. SOC curves
  4. Interpolate the two data sets
    - Given the same value of SOC, we have the corresponding value of V for both the curves



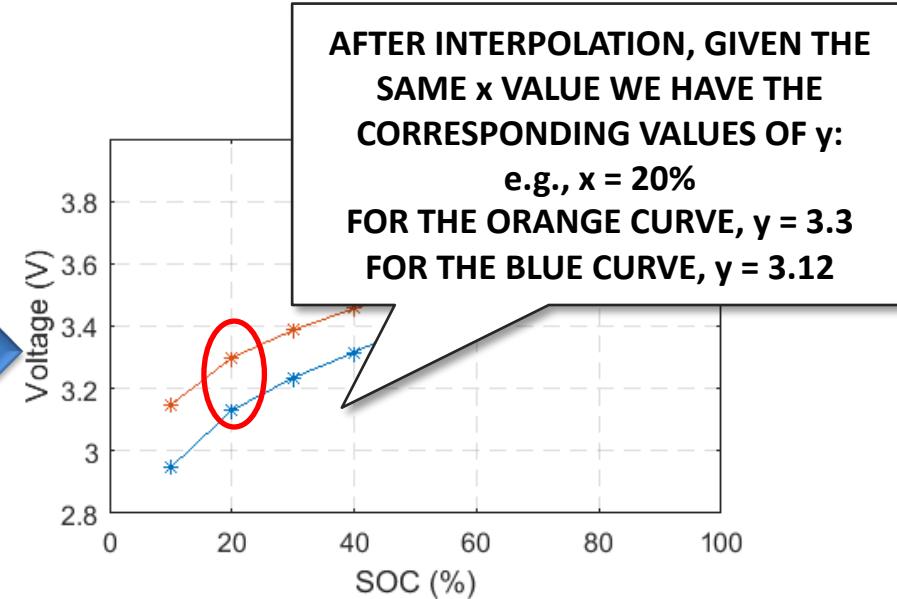
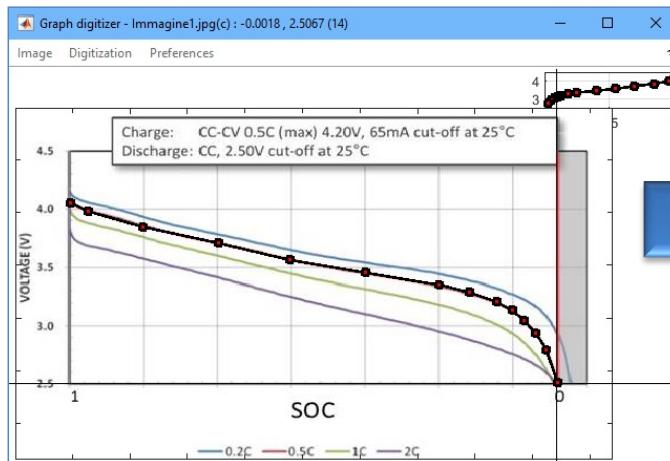
# Battery model

- Interpolation: Matlab function `interp1` ([link](#))
  - `newY = interp1(x, y, newX)`
    - Given the known samples: x values and corresponding y values
    - And the new x values `newX`
    - Calculate the corresponding y values `newY`
  - In our case:
    - x and y are the coordinates of the digitized samples
    - `newX` are samples of SOC (e.g., from 0 to 1 with step 0.1)



# Battery model

- Why do we need interpolation?
  - The samples of the two curves will most probably correspond to different values of SOC (i.e., x)
  - Thus we can not compare the two curves!
    - We have y values corresponding to different x coordinates!



