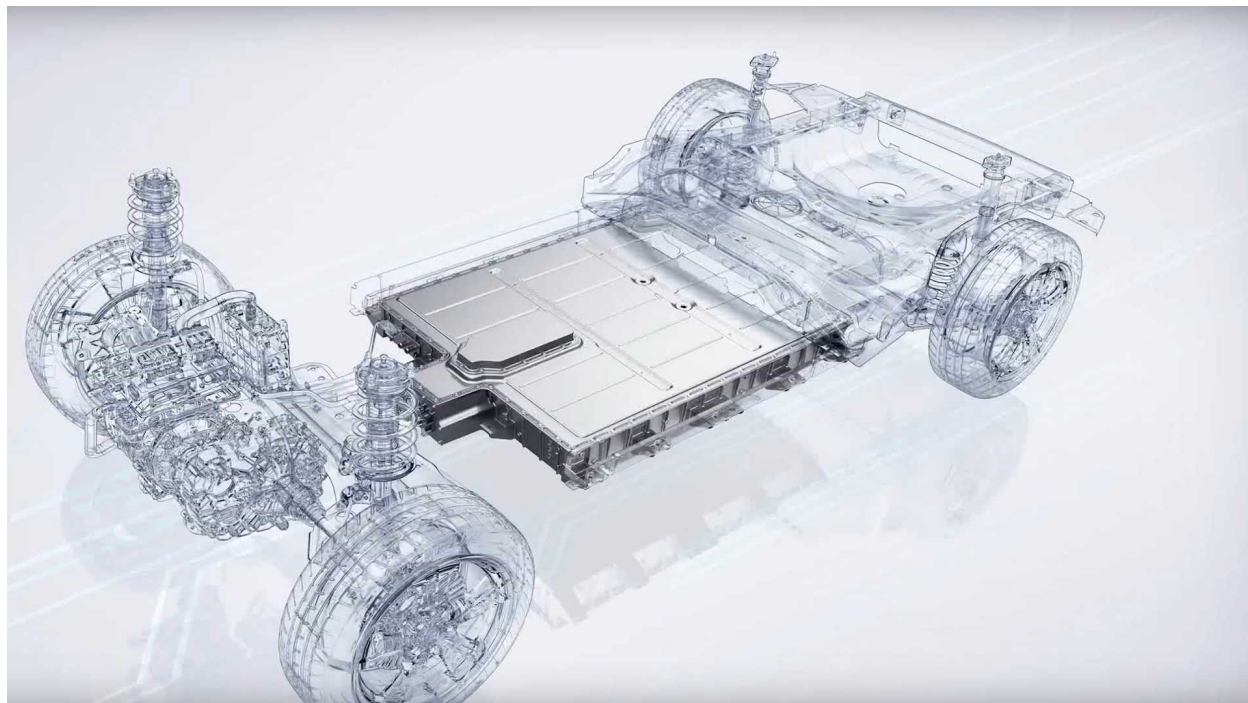


BATTERY STATE OF CHARGE ESTIMATION

UNDER THE GUIDANCE OF - PROF. VIVEK N AGARWAL



Faisal Aziz	190010022
Maddi Sanskar	190010041
Pondreti Dinesh	190010052
Apoorva Janawlekar	190010010
Riya Agrawal	19D110015

Introduction

An electric-vehicle battery (EVB) (also known as a traction battery) is a battery used to power the electric motors of a battery electric vehicle (BEV) or hybrid electric vehicle (HEV). These batteries are usually rechargeable (secondary) batteries and are typically lithium-ion batteries. These batteries are specifically designed for a high ampere-hour (or kilowatt-hour) capacity. The most common battery type in modern electric vehicles is lithium-ion and lithium-polymer, because of their high energy density compared to their weight. The amount of electricity (i.e. electric charge) stored in batteries is measured in ampere-hours or coulombs, with the total energy often measured in kilowatt-hours.

