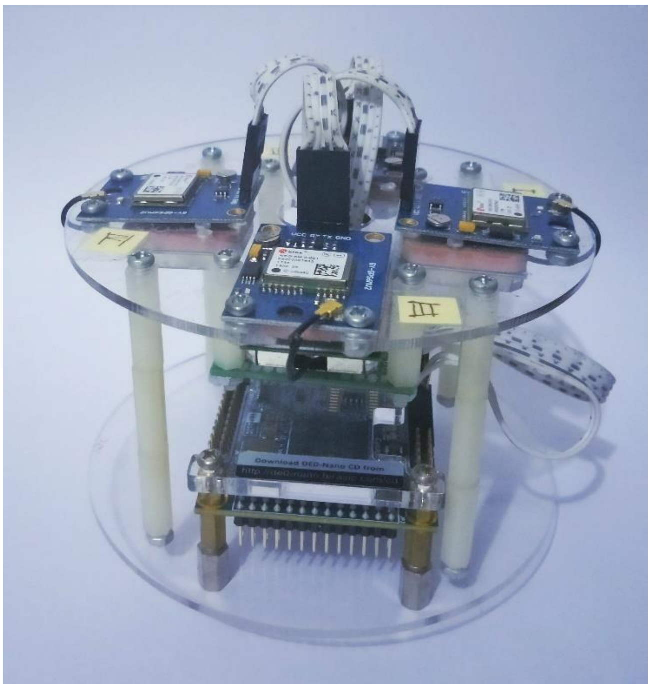
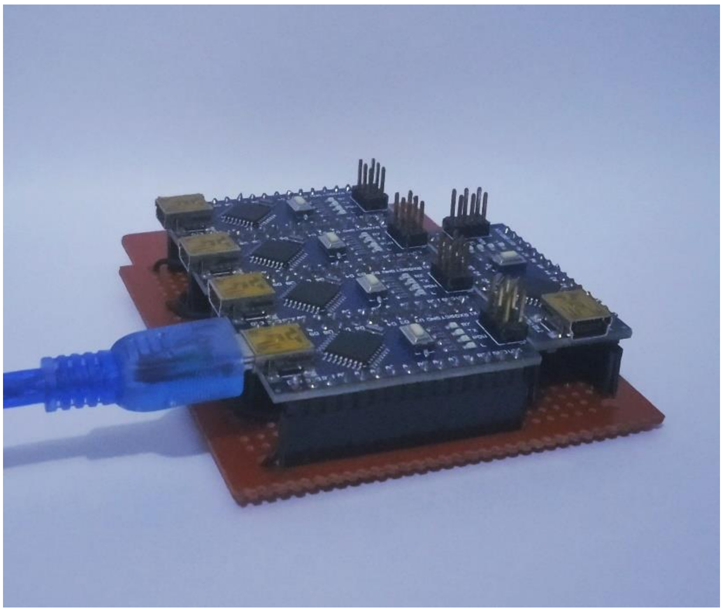
Prototype of FPGA-based system



Prototype of Arduino-based system



Resource usage summary

Resource Usage Summary	
Resource	**Usage**
Total logic elements	21,055 / 22,320 (94 %)
--Combinational with no register	12918
--register only	728
--Combinational with a register	7409
Logic element usage by number of LUT inputs	
-- 4 input functions	8283
-- 3 input functions	4713
-- <=2 input functions	7331
--register only	728
Total registers*	8,137 / 23,018 (35 %)
-- Dedicated logic registers	8,137 / 22,320 (36 %)
-- I/O registers	0 / 698 (0 %)
Total LABs: partially or completely used	1,395 / 1,395 (100 %)
Virtual pins	0
I/O pins	15 / 154 (10 %)
-- Clock pins	1 / 7 (14 %)
-- Dedicated input pins	0 / 9 (0 %)
M9Ks	0 / 66 (0 %)
Total block memory bits	0 / 608,256 (0 %)
Total block memory implementation bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	24 / 132 (18 %)
PLLs	1 / 4 (25 %)
Global clocks	1 / 20 (5 %)
JTAGs	0 / 1 (0 %)
CRC blocks	0 / 1 (0 %)
ASMI blocks	0 / 1 (0 %)
Oscillator blocks	0 / 1 (0 %)
Impedance control blocks	0 / 4 (0 %)

Average interconnect usage (total/H/V)	38.0% / 34.7% / 42.5%
Peak interconnect usage (total/H/V)	54.0% / 53.7% / 57.5%
Maximum fan-out	8149
Highest non-global fan-out	3889
Total fan-out	92655
Average fan-out	3.19

* Total register is not including register inside RAM or DSP blocks.

List VHDL program of Multi GPS.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity Multi_GPS is
    generic(
        clockFrequencyHz : integer := 50_000_000
    );
    port(
        clk   : in std_logic;
        rst   : in std_logic;
        gps1 : in std_logic;
        gps2 : in std_logic;
        gps3 : in std_logic;
        gps4 : in std_logic;
        led1 : out std_logic;
        led2 : out std_logic;
        led3 : out std_logic;
        led4 : out std_logic;
        led5 : out std_logic;
        led6 : out std_logic;
        led7 : out std_logic;
        led8 : out std_logic;
        tx    : out std_logic
    );
end entity;

architecture rtl of Multi_GPS is
    type state is (idle, buffer_ASCII, atoi, buff_module, average, itoa, digit1, digit2,
    digit3, digit4, digit5, digit6, digit7, digit8, digit9, digit10, digit11, digit12,
    tulis);
    signal control_state : state := idle;

    type gpsram is array (0 to 156) of std_logic_vector(7 downto 0);
    signal data_ram : gpsram;
    signal tx_byte   : std_logic_vector(7 downto 0) := (others => '0');

    constant clks_per_bit   : integer := 5208;
    constant counter        : integer := 57292 ; -- Clock per bytes = (50MHz x 11) : 9600
    constant counter_bytes  : integer := 52083; -- Clock cycle for 1 bytes , start bit and
    stop bit

    signal startbuff_ASCII : std_logic := '0';
    signal startbuff_module: std_logic := '0';
    signal startvalidator  : std_logic := '0';
    signal startitoa          : std_logic := '0';
    signal doneParsing    : std_logic := '0';
    signal donebuff_module: std_logic := '0';
    signal doneaverage    : std_logic := '0';
    signal doneitoa       : std_logic := '0';
    signal done_lat       : std_logic := '0';

    signal cin      : std_logic := '0';
    signal index    : integer := 0;
    signal cnt      : integer := 0;
    signal doneGPS1 : std_logic := '0';
    signal doneGPS2 : std_logic := '0';
    signal doneGPS3 : std_logic := '0';
    signal doneGPS4 : std_logic := '0';
    signal lat_val_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_val_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_val_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_val_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_val_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_val_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_val_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_val_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_val_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_val_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_val_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_val_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_val_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_val_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_val_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_val_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal lat_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal lon_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal alt_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_1    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_2    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_3    : std_logic_vector(7 downto 0) := (others => '0');
    signal sat_4    : std_logic_vector(7 downto 0) := (others => '0');
    signal out_lat  : std_logic_vector(7 downto 0) := (others => '0');
    signal out_lon  : std_logic_vector(7 downto 0) := (others => '0');
    signal out_alt  : std_logic_vector(7 downto 0) := (others => '0');
    signal tx_start : std_logic := '0';
    signal noDivisorFlag : std_logic_vector(7 downto 0) := (others => '0');
```

```vhdl
    signal led_signal : std_logic_vector(3 downto 0) := "0000";
    signal led_index     : integer := 0;
    signal noData_counter : integer := 0;

begin
    led1 <= doneGPS1;
    led2 <= doneGPS2;
    led3 <= doneGPS3;
    led4 <= doneGPS4;
    led5 <= led_signal(0);
    led6 <= led_signal(1);
    led7 <= led_signal(2);
    led8 <= led_signal(3);


