**Arys Garage – Electronics Assignment**

**1. Title & candidate details**

Title: Arys Garage – Electronics Assignment

Name: Divyashree M Narwade  
Branch: Electronics and Communication  
USN: 1MS22EC044  
Email: [divyamn612@gmail.com](mailto:divyamn612@gmail.com)  
Contact no: 8310992103

Github Link: <https://github.com/1ms22ec044/arys-electronics-divyashree-m-narwade.git>

**2. Abstract**

This assignment focuses on developing and validating electronic subsystems for battery management and power conversion. A fault detection finite state machine (FSM) was designed in SystemVerilog to monitor cell voltages, current, and temperature flags, with simulations verified using EDA Playground. A battery management system (BMS) was modeled in MATLAB/Simulink using Simscape Battery for cell-level behavior and Vehicle Network Toolbox for CAN communication, enabling state-of-charge estimation, fault detection, and real-time data transmission. For the DC-DC buck converter task, a preliminary circuit schematic was prepared outlining regulation and fault detection strategies.

**3. Tools & AI usage**

Q1 - Battery Management System (MATLAB/Simulink, CAN Integration):

* Used ChatGPT to locate relevant tutorials and YouTube resources for BMS modelling.
* Generated step-by-step methodology and suggested block connections based on available MATLAB Vehicle Network Toolbox CAN documentation link provided.

Q2 - HDL Fault Detection FSM (SystemVerilog):

* Used Google Gemini to review SystemVerilog code for logical loopholes.
* Generated a testbench using Gemini to validate FSM design under normal and fault conditions.

Q3 - DC-DC Buck Converter:

* Used Google Gemini to search for reference materials and design approaches.

Report Writing & Documentation:

Used ChatGPT to refine report sections, correct formatting/grammar, and structure the final submission.

**4. Design & methodology**

**Q1 – Battery Management System**

* The BMS model was developed using MATLAB/Simulink with Simscape Battery Toolbox and Vehicle Toolbox for CAN integration.
* Collected relevant tutorials, MATLAB documentation, and YouTube resources to understand Simscape Battery features and CAN communication setup.
* Constructed a 2s2p battery pack using Simscape Battery blocks and added voltage, current, temperature, and state of charge (SOC) logging.
* Implemented SOC estimation using the coulomb counting method.
* Integrated built-in monitoring blocks for over/undervoltage, overcurrent, and over/undertemperature detection.
* Implemented a passive cell balancing circuit for voltage equalization across series cells.
* Modeled parallel cooling paths to simulate thermal management during high loads or faults.
* Designed block connections for SOC data transmission over CAN and a basic VCU receiver to read and decode messages.

**Q2 – HDL Fault Detection FSM**

The entire design and verification process was carried out on EDA Playground, enabling iterative testing and debugging.

The fault detection system was implemented as a finite state machine (FSM) with four states:   
Normal → Warning → Fault → Shutdown.

1. FSM Diagram Development

A state transition diagram was created to outline the four operating states and the conditions under which transitions occur.

Inputs considered were cell voltages, pack current, and temperature flags.

1. Initial FSM Coding

A baseline SystemVerilog FSM was written to capture state encoding and basic transitions.

1. Fault Priority Handling

A priority mechanism was integrated to ensure that critical faults (e.g., overvoltage, overtemperature) override minor warnings.

1. Debounce & Persistence Counters

Debounce counter logic was added to filter out transient spikes.

Persistence counters were implemented to require a fault condition to remain active for a defined number of cycles before triggering a state change.

1. Masking Register Integration

A masking bits register was added to allow selective enabling/disabling of fault conditions, making the FSM more flexible in testing and operation.

1. Testbench Development

A SystemVerilog testbench was generated using Gemini by providing the FSM design code.

**Q3 – DC-DC Buck Converter**

* The input voltage is connected to a switch (usually a MOSFET), which is controlled by a control circuit.
* When the switch is closed, the input voltage is applied across the inductor, which starts to store energy in the form of a magnetic field.
* When the switch is opened, the stored energy in the inductor is released, and the energy is transferred to the output capacitor and load.
* The output voltage is regulated by adjusting the duty cycle of the switch. The duty cycle is the ratio of the time that the switch is closed to the time that it is open.
* By varying the duty cycle, the output voltage can be maintained at a constant level, even when the input voltage or load changes.
* A Buck Converter works by converting the higher voltage input to a lower voltage output by controlling the switch’s duty cycle to regulate the output voltage.