Problem Statement

A battery in an electric vehicle stores the energy that is then used to drive the vehicle. As a result, it's critical to fully charge the battery to ensure a good range of travel. There are many charging methods to ensure that the battery is charged quickly, safely, and effectively. The battery should not be deep-discharged or overcharged to prolong its lifespan. As a result, estimating the battery State of Charge (SoC) value, which means the amount of battery power available for use, becomes critical.

Implement a battery SoC estimator in Verilog HDL that uses the Coulomb-Counting method for each of the cases mentioned below. Assume that for all cases, the initial SoC and the time (T) after which the SoC must be estimated are given as inputs.

$$SoC = SoC_i + \int_0^T I(t)dt$$

1. Constant Current (CC) charging i.e., the current supplied to the battery (I_b) is constant, $I_b = k$, where k is given as an input.
2. The current supplied to the battery varies as a ramp (given as input), with a slope of α , i.e., $I_b = \alpha t$.
3. Assume that initially, the battery is charging with CC until its SoC reaches 70%. Then it shifts to Constant Voltage (CV), where the current decreases exponentially as $I_b = e^{-\beta t}$, where β is an exponential coefficient given as input.

Assumptions and parameters

- Battery: 21.5 kWh lithium-ion
- Battery Charge Time: 11.5h at 220V
- Maximum Charge: 350,000C (estimated from above values)
- $SoC_{initial}=10\%$ of total charge = 35,000C
- Time is taken in minutes

Calculations and Comparisons

1. **Constant current input:** We take $I(t)$ as some constant k .

$$SoC = SoC_i + \int_0^T I(t)dt$$

$$\Rightarrow SoC = SoC_i + kt$$

Analytical Calculations

- **Time of charging = 690 Minutes, Current = 7 A**

$$SoC_i = 10\%; \quad SoC_{t_o} = SoC_i + (I \times t \times 100)/Q_{max};$$

$$\Rightarrow SoC_{t_o} = 10\% + 7 \times 690 \times 60 \times 2.85 \times 10^{-4} = 92.593\%$$

- **Time of charging = 335 Minutes, Current = 8 A**

$$SoC_i = 10\%; \quad SoC_{t_o} = SoC_i + (I \times t \times 100)/Q_{max};$$

$$\Rightarrow SoC_{t_o} = 10\% + 8 \times 335 \times 60 \times 2.85 \times 10^{-4} = 55.828\%$$

- **Time of charging = 800 Minutes, Current = 5 A**

$$SoC_i = 10\%; \quad SoC_{t_o} = SoC_i + (I \times t \times 100)/Q_{max};$$

$$\Rightarrow SoC_{t_o} = 10\% + 5 \times 800 \times 60 \times 2.85 \times 10^{-4} = 78.4\%$$

Module Design

constant_int

This module gives the percentage of charge added to the SoC_i for a given value of *I* and *time*

```
module constant_int(                                //testbench
input [10:0] I,                                    module tb_const();
input [10:0] t,                                    reg [10:0] t,i;
output reg [10:0] c);                             wire [10:0] c;
real time_sec, Ic, factor, out;                  constant_int cl(i,t,c);
integer charge;                                  initial
always@(*)                                       begin
begin                                           #50
factor = 2.85e-4;                             i = 3'b111; t = 16'b00000001010110010;
time_sec=t*60;                                end
Ic=I;                                           endmodule
out = Ic*time_sec*factor+10;
charge = out;
c = charge;
end
endmodule
```

Input Ports

I (11-bit binary): Input port for the value of the current

t (11-bit binary): Input port for the value of time

Output Port

c (11-bit binary register): Output register to store the value of percentage of charge

