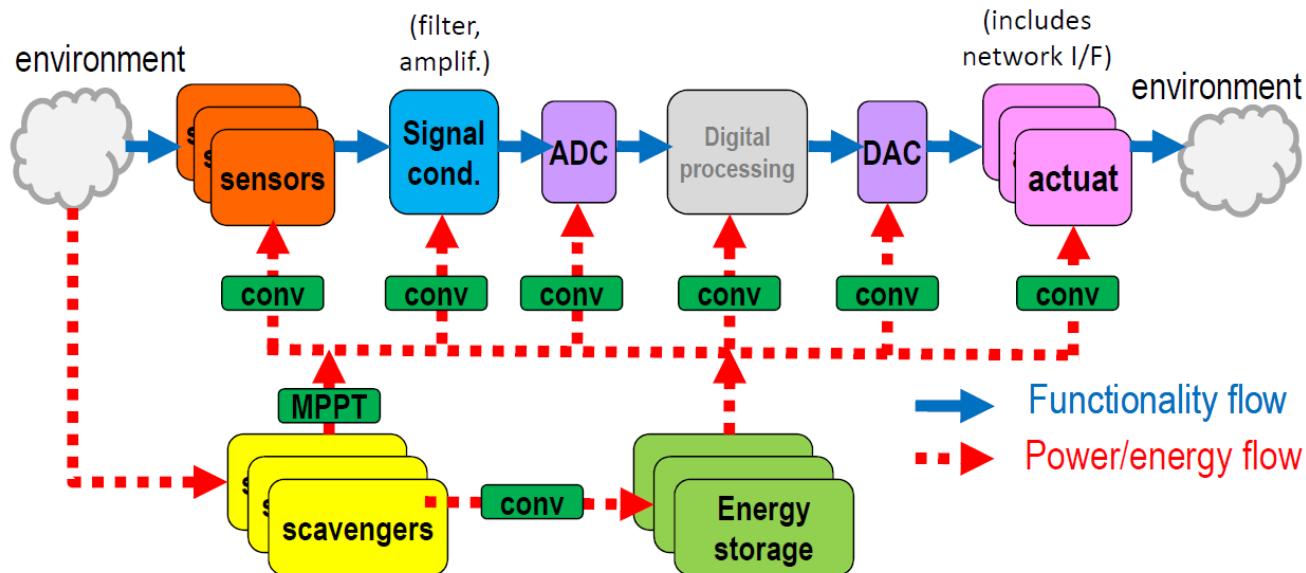


# **Lab 3**

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

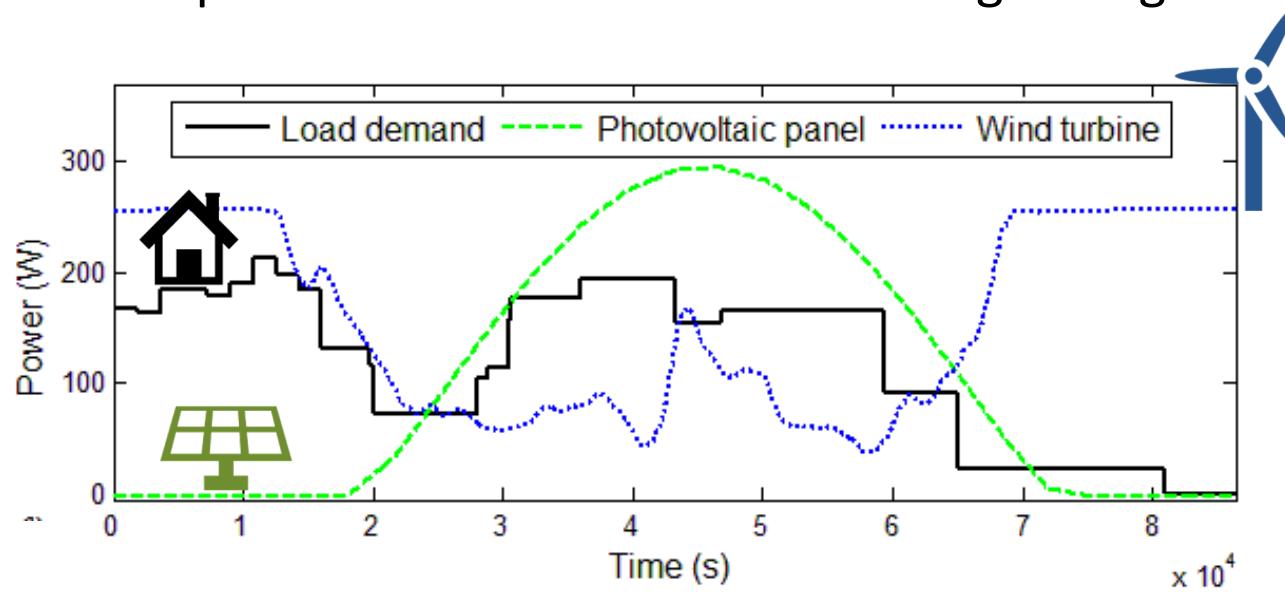
# Energy storage, generation and conversion

- 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...

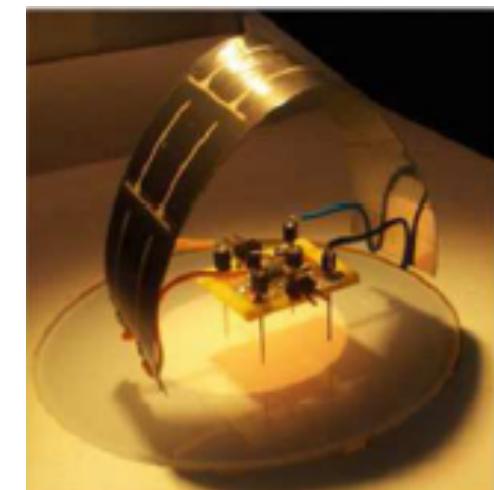


# Energy storage, generation and conversion

- 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
  - A module to transmit data over ZigBee
  - A battery with a DC-DC converter
  - A thin-film photovoltaic module operated at the MPP with a DC-DC converter



# Objective and organization

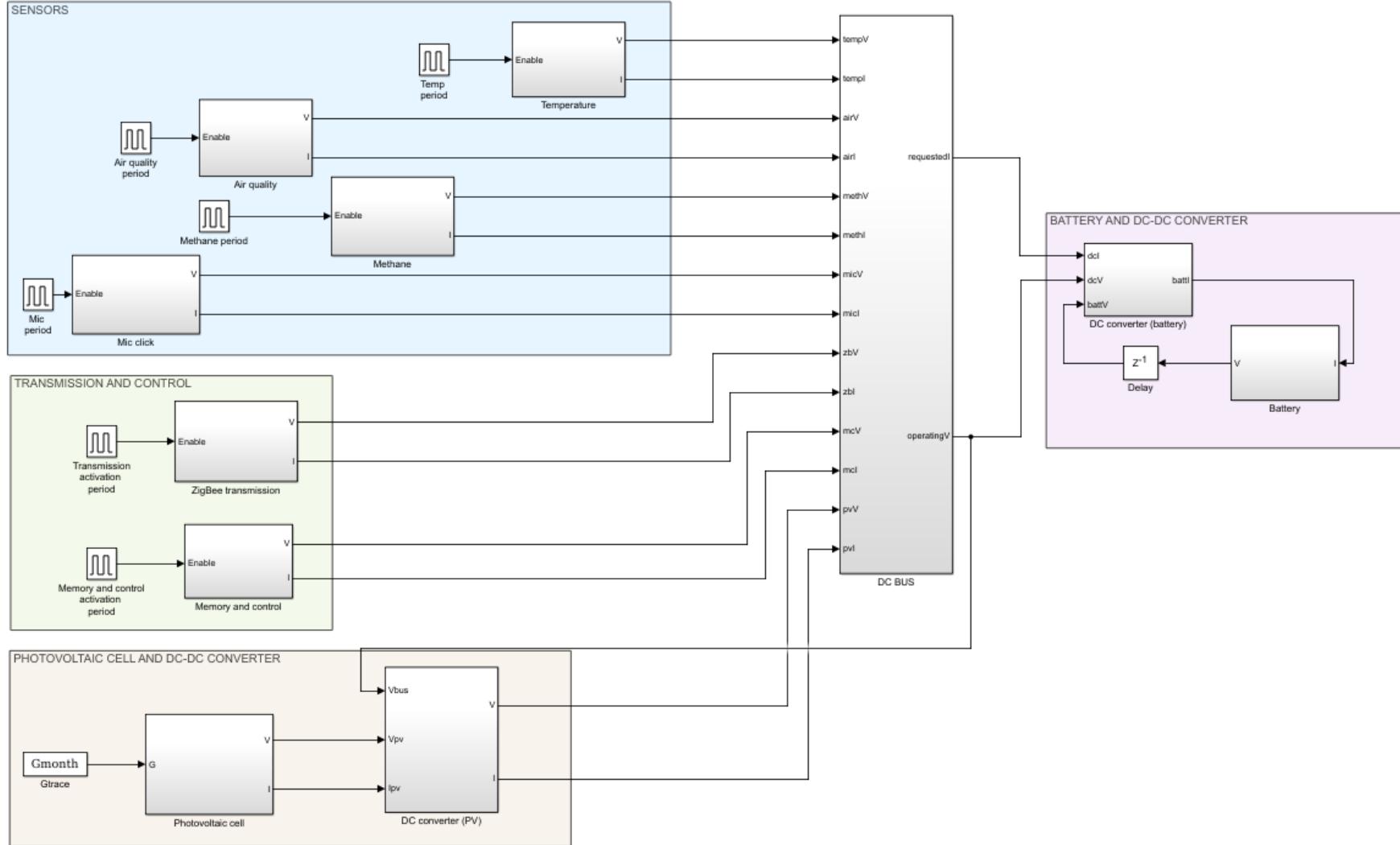
- Goal of this lab is to simulate an IoT device:
  - Implemented in Matlab/Simulink
    - Standard-de-facto for energy simulation
    - Predefined simulation skeleton
    - To be populated with models for the components
  - Populate the system, analyse and optimize its behavior

# Objective and organization

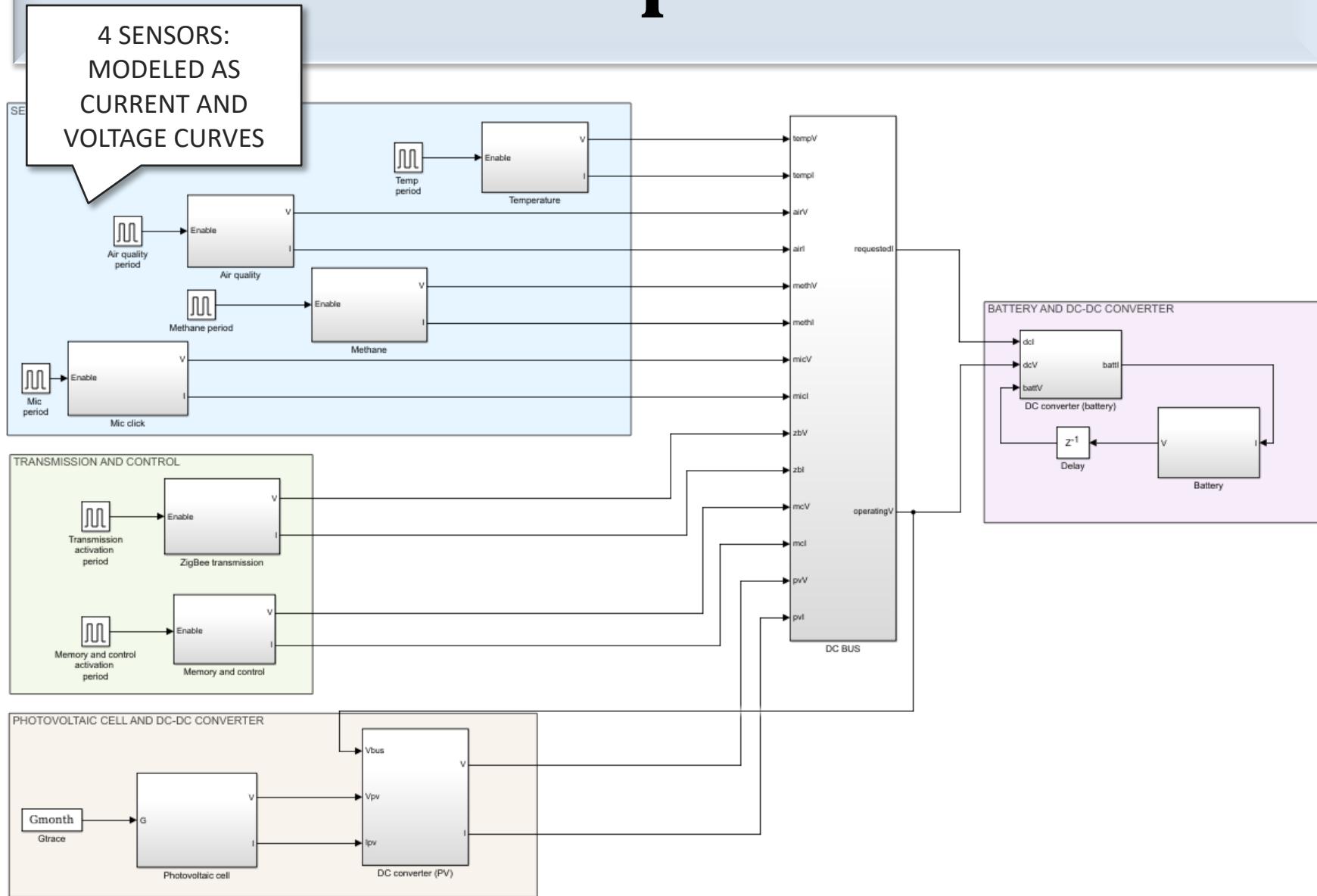
- Organization
  - 1 assignment to deliver
  - 2 days (+ 1 optional session)
    - *Jan 11th 4pm-7pm*
    - *Jan 12th 1pm-2:30pm*
  - Required software: Matlab, Simulink