    Top_Level_Process : process(cin,rst, control_state)
    begin
        if rst = '0' then
            control_state <= idle;
        elsif rising_edge(cin) then
            case control_state is
                when idle =>
                    data_ram(0) <= "00000000";
                    data_ram(1) <= "00000000";
                    data_ram(2) <= "00101100"; -- Koma
                    data_ram(3) <= "00000000";  -- latitude
                    data_ram(4) <= "00000000";
                    data_ram(5) <= "00000000";
                    data_ram(6) <= "00000000";
                    data_ram(7) <= "00000000";  -- titik
                    data_ram(8) <= "00000000";
                    data_ram(9) <= "00000000";
                    data_ram(10)<= "00000000";
                    data_ram(11)<= "00000000";
                    data_ram(12)<= "00000000";
                    data_ram(13)<= "00000000";
                    data_ram(14)<= "00000000";
                    data_ram(15)<= "00000000";
                    data_ram(16)<= "00000000";
                    data_ram(17)<= "00000000";
                    data_ram(18)<= "00000000";
                    data_ram(19)<= "00000000";
                    data_ram(20)<= "00000000";
                    data_ram(21)<= "00000000";
                    data_ram(22)<= "00000000";
                    data_ram(23)<= "00000000";
                    data_ram(24)<= "00000000";
                    data_ram(25)<= "00000000";
                    data_ram(26)<= "00000000";
                    data_ram(27)<= "00000000";
                    data_ram(28)<= "00000000";
data_ram(29)<= "00000000";
data_ram(30)<= "00000000";
data_ram(31)<= "00000000";
data_ram(32)<= "00000000";
data_ram(33)<= "00000000";
data_ram(34)<= "00000000";
data_ram(35)<= "00000000";
data_ram(36)<= "00000000";
data_ram(37)<= "00000000";
data_ram(38)<= "00000000";
data_ram(39)<= "00000000";
data_ram(40)<= "00000000";
data_ram(41)<= "00000000";
data_ram(42)<= "00000000";
data_ram(43)<= "00000000";
data_ram(44)<= "00000000";
data_ram(45)<= "00000000";
data_ram(46)<= "00000000";
data_ram(47)<= "00000000";  -- longitude
data_ram(48)<= "00000000";
data_ram(49)<= "00000000";
data_ram(50)<= "00000000";
data_ram(51)<= "00000000";
data_ram(52)<= "00000000";
data_ram(53)<= "00000000";
data_ram(54)<= "00000000";
data_ram(55)<= "00000000";
data_ram(56)<= "00000000";
data_ram(57)<= "00000000";
data_ram(58)<= "00000000";
data_ram(59)<= "00000000";
data_ram(60)<= "00000000";
data_ram(61)<= "00000000";
data_ram(62)<= "00000000";
data_ram(63)<= "00000000";
data_ram(64)<= "00000000";
data_ram(65)<= "00000000";
data_ram(66)<= "00000000";
data_ram(67)<= "00000000";
data_ram(68)<= "00000000";
data_ram(69)<= "00000000";
data_ram(70)<= "00000000";
data_ram(71)<= "00000000";
data_ram(72)<= "00000000";
data_ram(73)<= "00000000";
data_ram(74)<= "00000000";
data_ram(75)<= "00000000";
data_ram(76)<= "00000000";
data_ram(77)<= "00000000";
data_ram(78)<= "00000000";
data_ram(79)<= "00000000";
data_ram(80)<= "00000000";
data_ram(81)<= "00000000";
data_ram(82)<= "00000000";
data_ram(83)<= "00000000";
data_ram(84)<= "00000000";
data_ram(85)<= "00000000";
data_ram(86)<= "00000000";
data_ram(87)<= "00000000";
data_ram(88)<= "00000000";
data_ram(89)<= "00000000";
data_ram(90)<= "00000000";
data_ram(91)<= "00000000";
data_ram(92)<= "00000000";
data_ram(93)<= "00000000";
data_ram(94)<= "00000000";
data_ram(95)<= "00000000";  --Satellite
data_ram(96)<= "00000000";
data_ram(97)<= "00000000";
data_ram(98)<= "00000000";
data_ram(99)<= "00000000";
data_ram(100)<="00000000";
data_ram(101)<="00000000";
data_ram(102)<="00000000";
data_ram(103)<="00000000";
data_ram(104)<="00000000";
data_ram(105)<="00000000";
data_ram(106)<="00000000";
data_ram(107)<="00000000";  -- Altitude
data_ram(108)<="00000000";
data_ram(109)<="00000000";
data_ram(110)<="00000000";
data_ram(111)<="00000000";
data_ram(112)<="00000000";
data_ram(113)<="00000000";
data_ram(114)<="00000000";
data_ram(115)<="00000000";
data_ram(116)<="00000000";
data_ram(117)<="00000000";
data_ram(118)<="00000000";
data_ram(119)<="00000000";
data_ram(120)<="00000000";
data_ram(121)<="00000000";
data_ram(122)<="00000000";
data_ram(123)<="00000000";
data_ram(124)<="00000000";
data_ram(125)<="00000000";
data_ram(126)<="00000000";
data_ram(127)<="00000000";
data_ram(128)<="00000000";
data_ram(129)<="00000000";
data_ram(130)<="00101100";
```

```
                    data_ram(131)<="00000000";  -- Latitude Average
                    data_ram(132)<="00000000";
                    data_ram(133)<="00000000";
                    data_ram(134)<="00000000";
                    data_ram(135)<="00000000";
                    data_ram(136)<="00000000";
                    data_ram(137)<="00000000";
                    data_ram(138)<="00000000";
                    data_ram(139)<="00101100";
                    data_ram(140)<="00000000"; -- Longitude Average
                    data_ram(141)<="00000000";
                    data_ram(142)<="00000000";
                    data_ram(143)<="00000000";
                    data_ram(144)<="00000000";
                    data_ram(145)<="00000000";
                    data_ram(146)<="00000000";
                    data_ram(147)<="00000000";
                    data_ram(148)<="00000000";
                    data_ram(149)<="00000000";
                    data_ram(150)<="00101100";
                    data_ram(151)<="00000000"; -- Altitude Average
                    data_ram(152)<="00000000";
                    data_ram(153)<="00000000";
                    data_ram(154)<="00000000";
                    data_ram(155)<="00000000";
                    data_ram(156)<="00001010"; -- New Line
                    tx_start    <= '0';
                    led_signal(0) <= '0';
                    led_signal(1) <= '0';
                    led_signal(2) <= '0';
                    led_signal(3) <= '0';
                    if doneGPS1 = '1' or doneGPS2 = '1' or doneGPS3 = '1' or doneGPS4 = '1'
then        -- Counter for buffer
                        control_state <= buffer_ASCII;
                    else
                        control_state <= idle;
                    end if;
                when buffer_ASCII =>
                    if cnt = 6932291 then
                        cnt <= 0;
                        control_state <= atoi;
                    else
                        cnt <= cnt + 1;
                        control_state <= buffer_ASCII;
                    end if;
                when atoi =>
                        led_signal(0) <= '1';
                        led_signal(1) <= '0';
                        led_signal(2) <= '0';
                        led_signal(3) <= '0';
                    if cnt = 630193 then            -- Counter for ASCII to integer
conversion
                        cnt <= 0;
                        startbuff_module <= '1';
                        control_state <= buff_module;
                    else
                        cnt <= cnt + 1;
                        startbuff_module <= '0';
                        control_state <= atoi;
                    end if;
                when buff_module =>
                        led_signal(0) <= '0';
                        led_signal(1) <= '1';
                        led_signal(2) <= '0';
                        led_signal(3) <= '0';
                    if donebuff_module = '1' then
                        startbuff_module <= '0';
                        startvalidator <= '1';
                        control_state <= average;
                    else
                        startbuff_module <= '1';
                        startvalidator <= '0';
                        control_state <= buff_module;
                    end if;
                when average =>
                    if doneaverage = '1' then
                        startvalidator <= '0';
                        startitoa <= '1';
                        control_state <= itoa;
                    else
                        startvalidator <= '1';
                        startitoa <= '0';
                        control_state <= average;
                    end if;
                when itoa =>
                    if doneitoa = '1' then
                        startitoa <= '0';
                        startbuff_ASCII <= '1';
                        control_state <= digit1;
                    else
                        startitoa <= '1';
                        startbuff_ASCII <= '0';
                        control_state <= itoa;
                    end if;
                when digit1 =>
                    led_signal(0) <= '0';
                    led_signal(1) <= '0';
                    led_signal(2) <= '1';
                    led_signal(3) <= '0';
                    data_ram(1) <= noDivisorFlag;
                    data_ram(3) <= lat_1;
```

```
    data_ram(14)<= lat_2;
    data_ram(25)<= lat_3;
    data_ram(36)<= lat_4;
    data_ram(47)<= lon_1;
    data_ram(59)<= lon_2;
    data_ram(71)<= lon_3;
    data_ram(83)<= lon_4;
    data_ram(95)<= sat_1;
    data_ram(98)<= sat_2;
    data_ram(101)<= sat_3;
    data_ram(104)<= sat_4;
    data_ram(107)<= alt_1;
    data_ram(113)<= alt_2;
    data_ram(119)<= alt_3;
    data_ram(125)<= alt_4;
    data_ram(131)<= out_lat;
    data_ram(140)<= out_lon;
    data_ram(151)<= out_alt;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit1;
    else
        cnt <= 0;
        control_state <= digit2;
    end if;
when digit2 =>
    data_ram(2) <= "00101100";
    data_ram(4) <= lat_1;
    data_ram(15)<= lat_2;
    data_ram(26)<= lat_3;
    data_ram(37)<= lat_4;
    data_ram(48)<= lon_1;
    data_ram(60)<= lon_2;
    data_ram(72)<= lon_3;
    data_ram(84)<= lon_4;
    data_ram(96)<= sat_1;
    data_ram(99)<= sat_2;
    data_ram(102)<= sat_3;
    data_ram(105)<= sat_4;
    data_ram(108)<= alt_1;
    data_ram(114)<= alt_2;
    data_ram(120)<= alt_3;
    data_ram(126)<= alt_4;
    data_ram(132)<= out_lat;
    data_ram(141)<= out_lon;
    data_ram(152)<= out_alt;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit2;
    else
        cnt <= 0;
        control_state <= digit3;
    end if;
when digit3 =>
    data_ram(5) <= lat_1;
    data_ram(16)<= lat_2;
    data_ram(27)<= lat_3;
    data_ram(38)<= lat_4;
    data_ram(49)<= lon_1;
    data_ram(61)<= lon_2;
    data_ram(73)<= lon_3;
    data_ram(85)<= lon_4;
    data_ram(97)<= sat_1;
    data_ram(100)<= sat_2;
    data_ram(103)<= sat_3;
    data_ram(106)<= sat_4;
    data_ram(109)<= alt_1;
    data_ram(115)<= alt_2;
    data_ram(121)<= alt_3;
    data_ram(127)<= alt_4;
    data_ram(133)<= out_lat;
    data_ram(142)<= out_lon;
    data_ram(153)<= out_alt;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit3;
    else
        cnt <= 0;
        control_state <= digit4;
    end if;
when digit4 =>
    data_ram(6) <= lat_1;
    data_ram(17)<= lat_2;
    data_ram(28)<= lat_3;
    data_ram(39)<= lat_4;
    data_ram(50)<= lon_1;
    data_ram(62)<= lon_2;
    data_ram(74)<= lon_3;
    data_ram(86)<= lon_4;
    data_ram(110)<= alt_1;
    data_ram(116)<= alt_2;
    data_ram(122)<= alt_3;
    data_ram(128)<= alt_4;
    data_ram(134)<= out_lat;
    data_ram(143)<= out_lon;
    data_ram(154)<= out_alt;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit4;
    else
        cnt <= 0;
        control_state <= digit5;
    end if;
when digit5 =>
    data_ram(7) <= lat_1;
    data_ram(18)<= lat_2;
    data_ram(29)<= lat_3;
    data_ram(40)<= lat_4;
    data_ram(51)<= lon_1;
    data_ram(63)<= lon_2;
    data_ram(75)<= lon_3;
    data_ram(87)<= lon_4;
    data_ram(111)<= alt_1;
    data_ram(117)<= alt_2;
    data_ram(123)<= alt_3;
    data_ram(129)<= alt_4;
    data_ram(135)<= out_lat;
    data_ram(144)<= out_lon;
    data_ram(155)<= out_alt;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit5;
    else
        cnt <= 0;
        control_state <= digit6;
    end if;
when digit6 =>
    data_ram(8) <= lat_1;
    data_ram(19)<= lat_2;
    data_ram(30)<= lat_3;
    data_ram(41)<= lat_4;
    data_ram(52)<= lon_1;
    data_ram(64)<= lon_2;
    data_ram(76)<= lon_3;
    data_ram(88)<= lon_4;
    data_ram(112)<= "00101100";
    data_ram(118)<= "00101100";
    data_ram(124)<= "00101100";
    data_ram(130)<= "00101100";
    data_ram(136)<= out_lat;
    data_ram(145)<= out_lon;
    data_ram(156)<= "00001010";
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit6;
    else
        cnt <= 0;
        control_state <= digit7;
    end if;
when digit7 =>
    data_ram(9) <= lat_1;
    data_ram(20)<= lat_2;
    data_ram(31)<= lat_3;
    data_ram(42)<= lat_4;
    data_ram(53)<= lon_1;
    data_ram(65)<= lon_2;
    data_ram(77)<= lon_3;
    data_ram(89)<= lon_4;
    data_ram(137)<= out_lat;
    data_ram(146)<= out_lon;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit7;
    else
        cnt <= 0;
        control_state <= digit8;
    end if;
when digit8 =>
    data_ram(10)  <= lat_1;
    data_ram(21)<= lat_2;
    data_ram(32)<= lat_3;
    data_ram(43)<= lat_4;
    data_ram(54)<= lon_1;
    data_ram(66)<= lon_2;
    data_ram(78)<= lon_3;
    data_ram(90)<= lon_4;
    data_ram(138)<= out_lat;
    data_ram(147)<= out_lon;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit8;
    else
        cnt <= 0;
        control_state <= digit9;
    end if;
when digit9 =>
    data_ram(11)<= lat_1;
    data_ram(22)<= lat_2;
    data_ram(33)<= lat_3;
    data_ram(44)<= lat_4;
    data_ram(55)<= lon_1;
    data_ram(67)<= lon_2;
    data_ram(79)<= lon_3;
    data_ram(91)<= lon_4;
    data_ram(139)<= "00101100";
    data_ram(148)<= out_lon;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit9;
    else
        cnt <= 0;
        control_state <= digit10;
    end if;
when digit10=>
```