# Battery model

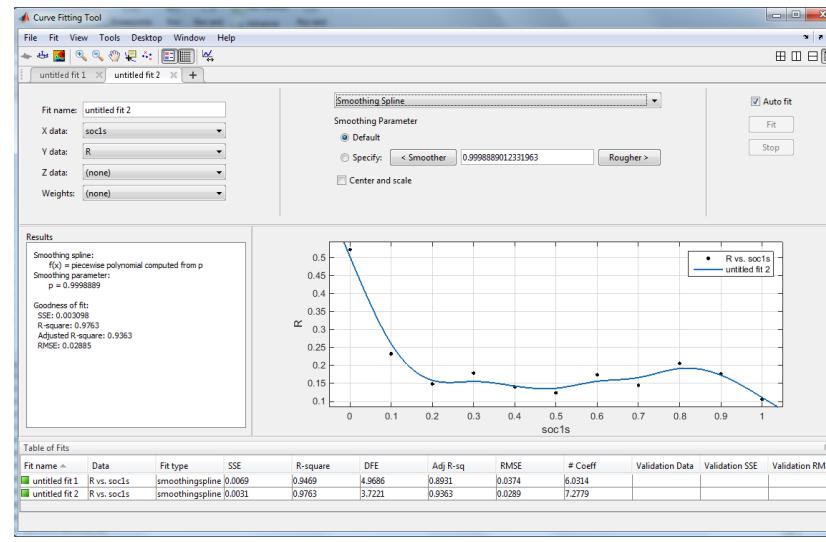
- Derive  $V_{OC}$  and  $R$ 
  - Solving the equations associated with the right hand side branch of the circuit
    - $V_{OC} = V_{curve1} + R \cdot I_{curve1}$
    - $R = \frac{V_{curve2} - V_{curve1}}{I_{curve1} - I_{curve2}}$
- Obtain values for the parameters of our battery model in *some points* of the SOC range
  - $SOC \rightarrow V \rightarrow V_{OC}, R$ 

WE KNOW FROM THE SAMPLES      WE DERIVE FROM THE EQUATIONS



# Battery model

- **Your Task:** fit the resulting data into functions by using the CurveFit app ([link](#))
  - Express  $V_{OC}$  and  $R$  as functions of  $V_{SOC}$ , i.e., of the state of charge
  - Allow to derive  $V_{OC}$  and  $R$  for any value of SOC as some function/polynomial

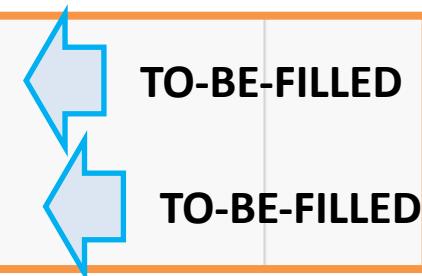


# Battery model

- This gives us all information to populate the battery model!
  - We derived the values or the equations modeling the various elements of the circuit

`src/battery_voc.cpp`

```
// SOC and battery Voc relationship  
v_oc.write(TO-BE-FILLED); // Place interpolated funct here  
  
// SOC and battery internal resistance relationship  
r_s.write(TO-BE-FILLED); // Place interpolated funct here
```