# **Simulink implementation**

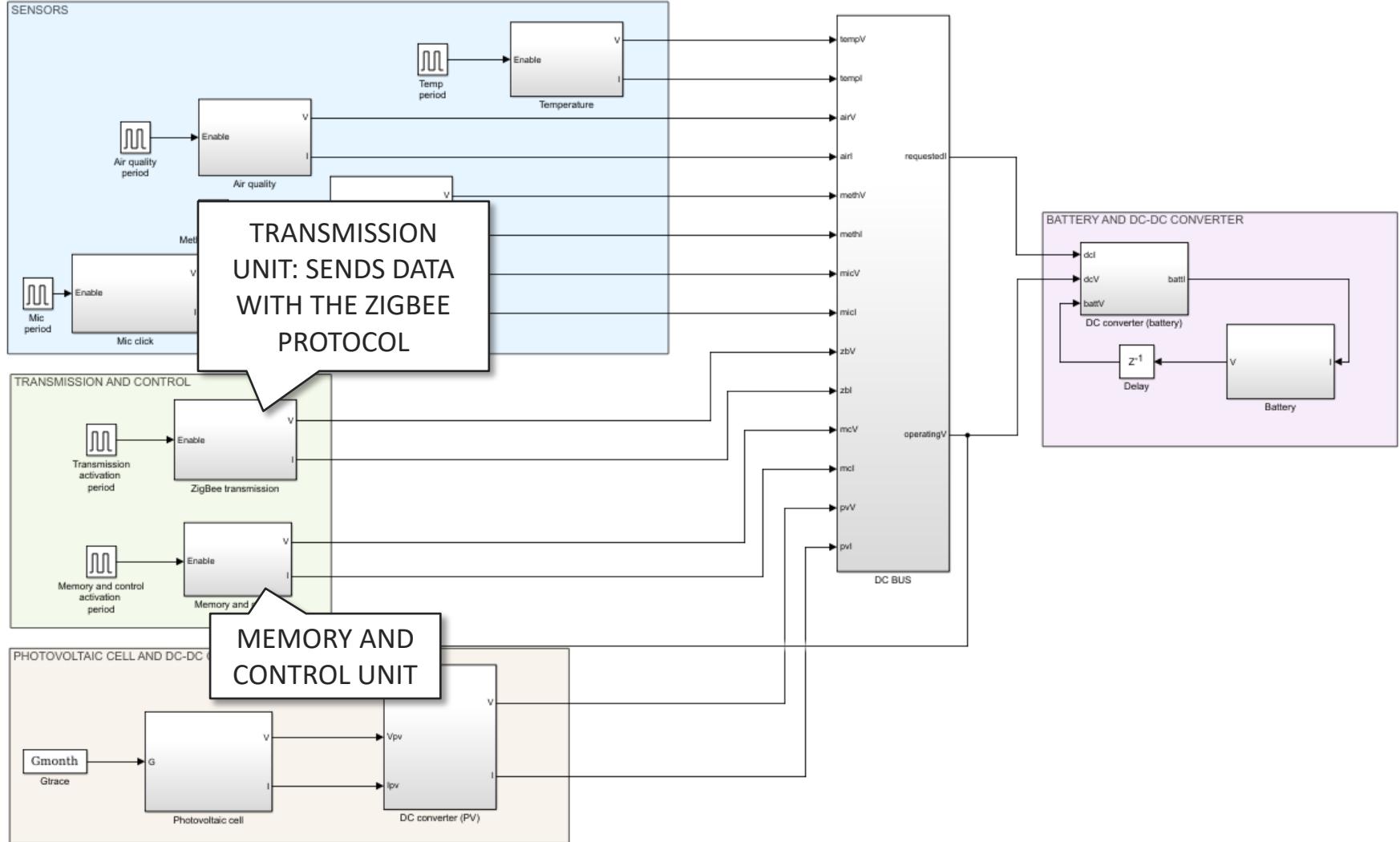
# Simulink implementation



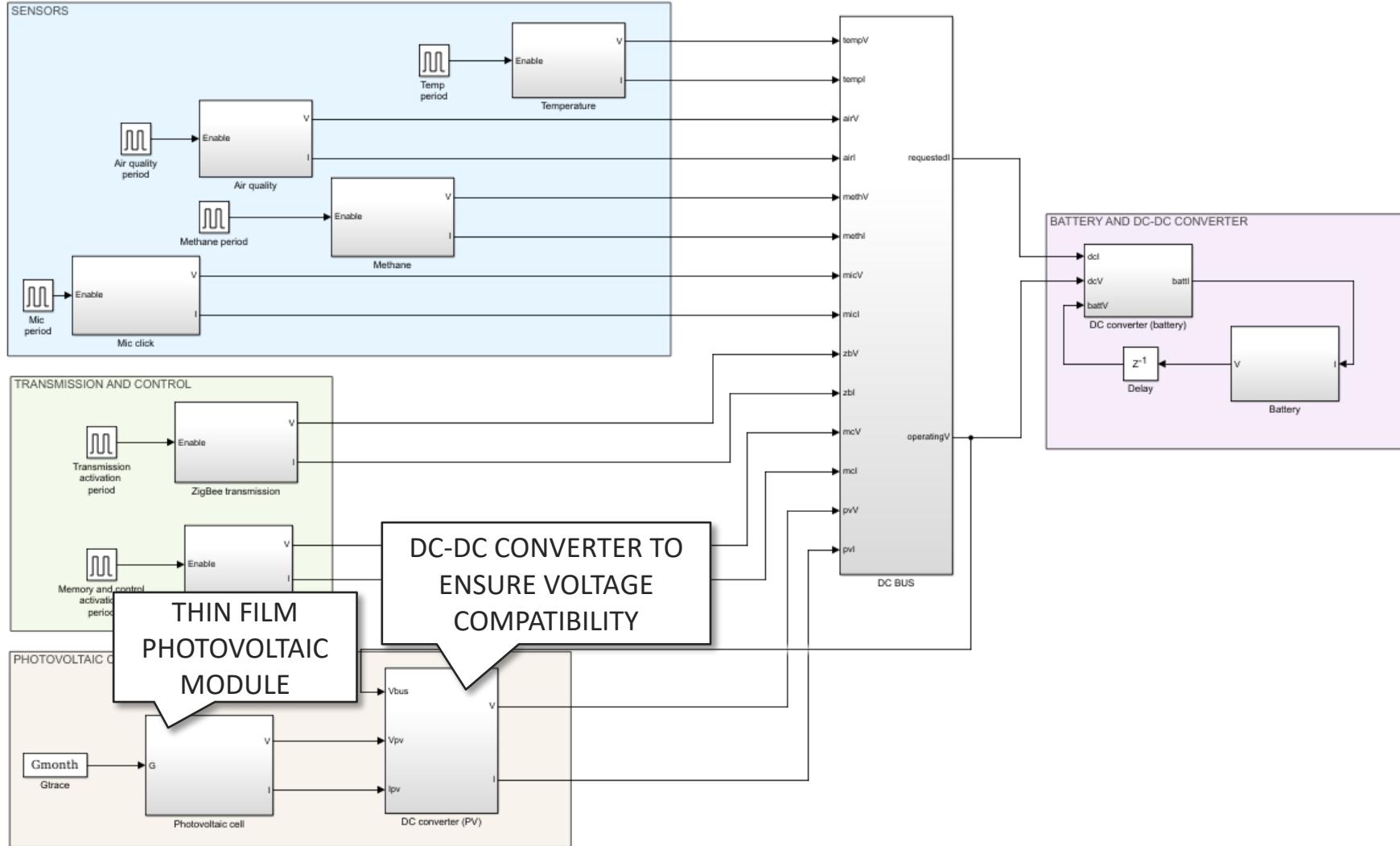
# Simulink implementation



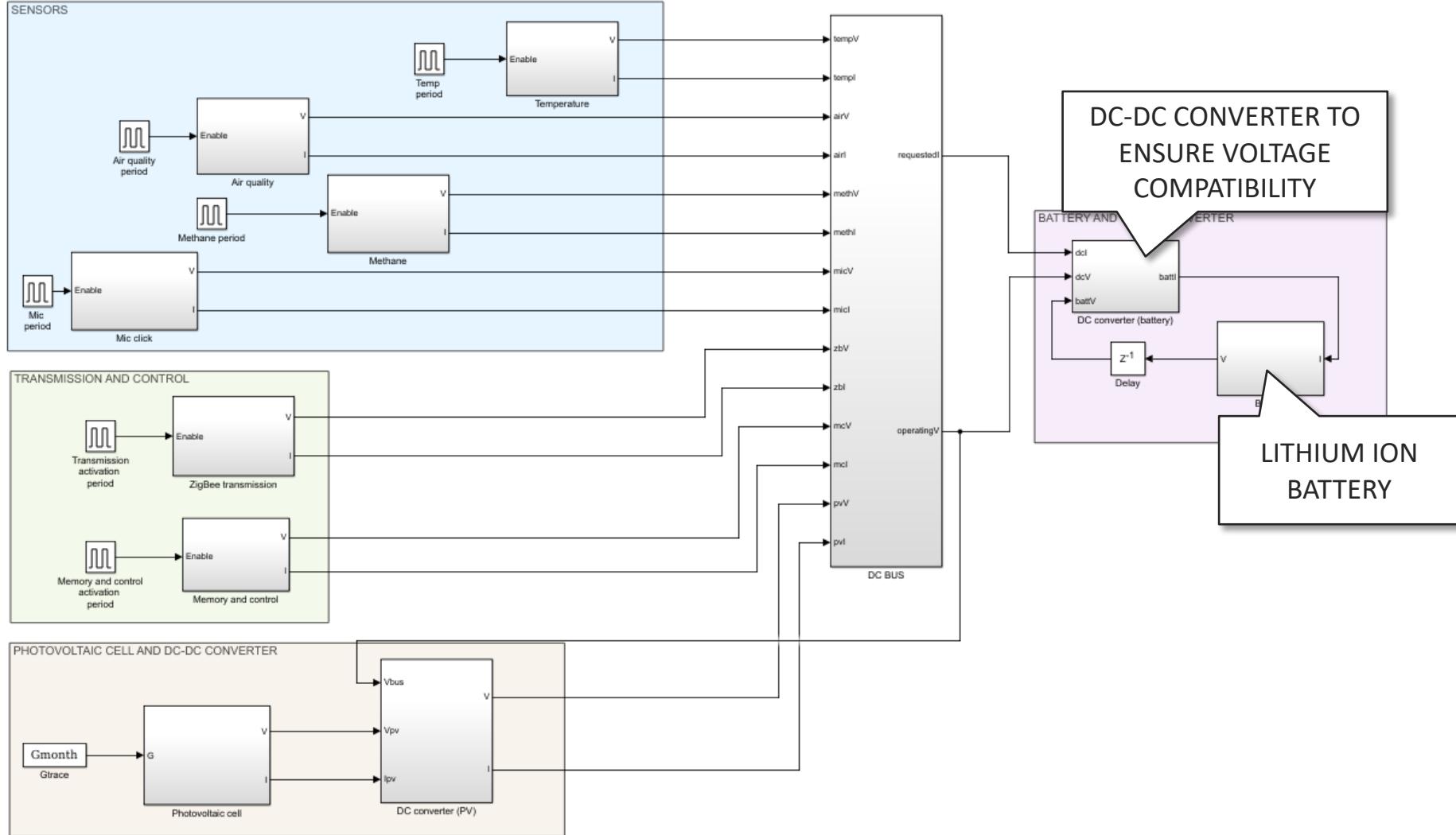
# Simulink implementation



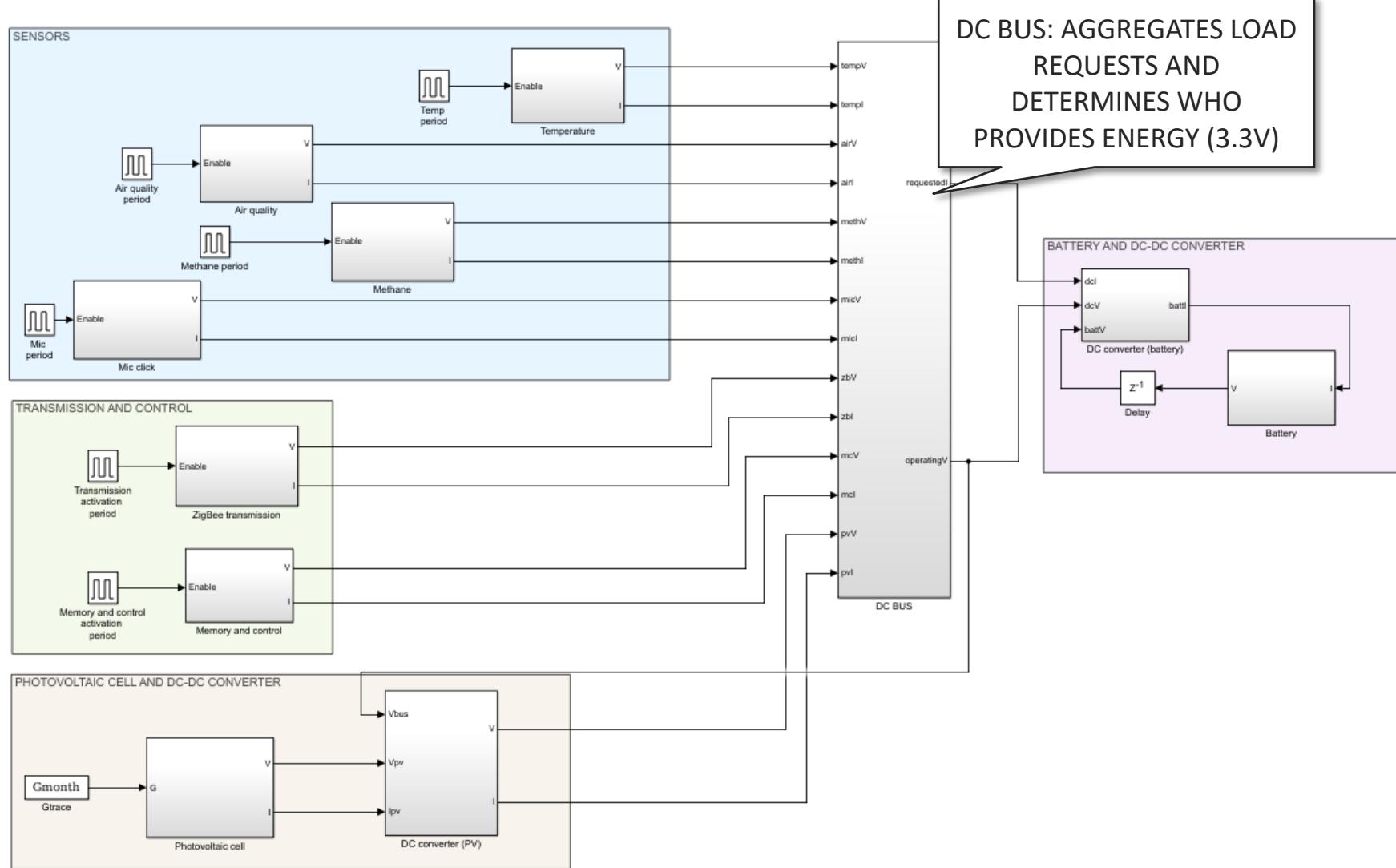
# Simulink implementation



# Simulink implementation



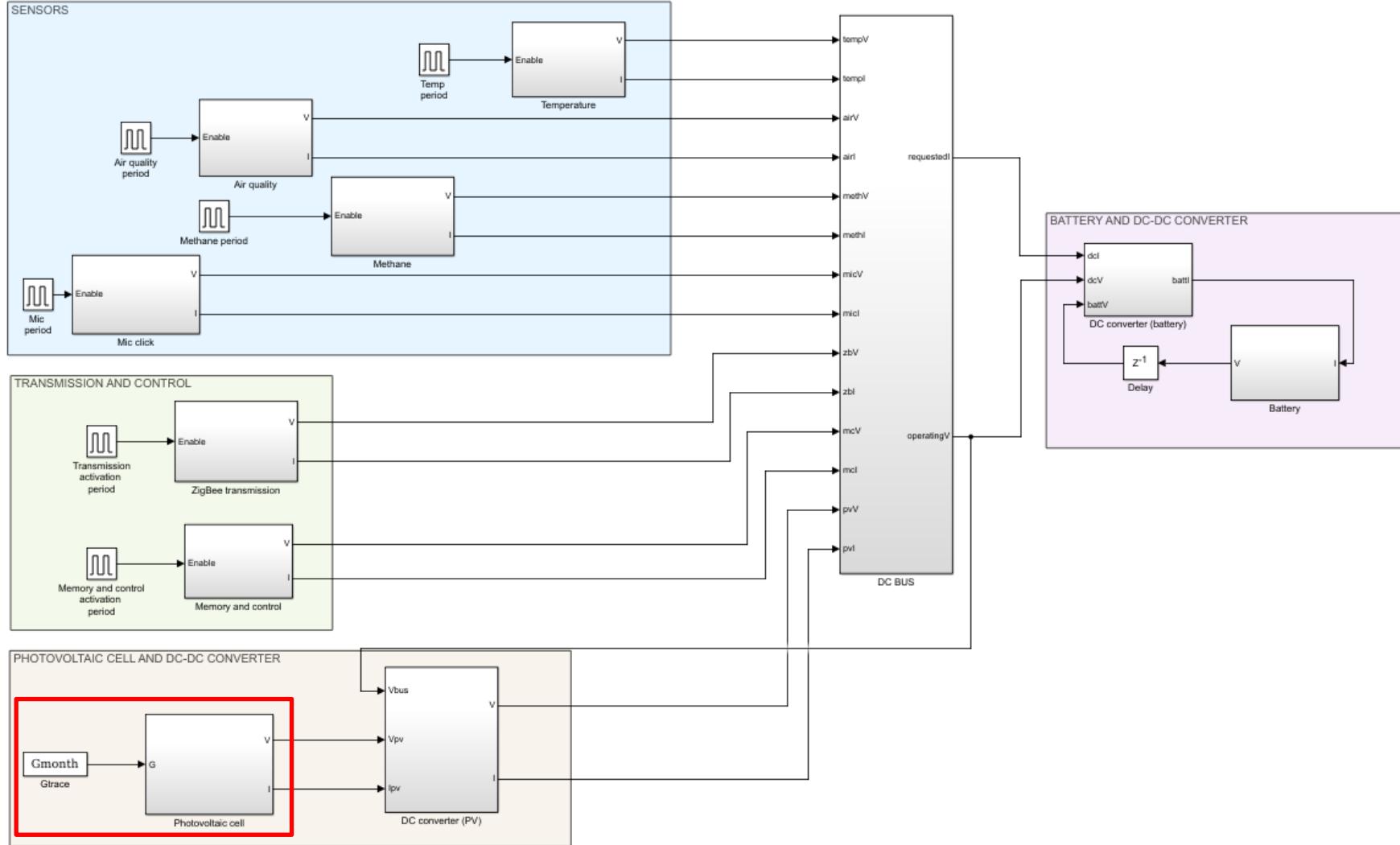
# Simulink implementation



# **Lab 3 – Part 1**

## **Model of the photovoltaic module**

# Simulink implementation



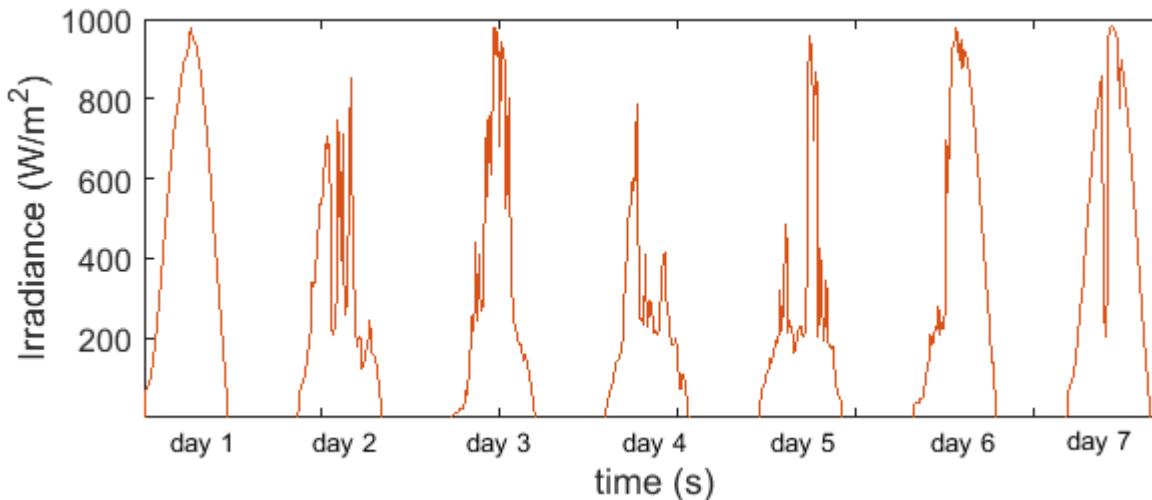
# Photovoltaic module

- Gmonth
  - Variable loaded from workspace
    - To load it, type:
      - `load ('gmonths.mat')`
    - 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

	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

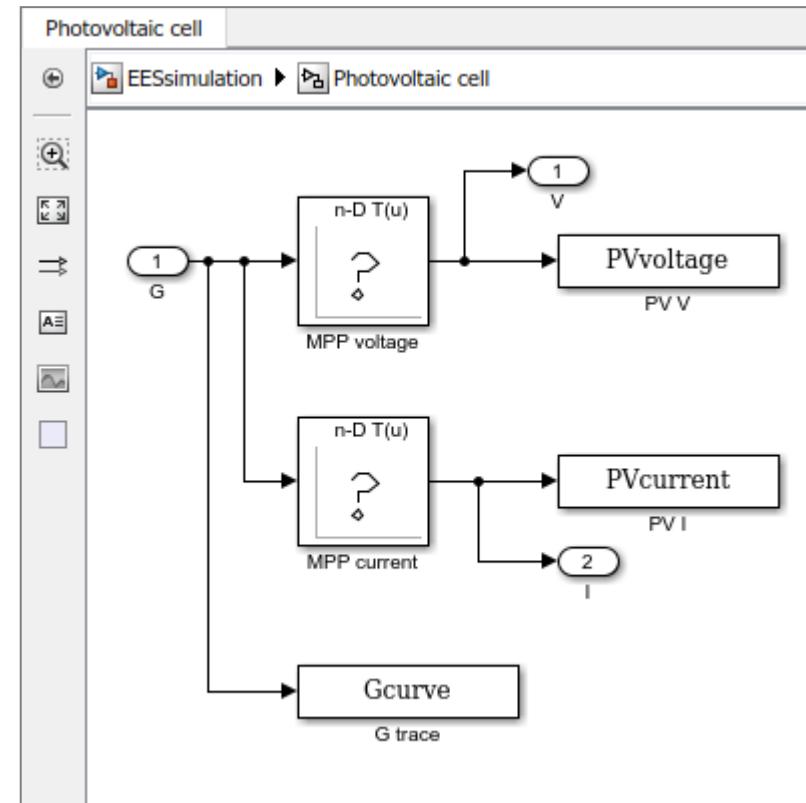
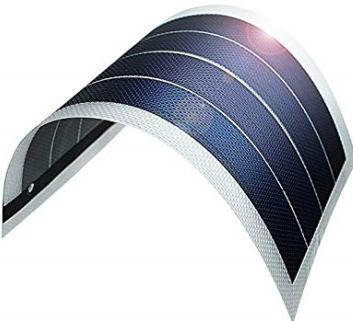
- Gmonth
  - Irradiance trace for about 3 months
    - Irradiance varies depending on weather (sunny/rainy/cloudy)
    - E.g.: figure  
`plot (Gmonth (:, 1), Gmonth (:, 2))`



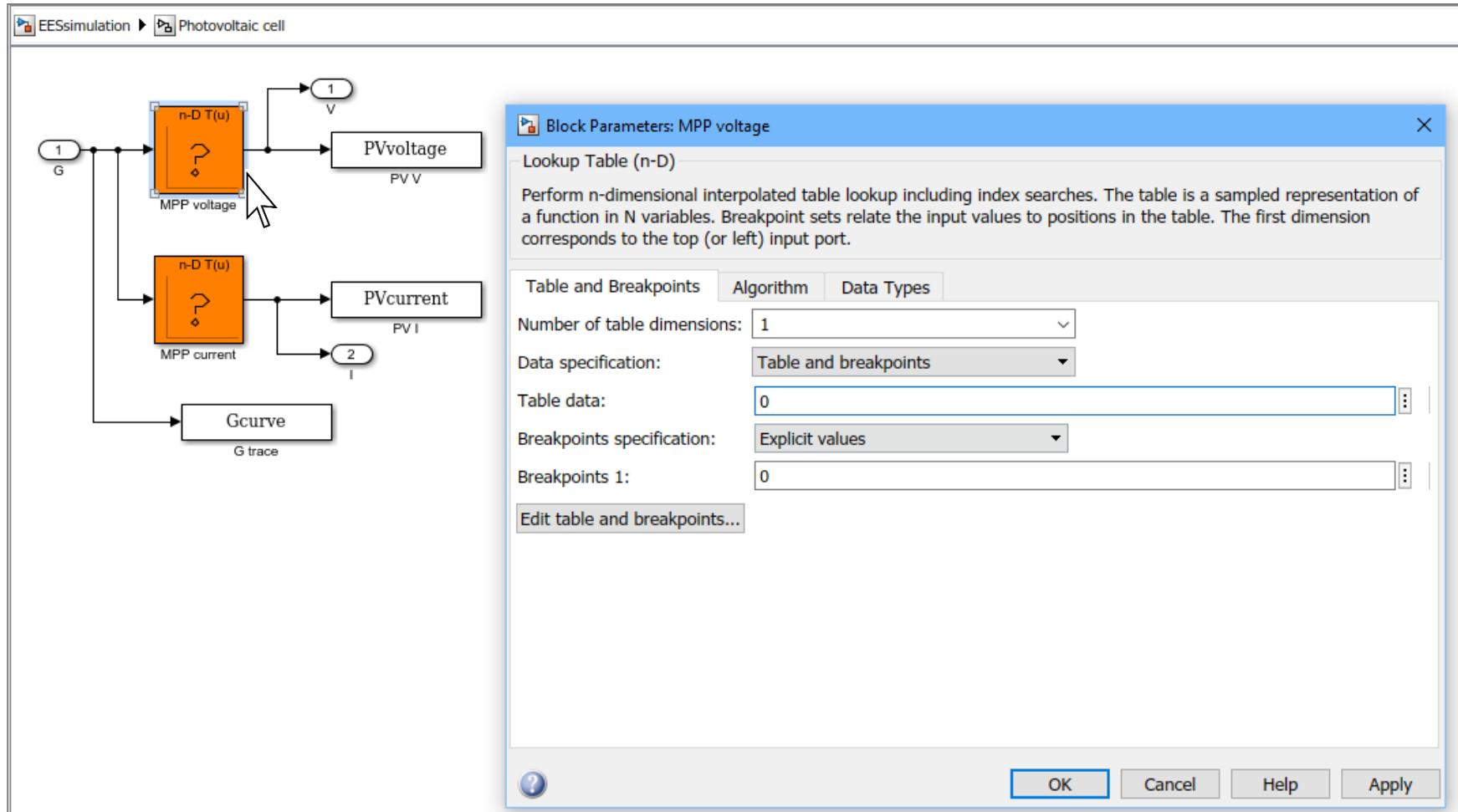
Editor - panel.m		
Gmonth		
1590x2 double		
1	0	0
2	900	67.6853
3	1800	77.7214
4	2700	87.3263
5	3600	100.5830
6	4500	124.0773
7	5400	155.2077
8	6300	191.0537
9	7200	233.1320
10	8100	278.4866
11	9000	324.2709
12	9900	370.6427
13	10800	422.2269
14	11700	469.9433
15	12600	514.6853
16	13500	563.9495
17	14400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

# Photovoltaic module

- Photovoltaic module
  - Open the subsystem by double-clicking on block
  - Two lookup tables
    - Voltage and current at the MPP given irradiance in input
    - Must populate the lookup tables!