```
    data_ram(12)<= lat_1;
    data_ram(23)<= lat_2;
    data_ram(34)<= lat_3;
    data_ram(45)<= lat_4;
    data_ram(56)<= lon_1;
    data_ram(68)<= lon_2;
    data_ram(80)<= lon_3;
    data_ram(92)<= lon_4;
    data_ram(149)<= out_lon;
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit10;
    else
        cnt <= 0;
        control_state <= digit11;
    end if;
when digit11=>
    data_ram(13)<= "00101100";
    data_ram(24)<= "00101100";
    data_ram(35)<= "00101100";
    data_ram(46)<= "00101100";
    data_ram(57)<= lon_1;
    data_ram(69)<= lon_2;
    data_ram(81)<= lon_3;
    data_ram(93)<= lon_4;
    data_ram(150)<= "00101100";
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit11;
    else
        cnt <= 0;
        control_state <= digit12;
    end if;
when digit12=>
    data_ram(58)<= "00101100";
    data_ram(70)<= "00101100";
    data_ram(82)<= "00101100";
    data_ram(94)<= "00101100";
    if cnt < counter - 1 then
        cnt <= cnt + 1;
        control_state <= digit12;
    else
        cnt <= 0;
        control_state <= tulis;
    end if;
when tulis =>
    led_signal(0) <= '0';
    led_signal(1) <= '0';
    led_signal(2) <= '0';
    led_signal(3) <= '1';
    startbuff_ASCII <= '0';
```

```
                tx_start    <= '1';
                if index < 157 then
                    if cnt < counter_bytes -1 then
                        cnt <= cnt + 1;
                        control_state <= tulis;
                        tx_byte <= data_ram(index);
                    else
                        cnt     <= 0;
                        tx_start<= '0';
                        index   <= index + 1;
                    end if;
                else
                    index <= 0;
                    control_state <= idle;
                end if;
        end case;
    end if;
end process;

Data_Parser_block : entity work.Data_Parser(rtl)
generic map(
    ClockFrequencyHz => ClockFrequencyHz
)
port map(
    Clk => cin,
    Reset   => rst,
    GPS1    => gps1,
    GPS2    => gps2,
    GPS3    => gps3,
    GPS4    => gps4,
    doneGPS1 => doneGPS1,
    doneGPS2 => doneGPS2,
    doneGPS3 => doneGPS3,
    doneGPS4 => doneGPS4,
    Lat_1   => lat_val_1,
    Lat_2   => lat_val_2,
    Lat_3   => lat_val_3,
    Lat_4   => lat_val_4,
    Lon_1   => lon_val_1,
    Lon_2   => lon_val_2,
    Lon_3   => lon_val_3,
    Lon_4   => lon_val_4,
    Alt_1   => alt_val_1,
    Alt_2   => alt_val_2,
    Alt_3   => alt_val_3,
    Alt_4   => alt_val_4,
    sat_1   => sat_val_1,
    sat_2   => sat_val_2,
    sat_3   => sat_val_3,
    sat_4   => sat_val_4
);
```

```
Data_Processing_block : entity work.Data_Processing(rtl)
port map(
    clk => cin,
    rst => rst,
   startbuff_ASCII => startbuff_ASCII,
   startbuff_module=> startbuff_module,
   startvalidator   => startvalidator,
   startitoa         => startitoa,
    Lat_GPS1    => lat_val_1,
    Lat_GPS2    => lat_val_2,
    Lat_GPS3    => lat_val_3,
    Lat_GPS4    => lat_val_4,
    Lon_GPS1    => lon_val_1,
    Lon_GPS2    => lon_val_2,
    Lon_GPS3    => lon_val_3,
    Lon_GPS4    => lon_val_4,
    Alt_GPS1    => alt_val_1,
    Alt_GPS2    => alt_val_2,
    Alt_GPS3    => alt_val_3,
    Alt_GPS4    => alt_val_4,
    Sat_GPS1    => sat_val_1,
    Sat_GPS2    => sat_val_2,
    Sat_GPS3    => sat_val_3,
    Sat_GPS4    => sat_val_4,
   doneatoi          => open,
   donebuff_module => donebuff_module,
   done_buff_ASCII => open,
   doneaverage   => doneaverage,
   doneitoa          => doneitoa,
    led_valid=> open,
    noDivisorFlag   => noDivisorFlag,
    lat_1   => lat_1,
    lat_2   => lat_2,
    lat_3   => lat_3,
    lat_4   => lat_4,
    lon_1   => lon_1,
    lon_2   => lon_2,
    lon_3   => lon_3,
    lon_4   => lon_4,
    alt_1   => alt_1,
    alt_2   => alt_2,
    alt_3   => alt_3,
    alt_4   => alt_4,
    sat_1   => sat_1,
    sat_2   => sat_2,
    sat_3   => sat_3,
    sat_4   => sat_4,
    out_lat => out_lat,
    out_lon => out_lon,
    out_alt => out_alt
    );

    Transmitter : entity work.uart_tx(rtl)
    generic map(
        clks_per_bit     => clks_per_bit,
        ClockFrequencyHz => ClockFrequencyHz
    )
    port map(
        clk       => cin,
        rst       => rst,
        tx_start  => tx_start,
        tx_byte   => tx_byte,
        tx_active => open,
        tx        => tx,
        tx_done   => open
    );

    pll_inst : entity work.pll
    PORT MAP (
        inclk0   => clk,
        c0       => cin
    );


end architecture;
```

List VHDL program of Data Parser.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Data_Parser IS
    GENERIC(
        ClockFrequencyHz : INTEGER
    );
    PORT(
        Clk : IN STD_LOGIC;
        Reset   : IN STD_LOGIC;
        GPS1    : IN STD_LOGIC;
        GPS2    : IN STD_LOGIC;
        GPS3    : IN STD_LOGIC;
        GPS4    : IN STD_LOGIC;
        doneGPS1 : out std_LOGIC;
        doneGPS2 : out std_LOGIC;
        doneGPS3 : out std_LOGIC;
        doneGPS4    : out std_LOGIC;
        Lat_1   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lat_2   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lat_3   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lat_4   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lon_1   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lon_2   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lon_3   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Lon_4   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Alt_1   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Alt_2   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Alt_3   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Alt_4   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Sat_1   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Sat_2   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Sat_3   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Sat_4   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END Data_Parser;

ARCHITECTURE rtl OF Data_Parser IS
BEGIN
Data_GPS1 : ENTITY WORK.Parser
    GENERIC MAP(
    ClockFrequencyHz => ClockFrequencyHz
    )
    PORT MAP(
    Clk         => Clk,
    Reset       => Reset,
    Rx          => GPS1,
    done        => doneGPS1,
    Lattitude   => Lat_1,
    Longitude   => Lon_1,
    Altitude    => Alt_1,
    Jml_Satelit => Sat_1
    );



Data_GPS2 : ENTITY WORK.Parser
    GENERIC MAP(
    ClockFrequencyHz => ClockFrequencyHz
    )
    PORT MAP(
    Clk         => Clk,
    Reset       => Reset,
    Rx          => GPS2,
    done        => doneGPS2,
    Lattitude   => Lat_2,
    Longitude   => Lon_2,
    Altitude    => Alt_2,
    Jml_Satelit => Sat_2
    );

Data_GPS3 : ENTITY WORK.Parser
    GENERIC MAP(
    ClockFrequencyHz => ClockFrequencyHz
    )
    PORT MAP(
    Clk         => Clk,
    Reset       => Reset,
    Rx          => GPS3,
    done        => doneGPS3,
    Lattitude   => Lat_3,
    Longitude   => Lon_3,
    Altitude    => Alt_3,
    Jml_Satelit => Sat_3
    );

Data_GPS4 : ENTITY WORK.Parser
    GENERIC MAP(
    ClockFrequencyHz => ClockFrequencyHz
    )
    PORT MAP(
    Clk         => Clk,
    Reset       => Reset,
    Rx          => GPS4,
    done        => doneGPS4,
    Lattitude   => Lat_4,
    Longitude   => Lon_4,
    Altitude    => Alt_4,
    Jml_Satelit => Sat_4
    );
END architecture;
```

