BATTERY STATE OF CHARGE ESTIMATION

UNDER THE GUIDANCE OF - PROF. VIVEK N AGARWAL

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.

PROBLEM STATEMENT

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 (Ib) 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 $I_b=e^{-ßt}$, where ß 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

CONSTANT CURRENT INPUT **01**

RAMP CURRENT INPUT **02**

CONSTANT CURRENT + EXPONENTIAL DECAY **03**

CALCULATIONS AND ASSUMPTIONS

01

CONSTANT CURRENT INPUT

ASSUMPTIONS

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%; SoC_{to}= SoC_i + (I x t x 100)/Q_{max} ;

  SoC_{to}= 10% + 7 x 690 x 60 x 2.85 x 10^{-4} = 92.593%

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

  SoC_i=10%; SoC_{to}= SoC_i + (I x t x 100)/Q_{max};

  SoC_{to}=10% + 8 x 335 x 60 x 2.85 x 10^{-4} = 55.828%

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

  SoC_i=10%; SoC_{to}=SoC_i + (I x t x 100)/Q_{max};

  SoC_{to} = 10% + 5 x 800 x 60 x 2.85 x 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*

```
//testbench
module tb_const();
reg [10:0] t,i;
wire [10:0] c;
constant_int c1(i,t,c);
initial
begin
#50
i = 3'b111; t = 16'b0000001010110010;
end
endmodule
```

```
module constant_int(
input [10:0] I,
input [10:0] t,
output reg [10:0] c);
real time_sec, Ic, factor, out;
integer charge;
always@(*)
begin
factor = 2.85e-4;
time_sec=t*60;
Ic=I;
out = Ic*time_sec*factor+10;
charge = out;
c = charge;
end
endmodule
```