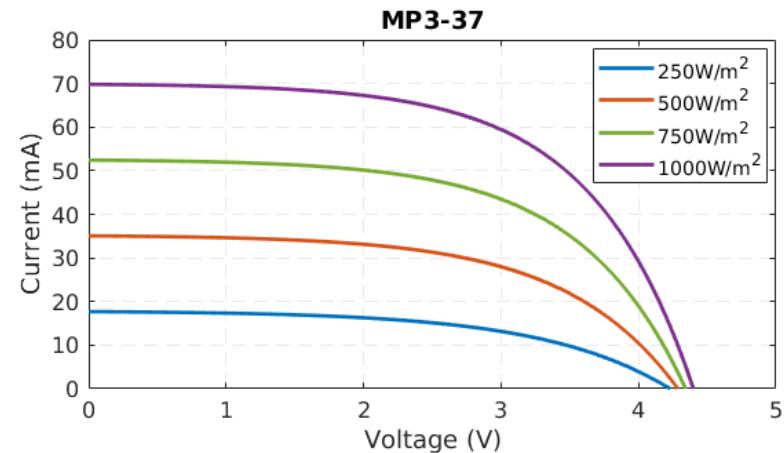


# Photovoltaic module



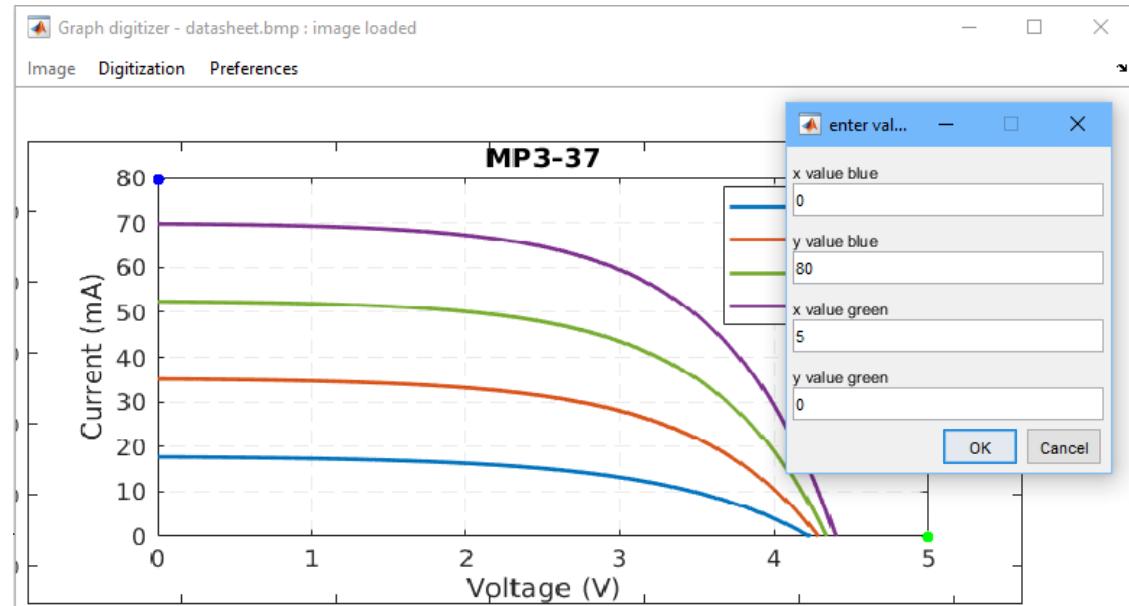
# Photovoltaic module

- Open the PVdatasheet.pdf file
- Provides a graph of current versus voltage given different irradiance values
- How to:
  - 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



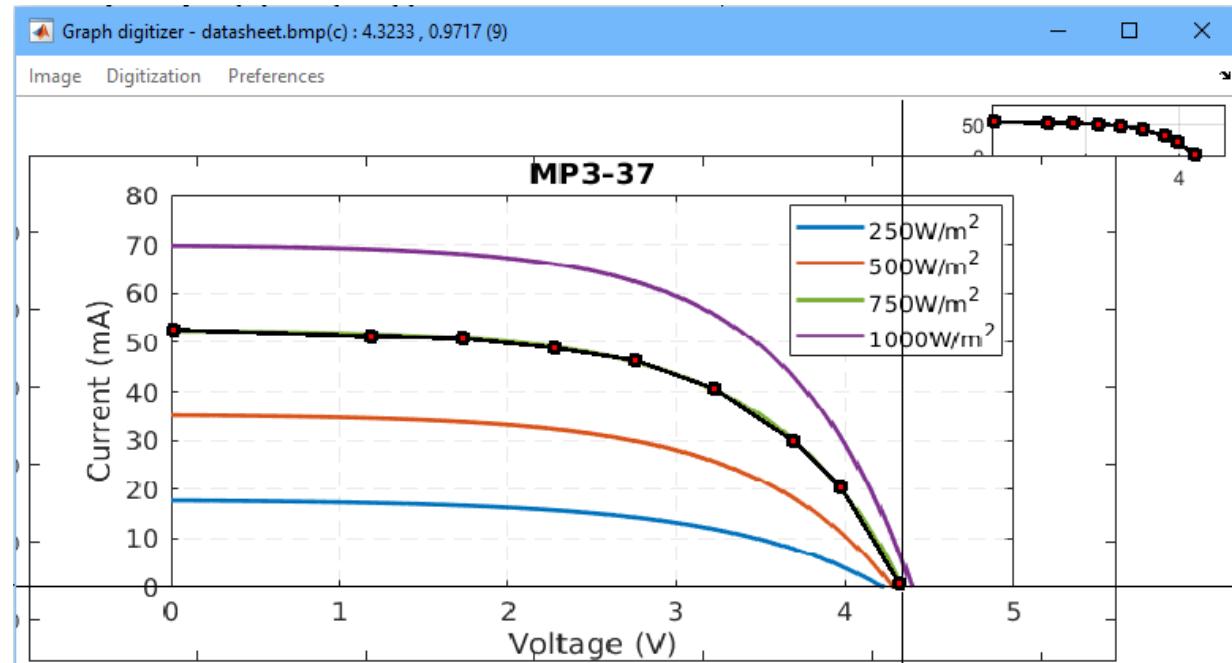
# Photovoltaic module

- How to use the digitizer 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

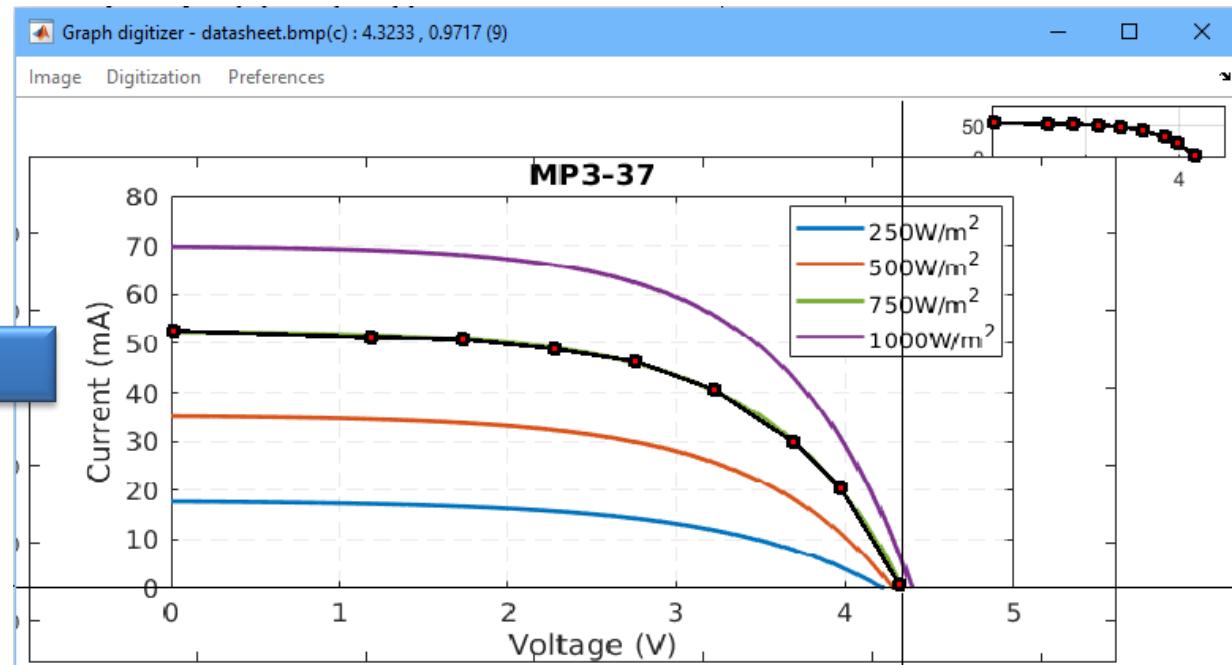
- How to use the digitizer tool:
  3. Select samples of one curve
    - Mouse left button = new sample
    - Mouse middle button = stop inserting samples



# Photovoltaic module

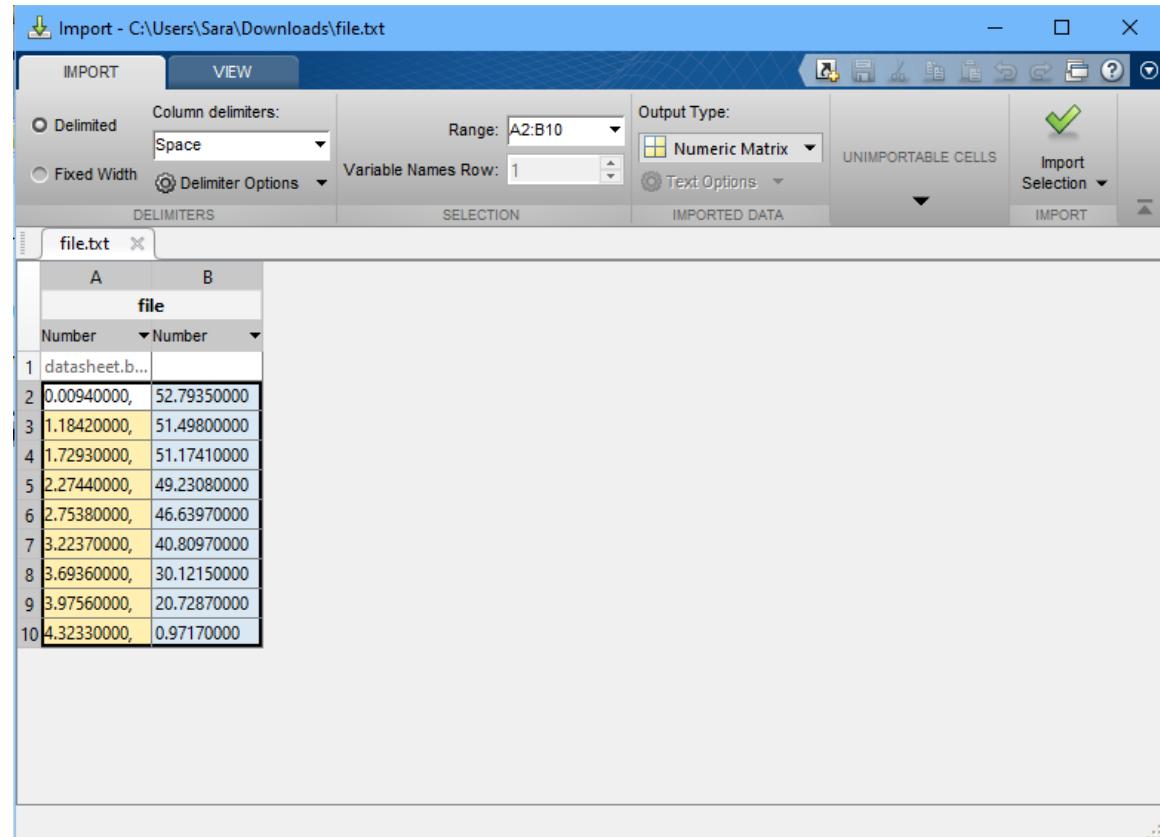
- How to use the digitizer tool:
  3. Select samples of one curve
    - Mouse left button = new sample
    - Mouse middle button = stop inserting samples
  4. Save to file

```
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
```



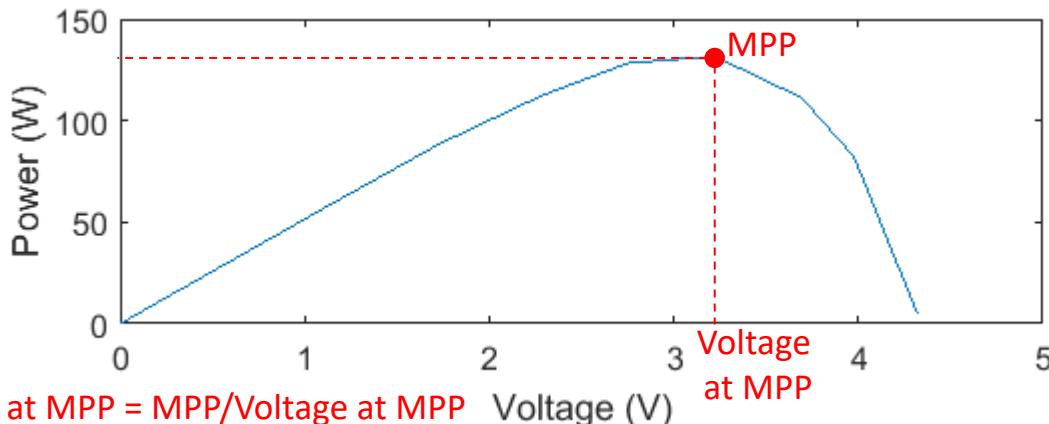
# Photovoltaic module

- 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



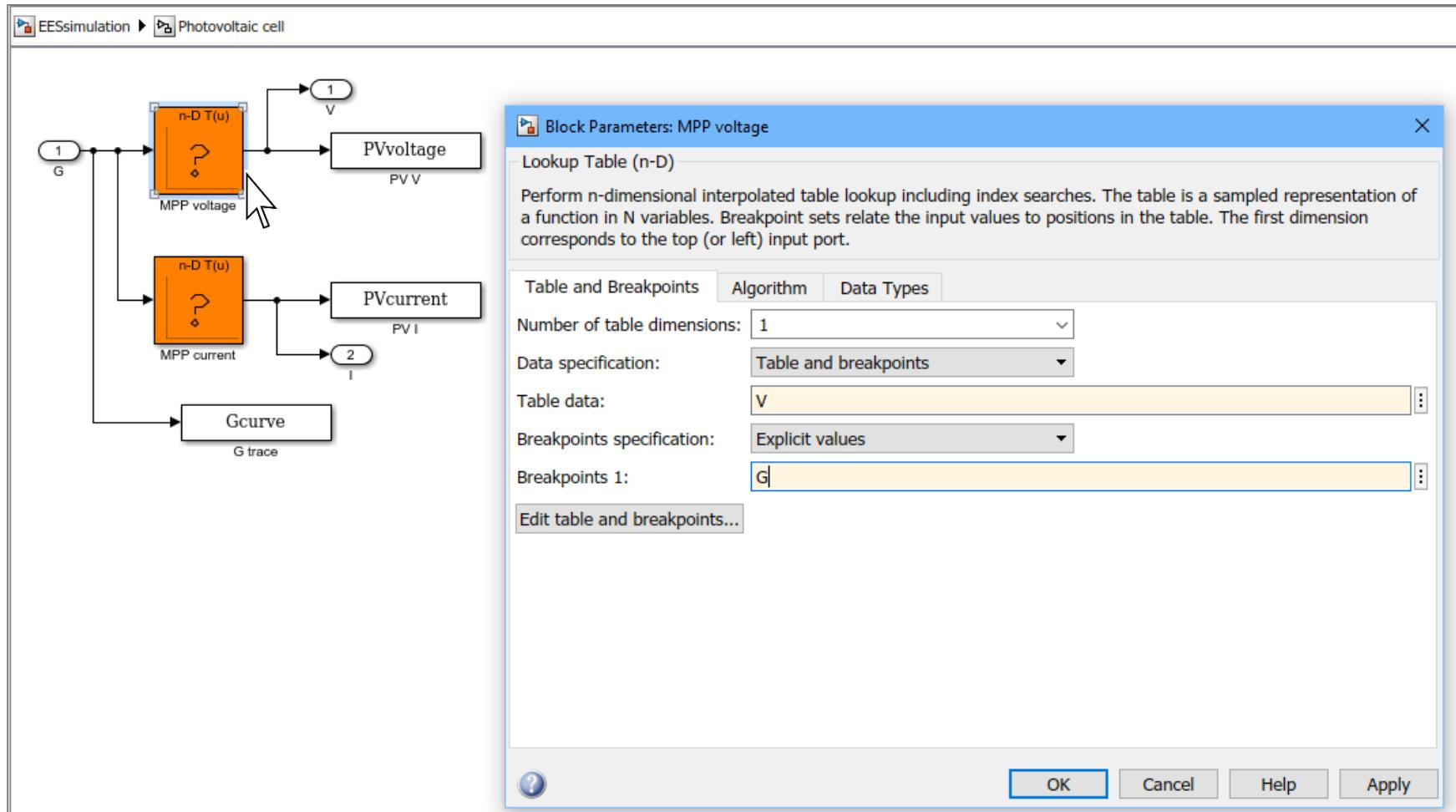
datasheet.txt

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

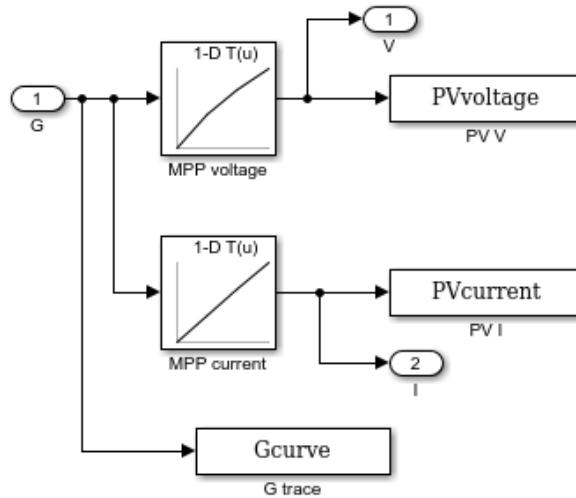
- Populate the lookup tables:
  - Vector of values of G:
    - $G = [250; 500; 750; 1000];$
  - Vector of corresponding values of current at the MPP
    - $I = [I_{250}; I_{500}; I_{750}; I_{1000}];$
  - Vector of corresponding values of voltage at the MPP
    - $V = [V_{250}; V_{500}; V_{750}; V_{1000}];$
  - Write the name of the variables in the configuration pane of the loopup tables
    - Breakpoints = G
    - Table values = V or I (corresponding values)

# Photovoltaic module



# Photovoltaic module

- If everything went well, your lookup tables should now look like:

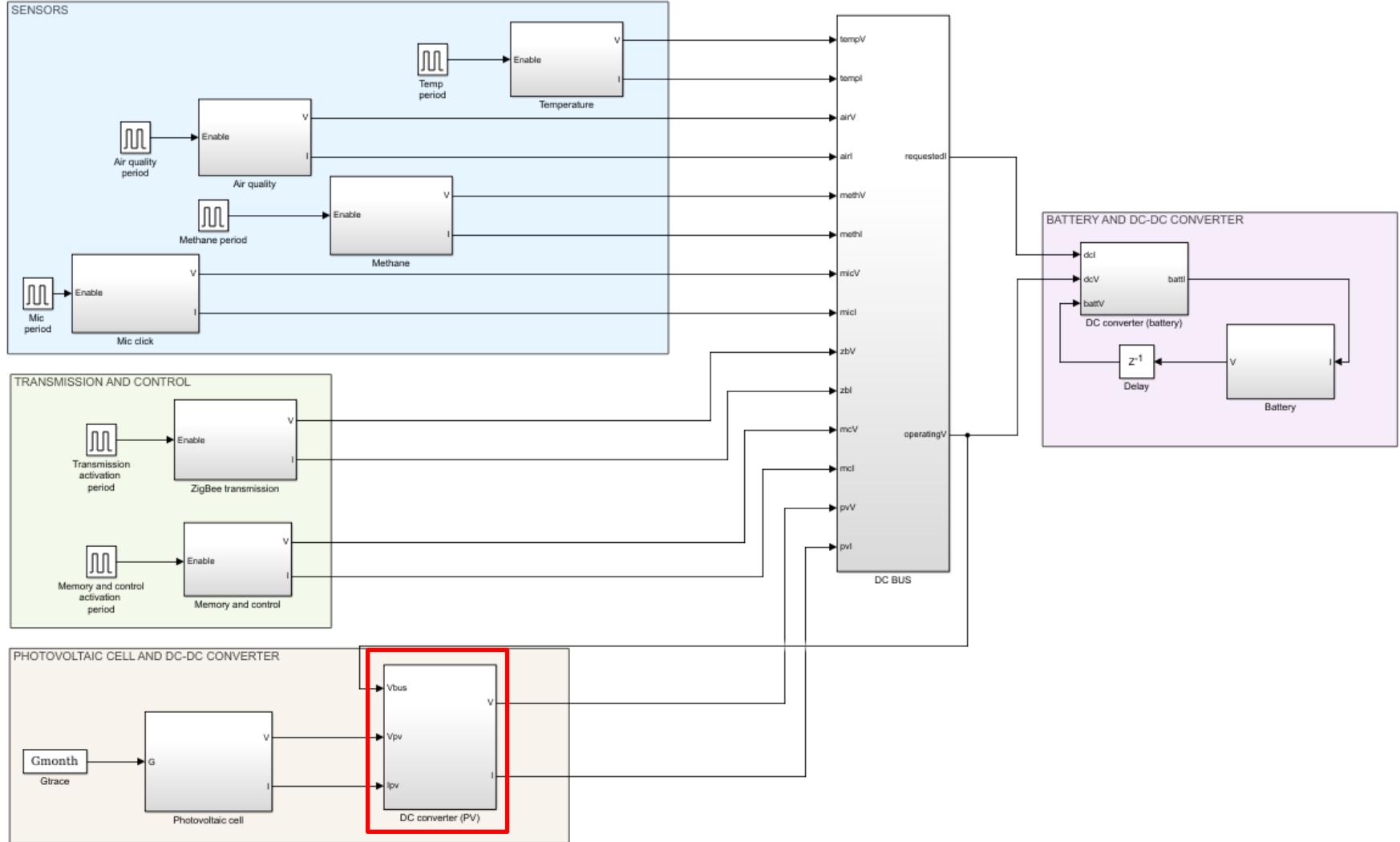


- Lookup table output:
  - If input irradiance is one of the sampled values, the lookup table outputs the corresponding value of voltage/current
  - Else, the lookup table interpolates between the available samples and estimate the value of voltage/current

# **Lab 3 – Part 2**

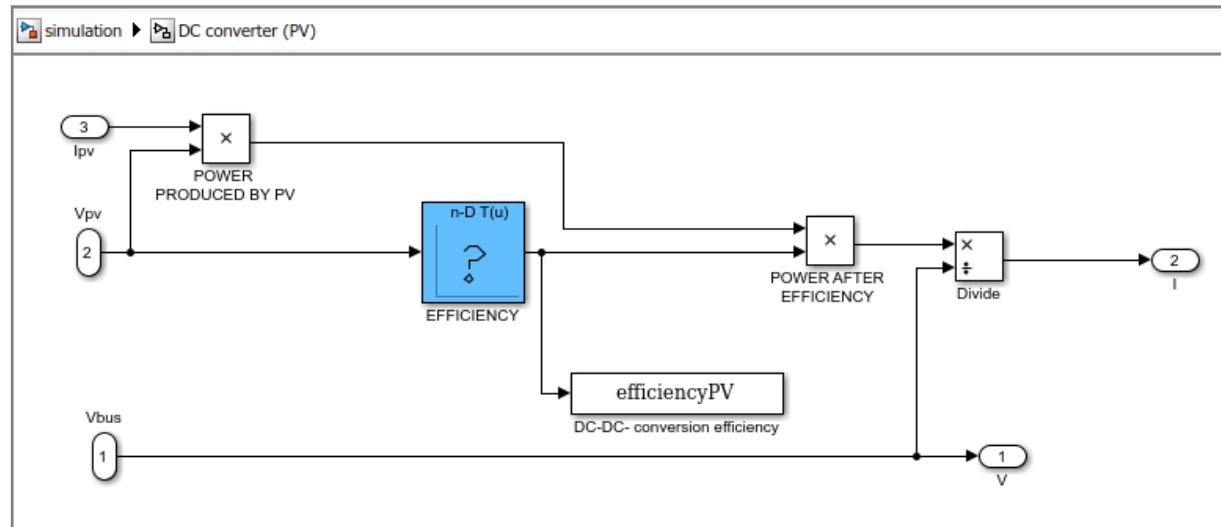
## **Model of DC-DC converters and battery**