List VHDL program of Data Processing.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Data_Processing is
    port(
        clk      : in std_logic;
        rst      : in std_logic;
       startbuff_ASCII : in std_logic;
       startbuff_module: in std_logic;
       startvalidator  : in std_logic;
       startitoa         : in std_logic;
        Lat_GPS1 : in std_logic_vector(7 downto 0);
        Lat_GPS2 : in std_logic_vector(7 downto 0);
        Lat_GPS3 : in std_logic_vector(7 downto 0);
        Lat_GPS4 : in std_logic_vector(7 downto 0);
        Lon_GPS1 : in std_logic_vector(7 downto 0);
        Lon_GPS2 : in std_logic_vector(7 downto 0);
        Lon_GPS3 : in std_logic_vector(7 downto 0);
        Lon_GPS4 : in std_logic_vector(7 downto 0);
        Alt_GPS1 : in std_logic_vector(7 downto 0);
        Alt_GPS2 : in std_logic_vector(7 downto 0);
        Alt_GPS3 : in std_logic_vector(7 downto 0);
        Alt_GPS4 : in std_logic_vector(7 downto 0);
        Sat_GPS1 : in std_logic_vector(7 downto 0);
        Sat_GPS2 : in std_logic_vector(7 downto 0);
        Sat_GPS3 : in std_logic_vector(7 downto 0);
        Sat_GPS4 : in std_logic_vector(7 downto 0);
        doneatoi        : out std_logic;
        donebuff_module : out std_logic;
        done_buff_ASCII : out std_logic;
        doneaverage     : out std_logic;
        doneitoa        : out std_logic;
        led_valid      : out std_logic_vector(3 downto 0);
        noDivisorFlag   : out std_logic_vector(7 downto 0);
        Lat_1    : out std_logic_vector(7 downto 0);
        Lat_2    : out std_logic_vector(7 downto 0);
        Lat_3    : out std_logic_vector(7 downto 0);
        Lat_4    : out std_logic_vector(7 downto 0);
        Lon_1    : out std_logic_vector(7 downto 0);
        Lon_2    : out std_logic_vector(7 downto 0);
        Lon_3    : out std_logic_vector(7 downto 0);
        Lon_4    : out std_logic_vector(7 downto 0);
        Alt_1    : out std_logic_vector(7 downto 0);
        Alt_2    : out std_logic_vector(7 downto 0);
        Alt_3    : out std_logic_vector(7 downto 0);
        Alt_4    : out std_logic_vector(7 downto 0);
        Sat_1    : out std_logic_vector(7 downto 0);
        Sat_2    : out std_logic_vector(7 downto 0);
        Sat_3    : out std_logic_vector(7 downto 0);
        Sat_4    : out std_logic_vector(7 downto 0);
        out_lat  : out std_logic_vector(7 downto 0);
        out_lon  : out std_logic_vector(7 downto 0);
        out_alt  : out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of Data_Processing is
    constant counter : integer := 57292;
    type atoi_state is (idle, turnON);
    signal ASCII_to_integer_s : atoi_state := idle;
    type ram is array (0 to 1) of std_logic_vector(7 downto 0);
    type state is (idle, send);
    signal flag_state : state; -- state untuk pengiriman data sinyal no divisor
    signal data_ram : ram;
    signal cnt  : integer := 0;
    signal index : integer := 0;
    signal start_val: std_logic := '0';
    signal start_buf: std_logic := '0';
    signal start_con: std_logic := '0';
    signal done_alt : std_logic := '0';
    signal done_sat : std_logic := '0';
    signal done_buf : std_logic := '0';
    signal Lat_ASCII1 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_ASCII2 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_ASCII3 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_ASCII4 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_ASCII1 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_ASCII2 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_ASCII3 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_ASCII4 : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_ASCII1 : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_ASCII2 : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_ASCII3 : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_ASCII4 : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_ASCII1 : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_ASCII2 : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_ASCII3 : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_ASCII4 : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_input_1  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_input_2  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_input_3  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_input_4  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_input_1  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_input_2  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_input_3  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_input_4  : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_input_1  : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_input_2  : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_input_3  : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_input_4  : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_input_1  : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_input_2  : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_input_3  : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_input_4  : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_int1 : integer := 0;
    signal Lat_int2 : integer := 0;
    signal Lat_int3 : integer := 0;
    signal Lat_int4 : integer := 0;
    signal Lon_int1 : integer := 0;
    signal Lon_int2 : integer := 0;
    signal Lon_int3 : integer := 0;
    signal Lon_int4 : integer := 0;
    signal Alt_int1 : integer := 0;
    signal Alt_int2 : integer := 0;
    signal Alt_int3 : integer := 0;
    signal Alt_int4 : integer := 0;
    signal Sat_int1 : integer := 0;
    signal Sat_int2 : integer := 0;
    signal Sat_int3 : integer := 0;
    signal Sat_int4 : integer := 0;
    signal Lat_Val_1: integer := 0;
    signal Lat_Val_2: integer := 0;
    signal Lat_Val_3: integer := 0;
    signal Lat_Val_4: integer := 0;
    signal Lon_Val_1: integer := 0;
    signal Lon_Val_2: integer := 0;
    signal Lon_Val_3: integer := 0;
    signal Lon_Val_4: integer := 0;
    signal Alt_Val_1: integer := 0;
    signal Alt_Val_2: integer := 0;
    signal Alt_Val_3: integer := 0;
    signal Alt_Val_4: integer := 0;
    signal Sat_Val_1: integer := 0;
    signal Sat_Val_2: integer := 0;
    signal Sat_Val_3: integer := 0;
    signal Sat_Val_4: integer := 0;
    signal lat_ave  : integer := 0;
    signal lon_ave  : integer := 0;
    signal alt_ave  : integer := 0;
    signal divisor  : integer := 0;
    signal noDivisor_sig     : std_logic := '0';
    signal noDivisorFlag_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Data_valid        : std_logic_vector(3 downto 0) := (others => '0');
    signal done_conv_sig     : std_logic := '0';
begin

    doneitoa <= done_conv_sig;
    noDivisorFlag <= noDivisorFlag_sig;
    led_valid <= Data_valid;
```

```
Lat_ASCII1 <= Lat_GPS1;
Lat_ASCII2 <= Lat_GPS2;
Lat_ASCII3 <= Lat_GPS3;
Lat_ASCII4 <= Lat_GPS4;
Lon_ASCII1 <= Lon_GPS1;
Lon_ASCII2 <= Lon_GPS2;
Lon_ASCII3 <= Lon_GPS3;
Lon_ASCII4 <= Lon_GPS4;
Alt_ASCII1 <= Alt_GPS1;
Alt_ASCII2 <= Alt_GPS2;
Alt_ASCII3 <= Alt_GPS3;
Alt_ASCII4 <= Alt_GPS4;
Sat_ASCII1 <= Sat_GPS1;
Sat_ASCII2 <= Sat_GPS2;
Sat_ASCII3 <= Sat_GPS3;
Sat_ASCII4 <= Sat_GPS4;


noDivisorFlag_proc :process(clk)
begin
    if rising_edge(clk) then
        case flag_state is
            when idle =>
                if done_conv_sig = '1' then
                    if noDivisor_sig = '0' then
                        data_ram(0) <= "00110000";
                        data_ram(1) <= "00101100";
                        flag_state <= send;
                    elsif noDivisor_sig = '1' then
                        data_ram(0) <= "00110001";
                        data_ram(1) <= "00101100";
                        flag_state <= send;
                    end if;
                end if;
            when send =>
                if index < 2 then
                    flag_state <= send;
                    noDivisorFlag_sig <= data_ram(index);
                    if cnt < counter -1 then
                        cnt <= cnt + 1;
                    else
                        cnt <= 0;
                        index <= index + 1;
                    end if;
                else
                    cnt <= 0;
                    index <= 0;
                    flag_state <= idle;
                end if;
        end case;
    end if;
end process;

ASCII_to_integer : entity work.ASCII_to_int(rtl)
port map(
    clk      => clk,
    rst      => rst,
    Lat_GPS1 => Lat_input_1,
    Lat_GPS2 => Lat_input_2,
    Lat_GPS3 => Lat_input_3,
    Lat_GPS4 => Lat_input_4,
    Lon_GPS1 => Lon_input_1,
    Lon_GPS2 => Lon_input_2,
    Lon_GPS3 => Lon_input_3,
    Lon_GPS4 => Lon_input_4,
    Alt_GPS1 => Alt_input_1,
    Alt_GPS2 => Alt_input_2,
    Alt_GPS3 => Alt_input_3,
    Alt_GPS4 => Alt_input_4,
    Sat_GPS1 => Sat_input_1,
    Sat_GPS2 => Sat_input_2,
    Sat_GPS3 => Sat_input_3,
    Sat_GPS4 => Sat_input_4,
    Lat_int1 => Lat_int1,
    Lat_int2 => Lat_int2,
    Lat_int3 => Lat_int3,
    Lat_int4 => Lat_int4,
    Lon_int1 => Lon_int1,
    Lon_int2 => Lon_int2,
    Lon_int3 => Lon_int3,
    Lon_int4 => Lon_int4,
    Alt_int1 => Alt_int1,
    Alt_int2 => Alt_int2,
    Alt_int3 => Alt_int3,
    Alt_int4 => Alt_int4,
    Sat_int1 => Sat_int1,
    Sat_int2 => Sat_int2,
    Sat_int3 => Sat_int3,
    Sat_int4 => Sat_int4,
    done_lon => doneatoi
);

Validator : entity work.Validator(rtl)
port map(
    Clk          => clk,
    rst          => rst,
    start_val    => startvalidator,
    satelit_gps1 => Sat_Val_1,
    satelit_gps2 => Sat_Val_2,
    satelit_gps3 => Sat_Val_3,
    satelit_gps4 => Sat_Val_4,
    done_val         => open,
    Data_valid   => Data_valid
);

