Generation of GPS L1C PRN Codes (L1CP and L1CD) for First 5 Satellites using Verilog

Mannava Venkatasai, Solanki Utsav, Anekar Vaishnavi, Tripathi Vaishnavi, Pavan sai Kumar Reddy Thummala, Gupta Vinayak, Vivek Sai Yerra

April 24, 2025

Abstract

This report documents the design and implementation of GPS L1C pilot (L1CP) and data (L1CD) PRN codes for the first five satellites using synthesizable Verilog. The generation method is based on the structure defined in the GPS Interface Specification. The Legendre sequence, Weil codes, and expansion sequence insertion steps are explained in detail. Simulation results confirm the correctness of the implementation.

1 Introduction

The GPS L1C signal introduces new ranging codes L1CP (pilot) and L1CD (data) that are based on a structured sequence design using Weil codes derived from the Legendre sequence. Each code is 10230 chips long, composed of a 10223-chip Weil code with a fixed 7-bit expansion sequence inserted at a specified index.

2 PRN Code Structure

2.1 L1CP and L1CD

For each PRN signal number i, the codes $L1CP_i(t)$ and $L1CD_i(t)$ are generated as:

- A Weil-code $W_i(t; w)$:

$$W_i(t; w) = L(t) \oplus L((t + w) \bmod 10223)$$

- A 7-bit expansion sequence `0110100` is inserted before the p^{th} bit of the Weil-code to produce the final code.

2.2 Legendre Sequence $L(t)$

$$L(0) = 0$$

$$L(t) = \begin{cases} 1 & \text{if } t \equiv x^2 \pmod{10223} \text{ for some } x \\ 0 & \text{otherwise} \end{cases}$$

for $t = 1$ to 10222.

2.3 Weil Code

Each satellite PRN is assigned a unique Weil index w and insertion index p. The Weil-code is the XOR of $L(t)$ and its shift by w.

2.4 Expansion Sequence

The fixed expansion sequence `0110100` is inserted before the p^{th} chip of the 10223-chip Weil-code to create the 10230-chip ranging code.

3 PRN Parameters for Satellites 1–5

PRN	L1CP Weil Index w	Insertion Index p	L1CD Weil Index w	Insertion Index p
1	5111	412	5097	181
2	5109	161	5110	359
3	5108	1	5079	72
4	5106	303	4403	1110
5	5103	207	4121	1480

Table 1: Weil and Insertion Indices for PRNs 1–5

4 Implementation in Verilog

The Verilog design includes:

- A ROM module to store the Legendre sequence $L(t)$.
- Logic to compute $W_i(t; w)$ using XOR.
- Expansion logic to insert the 7-bit sequence at position p.
- Output register to store the final 10230-chip code.

4.1 Module Description

Listing 1: Top Module

```
module prn_code_generator(
    input wire clk,
    input wire rst,
    input wire start,
    input wire [2:0] prn_id,
    input wire pd,
    output reg [13:0] addr,
    output reg out_valid,
    output reg prn_bit
);
```

```
parameter N = 10223;
reg [13:0] i;
reg [13:0] weil_index;
reg [13:0] insertion_index;
reg [13:0] insert_end;
reg [6:0] expansion_bits;

reg legendre_rom [0:N-1];

initial
begin
    expansion_bits[0] = 0;
    expansion_bits[1] = 1;
    expansion_bits[2] = 1;
    expansion_bits[3] = 0;
    expansion_bits[4] = 1;
    expansion_bits[5] = 0;
    expansion_bits[6] = 0;

    $readmemb("legendre.mem", legendre_rom);

end


always @(*)
begin
    case ({pd, prn_id})

        /* Data prn code settings */
        4'b1001: begin weil_index = 5097; insertion_index = 181; end
        4'b1010: begin weil_index = 5110; insertion_index = 359; end
        4'b1011: begin weil_index = 5079; insertion_index = 72;
end
        4'b1100: begin weil_index = 4403; insertion_index = 1110; end
        4'b1101: begin weil_index = 4121; insertion_index = 1480; end

        /* pilot prncodes settings */
        4'b0001: begin weil_index = 5111; insertion_index = 412; end
        4'b0010: begin weil_index = 5109; insertion_index = 161; end
        4'b0011: begin weil_index = 5108; insertion_index = 1;
end
        4'b0100: begin weil_index = 5106; insertion_index = 303; end
        4'b0101: begin weil_index = 5103; insertion_index = 207; end
        default: begin weil_index = 0; insertion_index = 0; end
    endcase
        insert_end = insertion_index + 7;
        //$display("pd = %d prn_id = %d\n",pd,prn_id);
```

```
        //$display("weil_index = %d insertion_index = %d insert_end = %d\n

end


assign legendre_i       = legendre_rom[i];
assign legendre_shifted = legendre_rom[(i + weil_index) % N];


always @(posedge clk or posedge rst)
begin
    if (rst)
    begin
        i <= 0;
        addr <= 0;
        out_valid <= 0;
    end
    else if (start)
    begin
        if (i < N)
        begin
            if (i >= insertion_index && i < insert_end)
            begin
                prn_bit <= expansion_bits[i - insertion_index];
            end
            else
            begin
                prn_bit <= legendre_i ^ legendre_shifted;
            end
            addr <= i;
            i <= i + 1;
            out_valid <= 1;
        end
        else
        begin
            out_valid <= 0;
        end
    end
end

endmodule
```