# Simulink implementation



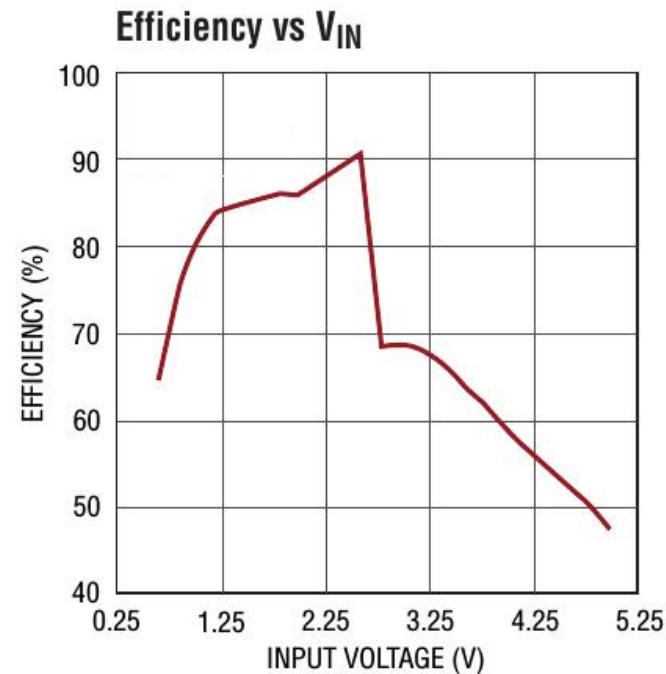
# DC-DC converter of PV module

- DC-DC converter of the photovoltaic module
  - Contains a lookup table of efficiency given input PV voltage
    - Efficiency in [0, 1]
  - Block implementation:
    - $(V_{PV} \cdot I_{PV}) \cdot \text{efficiency} = (V_{BUS} \cdot I)$
    - $V_{BUS} = 3.3V$  (DC bus voltage)
    - $I = (V_{PV} \cdot I_{PV} \cdot \text{efficiency}) / V_{BUS}$



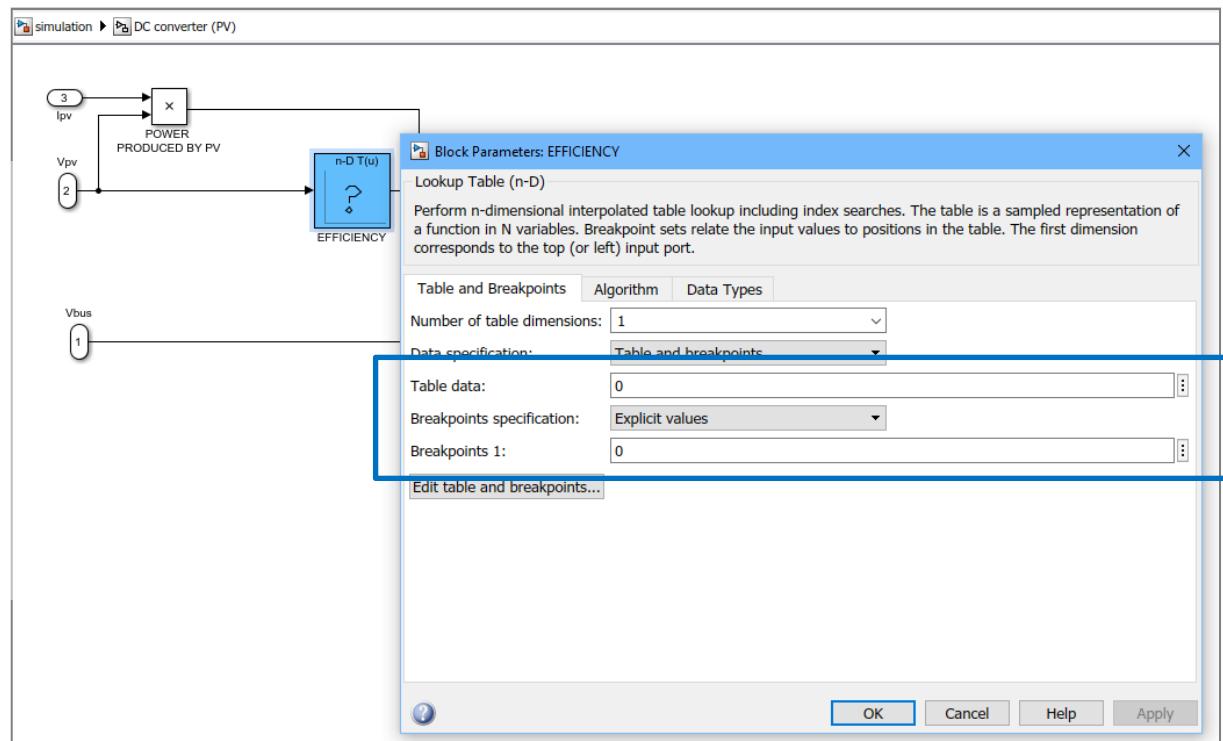
# DC-DC converter of PV module

- 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

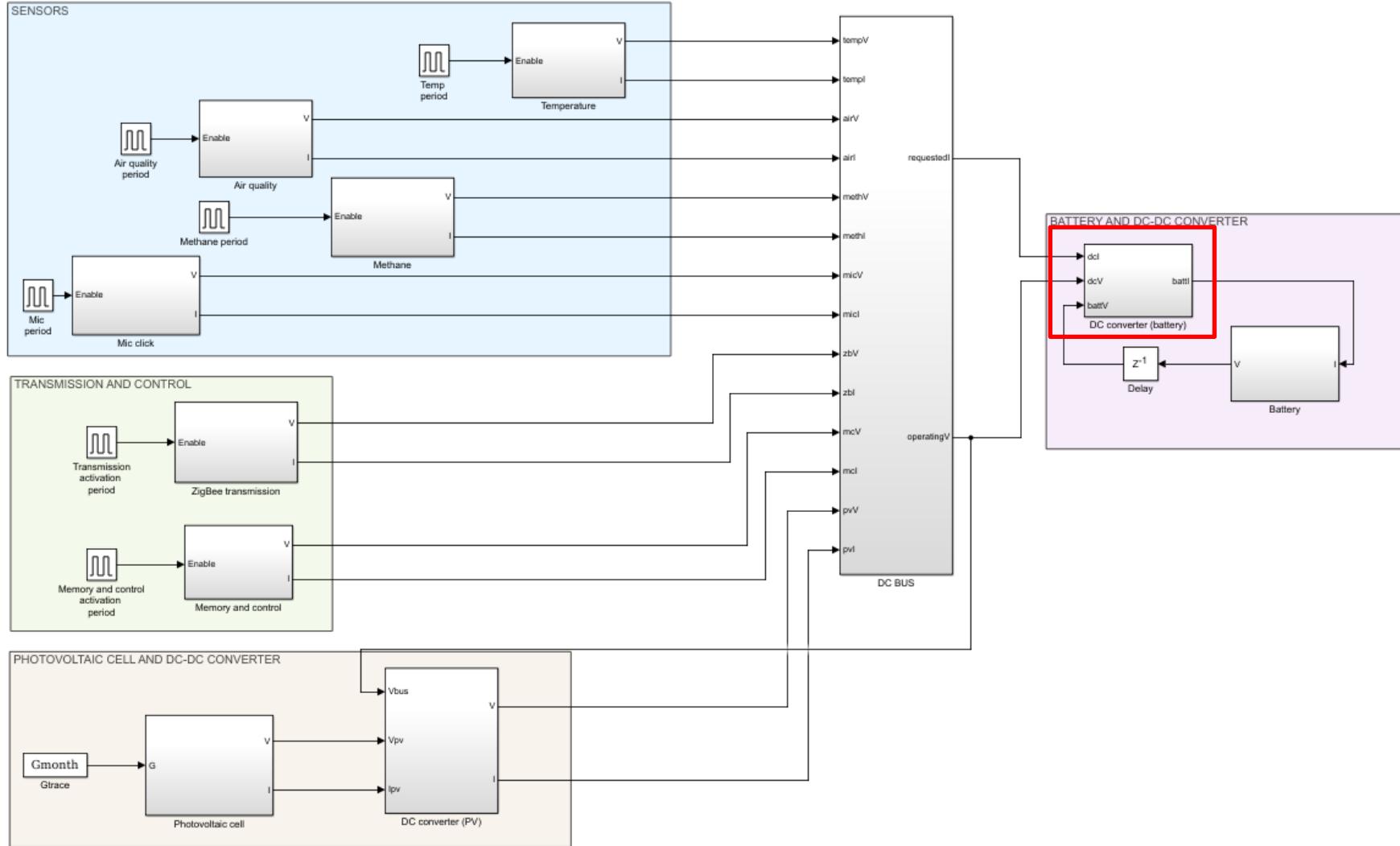


# DC-DC converter of PV module

- How to:
  1. Digitize the curve
  2. Import file in Matlab
  3. Set the lookup table parameters
    - Voltage samples = breakpoints
    - Efficiency samples = table data
    - More digitized samples = more accurate

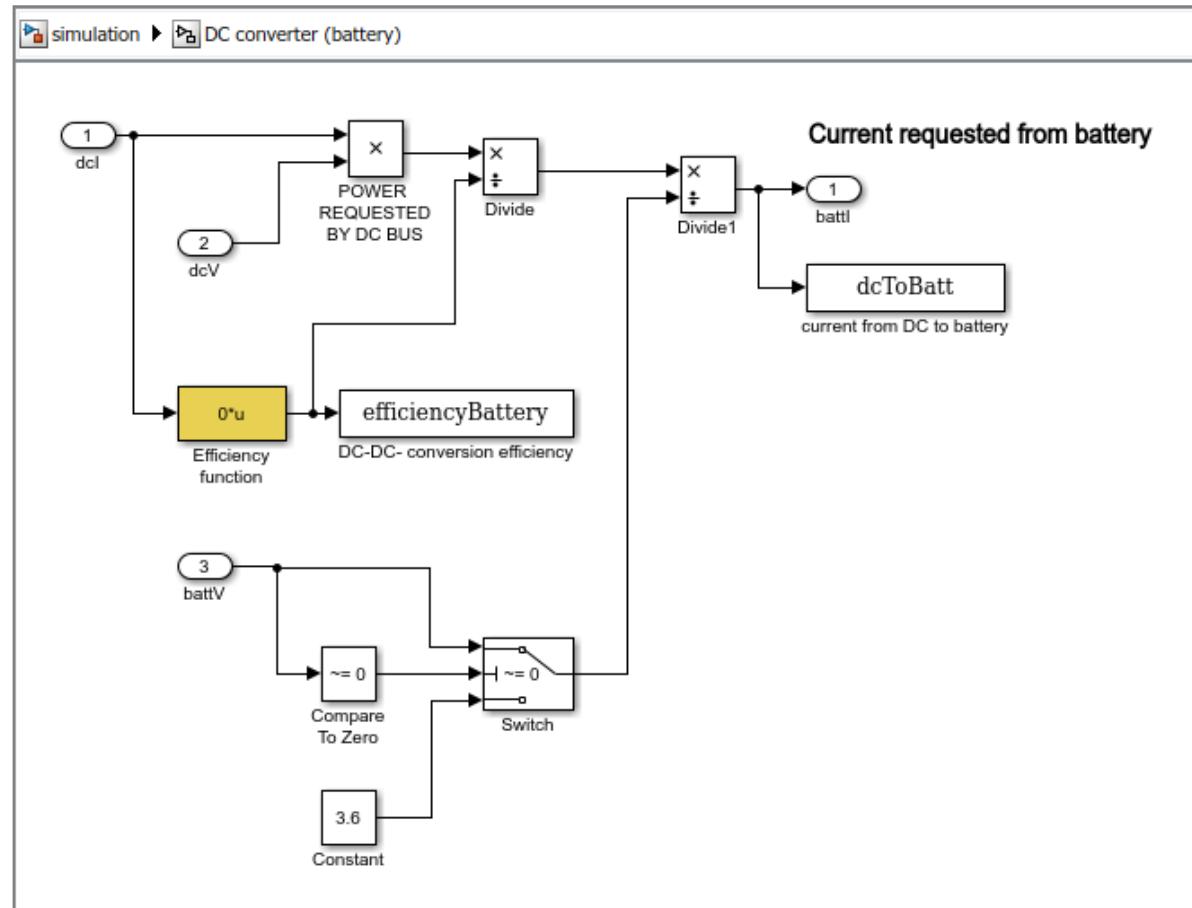


# Simulink implementation



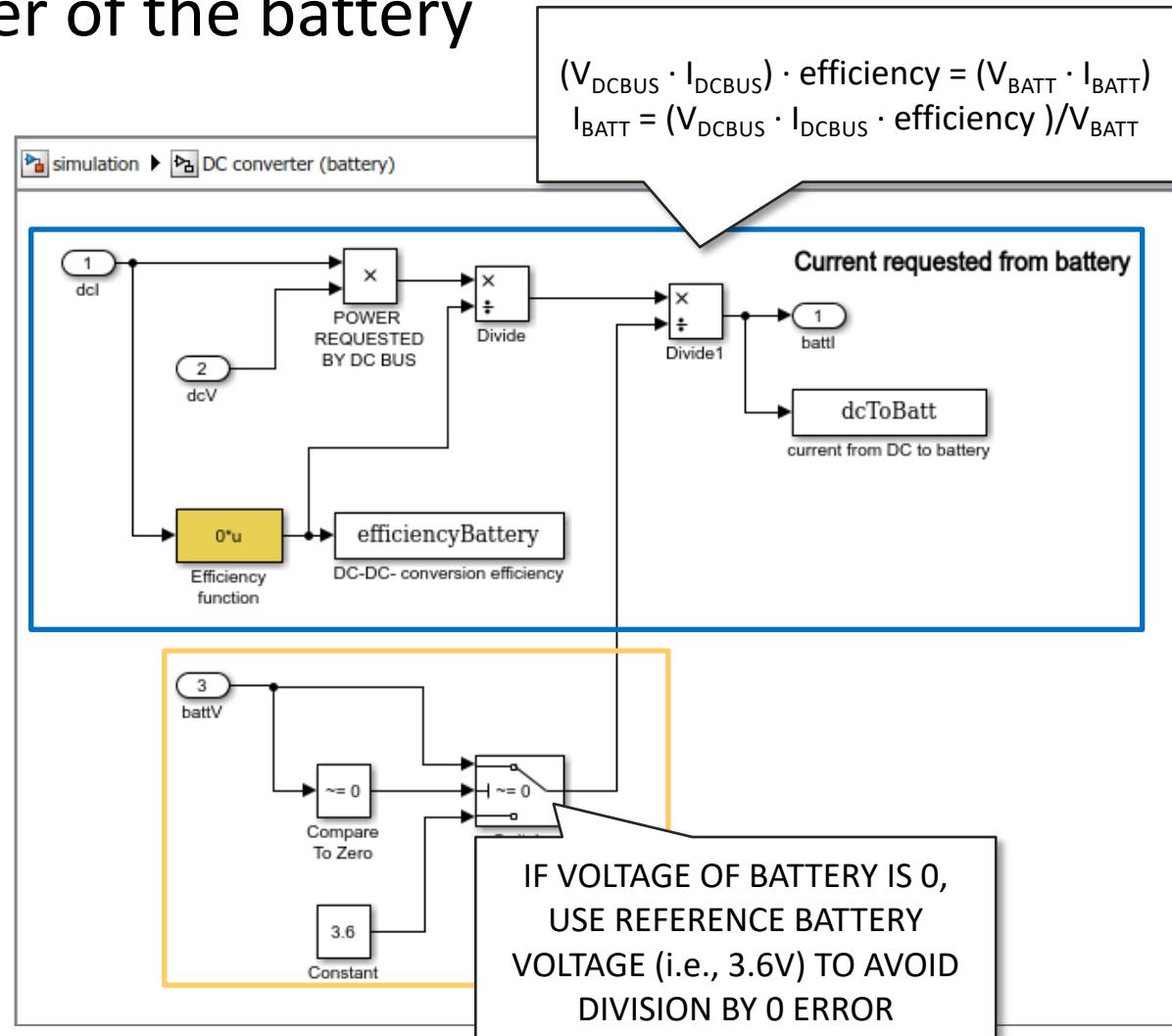
# DC-DC converter of battery

- DC-DC converter of the battery



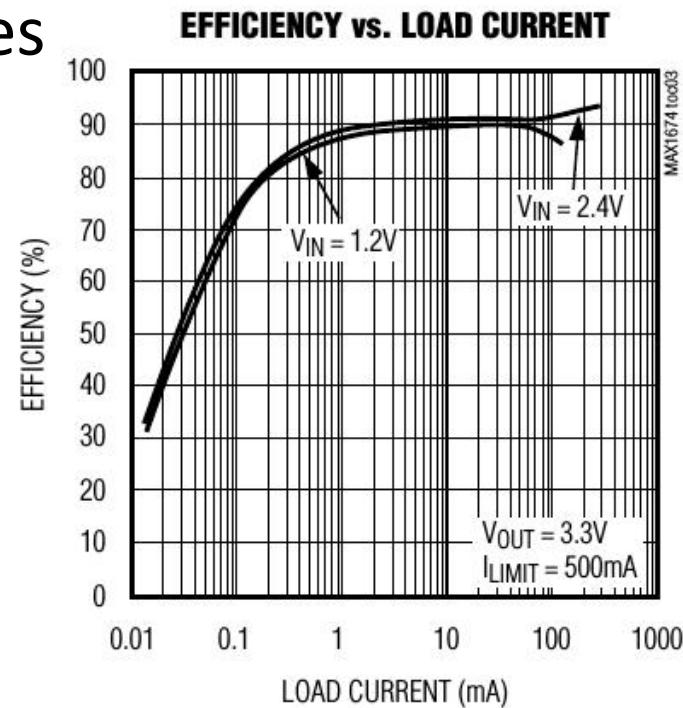
# DC-DC converter of battery

- DC-DC converter of the battery



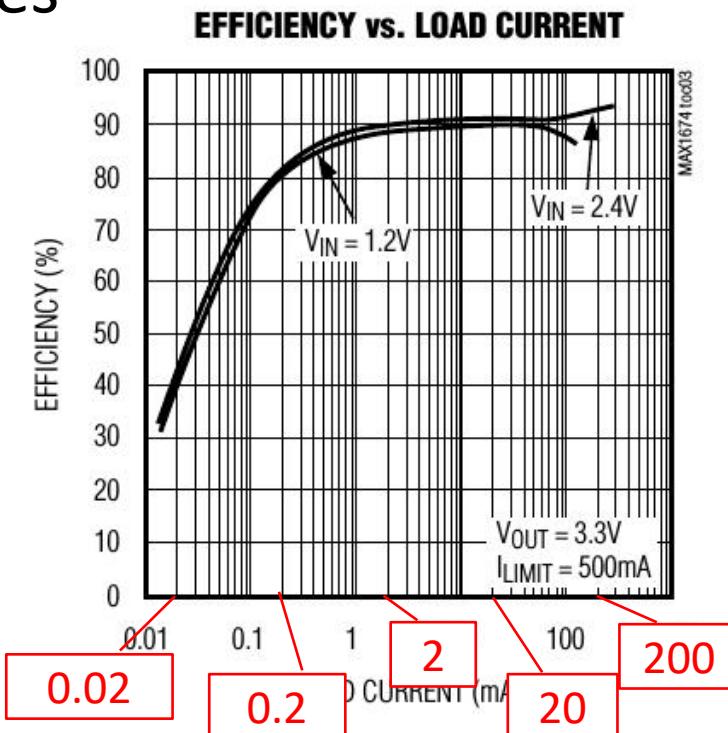
# DC-DC converter of battery

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

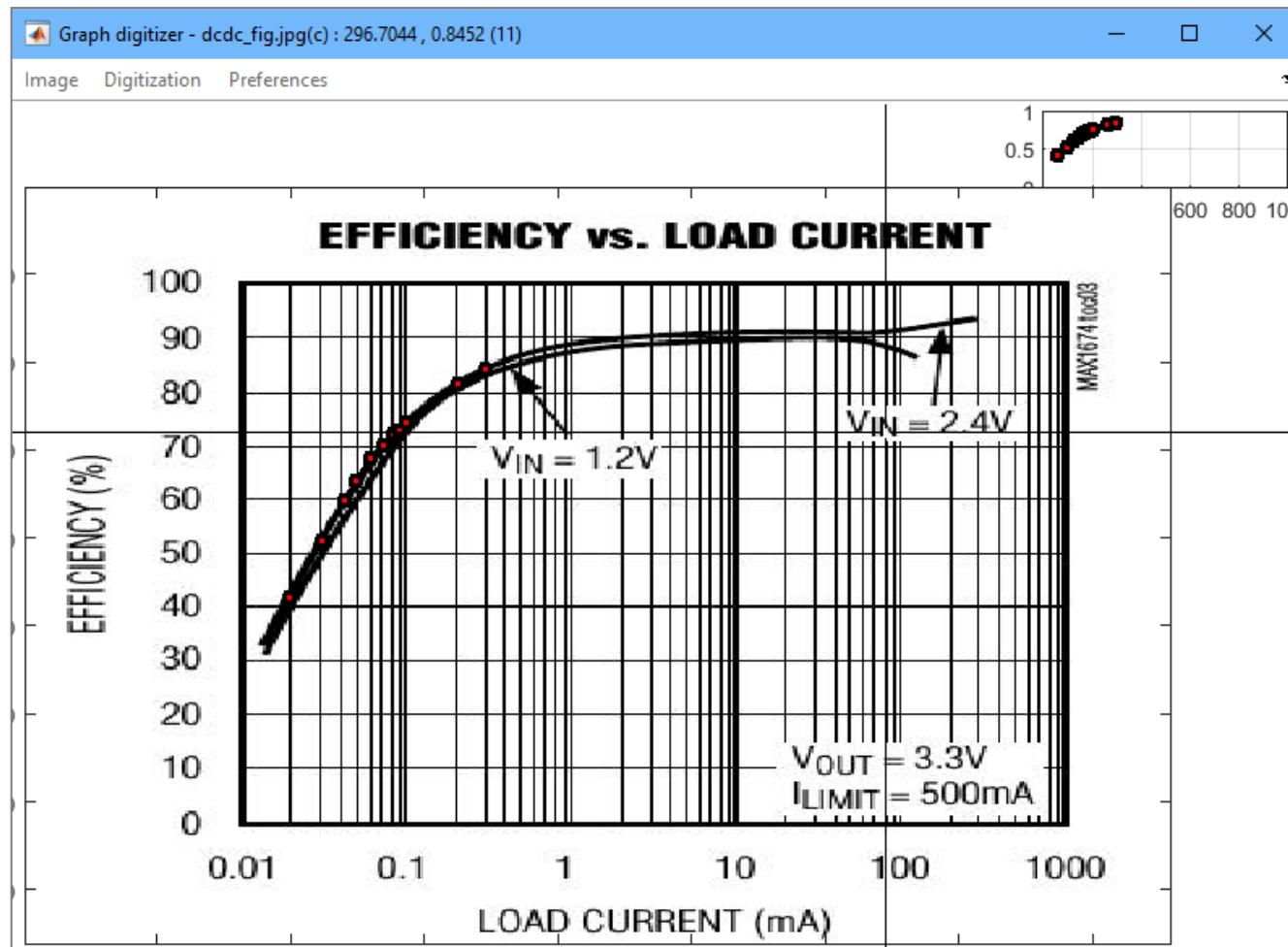


# DC-DC converter of battery

- Open the **BATT\_DCDCconv.pdf** file
  - Page 4: curve of efficiency given input current with voltage to the bus = 3.3V
- How to:
  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. Import the data in Matlab