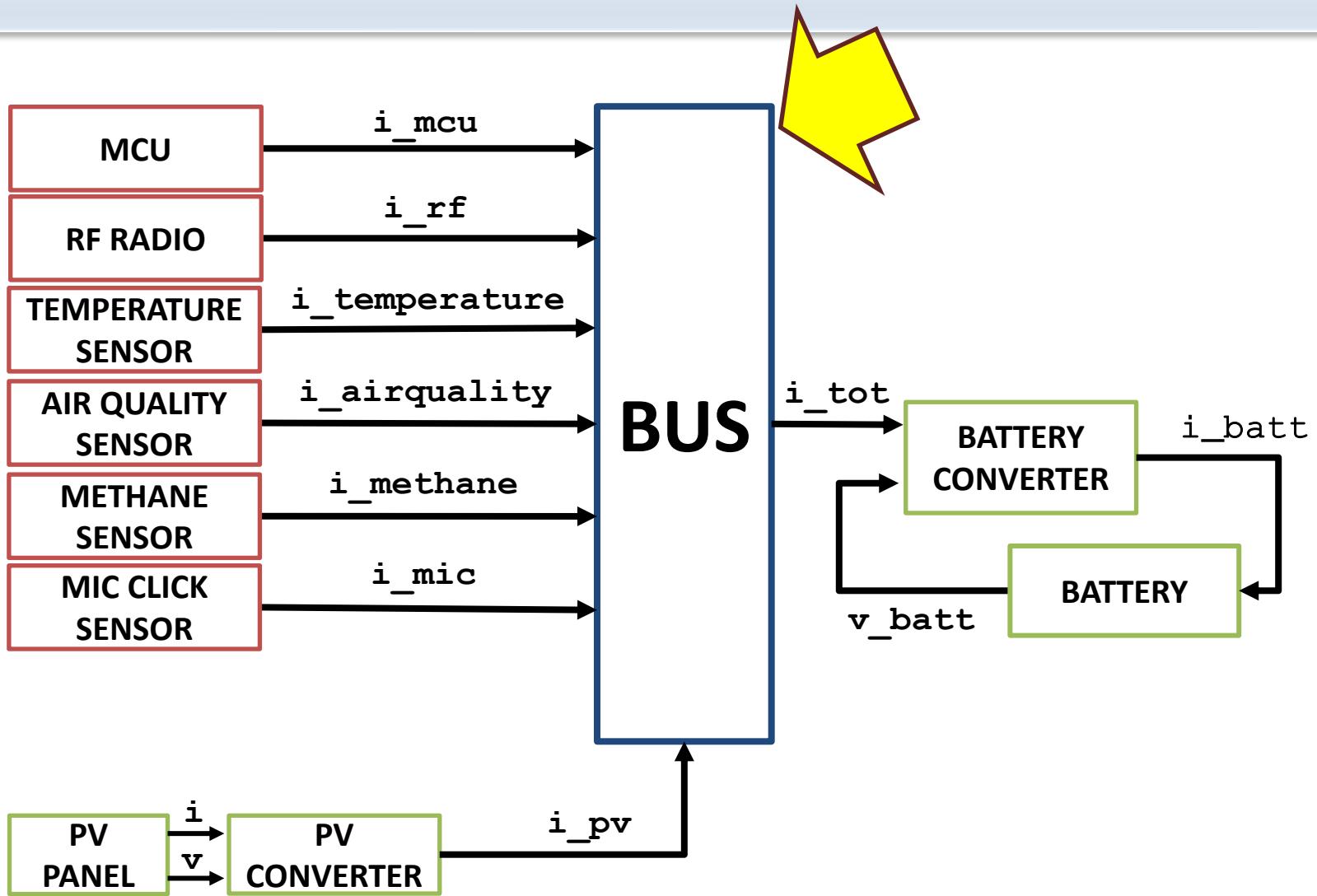
**TO-BE-FILLED**

**TO-BE-FILLED**

# **Lab 3 – Part 3**

## **Load modeling and scheduling**

# Simulator overview



# DC bus

- Two-fold goal:
  - Provide a **reference voltage** for the system (3.3V)
  - **Rule the energy flow** in the system
    - Determine what is the total power consumption of loads
    - Collect the power production of the photovoltaic module
    - Decide how to use the battery:
      - If the photovoltaic power is higher or equal to total load demand, no need to use the battery
        - » I can even charge the battery!
      - Else, estimate how much current must be drained from the battery
- **How long will my IoT system survive?**

# DC bus

- Estimate the energy flow
  1. Estimate total load power consumption
  2. Derive photovoltaic power production
  3. Calculate the difference:  $P_{LOAD} - P_{PV}$ 
    - If the difference is positive, that power must be provided by the battery (photovoltaic module is not enough!)
    - If the difference is negative, that power can be used to charge the battery (unused power!)
  4. Battery current is thus:  $\frac{P_{LOAD} - P_{PV}}{3.3V}$