**5. Implementation details**

**Q1 – Battery Management System**

Battery pack configuration was modeled as 2s2p (two cells in series, two in parallel).

SOC calculated using coulomb counting (SOC(t) = SOC(t-1) - (I \* Δt) / Capacity).

Blocks used for Fault detection:

* Battery Cell Contact Monitoring Monitor for battery cell contact (Since R2024a)
* Battery Current Monitoring Monitor for battery current (Since R2022b)
* Battery Temperature Monitoring Monitor for battery temperature (Since R2022b)
* Battery Voltage Monitoring Monitor for battery voltage (Since R2022b)
* Fault Qualification Fault qualification algorithm (Since R2022b)

Blocks used for CAN Communication:

* CAN Configuration Configure parameters for specified CAN device
* CAN Pack Pack individual signals into CAN message
* CAN Receive Receive CAN messages from specified CAN device
* CAN Transmit Transmit CAN message to selected CAN device
* CAN Unpack Unpack individual signals from CAN messages
* CAN Replay Replay logged CAN messages
* CAN Log Log received CAN messages

**Q2 – HDL Fault Detection FSM**

The FSM was implemented in SystemVerilog using a synchronous design approach with a clocked always block for state transitions and combinational logic for next-state determination.

State Encoding:

* Normal (00)
* Warning (01)
* Fault (10)
* Shutdown (11)

Transition Logic:

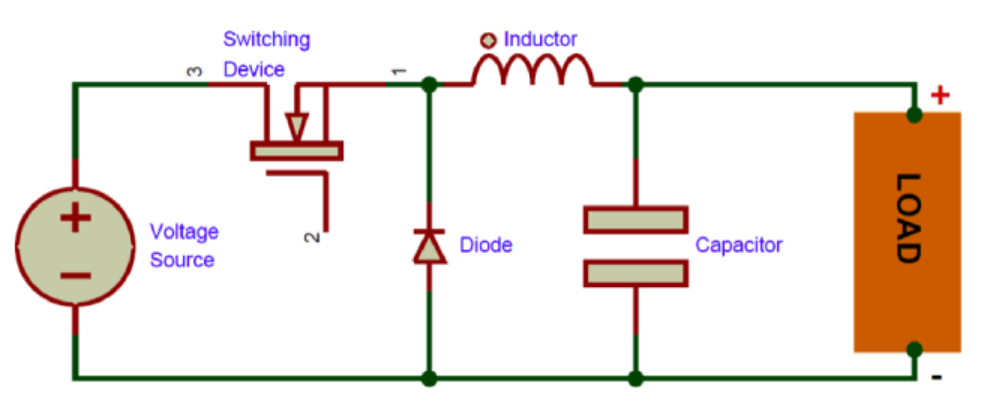
* Normal -> Warning: Triggered by non-critical faults such as undervoltage or temperature.
* Normal -> Shutdown: Triggered by highest priority fault overcurrent.
* Normal -> Fault: Triggered by overvoltage fault.
* Warning -> Fault: Triggered if fault conditions persist beyond debounce threshold or overvoltage occurs.
* Warning -> Shutdown: Triggered by highest priority fault overcurrent.
* Warning -> Normal: If all faults are cleared.
* Fault -> Shutdown: Triggered by critical fault overcurrent or if fault conditions persist beyond persistence threshold.
* Fault -> Normal: Only if faults are masked.
* Shutdown -> Normal: Only by manual reset.