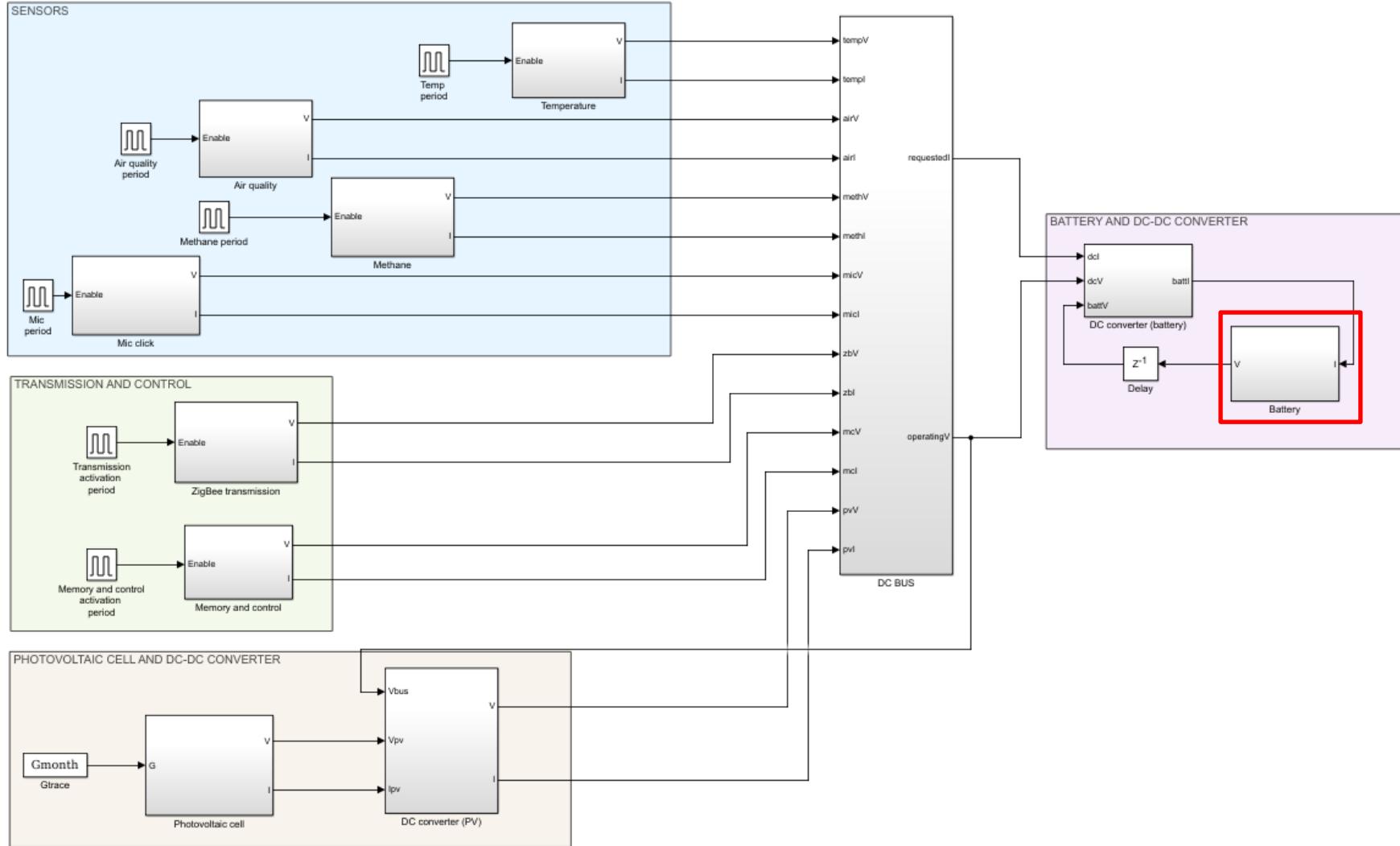
# DC-DC converter of battery



# DC-DC converter of battery

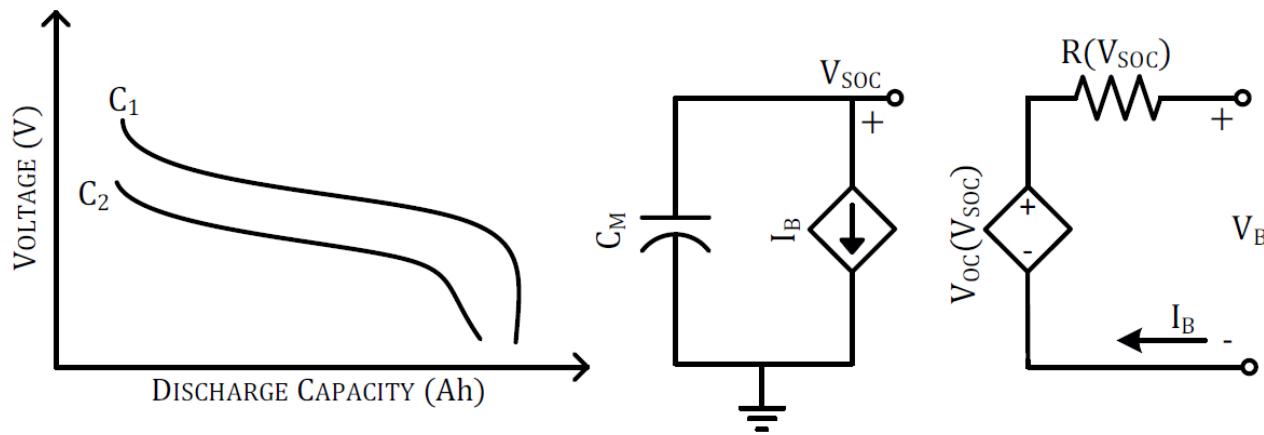
- How to:
  3. Set the lookup table parameters (as before):
    - Current samples = breakpoints
    - Efficiency samples = table data
    - More digitized samples = more accurate

# Simulink implementation



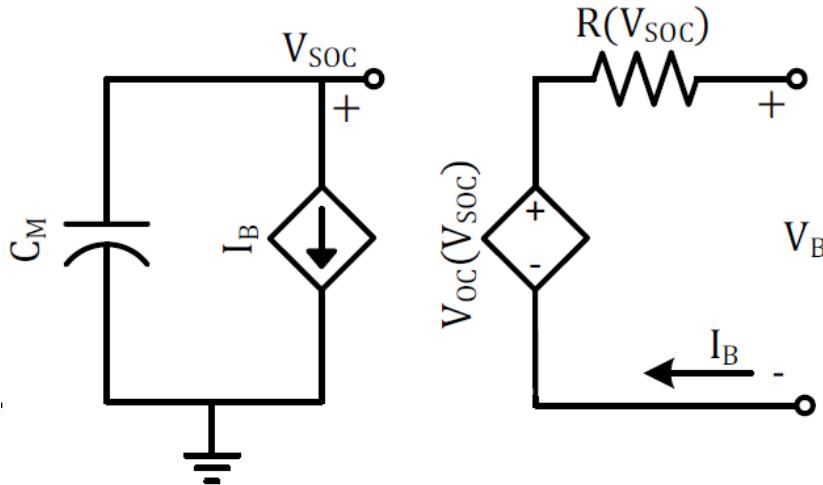
# Battery model

- 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



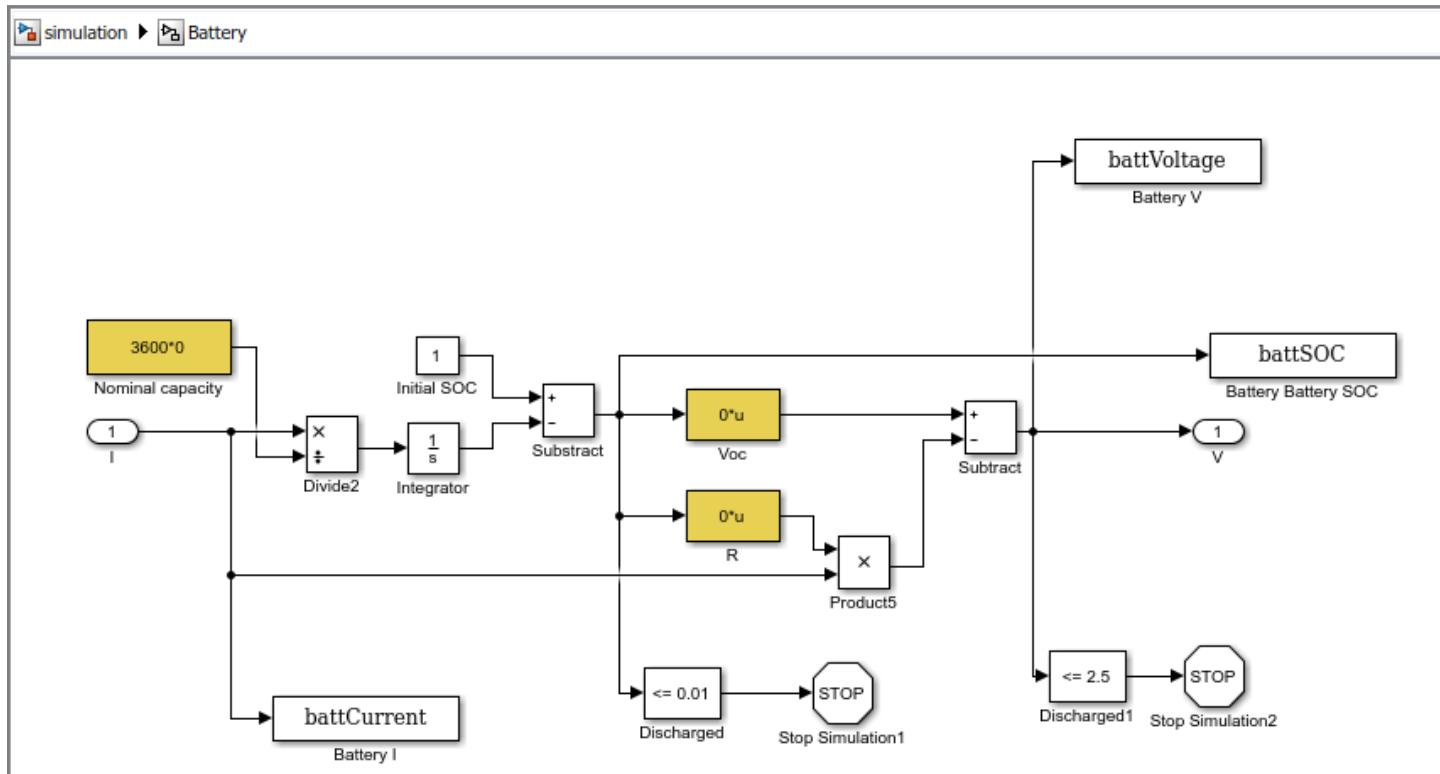
# Battery model

- 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



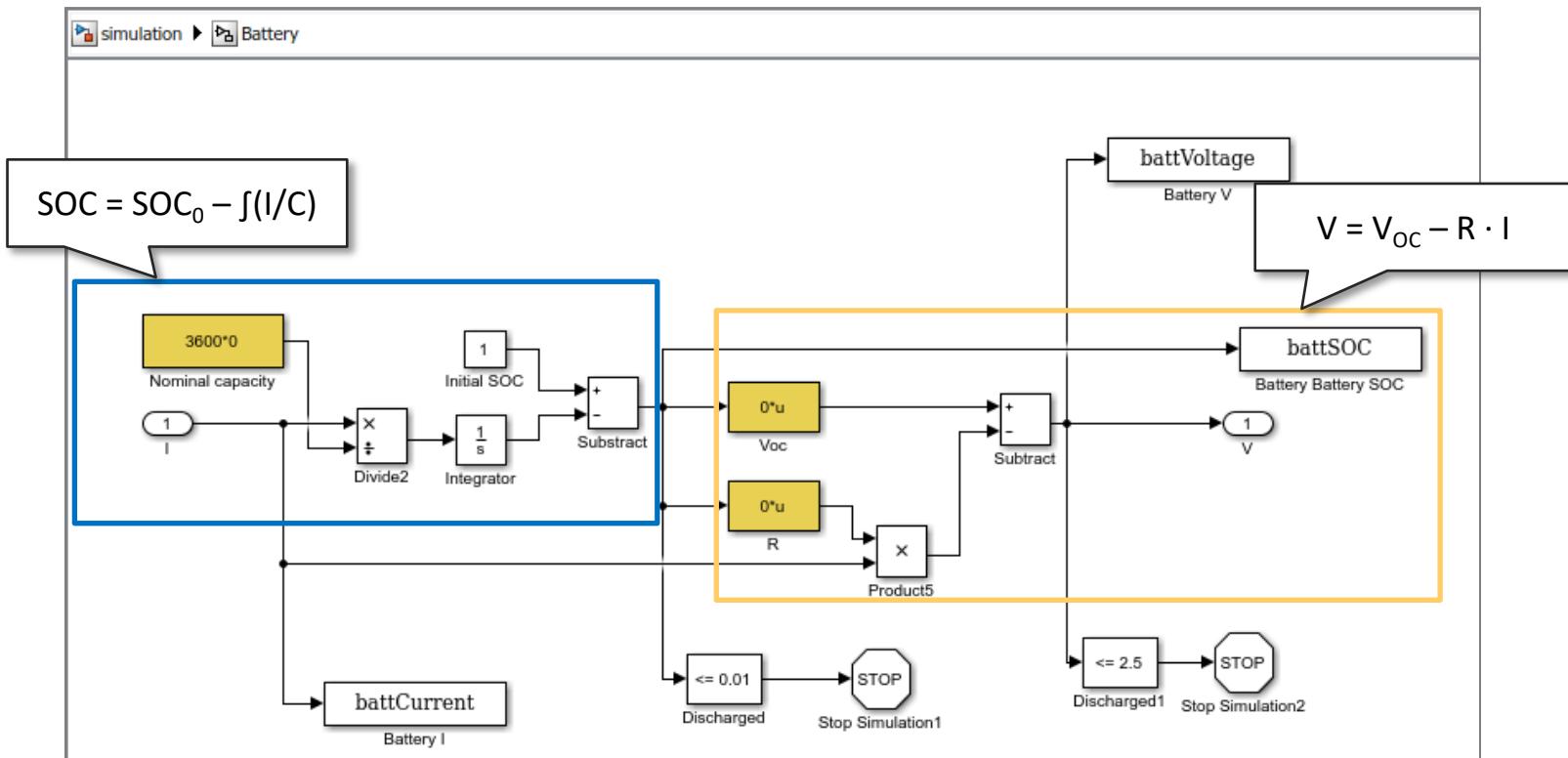
# Battery model

- Not implemented as a circuit but as equations solving the circuit
  - Simulation faster than implementing the circuit



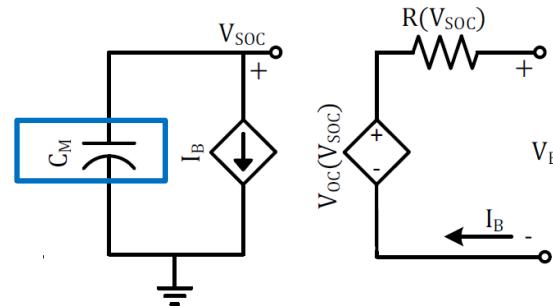
# Battery model

- Not implemented as a circuit but as equations solving the circuit
    - Simulation faster than implementing the circuit



# 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)

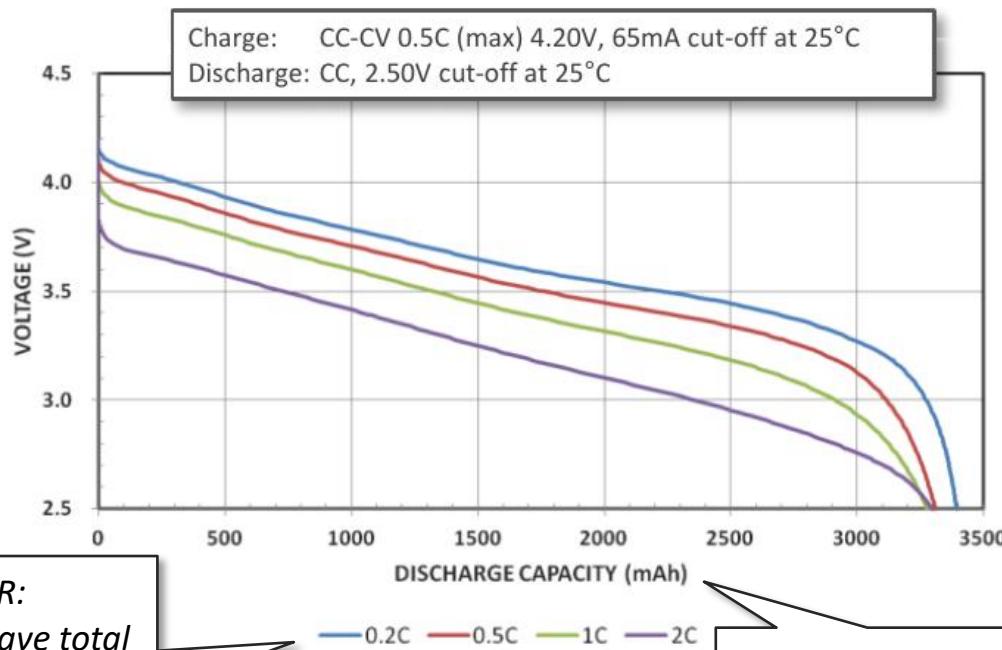


## Specifications

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

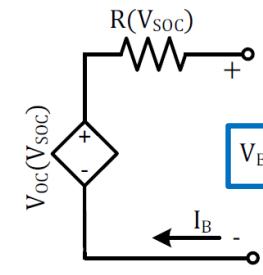
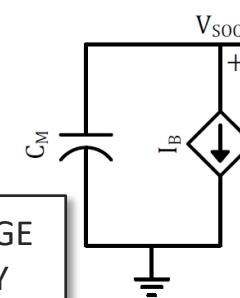
# Battery model

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



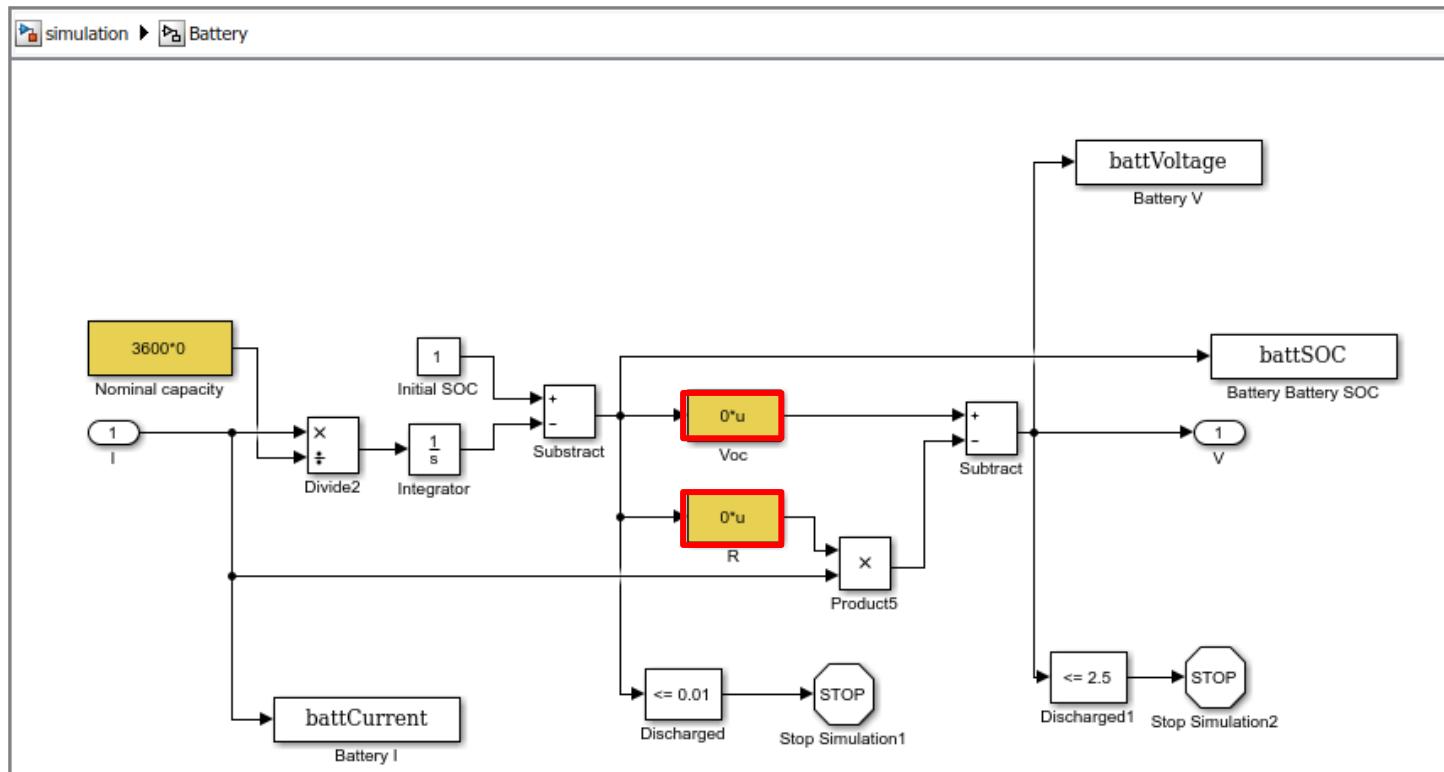
REMINDER:  
 $1C$  = current to have total  
discharge in 1 hour

≈ STATE OF CHARGE  
OF THE BATTERY



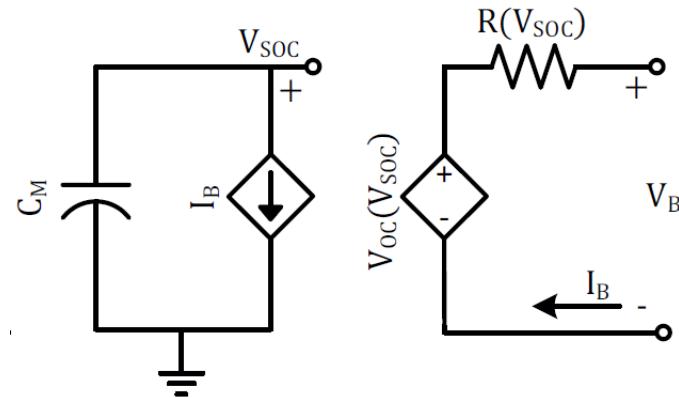
# Simulink implementation

- Simulink models the battery as  $V_{OC}$  and  $R$  as a function of  $SOC$ 
  - How can we relate  $V_{OC}$  and  $R$  to  $V$ , and thus  $SOC$ ?