# DC bus

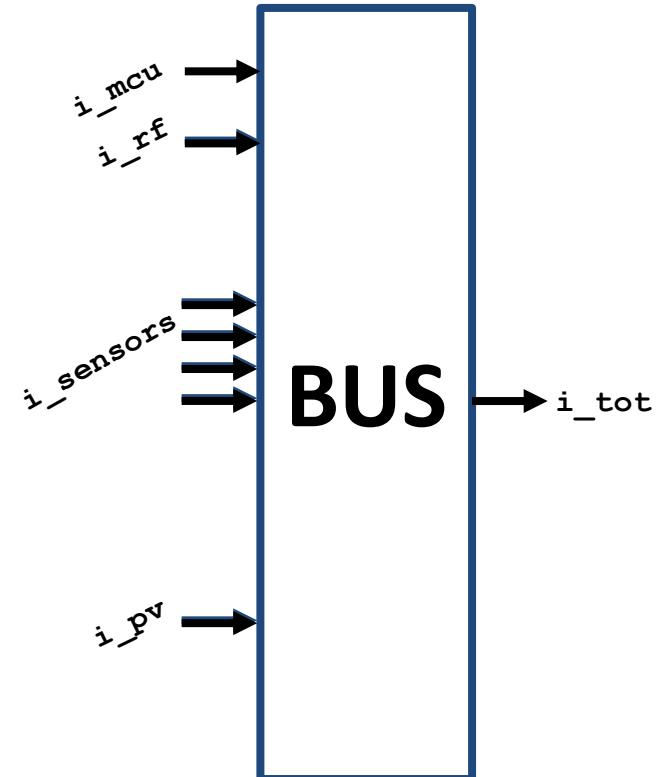
- Defined in `inc/bus.h` and implemented in `src/bus.cpp`

```
SCA_TDF_MODULE(bus)
{
    sca_tdf::sca_in<double> i_mcu; // Requested current from MCU
    sca_tdf::sca_in<double> i_rf; // Requested current from RF module
    sca_tdf::sca_in<double> i_air_quality_sensor; // Requested current from air_quality_sensor
    sca_tdf::sca_in<double> i_methane_sensor; // Requested current from methane_sensor
    sca_tdf::sca_in<double> i_temperature_sensor; // Requested current from temperature_sensor
    sca_tdf::sca_in<double> i_mic_click_sensor; // Requested current from mic_click_sensor
    sca_tdf::sca_in<double> real_i_pv; // Provided current from pv panel after conversion

    sca_tdf::sca_out<double> i_tot;

    SCACTOR(bus): i_tot("i_tot"),
                  i_mcu("i_mcu"),
                  i_rf("i_rf"),
                  i_air_quality_sensor("i_air_quality_sensor"),
                  i_methane_sensor("i_methane_sensor"),
                  i_temperature_sensor("i_temperature_sensor"),
                  i_mic_click_sensor("i_mic_click_sensor"),
                  real_i_pv("real_i_pv") {}

    void set_attributes();
    void initialize();
    void processing();
}
```



# DC bus

- Defined in `inc/bus.h` and implemented in `src/bus.cpp`

```
void bus::processing()
{
    // Compute total current consumption
    double tot_consumed = i_mcu.read() + i_rf.read()
        + i_air_quality_sensor.read()
        + i_methane_sensor.read()
        + i_temperature_sensor.read()
        + i_mic_click_sensor.read()
        ;
    double tot_scavenged = real_i_pv.read();
    double tot_requested = tot_consumed - tot_scavenged;
    i_tot.write(tot_requested); // tot_requested >= 0 ? pow_from_battery : pow_to_battery
}
```

