**1. Tree Memory Module (ROM/RAM)**

systemverilog

module tree\_memory #(

parameter TREE\_DEPTH = 512, // Max 512 nodes (9-bit address)

parameter NODE\_WIDTH = 95 // Total bits per node

)(

input wire clk,

input wire [8:0] node\_addr, // Node index to read

output reg [8:0] node\_id, // Current node ID

output reg [1:0] feature\_idx, // Feature index (00,01,10,11)

output reg [63:0] threshold, // Q32.32 threshold

output reg [8:0] left\_child, // Left child index

output reg [8:0] right\_child, // Right child index

output reg [1:0] prediction // Prediction value

);

**Input/Output:**

* clk: Clock
* node\_addr: Địa chỉ node cần đọc (9 bit)
* node\_id: ID của node hiện tại
* feature\_idx: Index của feature cần so sánh (2 bit: 00=CAN\_ID, 01=DLC, 10=Data, 11=Reserved)
* threshold: Ngưỡng Q32.32 (64 bit)
* left\_child: Index của node con trái
* right\_child: Index của node con phải
* prediction: Kết quả dự đoán (0=Normal, 1=Attack, -1=Non-leaf)

**2. Feature Extractor Module**

systemverilog

module feature\_extractor (

input wire clk,

input wire rst\_n,

// CAN Frame Input

input wire [28:0] can\_id, // CAN ID (29-bit extended)

input wire [3:0] can\_dlc, // Data Length Code

input wire [63:0] can\_data, // 8 bytes data

input wire frame\_valid,

// Feature Output (Q32.32 format)

output reg [63:0] feature\_00, // CAN ID as Q32.32

output reg [63:0] feature\_01, // DLC as Q32.32

output reg [63:0] feature\_10, // Data as Q32.32

output reg features\_ready

);

**Input/Output:**

* can\_id: CAN ID của frame
* can\_dlc: Data Length Code
* can\_data: 8 bytes dữ liệu
* frame\_valid: Signal báo có frame mới
* feature\_00/01/10: 3 features đã convert sang Q32.32
* features\_ready: Signal báo features đã sẵn sàng

**3. Comparator Module (So sánh Q32.32)**

systemverilog

module q32\_comparator (

input wire [63:0] operand\_a, // Feature value (Q32.32)

input wire [63:0] operand\_b, // Threshold (Q32.32)

output wire a\_less\_b, // operand\_a < operand\_b

output wire a\_equal\_b, // operand\_a == operand\_b

output wire a\_greater\_b // operand\_a > operand\_b

);

**Input/Output:**

* operand\_a: Giá trị feature
* operand\_b: Ngưỡng threshold
* a\_less\_b: a < b
* a\_greater\_b: a > b

**4. Decision Tree Engine (Core logic)**

systemverilog

module decision\_tree\_engine #(

parameter MAX\_DEPTH = 20 // Maximum tree depth

)(

input wire clk,

input wire rst\_n,

// Control signals

input wire start, // Start inference

output reg busy, // Engine is processing

output reg done, // Inference complete

// Input features (Q32.32)

input wire [63:0] feature\_00,

input wire [63:0] feature\_01,

input wire [63:0] feature\_10,

// Memory interface

output reg [8:0] mem\_addr, // Address to tree memory

input wire [8:0] node\_id,

input wire [1:0] feature\_idx,

input wire [63:0] threshold,

input wire [8:0] left\_child,

input wire [8:0] right\_child,

input wire [1:0] prediction,

// Output

output reg [1:0] result, // Final prediction

output reg is\_attack // 1=Attack, 0=Normal

);

**Input/Output:**

* start: Trigger bắt đầu inference
* busy: Đang xử lý
* done: Hoàn thành inference
* feature\_00/01/10: 3 features đầu vào
* mem\_addr: Địa chỉ gửi tới tree memory
* Node data từ memory
* result: Kết quả cuối cùng
* is\_attack: Flag đơn giản (1=tấn công, 0=bình thường)

**5. TOP Module (Tích hợp tất cả)**

systemverilog

module can\_attack\_detector\_top (

input wire clk,

input wire rst\_n,

// CAN Frame Input

input wire [28:0] can\_id,

input wire [3:0] can\_dlc,

input wire [63:0] can\_data,

input wire frame\_valid,

// Detection Output

output wire detection\_done,

output wire is\_attack,

output wire [1:0] attack\_class,

// Status

output wire busy,

output wire [8:0] current\_node

);

**Input/Output:**

* CAN frame inputs
* detection\_done: Hoàn thành detection
* is\_attack: Có tấn công hay không
* attack\_class: Loại tấn công (nếu có)
* busy: Đang xử lý
* current\_node: Node hiện tại (debug)

**State Machine cho Decision Tree Engine**

IDLE → LOAD\_ROOT → FETCH\_NODE → COMPARE\_FEATURE →

DECIDE\_BRANCH → CHECK\_LEAF → OUTPUT\_RESULT → IDLE

**States:**

1. **IDLE**: Chờ start signal
2. **LOAD\_ROOT**: Load node 0 (root)
3. **FETCH\_NODE**: Đọc node data từ memory
4. **COMPARE\_FEATURE**: So sánh feature với threshold
5. **DECIDE\_BRANCH**: Quyết định đi left hay right child
6. **CHECK\_LEAF**: Kiểm tra có phải leaf node không
7. **OUTPUT\_RESULT**: Xuất kết quả

**Memory Format (.mem file)**

Mỗi dòng 95 bits (hex format):

// Format: [Node\_ID(9)][Feature(2)][Threshold(64)][Left(9)][Right(9)][Pred(2)]

// Example: Node 0

0\_0\_0000000063D6C8E000000000\_0B2\_001\_3 // Hex representation