# Battery model

- Derive  $V_{OC}$  and  $R$  as a function of  $V$ 
  - 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

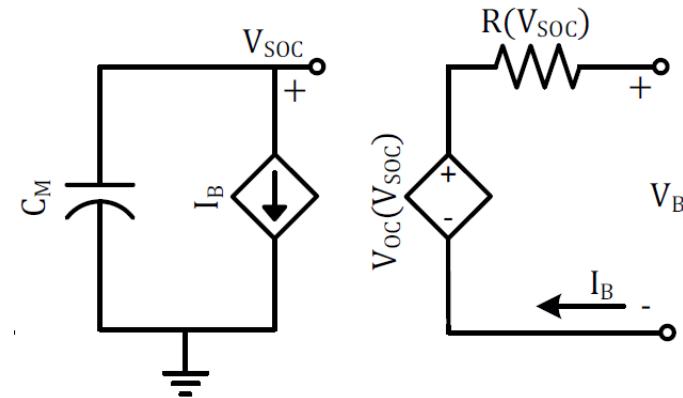
- In case you were wondering how we got here...
  - Equation solving the system:
    - $V_{OC} = R I + V$
  - Applied for both currents:
    - $\begin{cases} V_{OC} = RI_{curve1} + V_{curve1} \\ V_{OC} = RI_{curve2} + V_{curve2} \end{cases}$
  - $V_{OC} = RI_{curve1} + V_{curve1} = RI_{curve2} + V_{curve2}$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

# Battery model

- Derive  $V_{OC}$  and  $R$  as a function of  $V$ 
  - Solving the equations associated with the right hand side branch of the circuit

$$- V_{OC} = V_{curve1} + R \cdot I_{curve1}$$

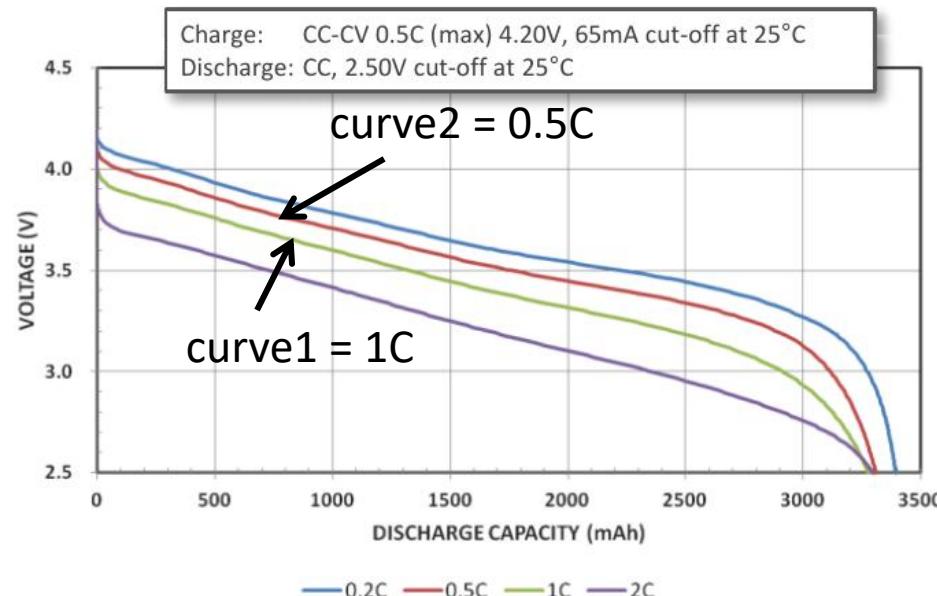
$$- R = \frac{V_{curve2} - V_{curve1}}{I_{curve1} - I_{curve2}}$$



- How do we populate the equations?

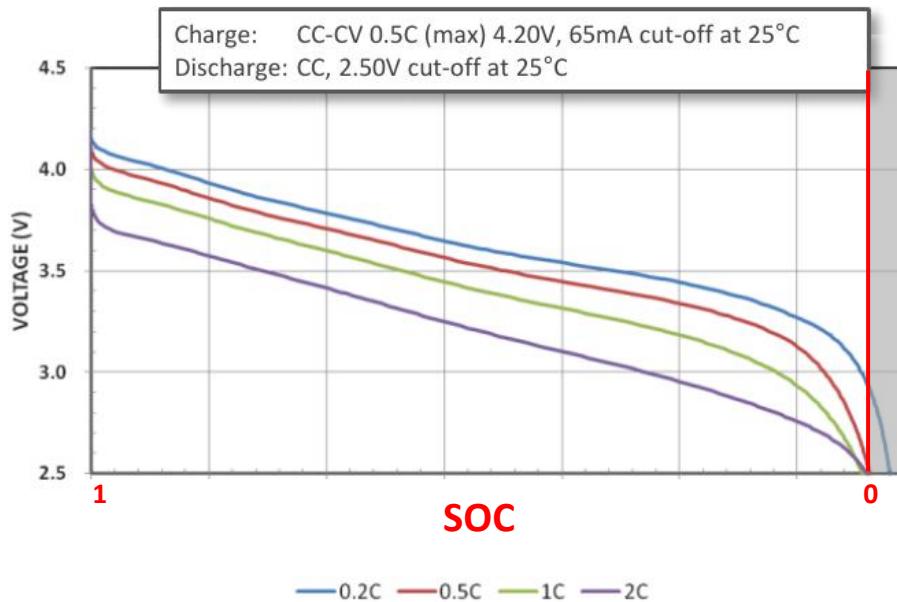
# Battery model

- 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



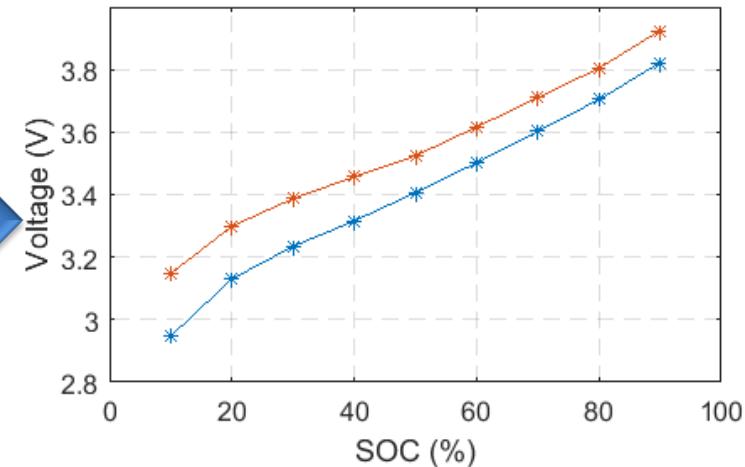
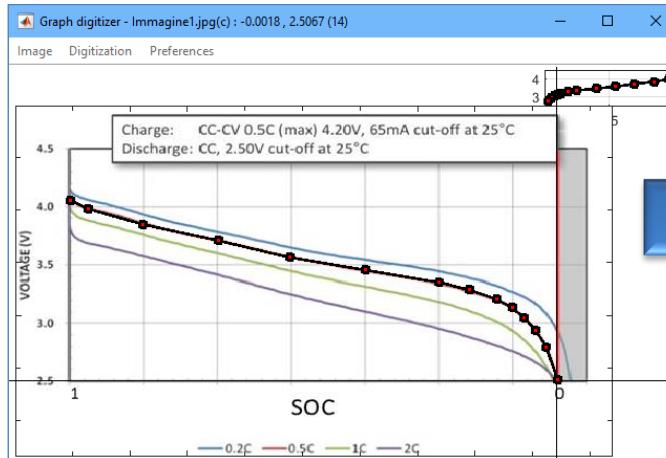
# Battery model

- $V$  battery voltage: how to
  1. Download the datasheet figure (course website)
  2. Determine SOC axis
    - Fix the origin where the lower curve ends



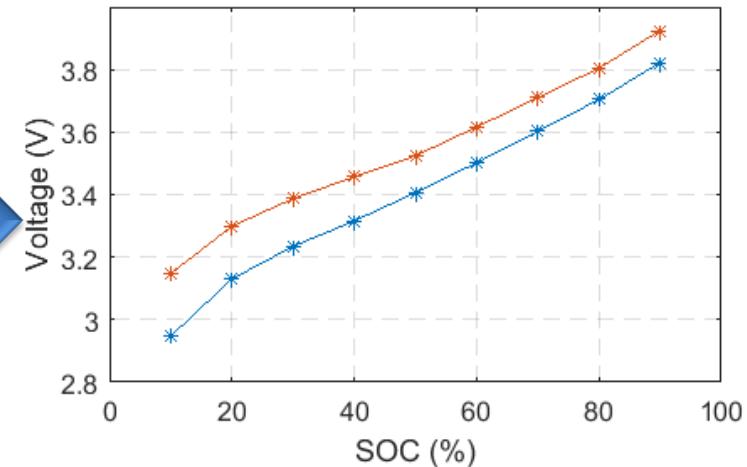
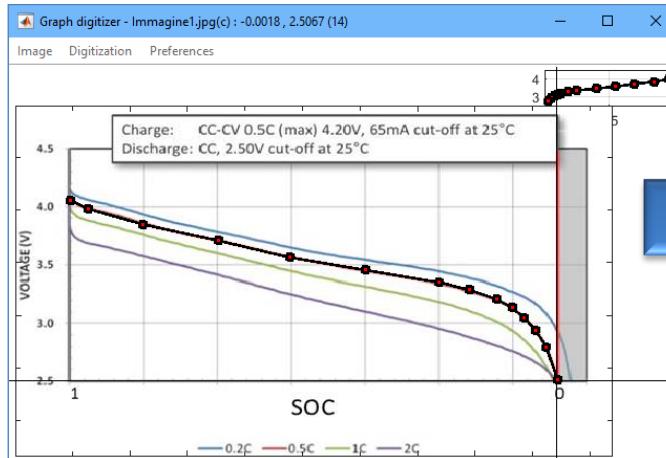
# Battery model

- $V_{\text{battery}}$  voltage: how to
  - 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



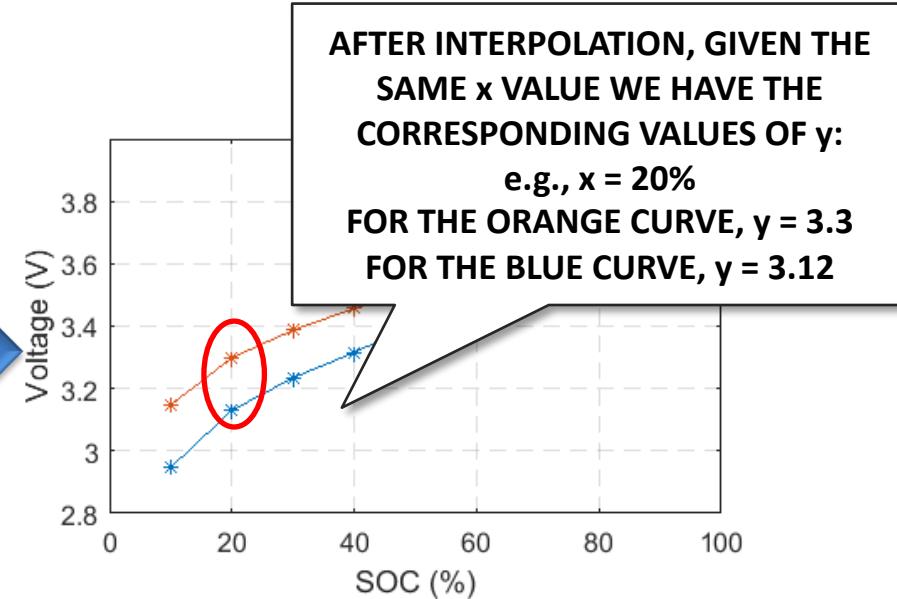
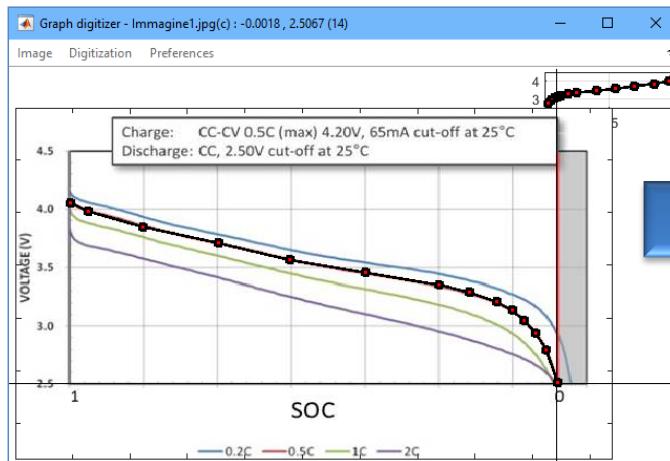
# Battery model

- 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!

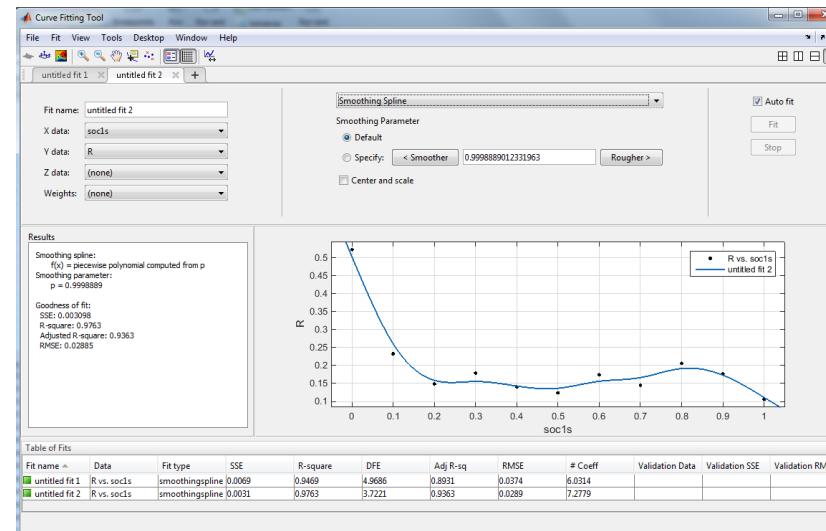


# Battery model

- 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
  - $SOC \rightarrow V \rightarrow V_{OC}, R$ 

# Battery model

- So we 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



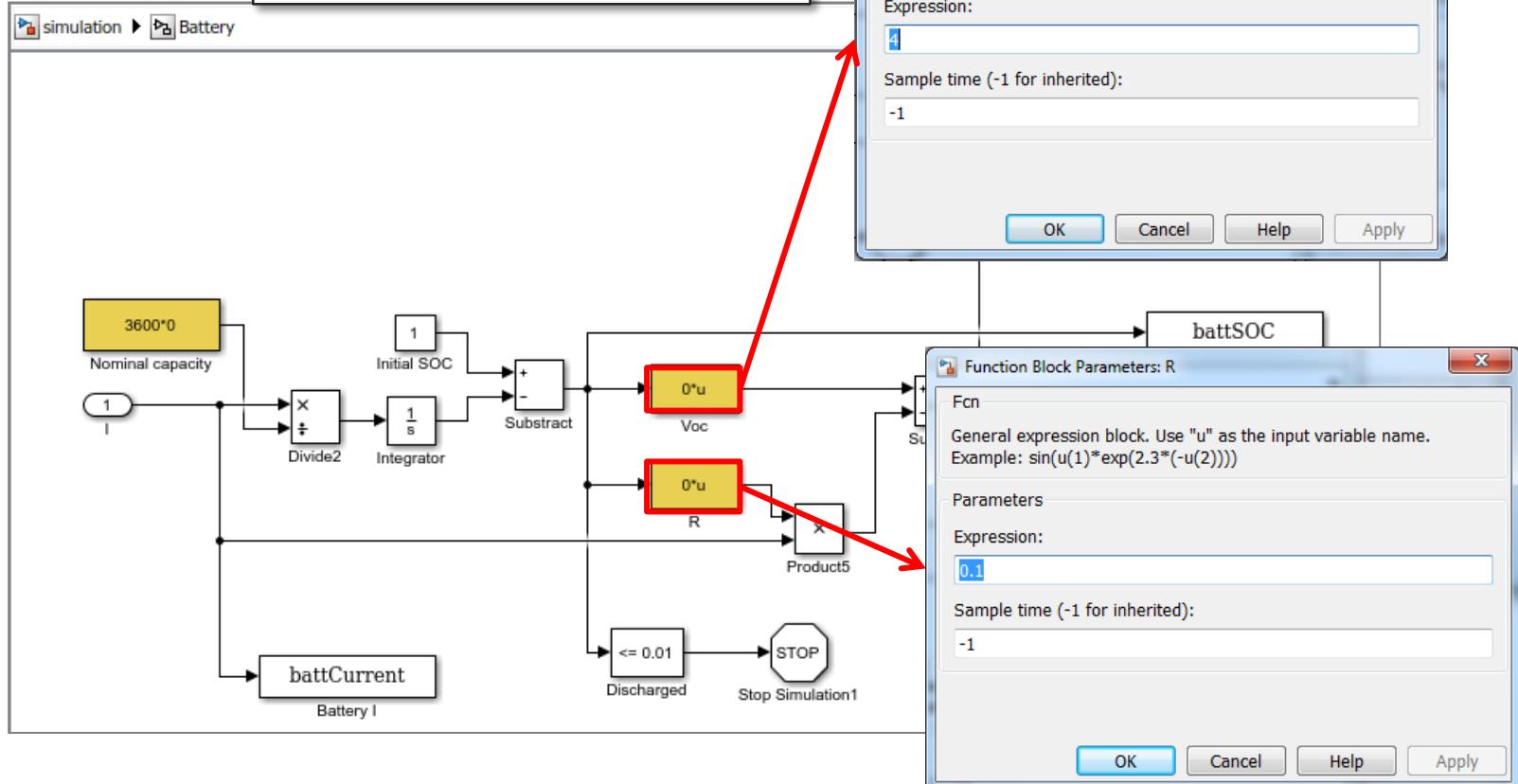
# Battery model

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

# Simulink implementation

PUT IN THE FUNCTION BLOCKS THE  
FUNCTION/POLYNOMIAL

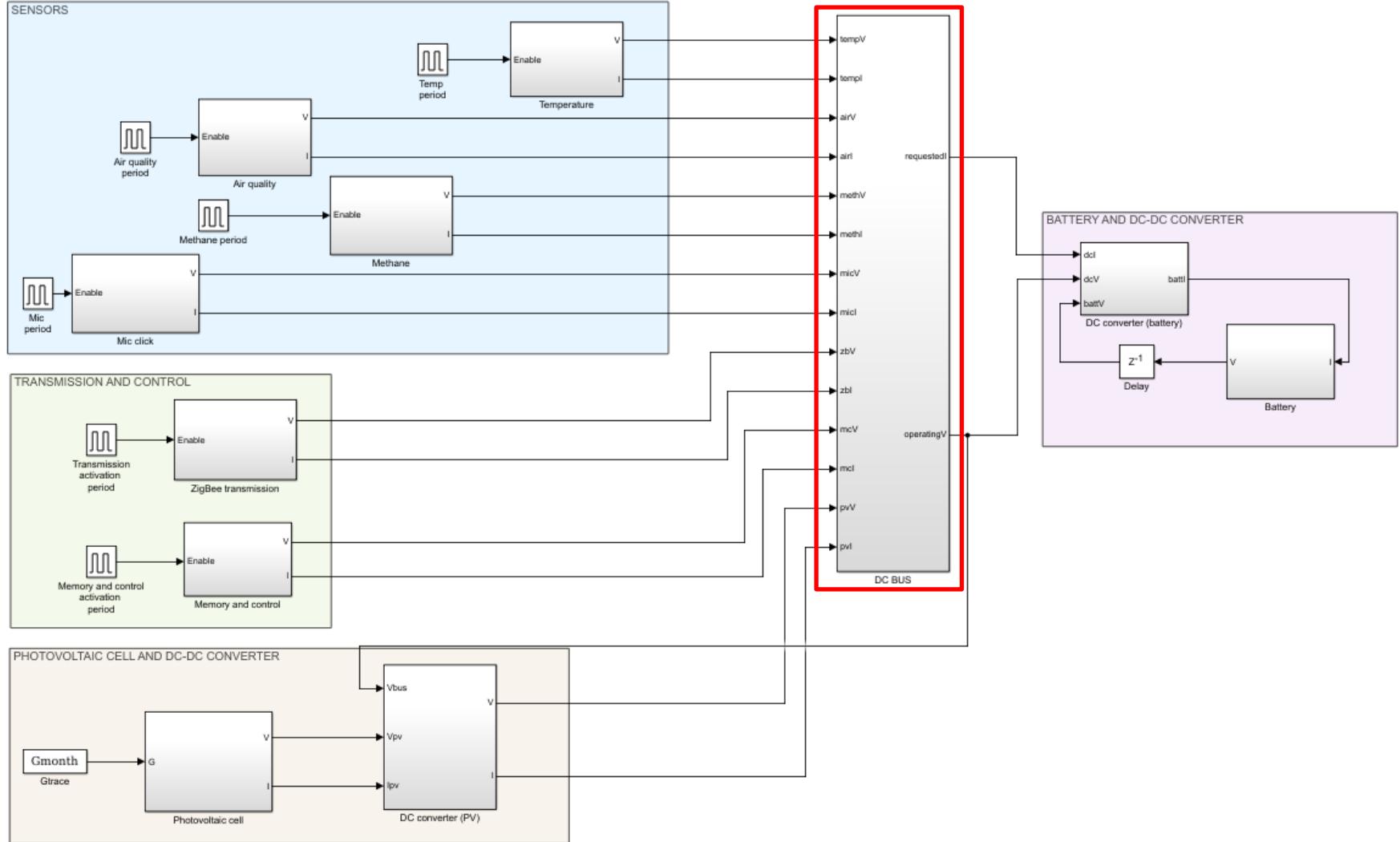
REMEMBER THAT THE VARIABLE  
REPRESENTING INPUT IS u (NOT x)



# **Lab 3 – Part 3**

## **Load modeling and scheduling**

# Simulink implementation

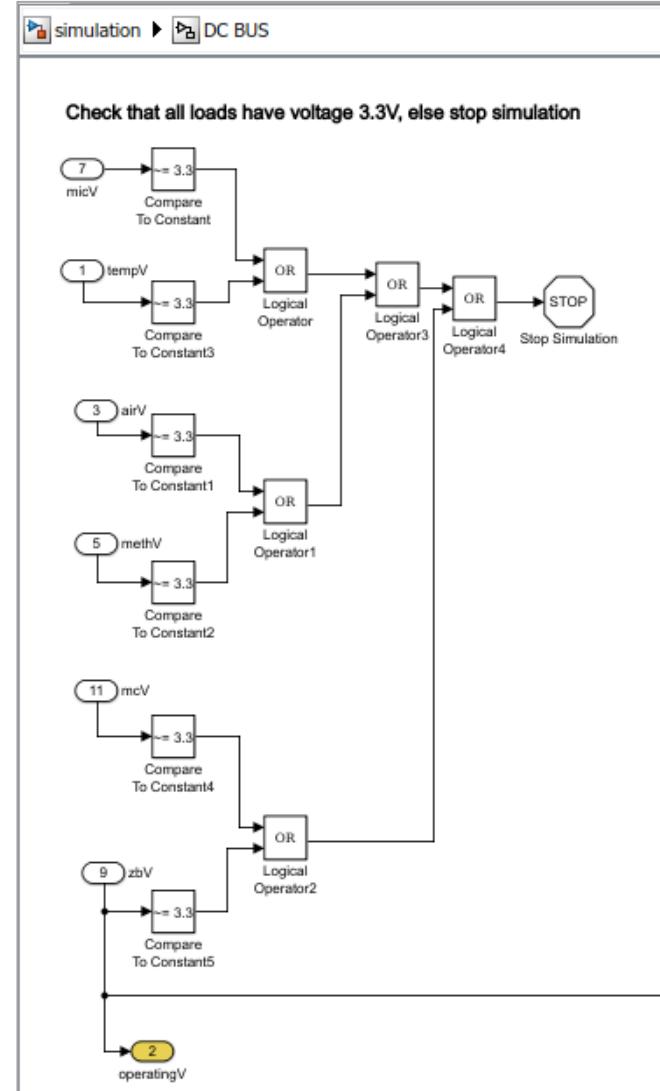


# DC bus

- 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

- Part 1:
  - Guarantee that all current requests happen at the same voltage
  - If any voltage is different from 3.3V, stop the simulation!
    - Maybe some DC-DC converter is missing?