Consumed Current

Generated Current

Requested Current

# Loads

- 6 loads
  - Temperature sensor
    - Humidity and temperature sensor
  - Methane sensor
    - CH4 sensor
  - Air quality sensor
    - Metal-oxide, H2 and Ethanol sensing
  - Mic click sensor
    - Silicon microphone
  - Memory and control unit
  - A module to transmit data over ZigBee

**MIKROE**  
be on time



AIR QUALITY  
SENSOR



MIC CLICK  
SENSOR



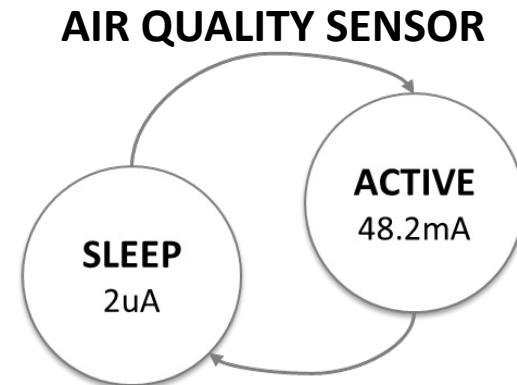
TEMPERATURE  
SENSOR



METHANE  
SENSOR

# Loads

- Loads are implemented as simple PSM
  - Fixed voltage, same as DC bus (3.3V)
  - Varying current
    - One active state (higher consumption)
    - One sleep state (lower consumption)
- Data taken from load datasheets
  - Typical current consumption depending on activity



LOAD	ACTIVE (mA)	SLEEP (mA)
Air quality sensor	48.2	0.002
Methane sensor	18	0.002
Temperature sensor	3	0.002
Mic click sensor	0.15	0.002
ZigBee transmission	0.1	0.001
Memory and control	13	0.002

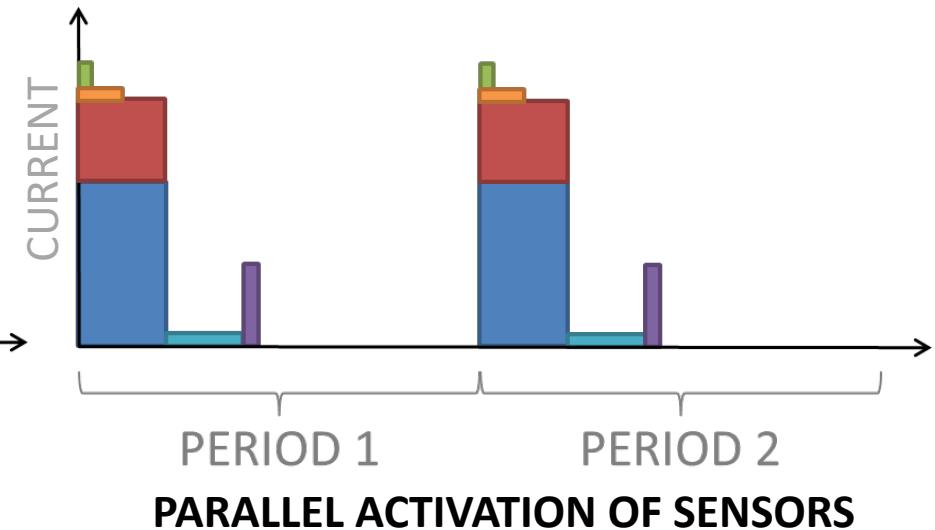
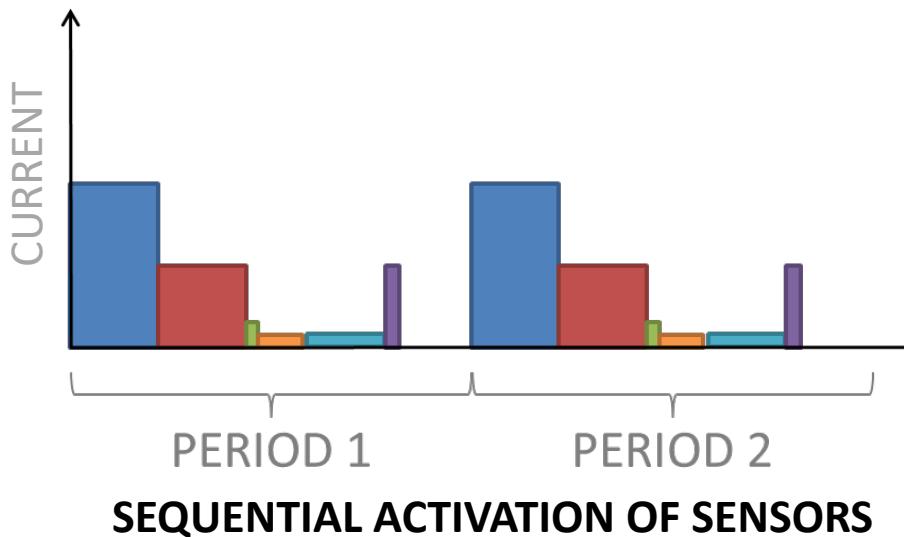
# Loads