Variables

real time_sec: Typecasting binary value of time into real data-type for computation

real factor: Multiplied to change the value of charge into percentage

real out: Store the calculated value of charge percentage

real Ic: Typecasting binary value of **current** into real data-type for computation

integer charge: Typecasting *real out* into integer

Calculations

$$time_{sec} = t \times 60; \quad I_c = I; \quad factor = 2.85 \times 10^{-4};$$

$$out = I_c \times time_{sec} \times factor;$$

Verilog Output

- **$t = 690$ minutes, $i = 7$ A**

Analytical Value: 92.593%

Verilog Output: 93%

Error in Computation: 0.439%

/tb_const/t	690	690
/tb_const/i	7	7
/tb_const/c	93	93

- **$t = 335$ minutes, $i = 8$ A**

Analytical Value: 55.828%

Verilog Output: 56%

Error in Computation: 0.308%

/tb_const/t	335	335
/tb_const/i	8	8
/tb_const/c	56	56

- **$t = 800$ minutes, $i = 5$ A**

Analytical Value: 78.4%

Verilog Output: 78%

Error in Computation: 0.51%

/tb_const/t	800	800
/tb_const/i	5	5
/tb_const/c	78	78

2. Ramp current input: We take $I(t)$ as $I(t) = \alpha t$.

Assumptions: $\alpha = 33 \times 10^{-5}$

$$SoC = SoC_i + \int_0^T I(t) dt$$

$$\Rightarrow SoC = SoC_i + \frac{\alpha t^2}{2}$$

Analytical Calculations

- **Time of charging = 690 Minutes**

$$SoC_i = 10\%; \quad SoC_{t_o} = SoC_i + (\alpha \times t^2 \times 100)/2 \times Q_{max};$$

$$\Rightarrow SoC_t = 10\% + 33 \times 10^{-5} \times (690 \times 60)^2 \times 2.85 \times 10^{-4}/2 = 90.599\%$$

- **Time of charging = 335 Minutes**

$$SoC_i = 10\%; \quad SoC_{t_o} = SoC_i + (\alpha \times t^2 \times 100)/2 \times Q_{max};$$

$$\Rightarrow SoC_t = 10\% + 33 \times 10^{-5} \times (335 \times 60)^2 \times 2.85 \times 10^{-4}/2 = 28.999\%$$

Module Design

```
module ramp_int(  
input [10:0] a,  
input [10:0] t,  
output reg [10:0] c);  
real time_sec, factor, out, slope;  
integer charge;  
always@(*)  
begin  
factor = 2.85e-4;  
time_sec=t*60;  
slope=a*1e-5;  
out = (slope*time_sec*time_sec*factor/2)+10;  
charge=out;  
c=charge;  
end  
endmodule
```

```
//testbench  
module tb_ramp();  
reg [10:0] a,t;  
wire [10:0] c;  
ramp_int cl(a,t,c);  
initial  
begin  
#50  
a = 8'b00100001;  
t = 16'b0000001010110010;  
end  
endmodule
```

Input Ports

a (11-bit binary): Input port for the value of the slope

t (11-bit binary): Input port for the value of time

Output Port

c (11-bit binary register): Output register to store the value of percentage of charge

Variables

real time_sec: Typecasting binary value of time into real data-type for computation

real factor: Multiplied to change the value of charge into percentage

real out: Store the calculated value of charge percentage

real slope: Typecasting binary value of *a* into real data-type for computation

integer charge: Typecasting *real out* into integer

Calculations

$$time_{sec} = t \times 60; \quad slope = a \times 10^{-5}; \quad factor = 2.85 \times 10^{-4};$$

$$out = (slope \times (time_{sec})^2 \times factor) / 2$$

Verilog Output

- *t* = 690 minutes, *i* = 7 A

Analytical Value: 90.559%

Verilog Output: 91%

Error in Computation: 0.443%

/tb_ramp/a	33	33
/tb_ramp/t	690	690
/tb_ramp/c	91	91

- *t* = 335 minutes, *i* = 8 A

Analytical Value: 28.999%

Verilog Output: 29%

Error in Computation: 0.0035%

/tb_ramp/a	33	33
/tb_ramp/t	335	335
/tb_ramp/c	29	29

3. Constant Current + Exponential Decay:

Assumptions: Constant Current (I_b) = 8A;