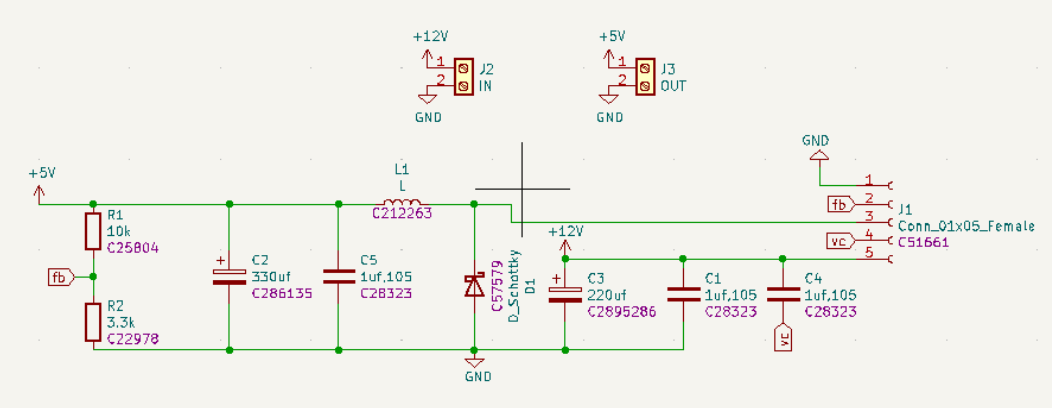
Debounce & Persistence: Implemented as counters that increment when a fault is present and reset when conditions normalize. Only when the counter exceeds a threshold does the state transition occur.

Masking Register is a configurable register allows disabling specific fault inputs for testing scenarios without altering the main FSM logic.

Testbench: Stimuli applied included nominal operating conditions, transient spikes, and sustained faults and the waveforms confirmed proper handling of persistence counters and priority rules.

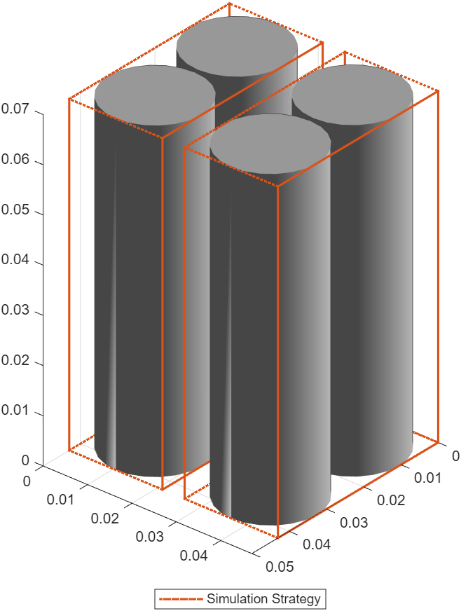
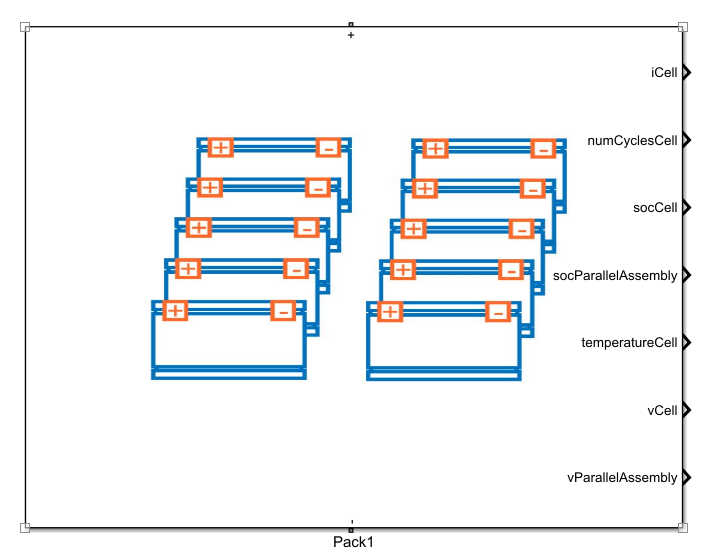
**Q3 – DC-DC Buck Converter**