- Loads behavior is periodic
  - Active time is fixed for each load
    - Depends on the requirements of the component
    - E.g., how long it takes for the sensor to make the measurements
    - Given as input specification

LOAD	TIME (s)
Air quality sensor	30
Methane sensor	30
Temperature sensor	6
Mic click sensor	12
ZigBee transmission	24
Memory and control	6

# Loads

- Loads behavior is periodic
  - Load activation depends on a schedule
    - Fixed order: first sensors, then memory and control, then transmission
    - Sensors can be activated sequentially or in parallel



# Loads

- Who controls load activation?
  - Defined in the second half of **sim\_setting/parallel.json**

```
"sensors" : [  
    {  
        "name": "air_quality_sensor",  
        "current_on": "48.2",  
        "current_idle": "0.002",  
        "activation_time": "0",  
        "time_on": "30"  
    },  
    {  
        "name": "methane_sensor",  
        "current_on": "18",  
        "current_idle": "0.002",  
        "activation_time": "0",  
        "time_on": "30"  
    },  
    ...  
]
```

Same activation time  
→ Parallel Execution

N.B., Current values  
are in mA !!

# Loads

- Who controls load activation?
  - Defined under `sim_setting/parallel.json`

```
{"mcu" : {  
    "states": [  
        {  
            "name": "ON",  
            "current": "13",  
            "time_on": "6"  
        }  
    ],  
    "current_idle": "0.002"  
},  
    "rf" : {  
        "states": [  
            {  
                "name": "ON",  
                "current": "0.1",  
                "time_on": "24"  
            }  
        ],  
        "current_idle": "0.001"  
    }  
}
```

Activated after  
sensors

```
"rf" : {  
    "states": [  
        {  
            "name": "ON",  
            "current": "0.1",  
            "time_on": "24"  
        }  
    ],  
    "current_idle": "0.001"  
}
```

Activated after MCU

# Simulation settings

- Who controls load activation?
  - Defined in the first half of  
**sim\_setting/parallel.json**
- Simulation Step: 1 second

```
{  
  "sim_step" : 1,  
  "sim_len": 7736400,  
  "period" : 120,  
  "vref_bus" : 3.3,  
  "soc_init" : 1.0,  
  "selfdisch_factor" : 0.0,  
  "sensors" : [
```

Length of  
Gmonth.txt file

# Simulation settings

- To run the simulation, in a terminal launch the bash script:
  - \$ source simulate.sh sim\_setting/parallel.json

```
#!/bin/bash

cd simulator

# Python codegen 1
cd codegen
sim_setup_path=$1
python codegen.py -f ${sim_setup_path}
cd ..

# Compile 2
make clean
make

# launch the simulation 3
.bin/run.x
```

**1 Generate simulation source code using a template-based python script directly from the sim\_setting/parallel.json file.**