MODULE DESIGN

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

MODULE DESIGN

Calculations

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

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

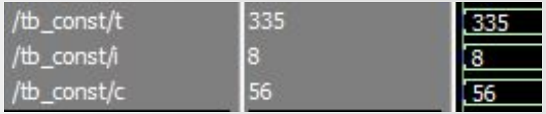
VERILOG OUTPUT

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

  Analytical Value: 92.593%
  Verilog Output: 93%
  Error in Computation: 0.439%

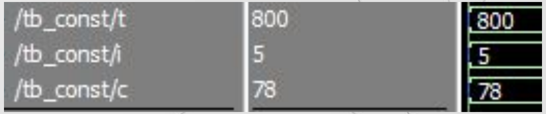
- ***t* = 335 minutes, *i* = 8 A**

  Analytical Value: 55.828%
  Verilog Output: 56%
  Error in Computation: 0.308%



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

  Analytical Value: 78.4%
  Verilog Output: 78%
  Error in Computation: 0.51%



02

RAMP CURRENT INPUT

We take I(t) as I(t)=t

ASSUMPTIONS

We take I(t) as $I(t) = \alpha t$ where $\alpha = 33 \times 10^{-5}$

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

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

ANALYTICAL CALCULATIONS

- ***Time of charging* = 690 Minutes**

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

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

- ***Time of charging* = 335 Minutes**

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

  SoC_t=10% + 33 x $10^{-5}(335 \times 60)^2 \times 2.85 \times 10^{-4}/2$ = 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 c1(a,t,c);
initial
begin
#50
a = 8'b00100001;
t = 16'b0000001010110010;
end
endmodule
```

MODULE DESIGN

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

MODULE DESIGN

Calculations

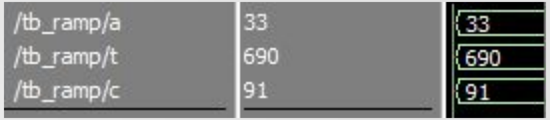
$time_{sec} = t \times 60$; $slope = \alpha \times 10^{-5}$; $factor = 2.85 \times 10^{-4}$;

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

VERILOG OUTPUT

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

  Analytical Value: 90.559%
  Verilog Output: 91%
  Error in Computation: 0.443%



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

  Analytical Value: 28.999%
  Verilog Output: 29%
  Error in Computation: 0.0035%

03

CONSTANT CURRENT + EXPONENTIAL DECAY

ASSUMPTIONS

1. Constant Current (I_b) = 8A;

   Beta = 11 x 10^{-5}/sec;

   a. **Constant Current Phase**: 10% < SoC < 70%

      Total Charge=350,000 C

      Time required for constant current charging t_O= (60% of Q_{total})/I_b=0.6 x 350,000/8 x 60 = 437.5 min

      Therefore, the range of time for Constant Current Case is 0 min < t < 437.5 min

   b. **Exponentially decaying Current Phase**: 70% < SoC < 90%

      For t>t_o :

$$SoC = SoC_i + kt_o + \int_0^{t-t_o} e^{-\beta t} dt$$

$$\Rightarrow SoC = SoC_i + kt_o + \frac{1-e^{-\beta(t-t_o)}}{\beta}$$

ANALYTICAL CALCULATIONS

- ***Time of charging* = 690 Minutes**

  For $0 < t < t_o = 437.5$ minutes

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

  $SoC_{to} = 10\% + 84 \times 37.5 \times 60 \times 2.85 \times 10^{-4} = 70\%$

  For $t_o < t <$ Time of charging = 690 minutes

  $SoC_i = 70\%;$ $SoC = SoC_i + (1 - \exp[-\beta(time_{min} - t_o)])/100 \times Q_{max};$

  $SoC_{to} = 70\% + 8 \times (1 - \exp[-11 \times 10^{-5}(252.5 \times 60)]) \times 2.85 \times 10^{-4} / 11 \times 10\text{-}5 = 86.811\%$

- ***Time of charging* = 400 minutes**

  $t < t_o = 437.5$ minutes

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

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

MODULE DESIGN

a. exp_int

This module gives the percentage of charge added to the SoC_i for a given value of 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
```

MODULE DESIGN

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 x 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

MODULE DESIGN

Calculations

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

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

MODULE DESIGN

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
```

MODULE DESIGN

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***

MODULE DESIGN

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(
input [10:0] b,
input [10:0] t,
input [10:0] t0,
input [10:0] I,
output reg [10:0] c);
wire [10:0] t1, t2, c1, c2;

comp C(t, t0, t1, t2);
exp_int E(b, t1, c1);
constant_int Ci(I, t2, c2);

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

endmodule
```

```
//testbench
module tb();
reg [10:0] b, t, t0, I;
wire [10:0] c;

case_3 C1(b, t, t0, I, c);

initial
begin
b = 4'b1011;
t = 10'b1010110010;
t0 = 9'b110110101;
I = 4'b1000;
end
endmodule
```

MODULE DESIGN

Input Ports

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

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**

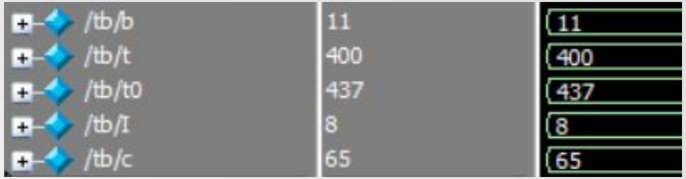
c1 (11 bit binary) : Output charge percentage value for **exp_int**

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**

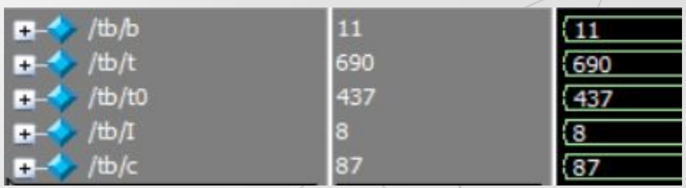


Analytical Value: 86.811%

Verilog Output: 87%

Error in Computation: 0.22%

- ***time_min* = 400 minutes**



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.

THANK YOU!