$$\text{Beta} = 11 \times 10^{-5} / \text{sec};$$

a. Constant Current Phase: $10\% < \text{SoC} < 70\%$

$$\text{Total Charge} = 350,000 \text{ C}$$

$$\text{Time required for constant current charging } t_o = \frac{60\% \text{ of } Q_{\text{total}}}{I_b} = \frac{0.6 \times 350,000}{8 \times 60} = 437.5$$

Therefore, the range of time for Constant Current Case is $0 \text{ min} < t < 437.5 \text{ min}$

b. Exponentially decaying Current Phase: $70\% < \text{SoC} < 90\%$

For $t > t_o$:

$$\begin{aligned} \text{SoC} &= \text{SoC}_i + kt_o + \int_0^{t-t_o} e^{-\beta t} dt \\ \Rightarrow \text{SoC} &= \text{SoC}_i + kt_o + \frac{1 - e^{-\beta(t-t_o)}}{\beta} \end{aligned}$$

Analytical Calculations

- Time of charging = 690 Minutes**

$$\text{For } 0 < t < t_o = 437.5 \text{ minutes}$$

$$\text{SoC}_i = 10\%; \quad \text{SoC}_{t_o} = \text{SoC}_i + (I \times t_o \times 100) / Q_{\text{max}};$$

$$\Rightarrow \text{SoC}_{t_o} = 10\% + 8 \times 437.5 \times 60 \times 2.85 \times 10^{-4} = 70\%$$

$$\text{For } t_o < t < \text{Time of charging} = 690 \text{ minutes}$$

$$\text{SoC}_i = 70\%; \quad \text{SoC} = \text{SoC}_i + (1 - \exp[-\beta(\text{time}_{\text{min}} - t_o)]) / \beta \times \frac{100}{Q_{\text{max}}};$$

$$\Rightarrow \text{SoC}_{t_o} = 70\% + \frac{8 \times (1 - \exp[-11 \times 10^{-5} (252.5 \times 60)]) \times 2.85 \times 10^{-4}}{11 \times 10^{-5}} = 86.811\%$$

- Time of charging = 400 minutes**

$$t < t_o = 437.5 \text{ minutes}$$

$$SoC_i = 10\%; \quad SoC = SoC_i + (I \times t_o \times 100)/Q_{max};$$

$$SoC = 10\% + 8 \times 400 \times 60 \times 2.85 \times 10^{-4} = 64.72\%$$

Module Design

a. exp_int

This module gives the percentage of charge added to the SoC_i for a given value of β and *time*

```
module exp_int(
input [10:0] beta,
input [10:0] time_min,
output reg [10:0] c);
real b, t, x, out, factor, I;
integer charge;

always@(*)
begin
I = 8;
factor = 2.85e-4;
b = beta*1e-5; t = time_min*60;
x = -(b*t);
out = (I*(5'd1 - $exp(x))/b)*factor;
charge = out;
c = charge;
end
endmodule
```

Input Ports

beta (11-bit binary): Input port for the value of β

time_min (11-bit binary): Input port for the value of **time** in minutes

Output Port

c (11-bit binary register): Output register to store the value of charge percentage

Variables

real b: Typecasting binary value of **beta** into real data type for computation

real t: Typecasting binary value of **time_min** into real data type for computation

real x: $x = -(b \times t)$

real out: Stores the percentage of charge to be added to SoC_i

real factor: Multiplied to convert the value of charge to percentage

real I: Value of constant current

integer charge: Typecasting real value of *out* into integer to round-off the value