Buffer_module : entity work.Buffer_Module(rtl)
port map(
    Clk         => clk,
    Reset       => rst,
    Gps_Valid   => Data_valid,
    Start_buffer=> startbuff_module,
    Lat_GPS1_in => Lat_int1,
    Lat_GPS2_in => Lat_int2,
    Lat_GPS3_in => Lat_int3,
    Lat_GPS4_in => Lat_int4,
    Lon_GPS1_in => Lon_int1,
    Lon_GPS2_in => Lon_int2,
    Lon_GPS3_in => Lon_int3,
    Lon_GPS4_in => Lon_int4,
    Alt_GPS1_in => Alt_int1,
    Alt_GPS2_in => Alt_int2,
    Alt_GPS3_in => Alt_int3,
    Alt_GPS4_in => Alt_int4,
    Sat_GPS1_in => Sat_int1,
    Sat_GPS2_in => Sat_int2,
    Sat_GPS3_in => Sat_int3,
    Sat_GPS4_in => Sat_int4,
    done_buffer => donebuff_module,
    Lat_Val_1   => Lat_Val_1,
    Lat_Val_2   => Lat_Val_2,
    Lat_Val_3   => Lat_Val_3,
    Lat_Val_4   => Lat_Val_4,
    Lon_Val_1   => Lon_Val_1,
    Lon_Val_2   => Lon_Val_2,
    Lon_Val_3   => Lon_Val_3,
    Lon_Val_4   => Lon_Val_4,
    Alt_Val_1   => Alt_Val_1,
    Alt_Val_2   => Alt_Val_2,
    Alt_Val_3   => Alt_Val_3,
    Alt_Val_4   => Alt_Val_4,
    Sat_Val_1   => Sat_Val_1,
    Sat_Val_2   => Sat_Val_2,
    Sat_Val_3   => Sat_Val_3,
    Sat_Val_4   => Sat_Val_4
);

Buffer_2 : entity work.Buffer_ASCII(rtl)
port map(
    clk         => clk,
    rst         => rst,
    eject_data  => startbuff_ASCII,
    Lat_GPS1_in => Lat_ASCII1,
    Lat_GPS2_in => Lat_ASCII2,
    Lat_GPS3_in => Lat_ASCII3,
    Lat_GPS4_in => Lat_ASCII4,
    Lon_GPS1_in => Lon_ASCII1,
    Lon_GPS2_in => Lon_ASCII2,
    Lon_GPS3_in => Lon_ASCII3,
    Lon_GPS4_in => Lon_ASCII4,
    Alt_GPS1_in => Alt_ASCII1,
    Alt_GPS2_in => Alt_ASCII2,
    Alt_GPS3_in => Alt_ASCII3,
    Alt_GPS4_in => Alt_ASCII4,
    Sat_GPS1_in => Sat_ASCII1,
    Sat_GPS2_in => Sat_ASCII2,
    Sat_GPS3_in => Sat_ASCII3,
    Sat_GPS4_in => Sat_ASCII4,
    done        => done_buff_ASCII,
    Lat_1_input => Lat_input_1,
    Lat_2_input => Lat_input_2,
    Lat_3_input => Lat_input_3,
    Lat_4_input => Lat_input_4,
    Lon_1_input => Lon_input_1,
    Lon_2_input => Lon_input_2,
    Lon_3_input => Lon_input_3,
    Lon_4_input => Lon_input_4,
    Alt_1_input => Alt_input_1,
    Alt_2_input => Alt_input_2,
    Alt_3_input => Alt_input_3,
    Alt_4_input => Alt_input_4,
    Sat_1_input => Sat_input_1,
    Sat_2_input => Sat_input_2,
    Sat_3_input => Sat_input_3,
    Sat_4_input => Sat_input_4,
    Lat_1       => Lat_1,
    Lat_2       => Lat_2,
    Lat_3       => Lat_3,
    Lat_4       => Lat_4,
    Lon_1       => Lon_1,
    Lon_2       => Lon_2,
    Lon_3       => Lon_3,
    Lon_4       => Lon_4,
    Alt_1       => Alt_1,
    Alt_2       => Alt_2,
    Alt_3       => Alt_3,
    Alt_4       => Alt_4,
    Sat_1       => Sat_1,
    Sat_2       => Sat_2,
    Sat_3       => Sat_3,
    Sat_4       => Sat_4
);


Average_module : entity work.Average(rtl)
```

```
    port map(
        clk   => clk,
        rst   => rst,
        Lat1  => Lat_Val_1,
        Lat2  => Lat_Val_2,
        Lat3  => Lat_Val_3,
        Lat4  => Lat_Val_4,
        Lon1  => Lon_Val_1,
        Lon2  => Lon_Val_2,
        Lon3  => Lon_Val_3,
        Lon4  => Lon_Val_4,
        Alt1  => Alt_Val_1,
        Alt2  => Alt_Val_2,
        Alt3  => Alt_Val_3,
        Alt4  => Alt_Val_4,
        noDivisor  => noDivisor_sig,
        data_valid => Data_valid,
        done     => doneaverage,
        Lat_ave => lat_ave,
        Lon_ave => lon_ave,
        Alt_ave => alt_ave
    );

    Integer_to_ASCII : entity work.int_to_ASCII(rtl)
    port map(
        clk       => clk,
        rst       => rst,
        start     => startitoa,
        lat_ave   => lat_ave,
        lon_ave   => lon_ave,
        alt_ave   => alt_ave,
        done      => done_conv_sig,
        out_lat   => out_lat,
        out_lon   => out_lon,
        out_alt   => out_alt
    );
end architecture;
```

List VHDL program of UART Transmitter.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uart_tx is
    generic(
        clks_per_bit    : integer := 5208;  -- 50MHz/9600
        ClockFrequencyHz: integer
    );

    port(
        clk         : in std_logic;
        rst         : in std_logic;
        tx_start    : in std_logic;
        tx_byte     : in std_logic_vector(7 downto 0);
        tx_active   : out std_logic;
        tx          : out std_logic;
        tx_done     : out std_logic
    );
end entity;

architecture rtl of uart_tx is
    type StateMachine is (idle, start, data, stop, cleanup);
    signal State     : StateMachine := idle;
    signal Clk_Cnt   : integer range 0 to clks_per_bit-1  := 0;
    signal Index     : integer range 0 to 7 := 0;
    signal Tx_Data   : std_logic_vector(7 downto 0)  := (others => '0');
    signal Tx_Done_Sig : std_logic := '0';
begin
    Transmitt_Process : process(clk,rst)
    begin
        if rst = '0' then
            state <= idle;
        elsif rising_edge(clk)  then
            case State is
                when idle =>
                    tx_active   <= '0';
                    tx          <= '1'; -- High for IDLE
                    Tx_Done_Sig <= '0';
                    Clk_Cnt     <= 0;
                    Index       <= 0;

                    if tx_start = '1' then
                        Tx_Data <= tx_byte;
                        State   <= start;
                    else
                        State       <= idle;
                    end if;
                -- Send Start Bit. Start Bit = 0
                when start =>
                    tx_active <= '1';
                    tx        <= '0';

                    if Clk_Cnt < clks_per_bit-1 then
                        Clk_Cnt <= Clk_Cnt + 1;
                        State   <= start;
                    else
                        Clk_Cnt <= 0;
                        State   <= data;
                    end if;

                when data =>
                    tx  <= Tx_Data(Index);
                    if Clk_Cnt < clks_per_bit-1 then
                        Clk_Cnt <= Clk_Cnt + 1;
                        State   <= data;
                    else
                        Clk_Cnt <= 0;

                        if Index < 7 then
                            Index   <= Index + 1;
                            State   <= data;
                        else
                            Index   <= 0;
                            State   <= stop;
                        end if;
                    end if;

                when stop =>
                    tx  <= '1';

                    if Clk_Cnt < clks_per_bit-1 then
                        Clk_Cnt <= Clk_Cnt + 1;
                        State   <= stop;
                    else
                        Tx_Done_Sig <= '1';
                        Clk_Cnt     <= 0;
                        State       <= cleanup;
                    end if;

                when cleanup =>
                    tx_active   <= '0';
                    Tx_Done_Sig <= '1';
                    State       <= idle;

                when others =>
                    State   <= idle;
            end case;
        end if;
```

```vhdl
    end process;

    tx_done <= Tx_Done_Sig;
end architecture;
```

List VHDL program of UART Receiver.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity uart_rx is
  generic (
    g_CLKS_PER_BIT : integer := 5208    -- Needs to be set correctly
    );
  port (
    i_Clk       : in  std_logic;
    rst         : in std_logic;
    i_RX_Serial : in  std_logic;
    o_RX_DV     : out std_logic;
    o_RX_Byte   : out std_logic_vector(7 downto 0)
    );
end entity;


architecture rtl of uart_rx is

  type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
                     s_RX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_RX_Data_R : std_logic := '0';
  signal r_RX_Data   : std_logic := '0';

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
  signal r_RX_DV     : std_logic := '0';

begin

  -- Purpose: Double-register the incoming data.
  -- This allows it to be used in the UART RX Clock Domain.
  -- (It removes problems caused by metastabiliy)
  p_SAMPLE : process (i_Clk)
  begin
    if rising_edge(i_Clk) then
      r_RX_Data_R <= i_RX_Serial;
      r_RX_Data   <= r_RX_Data_R;
    end if;
  end process p_SAMPLE;


  -- Purpose: Control RX state machine
  p_UART_RX : process (i_Clk,rst)
```

```
begin
   if rst = '0' then
      r_SM_Main <= s_Idle;
  elsif rising_edge(i_Clk) then

    case r_SM_Main is

      when s_Idle =>
        r_RX_DV     <= '0';
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if r_RX_Data = '0' then       -- Start bit detected
          r_SM_Main <= s_RX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;




      -- Check middle of start bit to make sure it's still low
      when s_RX_Start_Bit =>
        if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
          if r_RX_Data = '0' then
            r_Clk_Count <= 0;  -- reset counter since we found the middle
            r_SM_Main   <= s_RX_Data_Bits;
          else
            r_SM_Main   <= s_Idle;
          end if;
        else
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Start_Bit;
        end if;


      -- Wait g_CLKS_PER_BIT-1 clock cycles to sample serial data
      when s_RX_Data_Bits =>
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Data_Bits;
        else
          r_Clk_Count            <= 0;
          r_RX_Byte(r_Bit_Index) <= r_RX_Data;

          -- Check if we have sent out all bits
          if r_Bit_Index < 7 then
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main   <= s_RX_Data_Bits;
            else
              r_Bit_Index <= 0;
              r_SM_Main   <= s_RX_Stop_Bit;
            end if;
          end if;


        -- Receive Stop bit.  Stop bit = 1
        when s_RX_Stop_Bit =>
          -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
          if r_Clk_Count < g_CLKS_PER_BIT-1 then
            r_Clk_Count <= r_Clk_Count + 1;
            r_SM_Main   <= s_RX_Stop_Bit;
          else
            r_RX_DV     <= '1';
            r_Clk_Count <= 0;
            r_SM_Main   <= s_Cleanup;
          end if;