**2 Compile.**

**3 Run simulation.**

# Simulation output

- The simulation outputs are saved in the `sim_trace.txt` file.

```
1 %time soc i_tot i_mcu i_rf i_pv v_pv real_i_pv i_batt v_batt i_air_quality_sensor i_methane_sensor i_temperature_sensor i_mic_click_sensor
2 0 0 66.653 0.002 0.001 0 0 0 0 4.1138 48.2 18 0.3 0.15
3 1 1 66.653 0.002 0.001 0 0 0 58.8435823726 4.11379642475 48.2 18 0.3 0.15
4 2 0.999997446025 66.653 0.002 0.001 0 0 0 58.8436335128 4.11379126317 48.2 18 0.3 0.15
5 3 0.999992338073 66.653 0.002 0.001 0 0 0 58.8437073439 4.1137861016 48.2 18 0.3 0.15
6 4 0.999987230115 66.653 0.002 0.001 0 0 0 58.8437811753 4.11378094003 48.2 18 0.3 0.15
7 5 0.999982122152 66.653 0.002 0.001 0 0 0 58.8438550067 4.11377577846 48.2 18 0.3 0.15
8 6 0.999977014181 66.355 0.002 0.001 0 0 0 58.8439288383 4.1137706169 48.2 18 0.002 0.15
9 7 0.999971906205 66.355 0.002 0.001 0 0 0 58.5903200911 4.11376547075 48.2 18 0.002 0.15
10 8 0.999966809232 66.355 0.002 0.001 0 0 0 58.5903933851 4.11376033145 48.2 18 0.002 0.15
11 9 0.999961723263 66.355 0.002 0.001 0 0 0 58.5904665818 4.11375519215 48.2 18 0.002 0.15
12 10 0.999956637289 66.355 0.002 0.001 0 0 0 58.5905397786 4.11375005286 48.2 18 0.002 0.15
13 11 0.999951551307 66.355 0.002 0.001 0 0 0 58.5906129756 4.11374491357 48.2 18 0.002 0.15
14 12 0.99994646532 66.207 0.002 0.001 0 0 0 58.5906861727 4.11373977429 48.2 18 0.002 0.002
15 13 0.999941379326 66.207 0.002 0.001 0 0 0 58.4647381469 4.11373464266 48.2 18 0.002 0.002
16 14 0.999936298795 66.207 0.002 0.001 0 0 0 58.464811078 4.11372951444 48.2 18 0.002 0.002
17 15 0.999931223728 66.207 0.002 0.001 0 0 0 58.4648839609 4.11372438622 48.2 18 0.002 0.002
18 16 0.999926148655 66.207 0.002 0.001 0 0 0 58.4649568439 4.11371925801 48.2 18 0.002 0.002
19 17 0.999921073575 66.207 0.002 0.001 0 0 0 58.4650297271 4.1137141298 48.2 18 0.002 0.002
20 18 0.999915998489 66.207 0.002 0.001 0 0 0 58.4651026103 4.11370900159 48.2 18 0.002 0.002
21 19 0.999910923396 66.207 0.002 0.001 0 0 0 58.4651754937 4.11370387339 48.2 18 0.002 0.002
```

simulation  
timestamp

traced signals

# Simulation output

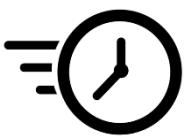
- How to control which signals are traced?
  - Refer to the `src/main.cpp` file:

```
// the following signals will be traced. Comment any signal you don't want to trace
sca_util::sca_trace(atf, soc, "soc" );
sca_util::sca_trace(atf, i_tot, "i_tot" );
sca_util::sca_trace(atf, i_mcu, "i_mcu" );
sca_util::sca_trace(atf, i_rf, "i_rf" );
sca_util::sca_trace(atf, i_pv, "i_pv" );
sca_util::sca_trace(atf, v_pv, "v_pv" );
sca_util::sca_trace(atf, real_i_pv, "real_i_pv" );
sca_util::sca_trace(atf, i_batt, "i_batt" );
sca_util::sca_trace(atf, v_batt, "v_batt" );
sca_util::sca_trace(atf, i_air_quality_sensor, "i_air_quality_sensor" );
sca_util::sca_trace(atf, i_methane_sensor, "i_methane_sensor" );
sca_util::sca_trace(atf, i_temperature_sensor, "i_temperature_sensor" );
sca_util::sca_trace(atf, i_mic_click_sensor, "i_mic_click_sensor" );
```