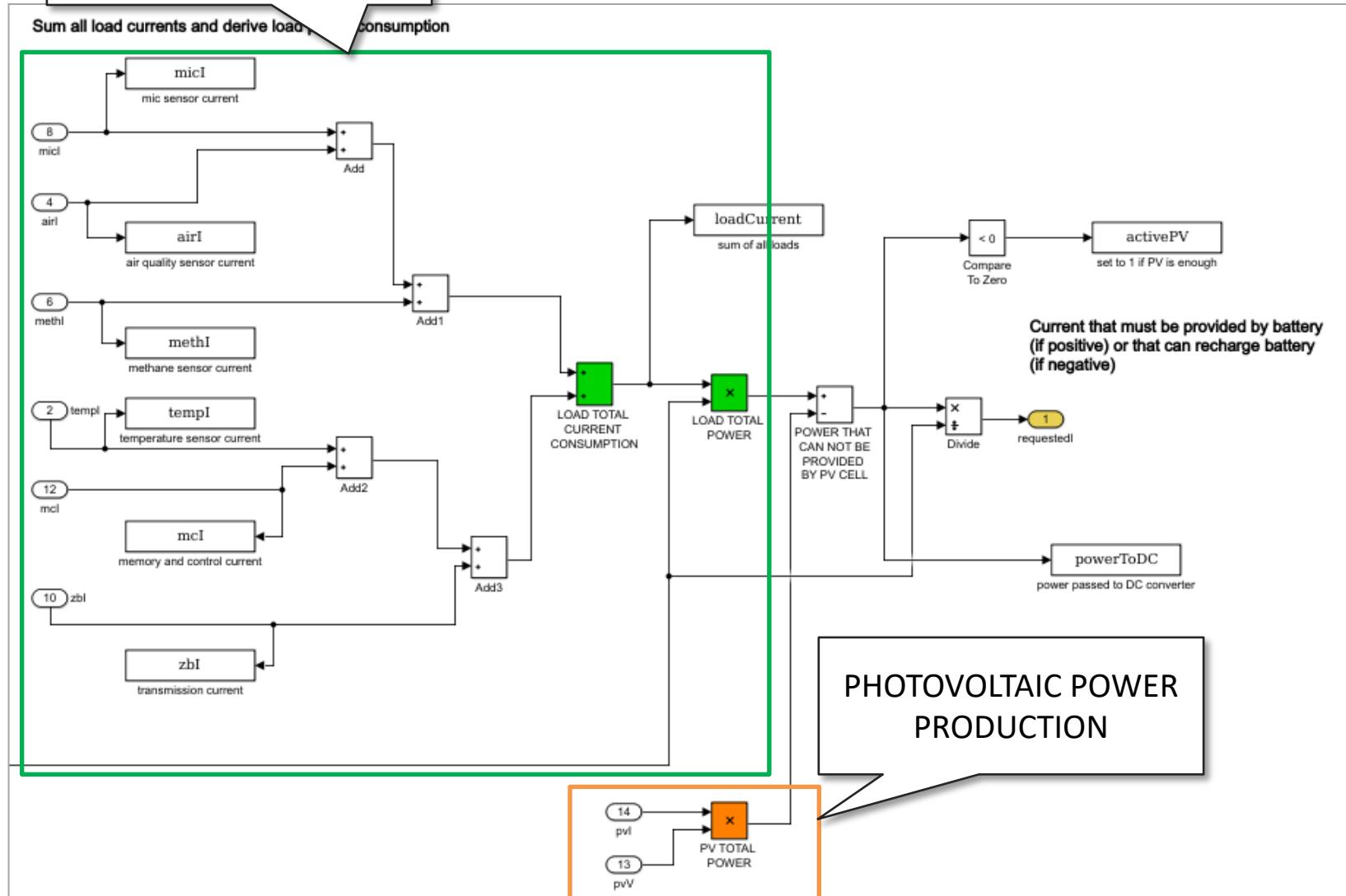


# DC bus

- **Part 2:**
  - 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

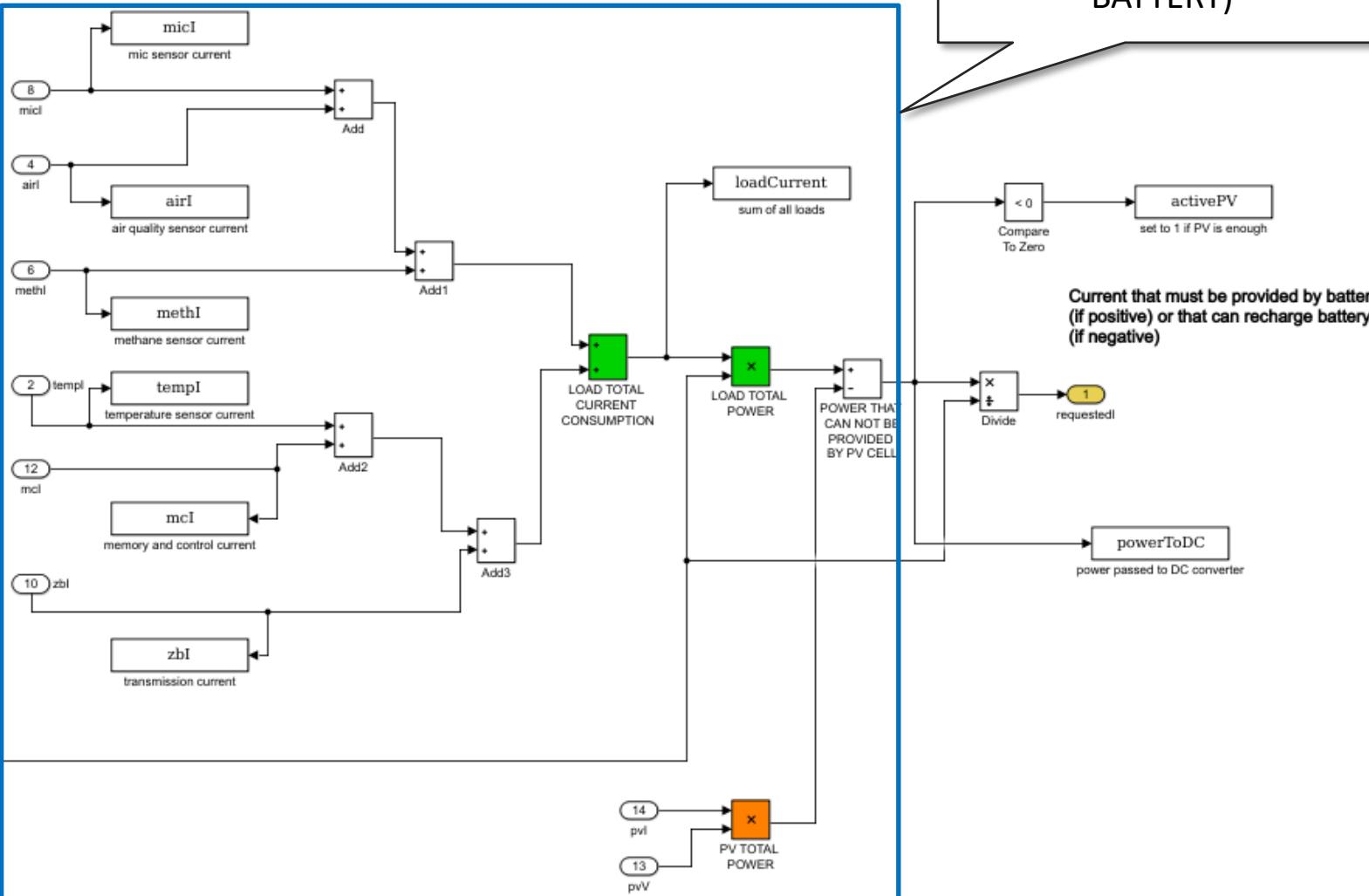
# TOTAL LOAD POWER CONSUMPTION



# DC bus

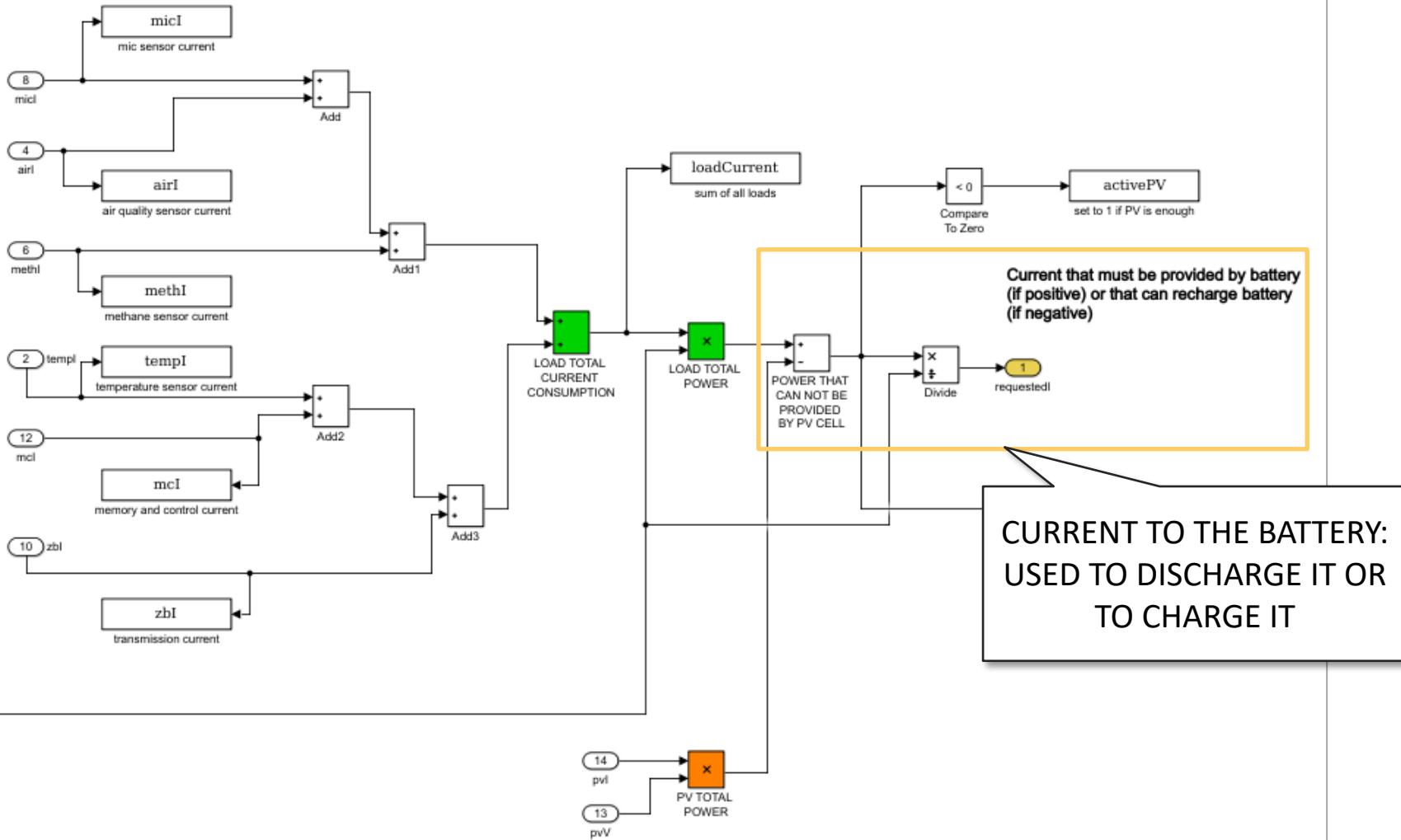
POWER MISSING/STILL AVAILABLE  
 (MUST DISCHARGE THE BATTERY/CAN CHARGE THE BATTERY)

Sum all load currents and derive load power consumption

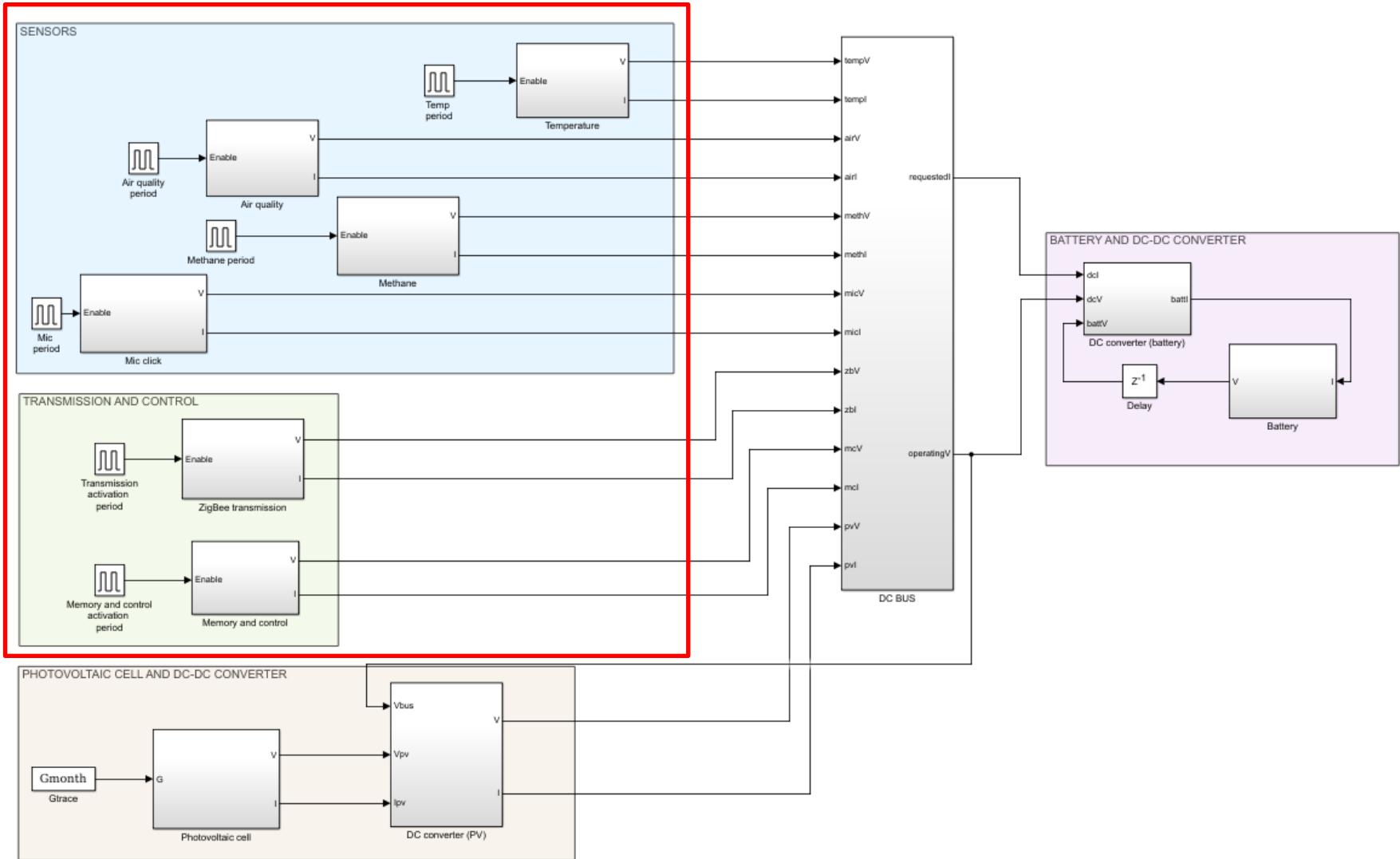


# DC bus

Sum all load currents and derive load power consumption



# Simulink implementation



# Loads

- 6 loads
  - Temperature sensor
    - Humidity and temperature sensor
  - Methane sensor
    - CH4 sensor
  - Air quality sensor
    - Metal-oxide, H<sub>2</sub> 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



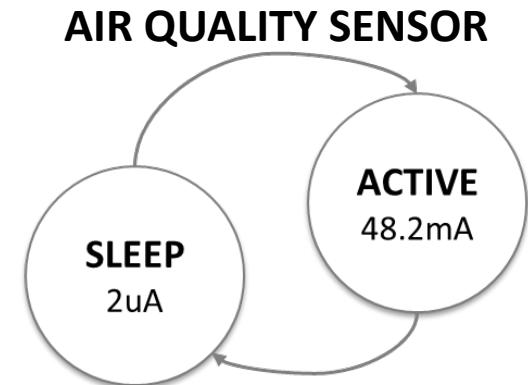
TEMPERATURE  
SENSOR



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

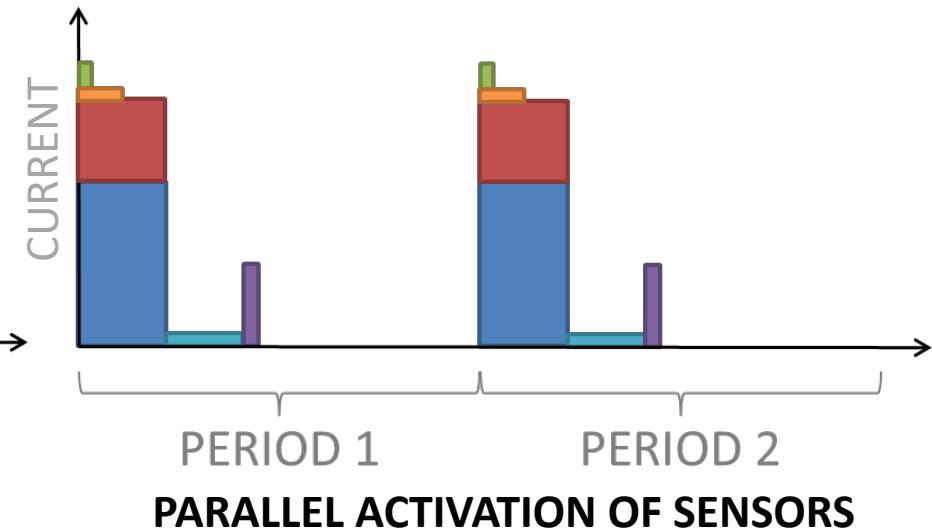
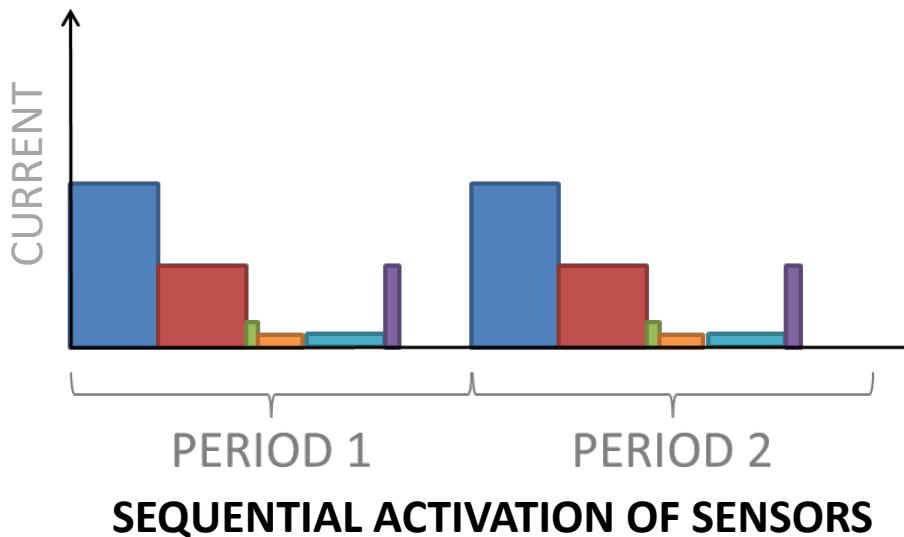
# Loads