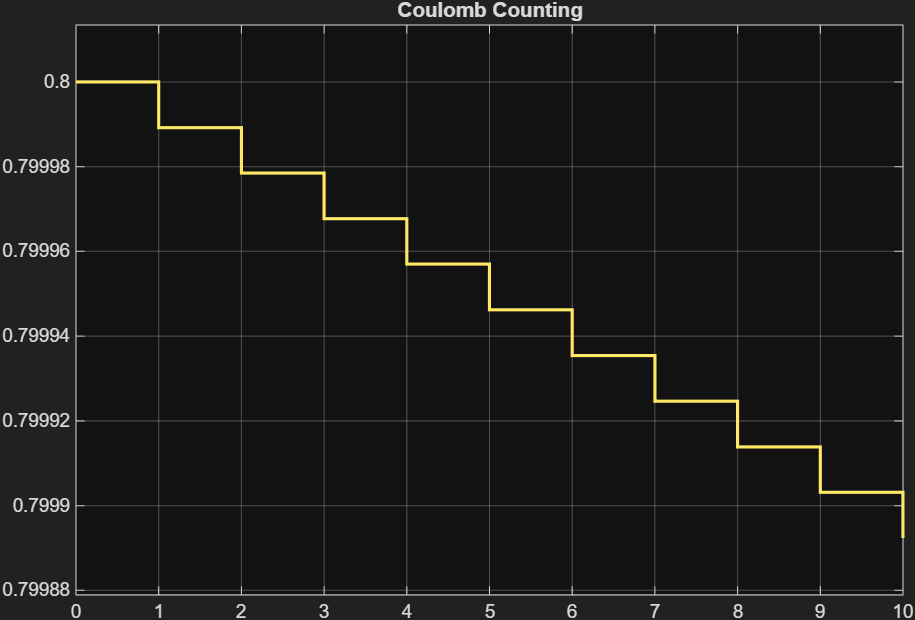
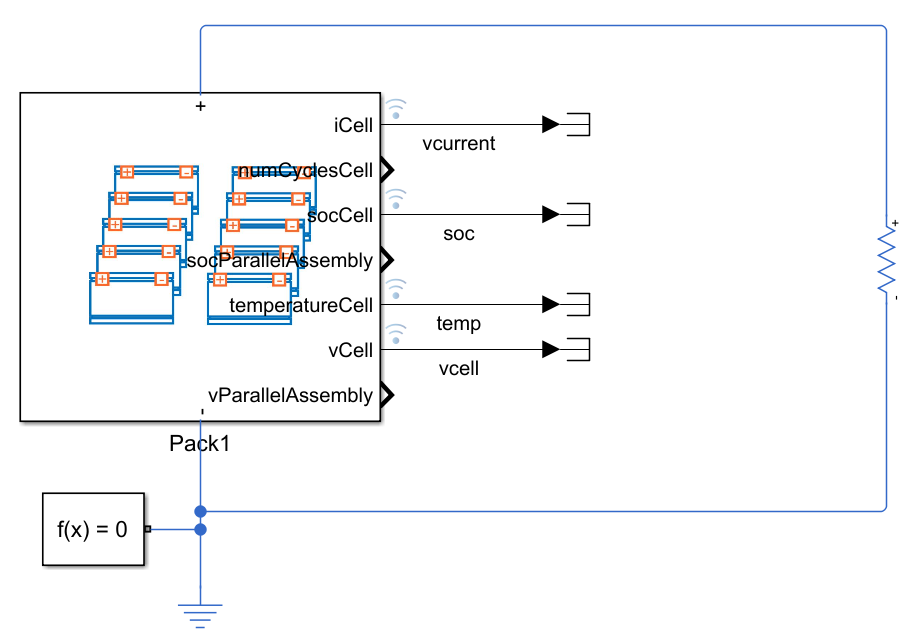