        -- Stay here 1 clock
        when s_Cleanup =>
          r_SM_Main <= s_Idle;
          r_RX_DV   <= '0';


        when others =>
          r_SM_Main <= s_Idle;

      end case;
    end if;
  end process p_UART_RX;

  o_RX_DV   <= r_RX_DV;
  o_RX_Byte <= r_RX_Byte;

end architecture;
```

List VHDL program of Parser.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Parser IS
    GENERIC(
    ClockFrequencyHz    : INTEGER
    );
    PORT(
    Clk         : IN STD_LOGIC;
    Reset       : IN STD_LOGIC;
    Rx          : IN STD_LOGIC;
    Lattitude   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
    Longitude   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
    Altitude    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
    done        : out std_LOGIC;
    Jml_Satelit : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0')
    );
END entity;

ARCHITECTURE rtl OF Parser IS
    TYPE ParseState     IS (Dolar,DetG,DetP,DetG2,DetG3,DetA,DetKoma,ParsingData);
    SIGNAL State            : ParseState := Dolar;
    SIGNAL RX_DV            : STD_LOGIC;
    SIGNAL Rx_Byte          : STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL Cnt_Comma        : INTEGER := 0;
    signal done_sig         : std_LOGIC := '0';
    signal cnt              : integer := 0;

BEGIN
    done <= done_sig;
 -- Counter koma
    PROCESS(Clk,Reset)
    BEGIN
        IF Reset = '0' THEN
            Cnt_Comma <= 0;
        ELSIF(RISING_EDGE(clk)) THEN
            IF(RX_DV = '1') THEN
                IF(Rx_Byte = "00001010") THEN -- Apabila bertemu new line, reset counter
                kembali ke 0
                    Cnt_Comma <= 0;
                ELSIF(Rx_Byte = "00101100") THEN    -- Apabila mendeteksi koma
                    Cnt_Comma <= Cnt_Comma + 1;
                END IF;
            END IF;
        END IF;
    END PROCESS;
```

```
-- Parsing GPGGA (Menghasilkan Lattitude, Longitude, dan Jumlah Satelit)
   PROCESS(Clk, Reset)
   BEGIN
       IF Reset = '0' THEN
           Lattitude   <= (OTHERS => '0');
           Longitude   <= (OTHERS => '0');
           Altitude    <= (OTHERS => '0');
           Jml_Satelit <= (OTHERS => '0');
       ELSIF(RISING_EDGE(clk)) THEN
           IF(RX_DV = '1') THEN
           CASE State IS
               WHEN Dolar =>
                   Lattitude <= "00000000";
                   Longitude <= "00000000";
                   Altitude  <= "00000000";
                   Jml_Satelit <= "00000000";
                   IF(Rx_Byte = "00100100" ) THEN -- $
                       State   <= DetG;
                   ELSE
                       State   <= Dolar;
                   END IF;
               WHEN DetG =>
                   IF(Rx_Byte = "01000111") THEN -- Deteksi G
                       State   <= DetP;
                   ELSE
                       State       <= Dolar;
                   END IF;
               WHEN DetP =>
                   IF(Rx_Byte = "01010000") THEN -- Deteksi P
                       State   <= DetG2;
                   ELSE
                       State       <= Dolar;
                   END IF;
               WHEN DetG2 =>
                   IF(Rx_Byte = "01000111") THEN -- Deteksi G
                       State   <= DetG3;
                   ELSE
                       State       <= Dolar;
                   END IF;
               WHEN DetG3 =>
                   IF(Rx_Byte = "01000111") THEN -- Deteksi G
                       State   <= DetA;
                   ELSE
                       State       <= Dolar;
                   END IF;
               WHEN DetA =>
                   IF(Rx_Byte = "01000001") THEN -- Deteksi A
                       State   <= DetKoma;
                   ELSE
                       State       <= Dolar;
                   END IF;
```

```
                WHEN DetKoma =>
                    IF(Rx_Byte = "00101100") THEN -- Deteksi koma
                        State   <= ParsingData;
                    ELSE
                        State       <= Dolar;
                    END IF;
                WHEN ParsingData =>
                    done_sig <= '1';
                    IF(Rx_Byte = "00001010") THEN -- LF(Line Feed)
                        done_sig <= '0';
                        State       <= Dolar;
                    else
                        IF(Cnt_Comma = 2) THEN -- Lattitude
                            IF(Rx_Byte = "00101100") THEN
                                Lattitude <= "00000000";
                            ELSE
                                Lattitude <= Rx_Byte;
                            END IF;
                        ELSIF(Cnt_Comma = 4) THEN -- Longitude
                            IF(Rx_Byte = "00101100") THEN
                                Longitude <= "00000000";
                            ELSE
                                Longitude <= Rx_Byte;
                            END IF;
                        ELSIF(Cnt_Comma = 7) THEN -- Jumlah Satelit
                            IF(Rx_Byte = "00101100") THEN
                                Jml_Satelit <= "00000000";
                            ELSE
                                Jml_Satelit <= Rx_Byte;
                            END IF;
                        ELSIF(Cnt_Comma = 9) THEN -- Altitude
                            IF(Rx_Byte = "00101100") THEN
                                Altitude <= "00000000";
                            ELSE
                                Altitude    <= Rx_Byte;
                            END IF;
                        END IF;
                    end if;
                WHEN OTHERS =>
                        State <= Dolar;
            END CASE;
            END IF;
        END IF;
    END PROCESS;

Receiver: ENTITY WORK.uart_rx(rtl)
    PORT MAP(
    i_Clk       => Clk,
    rst     => Reset,
    i_RX_Serial => Rx,
    o_RX_DV     => RX_DV,
    o_RX_Byte   => Rx_Byte
    );
END architecture;
```

List VHDL program of Buffer_ASCII.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Buffer_ASCII is
    port(
        clk         : in std_logic;
        rst         : in std_logic;
        eject_data  : in std_logic;
        Lat_GPS1_in : in std_logic_vector(7 downto 0);
        Lat_GPS2_in : in std_logic_vector(7 downto 0);
        Lat_GPS3_in : in std_logic_vector(7 downto 0);
        Lat_GPS4_in : in std_logic_vector(7 downto 0);
        Lon_GPS1_in : in std_logic_vector(7 downto 0);
        Lon_GPS2_in : in std_logic_vector(7 downto 0);
        Lon_GPS3_in : in std_logic_vector(7 downto 0);
        Lon_GPS4_in : in std_logic_vector(7 downto 0);
        Alt_GPS1_in : in std_logic_vector(7 downto 0);
        Alt_GPS2_in : in std_logic_vector(7 downto 0);
        Alt_GPS3_in : in std_logic_vector(7 downto 0);
        Alt_GPS4_in : in std_logic_vector(7 downto 0);
        Sat_GPS1_in : in std_logic_vector(7 downto 0);
        Sat_GPS2_in : in std_logic_vector(7 downto 0);
        Sat_GPS3_in : in std_logic_vector(7 downto 0);
        Sat_GPS4_in : in std_logic_vector(7 downto 0);
        done        : out std_logic;
        Lat_1_input : out std_logic_vector(7 downto 0);
        Lat_2_input : out std_logic_vector(7 downto 0);
        Lat_3_input : out std_logic_vector(7 downto 0);
        Lat_4_input : out std_logic_vector(7 downto 0);
        Lon_1_input : out std_logic_vector(7 downto 0);
        Lon_2_input : out std_logic_vector(7 downto 0);
        Lon_3_input : out std_logic_vector(7 downto 0);
        Lon_4_input : out std_logic_vector(7 downto 0);
        Alt_1_input : out std_logic_vector(7 downto 0);
        Alt_2_input : out std_logic_vector(7 downto 0);
        Alt_3_input : out std_logic_vector(7 downto 0);
        Alt_4_input : out std_logic_vector(7 downto 0);
        Sat_1_input : out std_logic_vector(7 downto 0);
        Sat_2_input : out std_logic_vector(7 downto 0);
        Sat_3_input : out std_logic_vector(7 downto 0);
        Sat_4_input : out std_logic_vector(7 downto 0);
        Lat_1       : out std_logic_vector(7 downto 0);
        Lat_2       : out std_logic_vector(7 downto 0);
        Lat_3       : out std_logic_vector(7 downto 0);
        Lat_4       : out std_logic_vector(7 downto 0);
        Lon_1       : out std_logic_vector(7 downto 0);
        Lon_2       : out std_logic_vector(7 downto 0);
        Lon_3       : out std_logic_vector(7 downto 0);
        Lon_4       : out std_logic_vector(7 downto 0);
        Alt_1       : out std_logic_vector(7 downto 0);
        Alt_2       : out std_logic_vector(7 downto 0);
        Alt_3       : out std_logic_vector(7 downto 0);
        Alt_4       : out std_logic_vector(7 downto 0);
        Sat_1       : out std_logic_vector(7 downto 0);
        Sat_2       : out std_logic_vector(7 downto 0);
        Sat_3       : out std_logic_vector(7 downto 0);
        Sat_4       : out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of Buffer_ASCII is
    constant counter : integer := 57292;
    type lat_s     is (idle, digit1, digit2, digit3, digit4, digit5,
                      digit6, digit7, digit8, digit9, digit10,send);
    type lon_s     is (idle, digit1, digit2, digit3, digit4, digit5,
                      digit6, digit7, digit8, digit9, digit10, digit11,send);
    type sat_s     is (idle, digit1, digit2,send);
    type alt_s     is (idle, digit1, digit2, digit3, digit4, digit5,send);
    type lat_array is array (0 to 43) of std_logic_vector(7 downto 0);
    type lon_array is array (0 to 47) of std_logic_vector(7 downto 0);
    type sat_array is array (0 to 11) of std_logic_vector(7 downto 0);
    type alt_array is array (0 to 23) of std_logic_vector(7 downto 0);
    signal Lat1_State : lat_s;
    signal Lat2_State : lat_s;
    signal Lat3_State : lat_s;
    signal Lat4_State : lat_s;
    signal Lon1_State : lon_s;
    signal Lon2_State : lon_s;
    signal Lon3_State : lon_s;
    signal Lon4_State : lon_s;
    signal Sat1_State : sat_s;
    signal Sat2_State : sat_s;
    signal Sat3_State : sat_s;
    signal Sat4_State : sat_s;
    signal Alt1_State : alt_s;
    signal Alt2_State : alt_s;
    signal Alt3_State : alt_s;
    signal Alt4_State : alt_s;
    signal lat_data  : lat_array :=
(X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",
X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X
"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00");
    signal lon_data  : lon_array :=
(X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",
X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X
"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"
00",X"00",X"00");
    signal sat_data  : sat_array :=
(X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00");
```