- 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

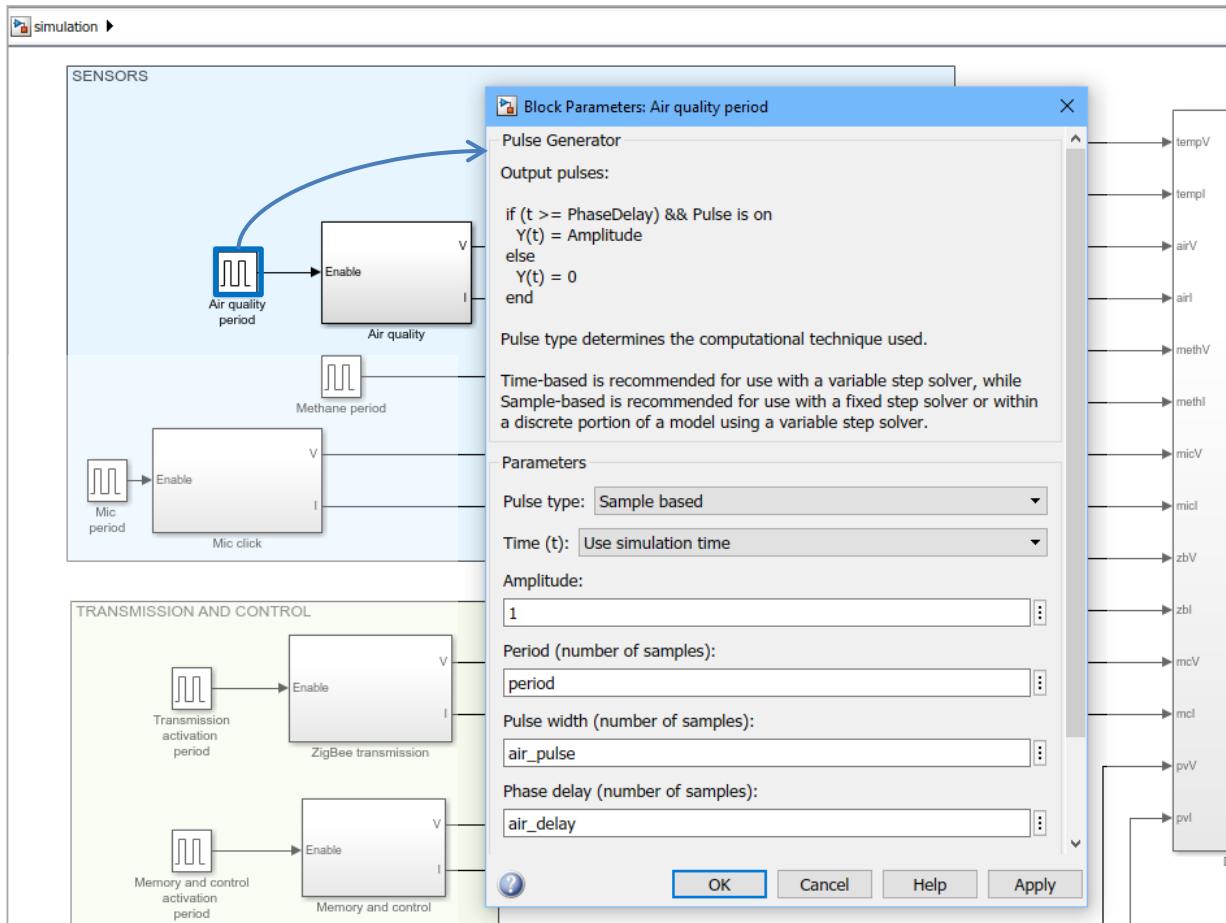
# Loads

- 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



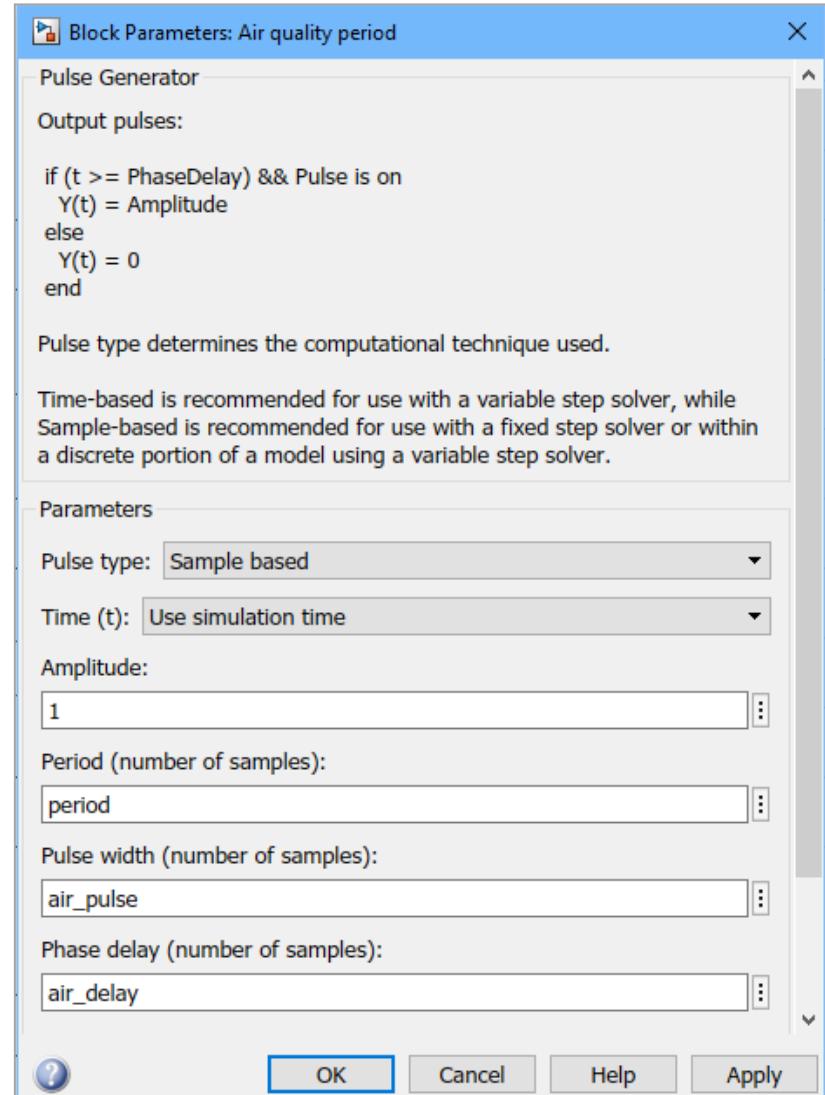
# Loads

- Who controls load activation?



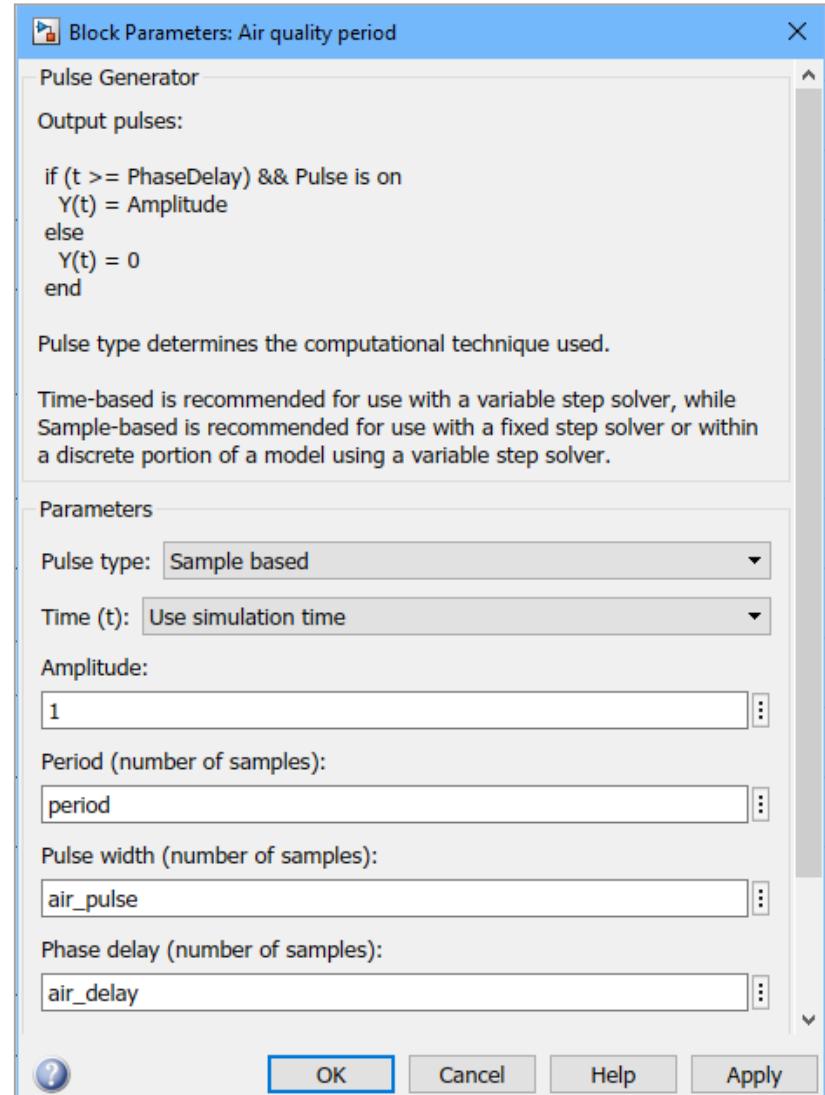
# Loads

- **Pulse generator:**
  - Define the *period*
    - Equal for all loads
  - Define the *delay*
    - 0 if load starts at the beginning of each period
    - Else equal to the computation time of preceding loads
  - Define *pulse width*
    - % of period in which the load is active
    - $\frac{\text{ACTIVE TIME} \cdot 100}{\text{PERIOD}}$



# Loads

- **Pulse generator:**
  - Loads variables from the workspace
    - E.g., for the air quality sensor:
      - Period = `period` variable
      - Delay = `air_delay` variable
      - Pulse width = `air_pulse` variable
    - Starting values provided in the **config.m** file
      - Must be executed before you run the simulation!



LOAD IRRADIANCE TRACE

LENGTH OF  
ACTIVE TIME FOR  
EACH LOAD

ACTIVATION DELAY FOR EACH LOAD  
(0 MEANS THAT LOAD IS ACTIVATED AT  
THE BEGINNING OF EACH PERIOD)  
HERE SENSORS ARE ACTIVATED IN  
PARALLEL

ACTIVATION PERIOD  
(HERE 120s)

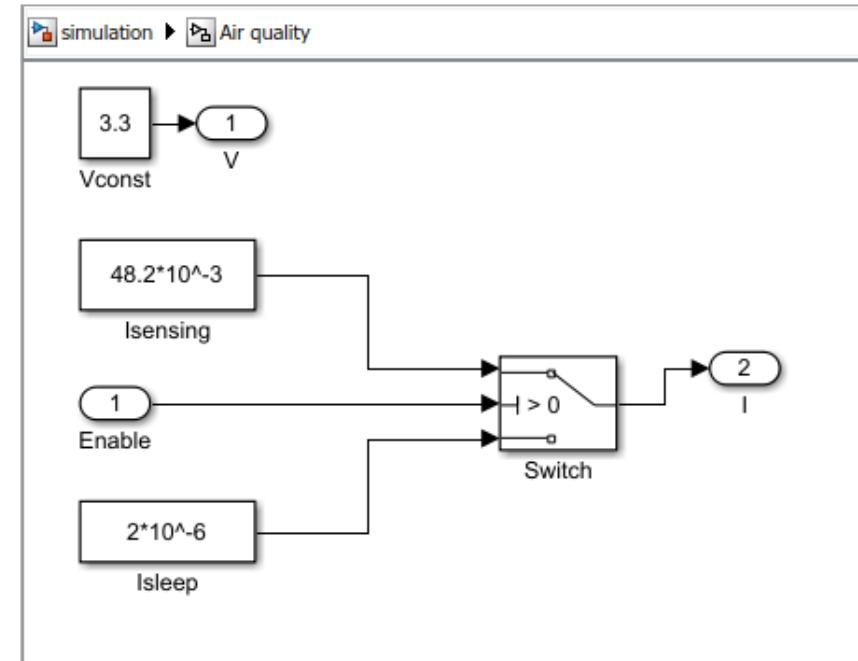
PULSE WIDTH FOR EACH LOAD  
$$\left( \frac{\text{ACTIVE TIME} \cdot 100}{\text{PERIOD}} \right)$$

SIMULATION LENGTH  
(HERE 30 days)

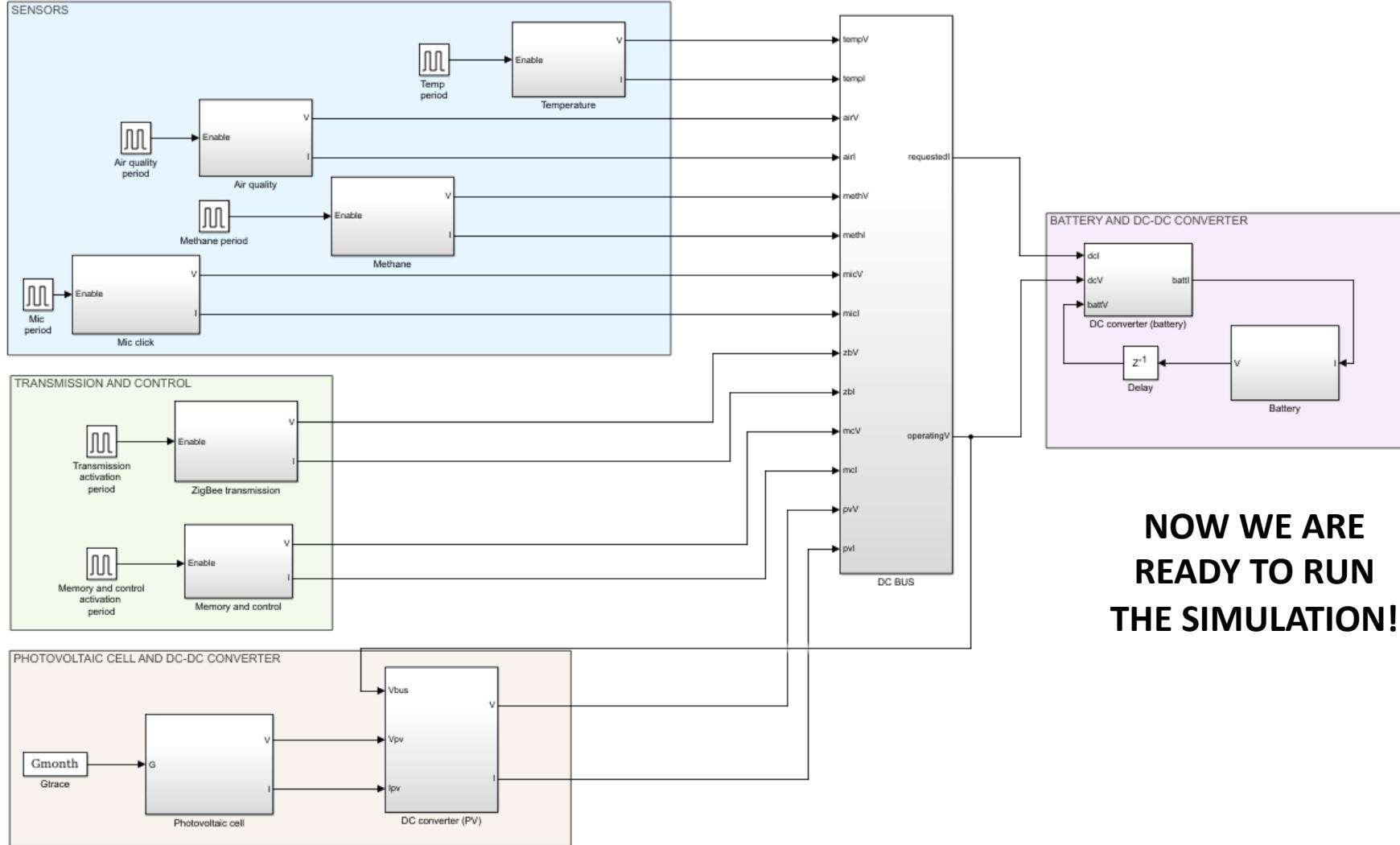
```
config.m x +  
1 -     load('gmonth.mat')  
2 -  
3 -     % sensor activation duration  
4 -     air_time = 30;  
5 -     methane_time = 30;  
6 -     temp_time = 6;  
7 -     mic_time = 12;  
8 -     transmit_time = 24;  
9 -     mc_time = 6;  
10 -  
11 -     % activation delay  
12 -     % parallel exec  
13 -     air_delay = 0;  
14 -     methane_delay = 0;  
15 -     temp_delay = 0;  
16 -     mic_delay = 0;  
17 -     mc_delay = 30;  
18 -     transmit_delay = mc_delay + mc_time;  
19 -  
20 -     % period  
21 -     % setting 1: 120 (2 minutes)  
22 -     % setting 2: 60*10 (10 minutes)  
23 -     period = 120;  
24 -  
25 -     % pulse width: % of period when sensor is active  
26 -     air_pulse = (air_time * 100)/period;  
27 -     methane_pulse = (methane_time * 100)/period;  
28 -     temp_pulse = (temp_time * 100)/period;  
29 -     mic_pulse = (mic_time * 100)/period;  
30 -     mc_pulse = (mc_time * 100)/period;  
31 -     transmit_pulse = (transmit_time * 100)/period;  
32 -  
33 -     % simulation length  
34 -     % max simulation length = 1 month (30 days)  
35 -     length = 30*24*60*60;
```

# Loads

- Loads react to the pulse:
  - Pulse is 1 → higher current value
    - Active load
  - Pulse is 0 → lower current value
    - Non active load
  - Voltage is fixed
    - Always 3.3V



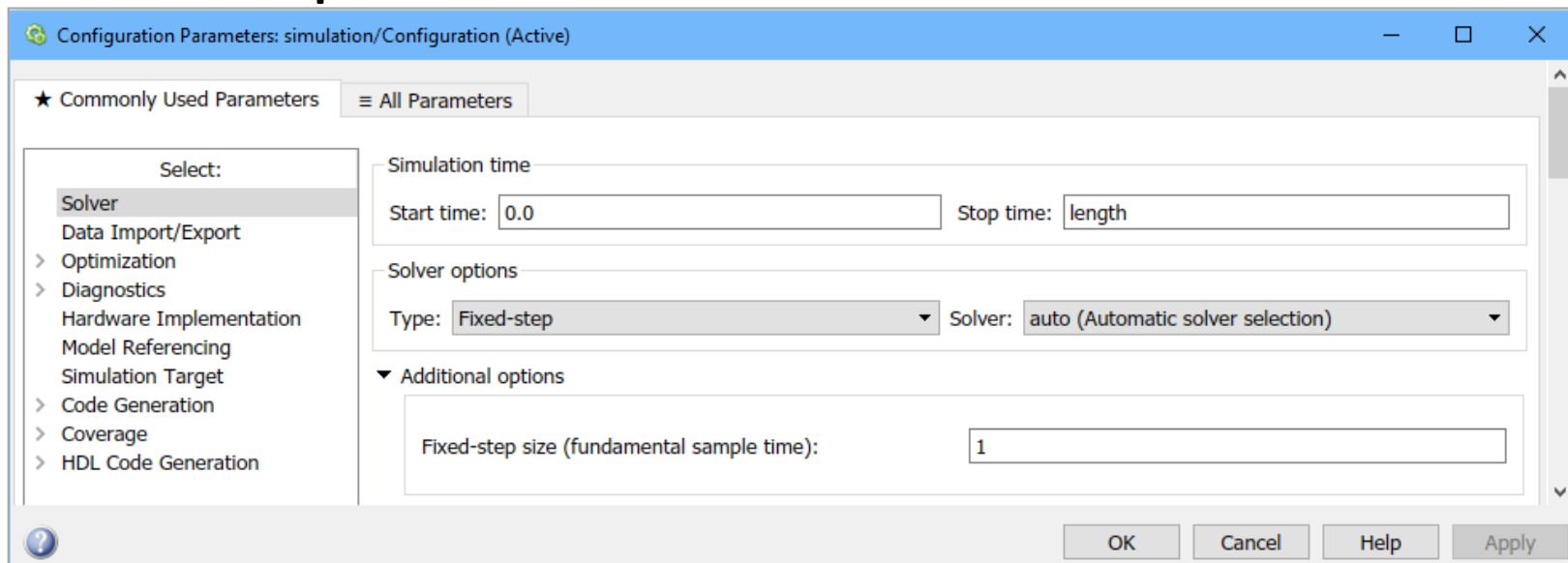
# Simulink implementation



**NOW WE ARE  
READY TO RUN  
THE SIMULATION!**

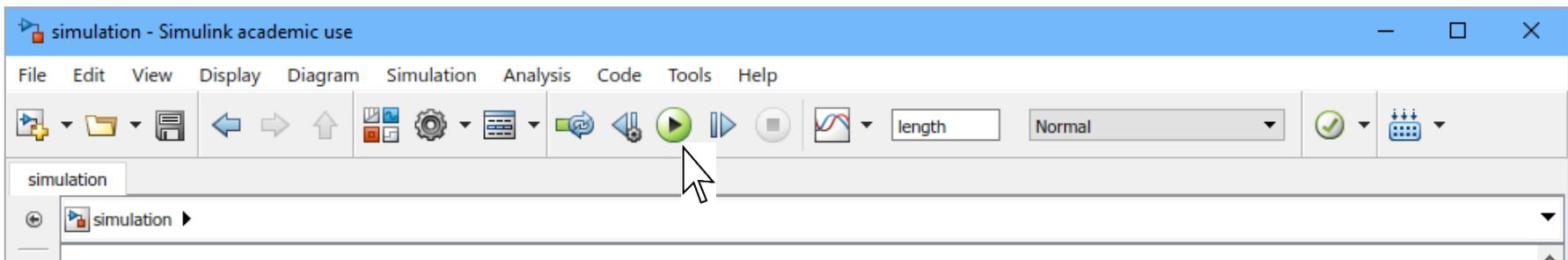
# Simulation settings

- Already set – no need to change them!
- Simulation length: `sim_length` variable
  - Set equal to the time spanned by the Gmonths trace.
- Timestep: 1 second



# Simulation settings

- To run the simulation:
  - Click on the green play button
  - Simulation stops when:
    - You reached the defined simulation length (30 days)
    - Any stop has been activated
      - E.g., battery discharged



# 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 qualities
    - 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
      - Any impact on the evolution battery?
    - **If no impact, why???**

# 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
  - Longer irradiance trace available (Gmonths.mat)
    - Up to 3 months

# Things to bear in mind:

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