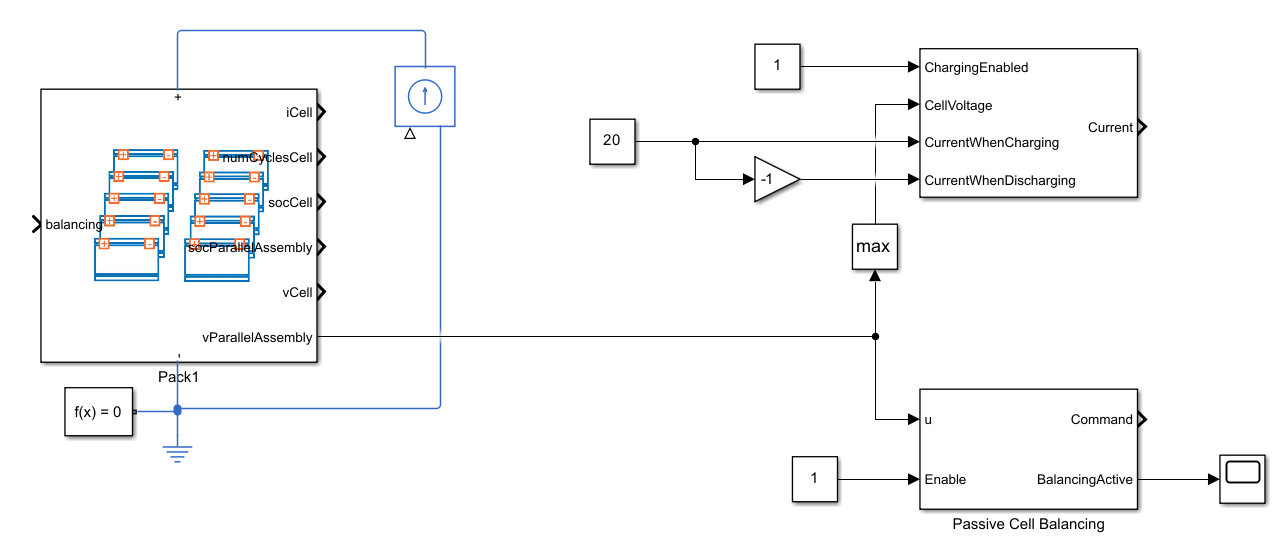
**6. Results (plots/screenshots)**

**Q1 – Battery Management System**

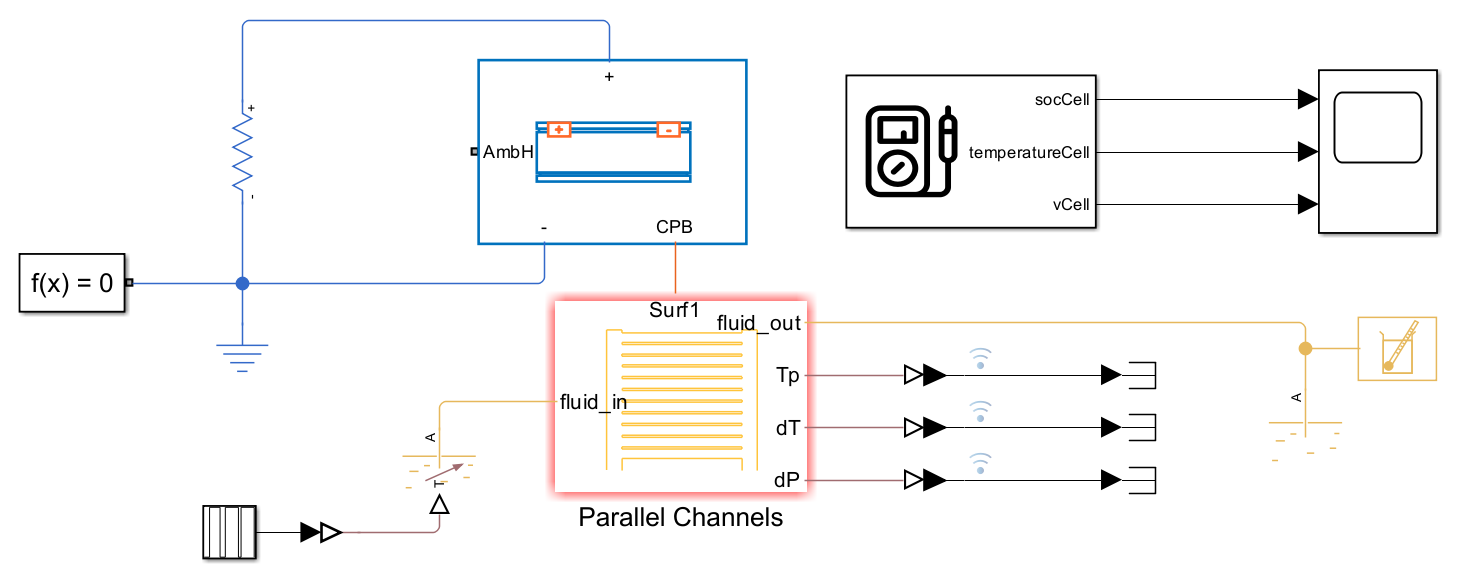
 