```
    signal alt_data  : alt_array :=
(X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",
X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00",X"00");
    signal Lat_1_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_2_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_3_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_4_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_1_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_2_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_3_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_4_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_1_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_2_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_3_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_4_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_1_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_2_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_3_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_4_sig : std_logic_vector(7 downto 0) := (others => '0');


    signal Lat_1_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_2_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_3_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lat_4_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_1_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_2_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_3_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Lon_4_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_1_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_2_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_3_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Alt_4_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_1_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_2_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_3_out_sig : std_logic_vector(7 downto 0) := (others => '0');
    signal Sat_4_out_sig : std_logic_vector(7 downto 0) := (others => '0');


    signal done_Lat_1 : std_logic := '0';
    signal done_Lat_2 : std_logic := '0';
    signal done_Lat_3 : std_logic := '0';
    signal done_Lat_4 : std_logic := '0';
    signal done_Lon_1 : std_logic := '0';
    signal done_Lon_2 : std_logic := '0';
    signal done_Lon_3 : std_logic := '0';
    signal done_Lon_4 : std_logic := '0';
    signal done_Alt_1 : std_logic := '0';
    signal done_Alt_2 : std_logic := '0';
    signal done_Alt_3 : std_logic := '0';
    signal done_Alt_4 : std_logic := '0';
signal done_Sat_1 : std_logic := '0';
signal done_Sat_2 : std_logic := '0';
signal done_Sat_3 : std_logic := '0';
signal done_Sat_4 : std_logic := '0';

signal keluarkan_data : std_logic := '0';
signal lat_index1: integer := 0;
signal lat_index2: integer := 11;
signal lat_index3: integer := 22;
signal lat_index4: integer := 33;
signal lon_index1: integer := 0;
signal lon_index2: integer := 12;
signal lon_index3: integer := 24;
signal lon_index4: integer := 36;
signal alt_index1: integer := 0;
signal alt_index2: integer := 6;
signal alt_index3: integer := 12;
signal alt_index4: integer := 18;
signal sat_index1: integer := 0;
signal sat_index2: integer := 3;
signal sat_index3: integer := 6;
signal sat_index4: integer := 9;

signal lat_index1_done: integer := 0;
signal lat_index2_done: integer := 11;
signal lat_index3_done: integer := 22;
signal lat_index4_done: integer := 33;
signal lon_index1_done: integer := 0;
signal lon_index2_done: integer := 12;
signal lon_index3_done: integer := 24;
signal lon_index4_done: integer := 36;
signal alt_index1_done: integer := 0;
signal alt_index2_done: integer := 6;
signal alt_index3_done: integer := 12;
signal alt_index4_done: integer := 18;
signal sat_index1_done: integer := 0;
signal sat_index2_done: integer := 3;
signal sat_index3_done: integer := 6;
signal sat_index4_done: integer := 9;

signal cnt_lat   : integer := 0;
signal cnt_lon   : integer := 0;
signal cnt_alt   : integer := 0;
signal cnt_sat   : integer := 0;
signal cnt_lat2  : integer := 0;
signal cnt_lon2  : integer := 0;
signal cnt_alt2  : integer := 0;
signal cnt_sat2  : integer := 0;

signal lat1_cnt  : integer := 0;
signal lat2_cnt  : integer := 0;
```

```
    signal lat3_cnt  : integer := 0;
    signal lat4_cnt  : integer := 0;
    signal lon1_cnt  : integer := 0;
    signal lon2_cnt  : integer := 0;
    signal lon3_cnt  : integer := 0;
    signal lon4_cnt  : integer := 0;
    signal alt1_cnt  : integer := 0;
    signal alt2_cnt  : integer := 0;
    signal alt3_cnt  : integer := 0;
    signal alt4_cnt  : integer := 0;
    signal sat1_cnt  : integer := 0;
    signal sat2_cnt  : integer := 0;
    signal sat3_cnt  : integer := 0;
    signal sat4_cnt  : integer := 0;

    signal done_sig  : std_logic := '0';
    signal done_param: std_logic_vector(15 downto 0) := (others => '0');
    signal done_altitude : std_logic := '0';
begin
    keluarkan_data <= eject_data;
    done    <= done_sig;

    Lat_1_input <= Lat_1_sig;
    Lat_2_input <= Lat_2_sig;
    Lat_3_input <= Lat_3_sig;
    Lat_4_input <= Lat_4_sig;
    Lon_1_input <= Lon_1_sig;
    Lon_2_input <= Lon_2_sig;
    Lon_3_input <= Lon_3_sig;
    Lon_4_input <= Lon_4_sig;
    Alt_1_input <= Alt_1_sig;
    Alt_2_input <= Alt_2_sig;
    Alt_3_input <= Alt_3_sig;
    Alt_4_input <= Alt_4_sig;
    Sat_1_input <= Sat_1_sig;
    Sat_2_input <= Sat_2_sig;
    Sat_3_input <= Sat_3_sig;
    Sat_4_input <= Sat_4_sig;

    Lat_1   <= Lat_1_out_sig;
    Lat_2   <= Lat_2_out_sig;
    Lat_3   <= Lat_3_out_sig;
    Lat_4   <= Lat_4_out_sig;
    Lon_1   <= Lon_1_out_sig;
    Lon_2   <= Lon_2_out_sig;
    Lon_3   <= Lon_3_out_sig;
    Lon_4   <= Lon_4_out_sig;
    Alt_1   <= Alt_1_out_sig;
    Alt_2   <= Alt_2_out_sig;
    Alt_3   <= Alt_3_out_sig;
    Alt_4   <= Alt_4_out_sig;
```

```
Sat_1   <= Sat_1_out_sig;
Sat_2   <= Sat_2_out_sig;
Sat_3   <= Sat_3_out_sig;
Sat_4   <= Sat_4_out_sig;

done_Process : process(clk) is
begin
if rising_edge(clk) then
if done_Alt_1 = '1' and done_Alt_2 = '1' and done_Alt_3 = '1' and done_Alt_4 = '1' then
     done_altitude <= '1';
elsif done_Alt_1 = '0' and done_Alt_2 = '0' and done_Alt_3 = '0'
    and done_Alt_4 = '0' and  done_Lon_1 = '1' and done_Lon_2 = '1' and done_Lon_3 = '1'
    and done_Lon_4 = '1' then
     done_altitude <= '0';
end if;
end if;
end process;

Lattitude1 : process(clk,rst)
begin
    if rst = '0' then
        Lat_1_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lat1_State is
            when idle  =>
                Lat_1_sig <= "00000000";
                if Lat_GPS1_in = "00000000"  then
                    Lat1_State <= idle;
                else
                    Lat1_State <= digit1;
                    lat_data(0) <= Lat_GPS1_in;
                end if;
            when digit1 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit1;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit2;
                    lat_data(1) <= Lat_GPS1_in;
                end if;
            when digit2 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit2;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit3;
                    lat_data(2) <= Lat_GPS1_in;
                end if;
            when digit3 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit3;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit4;
                    lat_data(3) <= Lat_GPS1_in;
                end if;
            when digit4 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit4;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit5;
                    lat_data(4) <= Lat_GPS1_in;
                end if;
            when digit5 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit5;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit6;
                    lat_data(5) <= Lat_GPS1_in;
                end if;
            when digit6 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit6;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit7;
                    lat_data(6) <= Lat_GPS1_in;
                end if;
            when digit7 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit7;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit8;
                    lat_data(7) <= Lat_GPS1_in;
                end if;
            when digit8 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit8;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit9;
```

```
                    lat_data(8) <= Lat_GPS1_in;
                end if;
            when digit9 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit9;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= digit10;
                    lat_data(9) <= Lat_GPS1_in;
                    lat_data(10)<= "00101100";
                end if;
            when digit10 =>
                if lat1_cnt < counter - 1 then
                    lat1_cnt <= lat1_cnt + 1;
                    Lat1_State <= digit10;
                else
                    lat1_cnt      <= 0;
                    Lat1_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if lat_index1_done < 11  then
                        Lat_1_sig   <= lat_data(lat_index1_done);
                        Lat1_State  <= send;
                        if lat1_cnt < counter-1 then
                            lat1_cnt <= lat1_cnt + 1;
                        else
                            lat1_cnt <= 0;
                            lat_index1_done <= lat_index1_done + 1;
                        end if;
                    else
                        Lat1_State <= idle;
                        lat_index1_done <= 0;
                    end if;
                else
                    Lat1_State <= send;
                end if;
            end case;
    end if;
end process;

Lattitude2 : process(clk,rst)
begin
    if rst = '0' then
        Lat_2_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lat2_State is
            when idle  =>
                Lat_2_sig <= "00000000";
                if Lat_GPS2_in = "00000000" then
        Lat2_State <= idle;
    else
        Lat2_State <= digit1;
        lat_data(11)<= Lat_GPS2_in;
    end if;
when digit1 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit1;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit2;
        lat_data(12)<= Lat_GPS2_in;
    end if;
when digit2 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit2;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit3;
        lat_data(13)<= Lat_GPS2_in;
    end if;
when digit3 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit3;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit4;
        lat_data(14)<= Lat_GPS2_in;
    end if;
when digit4 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit4;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit5;
        lat_data(15)<= Lat_GPS2_in;
    end if;
when digit5 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit5;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit6;
        lat_data(16)<= Lat_GPS2_in;
    end if;
when digit6 =>
```

```
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit6;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit7;
        lat_data(17)<= Lat_GPS2_in;
    end if;
when digit7 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit7;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit8;
        lat_data(18)<= Lat_GPS2_in;
    end if;
when digit8 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit8;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit9;
        lat_data(19)<= Lat_GPS2_in;
    end if;
when digit9 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit9;
    else
        lat2_cnt      <= 0;
        Lat2_State <= digit10;
        lat_data(20)<= Lat_GPS2_in;
        lat_data(21)<= "00101100";
    end if;
when digit10 =>
    if lat2_cnt < counter - 1 then
        lat2_cnt <= lat2_cnt + 1;
        Lat2_State <= digit10;
    else
        lat2_cnt      <= 0;
        Lat2_State <= send;
    end if;
when send =>
    if done_altitude = '1' then
        if lat_index2_done < 22 then
            Lat2_State  <= send;
            Lat_2_sig   <= lat_data(lat_index2_done);
            if lat2_cnt < counter-1 then
                lat2_cnt <= lat2_cnt + 1;
```

