trackforce.vivado\trackforce.srcs\sources_1\imports\imports\source\gpsdecode.sv

```systemverilog
// Macros for message ASCII
`define LF 10
`define CR 13
`define DOLLAR 36
`define COMMA 44
`define STAR 42


// Macro for message bits
`define CHAR_LENGTH 8

// gpsdecode
// ------------------------------
// Receives a NEMA message from the UART interface and decode it to signals
// used by other modules
// Only cares about GPVTG message
// Ready Valid& interface
//
// Inputs
// ------------------------------
// clk_i - Clock
// rst_i - Reset
// [7:0] data_i - ASCII char read from FIFO
// valid_i - Input message is valid
// ready_i - Next module is ready to receive data
//
// Outputs
// ------------------------------
//
// ready_o - Decoder is ready to receive a new message
// vkmh_o - Speed in km/h
//
// Example data from GPS
// $GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32\r\n
//
// TalkerID, course, Reference, Course, Reference, Speed, Unit (knot), Speed, Unit (km/h),
Mode, Checksum, CR, LF
// should get Speed before Unit (km/h)


module gpsdecode(
    input [0:0] clk_i,
    input [0:0] rst_i,
    input [7 :0] data_i,
    // data from FIFO is valid
    input [0:0] valid_i,
    // next module is ready to receive data
    input [0:0] ready_i,
    // data is ready to be read
    output [0:0] ready_o,
    output [0:0] valid_o,
    output [7:0] vkmh_o
);
```

```


logic [0:0] ready_r;
logic [0:0] valid_r;
logic [7:0] vkmh_r;

//
// State machine
//
// 0 - Idle      // Wait for $
// 1 - MsgID     // Parse message ID, go to Continue if parsed as GPVTG. When msgidindex_r == 3 and msgidmatch_r == 1, stay in msgID and go to field when msgidindex_r == 5. Else go to ignore
// 2 - Continue  // Parse message until 7th comma
// 3 - Speed     // Parse Speed
// 4 - Done      // Done parsing speed, set valid_o to 1.

typedef enum logic [2:0] {
    IDLE,
    MSGID,
    FIELD,
    IGNORE,
    DONE
} state_t;

state_t state_r, state_n;

logic [7:0] msgidindex_r;
logic [7:0] commaindex_r;
logic [7:0] speedindex_r;

logic [0:0] msgidmatch_r;

 always_ff @(posedge clk_i) begin
    if (rst_i) begin
        state_r <= IDLE;
    end
    else begin
        state_r <= state_n;
    end
 end

 // state transitions

    always_comb begin
        if (rst_i) begin
            state_n = IDLE;
        end
        else begin
            case (state_r)
                IDLE: begin
                    if (data_i == "$") begin
                        state_n = MSGID;
                    end
                    else begin
                        state_n = IDLE;
                    end
                end
                MSGID: begin
                    if (msgidmatch_r == 1 && msgidindex_r == 5) begin
                        state_n = FIELD;
                    end
                    else if (msgidmatch_r == 0 && msgidindex_r > 3) begin
                        state_n = IGNORE;
                    end
                    else begin
                        state_n = MSGID;
                    end
                end
                FIELD: begin
                    if (speedindex_r == 3) begin
                        state_n = DONE;
                    end
                    else begin
                        state_n = FIELD;
                    end
                end
                IGNORE: begin
                    if (data_i == "$") begin
                        state_n = MSGID;
                    end
                    else begin
                        state_n = IGNORE;
                    end
                end
                DONE: begin
                    state_n = IDLE;
                end
                default : begin
                    state_n = IDLE;
                end


            endcase
        end

    end


// msgindex logic
// increment once whenever a new char is read (pulse of valid_i) when in msgid state

always_ff @(posedge clk_i) begin
    if (rst_i) begin
        msgidindex_r <= 0;
    end
    else begin
        case (state_r)
            IDLE: begin
                msgidindex_r <= 0;
            end
            MSGID: begin
                if (valid_i) begin
```

```
                    msgidindex_r <= msgidindex_r + 1;
                end
                else begin
                    msgidindex_r <= msgidindex_r;
                end
            end
            FIELD: begin
                msgidindex_r <= 0;
            end
            IGNORE: begin
                msgidindex_r <= 0;
            end
            DONE: begin
                msgidindex_r <= 0;
            end
            default : begin
                msgidindex_r <= 0;
            end
        endcase
    end
end


// msgid match logic
//
// check at index 3 if the char is V
// if it is, msgidmatch_r = 1
// else msgidmatch_r = 0

always_ff @(posedge clk_i) begin
    if (rst_i) begin
        msgidmatch_r <= 0;
    end
    else begin
        case (state_r)
            IDLE: begin
                msgidmatch_r <= 0;
            end
            MSGID: begin
                if (msgidindex_r == 3) begin
                    if (data_i == "V") begin
                        msgidmatch_r <= 1;
                    end
                    else begin
                        msgidmatch_r <= msgidmatch_r;
                    end
                end
                else begin
                    msgidmatch_r <= msgidmatch_r;
                end
            end
            FIELD: begin
                msgidmatch_r <= 0;
            end