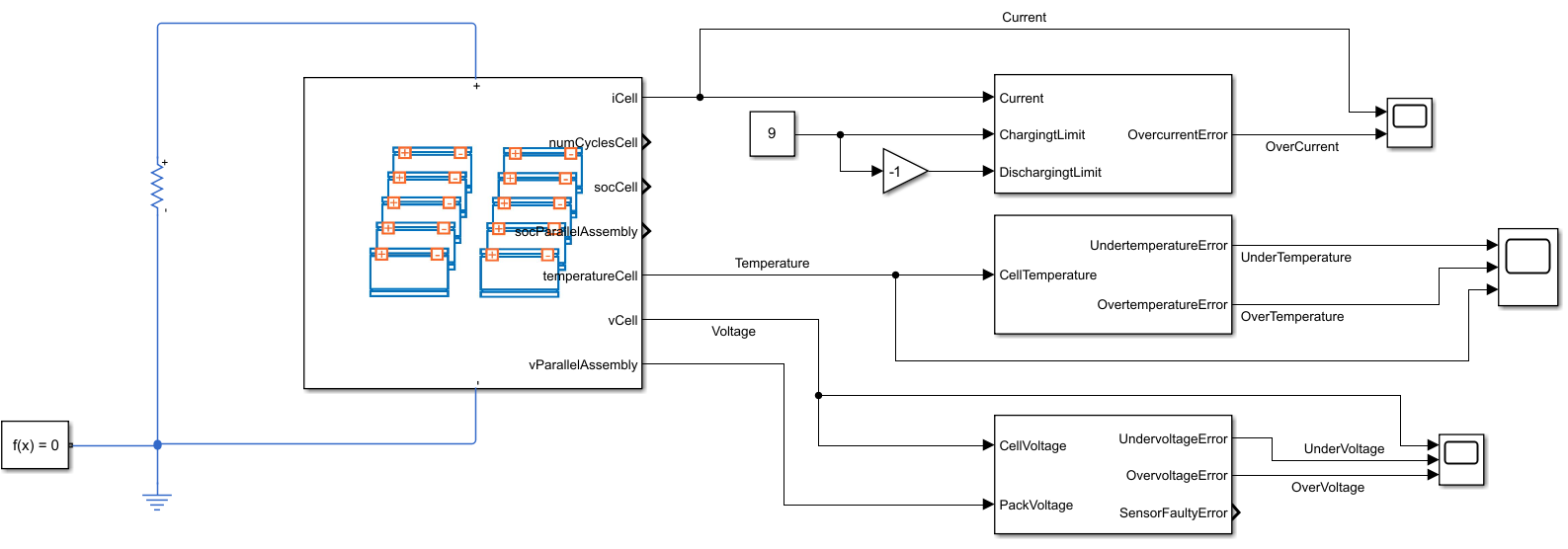
Passive Cell Balancing :



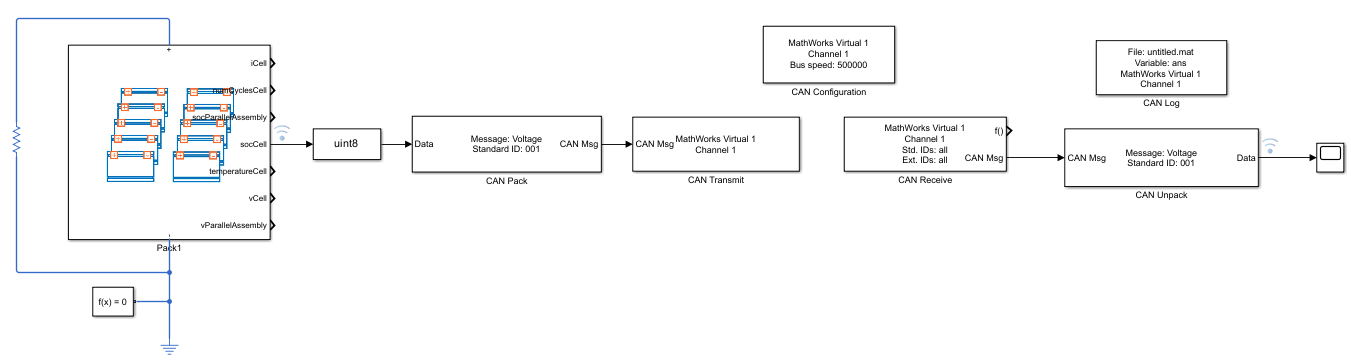
Parallel Cooling Plate Implementation :