Calculations

$$b = \beta \times 10^{-5}; \quad t = time_{min} \times 60; \quad x = -(b \times t);$$

$$factor = 2.85 \times 10^{-4}; \quad out = I \times \frac{1 - \exp(x)}{\beta} \times factor$$

b. comp

We need to know that for the given value of *time_min*, does the charge cross 70% of the total charge or not. So for a given **time_min**, this module checks whether the charge corresponding to the given time crosses 70% or not, and provides the time values to be given to modules **constant_int** and **exp_int** separately.

```
module comp (
input [10:0] In1,
input [10:0] In2,
output reg [10:0] t1,
output reg [10:0] t2);

always @ (In1 or In2)
begin
    t1 = (In1 > In2)? (In1 - In2) : 1'b0;
    t2 = (In1 > In2)? In2 : In1;
end
endmodule
```

Input Ports

In1 (11-bit binary): Input port for the value of **time_min**

In2 (11-bit binary): Input port for the value of time (**t0**) corresponding to 70% charge

Output Ports

t1 (11-bit binary register): Output register to store the value of the time for **exp_int**

Logic: If the value of **time_min** is greater than **t0**, **exp_int** would get the time input as (**time_min - t0**), otherwise, **exp_int** would get the time input as **0**

t2 (11-bit binary register): Output register to store the value of the time for **constant_int**

Logic: If the value of **time_min** is greater than **t0**, **constant_int** will run for **t0** time duration to charge the SoC_i up to 70%, else, **constant_int** will run for the entire period of **time_min**

c. case_3

This module is to integrate together with the entire logic for the third case. It gives the final amount of charge percentage for a given amount of time t for the third case

```
module case_3(                                     //testbench
input [10:0] b,                                     module tb();
input [10:0] t,                                     reg [10:0] b, t, t0, I;
input [10:0] t0,                                    wire [10:0] c;
input [10:0] I,                                     case_3 Cl(b, t, t0, I, c);
output reg [10:0] c);
wire [10:0] t1, t2, c1, c2;

comp C(t, t0, t1, t2);                             initial
exp_int E(b, t1, c1);                               begin
constant_int Ci(I, t2, c2);                         b = 4'b1011;
                                                    t = 10'b1010110010;
                                                    t0 = 9'b110110101;
                                                    I = 4'b1000;
                                                    end
                                                    endmodule

always@(*)
begin
c = c1 + c2;
end

endmodule
```

Input Ports

b (11-bit binary): Input port for the value of β

t (11-bit binary): Input port for the value of **time_{min}**

t0 (11-bit binary): Input port for the value of **t0** (defined in *module comp*)

I (11-bit binary): Input port for the value of **constant current**

Output Port

c (11-bit binary register): Output register to store the value of SoC_i

Wire

t1 (11 bit binary) : Input time value for **exp_int**

t2 (11 bit binary) : Input time value for **constant_int**











c1 (11 bit binary) : Output charge percentage value for **exp_int**

c2 (11 bit binary) : Output charge percentage value for **constant_int**

Note: *module testbench* is just to initiate the input values with ease

Verilog Output

- ***time_min* = 690 minutes**











  /tb/b	11	11
  /tb/t	690	690
  /tb/t0	437	437
  /tb/I	8	8
  /tb/c	87	87

Analytical Value: 86.811%

Verilog Output: 87%

Error in Computation: 0.22%

- ***time_min* = 400 minutes**

  /tb/b	11	11
  /tb/t	400	400
  /tb/t0	437	437
  /tb/I	8	8
  /tb/c	65	65

Analytical Value: 64.72%

Verilog Output: 65%

Error in computation: 0.43%

Conclusion

Finally, after testing out different functions of current, we conclude that the Constant Current + Exponential Decay function is the most efficient since the error obtained is the least and comes in the range of 0.22% to 0.43%. We also notice that the output obtained via Verilog matches closely with the

analytical value, and hence Verilog can be used while designing and implementing an efficient SoC of an EV battery based on the most efficient current function obtained following the above algorithm.