            IGNORE: begin
                msgidmatch_r <= 0;
            end
            DONE: begin
                msgidmatch_r <= 0;
            end
            default : begin
                msgidmatch_r <= 0;
            end
        endcase
    end
end






// commaindex logic

always_ff @(posedge clk_i) begin
    if (rst_i) begin
        commaindex_r <= 0;
    end
    else begin
        case (state_r)
            IDLE: begin
                commaindex_r <= 0;
            end
            MSGID: begin
                commaindex_r <= 0;
            end
            FIELD: begin
                if (valid_i) begin
                if (data_i == ",")
                commaindex_r <= commaindex_r + 1;
                end
                else begin
                    commaindex_r <= commaindex_r;
                end
            end
            IGNORE: begin
                commaindex_r <= 0;
            end
            DONE: begin
                commaindex_r <= 0;
            end
            default : begin
                commaindex_r <= 0;
            end
        endcase
    end
end


// speedindex logic

always_ff @(posedge clk_i) begin
```

```
    if (rst_i) begin
        speedindex_r <= 0;
    end
    else begin
        case (state_r)
            IDLE: begin
                speedindex_r <= 0;
            end
            MSGID: begin
                speedindex_r <= 0;
            end
            FIELD: begin
                if (valid_i) begin
                if (commaindex_r == 7) begin
                    speedindex_r <= speedindex_r + 1;
                end
                end
                else begin
                    speedindex_r <= speedindex_r;
                end
            end
            IGNORE: begin
                speedindex_r <= 0;
            end
            DONE: begin
                speedindex_r <= 0;
            end
            default : begin
                speedindex_r <= 0;
            end
        endcase
    end
end

// speed logic
logic [7:0] speed_r;

always_ff @(posedge clk_i) begin
    if (rst_i) begin
        speed_r <= 0;
    end
    else begin
        case (state_r)
            IDLE: begin
                speed_r <= 0;
            end
            MSGID: begin
                speed_r <= 0;
            end
            FIELD: begin
                if (valid_i) begin
                if (commaindex_r == 7) begin
                    if (speedindex_r == 0) begin
                        case (data_i)
                        "0" : speed_r <= 0;
                        "1" : speed_r <= 10;
                        "2" : speed_r <= 20;
                        "3" : speed_r <= 30;
                        "4" : speed_r <= 40;
                        "5" : speed_r <= 50;
                        "6" : speed_r <= 60;
                        "7" : speed_r <= 70;
                        "8" : speed_r <= 80;
                        "9" : speed_r <= 90;
                        default: speed_r <= 0;
                        endcase
                    end
                    else if (speedindex_r == 1) begin
                        case (data_i)
                        "0" : speed_r <= speed_r + 0;
                        "1" : speed_r <= speed_r + 1;
                        "2" : speed_r <= speed_r + 2;
                        "3" : speed_r <= speed_r + 3;
                        "4" : speed_r <= speed_r + 4;
                        "5" : speed_r <= speed_r + 5;
                        "6" : speed_r <= speed_r + 6;
                        "7" : speed_r <= speed_r + 7;
                        "8" : speed_r <= speed_r + 8;
                        "9" : speed_r <= speed_r + 9;
                        // this should handle empty msg
                        default: speed_r <= speed_r + 0;
                        endcase
                    end
                end
                end
                else begin
                    speed_r <= speed_r;
                end
            end
            IGNORE: begin
                speed_r <= 0;
            end
            DONE: begin
                speed_r <= 0;
            end
            default : begin
                speed_r <= 0;
            end
        endcase
    end
end



//


    always_ff @(posedge clk_i) begin
        if (rst_i) begin
            ready_r <= 0;
            valid_r <= 0;
```

```verilog
        end
        else begin
            case (state_r)
                IDLE: begin
                    ready_r <= 1;
                    valid_r <= 0;
                end
                MSGID: begin
                    ready_r <= 1;
                    valid_r <= 0;
                end
                FIELD: begin
                    if (speedindex_r == 3) begin
                    ready_r <= 0;
                    valid_r <= 1;
                    end
                    else begin
                    ready_r <= 1;
                    valid_r <= 0;
                    end
                end
                IGNORE: begin
                    ready_r <= 1;
                    valid_r <= 0;
                end
                DONE: begin
                    ready_r <= 0;
                    valid_r <= 0;
                end
                default : begin
                    ready_r <= 0;
                    valid_r <= 0;
                end
            endcase
        end
    end


    // outputs

    assign ready_o = ready_r;
    assign valid_o = valid_r;
    assign vkmh_r = speed_r;
    assign vkmh_o = vkmh_r;



endmodule
```