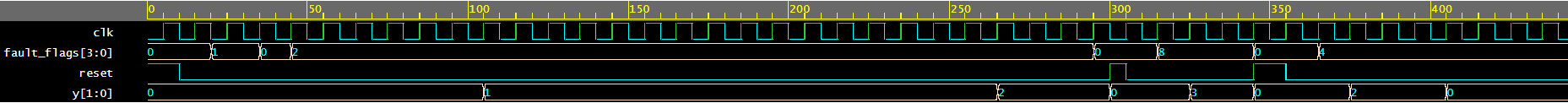
Fault Detection :

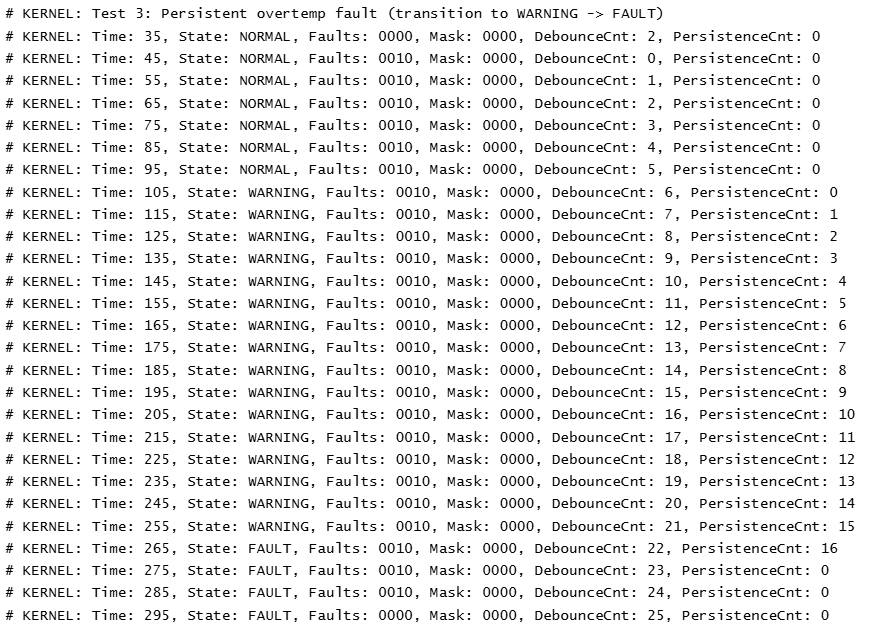


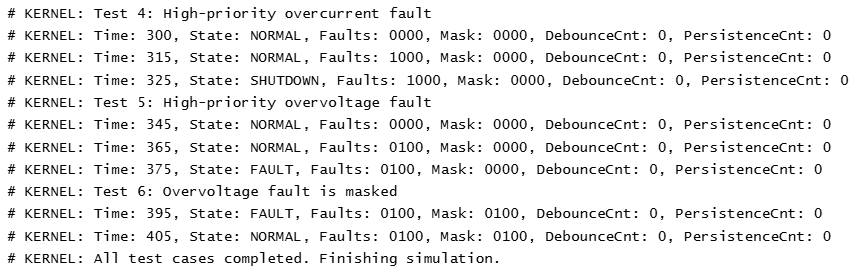
CAN Communication :



**Q2 – HDL Fault Detection FSM**







1. **Challenges & limitations**

* Gaps in CAN Integration Documentation: MATLAB resources on CAN protocol implementation were limited, with most guides covering only partial configurations rather than full end-to-end setups.
* Certain BMS features (fault detection, SOC estimation, cooling) had to be modeled in separate simulations rather than as a fully integrated system.
* Fault Threshold Selection: Voltage, current, and temperature limits were chosen from generic datasheets and may not directly apply to specific cell chemistries or hardware setups.
* CAN Bus Modeling: The CAN implementation in MATLAB was simplified to encode/decode SOC and basic signals. It does not fully replicate arbitration, bus loading, or error handling.
* Time and Scope Constraints: Due to limited time, the DC-DC converter task remained at the schematic stage, no full simulation of PI controller tuning or load fault response was completed.