Listing 2: Test bench

```
`timescale 1ns / 1ps
module prn_code_generator_tb;
    reg clk;
    reg rst;
    reg start;
```

```
reg [2:0] prn_id;
reg pd;

wire [13:0] addr;
wire out_valid;
wire prn_bit;

prn_code_generator uut (
    .clk(clk),
    .rst(rst),
    .start(start),
    .prn_id(prn_id),
    .pd(pd),
    .addr(addr),
    .out_valid(out_valid),
    .prn_bit(prn_bit)
);

always #5 clk = ~clk;

integer f,i;

initial
begin

    $dumpfile("wave.vcd");
    $dumpvars(0, prn_code_generator_tb);
    clk = 0;
    rst = 1;
    start = 0;
    prn_id = 3'b001;
    pd = 1'b1;          //pd = 1 for data and pd = 0 for pilot

    f = $fopen("prn_code_output.txt", "w");
    if (!f)
    begin
        $display("Error: Could not open file.");
        $finish;
    end


    #20;
    rst = 0;
    start = 1;
    i = 0;
    prn_id = 3'b001;  //prn id
    pd = 1'b1;          //pd = 1 for data and pd = 0 for pilot
    $display("pd = %d prn_id = %d\n",pd,prn_id);
```

```
        while (i < 10230)
        begin
            @(posedge clk);
            if (out_valid)
            begin
                $fwrite(f, "%b\n", prn_bit);
                // $display("%d\n",prn_bit);
            end
            i=i+1;
        end
        $finish;
        $fclose(f);
    end
endmodule
```

5 Simulation and Results

The Legendre sequence of length 10223 is generated externally using python and imported in to the verilog module.

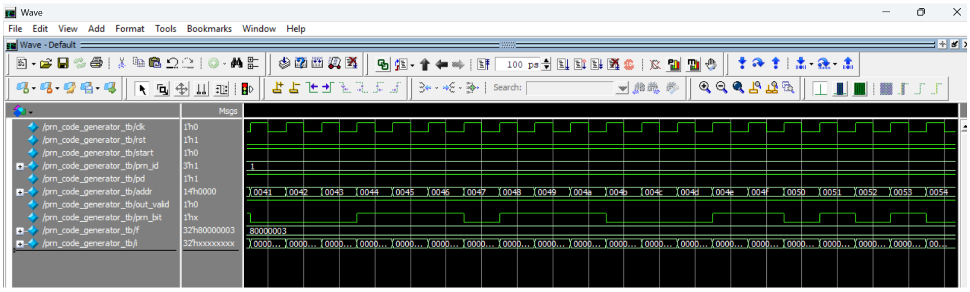

Figure 1: simulation waveforms.

```
o.dat
Data PRN 1 First 24 bits -> Expected: 0o77001425, Got: 0o77001425 -> Match: True
Data PRN 1 Last  24 bits -> Expected: 0o52231646, Got: 0o52231646 -> Match: True
Data PRN 2 First 24 bits -> Expected: 0o23342754, Got: 0o23342754 -> Match: True
Data PRN 2 Last  24 bits -> Expected: 0o46703351, Got: 0o46703351 -> Match: True
Data PRN 3 First 24 bits -> Expected: 0o30523404, Got: 0o30523404 -> Match: True
Data PRN 3 Last  24 bits -> Expected: 0o145161, Got: 0o145161 -> Match: True
Data PRN 4 First 24 bits -> Expected: 0o3777635, Got: 0o3777635 -> Match: True
Data PRN 4 Last  24 bits -> Expected: 0o11261273, Got: 0o11261273 -> Match: True
Data PRN 5 First 24 bits -> Expected: 0o10505640, Got: 0o10505640 -> Match: True
Data PRN 5 Last  24 bits -> Expected: 0o71364603, Got: 0o71364603 -> Match: True


Pilot PRN 1 First 24 bits -> Expected: 0o5752067, Got: 0o5752067 -> Match: True
Pilot PRN 1 Last  24 bits -> Expected: 0o20173742, Got: 0o20173742 -> Match: True
Pilot PRN 2 First 24 bits -> Expected: 0o70146401, Got: 0o70146401 -> Match: True
Pilot PRN 2 Last  24 bits -> Expected: 0o35437154, Got: 0o35437154 -> Match: True
Pilot PRN 3 First 24 bits -> Expected: 0o32066222, Got: 0o15044567 -> Match: False
Pilot PRN 3 Last  24 bits -> Expected: 0o161056, Got: 0o161056 -> Match: True
Pilot PRN 4 First 24 bits -> Expected: 0o72125121, Got: 0o72125121 -> Match: True
Pilot PRN 4 Last  24 bits -> Expected: 0o71435437, Got: 0o71435437 -> Match: True
Pilot PRN 5 First 24 bits -> Expected: 0o42323273, Got: 0o42323273 -> Match: True
Pilot PRN 5 Last  24 bits -> Expected: 0o15035661, Got: 0o15035661 -> Match: True
```

Figure 2: Verilog output compared with actual output and compared in python

6 Conclusion

The PRN codes for GPS L1C signals (L1CP and L1CD) for the first five satellites were successfully implemented using Verilog.