N.B., if you comment some lines DO NOT perform the “Python codegen” step of `simulate.sh` script, but perform directly compilation and the run simulation.

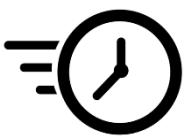
# First analysis of the Lab:

- In the report, no need to comment on the construction of the components
- **Discuss the main features of the simulation** given the provided schedule (parallel sensors)
  1. Exhibit one example trace of the simulation, showing meaningful quantities
    - E.g., loads power, photovoltaic power, battery power, and battery SOC to show that the system behaves correctly
  2. Efficiency of the converters, to see if we fall in a good area (high efficiency) or not (low efficiency = a lot of wasted power)
  3. How often the battery has to be used to supply power
    - I.e., photovoltaic power is not enough



# Second analysis of the Lab:

- Determine the best scheduling
    - Allows longer lifetime of the system
    - Compare:
      - Different scheduling
        - All sensors in parallel
        - Sequential activation of the sensor
      - Does the scheduling impacts on the evolution of battery lifetime?
      - If no impact, why???
- 
- You need to develop a new json file in sim\_setting



# Third analysis of the Lab:

- Propose solutions to increase the lifetime of the IoT system
  - Should be autonomous
  - Should not require frequent battery changes

IoT Wireless Sensors  
and the Problem of

## Short Battery Life

By Carlo Canziani, Keysight Technologies, Inc.

Wireless sensors provide great insight in applications like monitoring environmental conditions or industrial plants and machinery. Because they are simple to install, they can be deployed in a multitude of situations. In coming years, we will see an explosion of new uses for wireless sensors as the "Internet of Things," or "IoT," is widely deployed. But one of the factors that most limits the use of wireless sensors is their limited ability to do the job for a reasonable amount of time. When a wireless sensor's operation is fully dependent on a battery, and the battery is depleted, it becomes just a piece of junk.

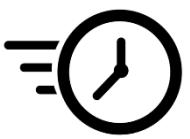
WHEN A WIRELESS SENSOR'S OPERATION IS FULLY DEPENDENT ON A BATTERY,  
AND THE BATTERY IS DEPLETED, IT BECOMES JUST A PIECE OF JUNK.

!!!



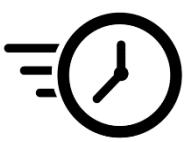
# Third analysis of the Lab:

- Possible solutions:
  1. Increase **photovoltaic power production**
    - E.g., one more identical photovoltaic module connected in series/in parallel
  2. Increase **battery capacity**
    - E.g., one more identical battery connected in series/parallel
    - Consider that adding one battery or one PV module implies a different value of current/voltage depending on the connection
      - DC-DC converter efficiency may change!
  3. ... Any other idea that comes to your mind!



# Third analysis of the Lab:

- Bear in mind costs in your exploration:
  - Cost of one battery: \$4.99
  - Cost of one PV module: \$5.50
  - E.g., maximum additional cost: \$11.00
- Compare possible solutions and the resulting lifetime



# Things to bear in mind:

- Efficiency is a number between 0 and 1 for all converters
  - 0 = 0%, 1 = 100%
- When digitizing, take care of axis prefix (e.g., mA = milli Ampere) – fix the digitization to the correct scale!
  - In the simulation, 1 = 1mA, 1V,...
- Don't forget to set battery capacity