1. **Conclusion**

The assignment demonstrated the design and simulation of key electronic subsystems, including a BMS with CAN integration, an HDL-based fault detection FSM, and a preliminary DC-DC converter. Core objectives such as SOC estimation, fault handling, and communication were successfully modeled and verified in simulation environments. The work highlighted practical problem-solving using MATLAB/Simulink and SystemVerilog.

1. **References**

[Simscape Battery Essentials](https://www.youtube.com/playlist?list=PLn8PRpmsu08qVPY7_YXjMzqzFqh12suV5)

<https://in.mathworks.com/help/simscape-battery/ug/build-battery-pack.html>

<https://in.mathworks.com/help/simscape-battery/ug/battery-current-temperature-fault-monitoring.html>

<https://in.mathworks.com/help/simscape-battery/bms-block-libraries.html?s_tid=CRUX_lftnav>

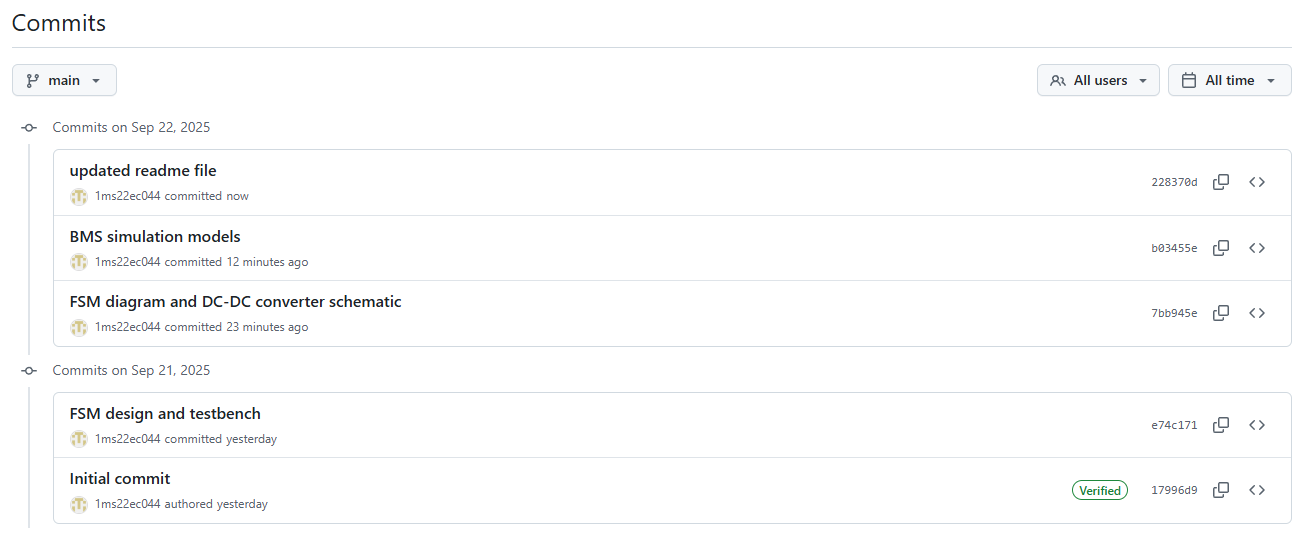
<https://in.mathworks.com/help/simscape-battery/ug/build-battery-module-thermal-effects.html>

<https://in.mathworks.com/help/vnt/can-simulink-communication.html>

<https://in.mathworks.com/help/vnt/ug/build-can-communication-simulink-models.html>

[LM2596 SIMPLE SWITCHER 4.5V to 40V](https://robu.in/wp-content/uploads/2016/03/lm2596.pdf)

**10. Appendix: Git log + run instructions**

****