```
                        else
                            lat2_cnt <= 0;
                            lat_index2_done <= lat_index2_done + 1;
                        end if;
                    else
                        Lat2_State <= idle;
                        lat_index2_done <= 11;
                    end if;
                else
                    Lat2_State <= send;
                end if;
            end case;
    end if;
end process;

Lattitude3 : process(clk,rst)
begin
    if rst = '0' then
        Lat_3_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lat3_State is
            when idle  =>
                Lat_3_sig <= "00000000";
                if Lat_GPS3_in = "00000000" then
                    Lat3_State <= idle;
                else
                    Lat3_State <= digit1;
                    lat_data(22)<= Lat_GPS3_in;
                end if;
            when digit1 =>
                if lat3_cnt < counter - 1 then
                    lat3_cnt <= lat3_cnt + 1;
                    Lat3_State <= digit1;
                else
                    lat3_cnt      <= 0;
                    Lat3_State <= digit2;
                    lat_data(23)<= Lat_GPS3_in;
                end if;
            when digit2 =>
                if lat3_cnt < counter - 1 then
                    lat3_cnt <= lat3_cnt + 1;
                    Lat3_State <= digit2;
                else
                    lat3_cnt      <= 0;
                    Lat3_State <= digit3;
                    lat_data(24)<= Lat_GPS3_in;
                end if;
            when digit3 =>
                if lat3_cnt < counter - 1 then
                    lat3_cnt <= lat3_cnt + 1;
                    Lat3_State <= digit3;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit4;
        lat_data(25)<= Lat_GPS3_in;
    end if;
when digit4 =>
    if lat3_cnt < counter - 1 then
        lat3_cnt <= lat3_cnt + 1;
        Lat3_State <= digit4;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit5;
        lat_data(26)<= Lat_GPS3_in;
    end if;
when digit5 =>
    if lat3_cnt < counter - 1 then
        lat3_cnt <= lat3_cnt + 1;
        Lat3_State <= digit5;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit6;
        lat_data(27)<= Lat_GPS3_in;
    end if;
when digit6 =>
    if lat3_cnt < counter - 1 then
        lat3_cnt <= lat3_cnt + 1;
        Lat3_State <= digit6;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit7;
        lat_data(28)<= Lat_GPS3_in;
    end if;
when digit7 =>
    if lat3_cnt < counter - 1 then
        lat3_cnt <= lat3_cnt + 1;
        Lat3_State <= digit7;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit8;
        lat_data(29)<= Lat_GPS3_in;
    end if;
when digit8 =>
    if lat3_cnt < counter - 1 then
        lat3_cnt <= lat3_cnt + 1;
        Lat3_State <= digit8;
    else
        lat3_cnt      <= 0;
        Lat3_State <= digit9;
        lat_data(30)<= Lat_GPS3_in;
    end if;
when digit9 =>
```

```
                if lat3_cnt < counter - 1 then
                    lat3_cnt <= lat3_cnt + 1;
                    Lat3_State <= digit9;
                else
                    lat3_cnt      <= 0;
                    Lat3_State <= digit10;
                    lat_data(31)<= Lat_GPS3_in;
                    lat_data(32)<= "00101100";
                end if;
            when digit10 =>
                if lat3_cnt < counter - 1 then
                    lat3_cnt <= lat3_cnt + 1;
                    Lat3_State <= digit10;
                else
                    lat3_cnt      <= 0;
                    Lat3_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if lat_index3_done < 33 then
                        Lat3_State  <= send;
                        Lat_3_sig   <= lat_data(lat_index3_done);
                        if lat3_cnt < counter-1 then
                            lat3_cnt <= lat3_cnt + 1;
                        else
                            lat3_cnt <= 0;
                            lat_index3_done <= lat_index3_done + 1;
                        end if;
                    else
                        Lat3_State <= idle;
                        lat_index3_done <= 22;
                    end if;
                else
                    Lat3_State <= send;
                end if;
            end case;
    end if;
end process;

Lattitude4 : process(clk,rst)
begin
    if rst = '0' then
        Lat_4_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lat4_State is
            when idle  =>
                Lat_4_sig <= "00000000";
                if Lat_GPS4_in = "00000000" then
                    Lat4_State <= idle;
                else
                    Lat4_State <= digit1;
        lat_data(33)<= Lat_GPS4_in;
    end if;
when digit1 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit1;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit2;
        lat_data(34)<= Lat_GPS4_in;
    end if;
when digit2 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit2;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit3;
        lat_data(35)<= Lat_GPS4_in;
    end if;
when digit3 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit3;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit4;
        lat_data(36)<= Lat_GPS4_in;
    end if;
when digit4 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit4;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit5;
        lat_data(37)<= Lat_GPS4_in;
    end if;
when digit5 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit5;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit6;
        lat_data(38)<= Lat_GPS4_in;
    end if;
when digit6 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit6;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit7;
        lat_data(39)<= Lat_GPS4_in;
    end if;
when digit7 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit7;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit8;
        lat_data(40)<= Lat_GPS4_in;
    end if;
when digit8 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit8;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit9;
        lat_data(41)<= Lat_GPS4_in;
    end if;
when digit9 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit9;
    else
        lat4_cnt      <= 0;
        Lat4_State <= digit10;
        lat_data(42)<= Lat_GPS4_in;
        lat_data(43)<= "00101100";
    end if;
when digit10 =>
    if lat4_cnt < counter - 1 then
        lat4_cnt <= lat4_cnt + 1;
        Lat4_State <= digit10;
    else
        lat4_cnt      <= 0;
        Lat4_State <= send;
    end if;
when send =>
    if done_altitude = '1' then
        if lat_index4_done < 44  then
            Lat4_State  <= send;
            Lat_4_sig   <= lat_data(lat_index4_done);
            if lat4_cnt < counter-1 then
                lat4_cnt <= lat4_cnt + 1;
            else
                lat4_cnt <= 0;
                lat_index4_done <= lat_index4_done + 1;
```

```
                        end if;
                    else
                        Lat4_State <= idle;
                        lat_index4_done <= 33;
                    end if;
                else
                    Lat4_State <= send;
                end if;
            end case;
    end if;
end process;

Longitude1 : process(clk,rst)
begin
    if rst = '0' then
        Lon_1_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lon1_State is
            when idle  =>
                done_Lon_1 <= '0';
                Lon_1_sig <= "00000000";
                if Lon_GPS1_in = "00000000" then
                    Lon1_State <= idle;
                else
                    Lon1_State <= digit1;
                    lon_data(0) <= Lon_GPS1_in;
                end if;
            when digit1 =>
                if lon1_cnt < counter - 1 then
                    lon1_cnt <= lon1_cnt + 1;
                    Lon1_State <= digit1;
                else
                    lon1_cnt      <= 0;
                    Lon1_State <= digit2;
                    lon_data(1) <= Lon_GPS1_in;
                end if;
            when digit2 =>
                if lon1_cnt < counter - 1 then
                    lon1_cnt <= lon1_cnt + 1;
                    Lon1_State <= digit2;
                else
                    lon1_cnt      <= 0;
                    Lon1_State <= digit3;
                    lon_data(2) <= Lon_GPS1_in;
                end if;
            when digit3 =>
                if lon1_cnt < counter - 1 then
                    lon1_cnt <= lon1_cnt + 1;
                    Lon1_State <= digit3;
                else
                    lon1_cnt      <= 0;
        Lon1_State <= digit4;
        lon_data(3) <= Lon_GPS1_in;
    end if;
when digit4 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
        Lon1_State <= digit4;
    else
        lon1_cnt      <= 0;
        Lon1_State <= digit5;
        lon_data(4) <= Lon_GPS1_in;
    end if;
when digit5 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
        Lon1_State <= digit5;
    else
        lon1_cnt      <= 0;
        Lon1_State <= digit6;
        lon_data(5) <= Lon_GPS1_in;
    end if;
when digit6 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
        Lon1_State <= digit6;
    else
        lon1_cnt      <= 0;
        Lon1_State <= digit7;
        lon_data(6) <= Lon_GPS1_in;
    end if;
when digit7 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
        Lon1_State <= digit7;
    else
        lon1_cnt      <= 0;
        Lon1_State <= digit8;
        lon_data(7) <= Lon_GPS1_in;
    end if;
when digit8 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
        Lon1_State <= digit8;
    else
        lon1_cnt      <= 0;
        Lon1_State <= digit9;
        lon_data(8) <= Lon_GPS1_in;
    end if;
when digit9 =>
    if lon1_cnt < counter - 1 then
        lon1_cnt <= lon1_cnt + 1;
```

```
                    Lon1_State <= digit9;
                else
                    lon1_cnt      <= 0;
                    Lon1_State <= digit10;
                    lon_data(9) <= Lon_GPS1_in;
                end if;
            when digit10 =>
                if lon1_cnt < counter - 1 then
                    lon1_cnt <= lon1_cnt + 1;
                    Lon1_State <= digit10;
                else
                    lon1_cnt      <= 0;
                    Lon1_State <= digit11;
                    lon_data(10)<= Lon_GPS1_in;
                    lon_data(11)<= "00101100";
                end if;
            when digit11 =>
                if lon1_cnt < counter - 1 then
                    lon1_cnt <= lon1_cnt + 1;
                    Lon1_State <= digit11;
                else
                    lon1_cnt      <= 0;
                    Lon1_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if lon_index1_done < 12 then
                        done_sig <= '1';
                        Lon1_State  <= send;
                        Lon_1_sig   <= lon_data(lon_index1_done);
                        if lon1_cnt < counter-1 then
                            lon1_cnt <= lon1_cnt + 1;
                        else
                            lon1_cnt <= 0;
                            lon_index1_done <= lon_index1_done + 1;
                        end if;
                    else
                        done_Lon_1 <= '1';
                        done_sig <= '0';
                        Lon1_State <= idle;
                        lon_index1_done <= 0;
                    end if;
                else
                    Lon1_State <= send;
                end if;
            end case;
    end if;
end process;

Longitude2 : process(clk,rst)
begin
```

```vhdl
if rst = '0' then
    Lon_2_sig <= (others => '0');
elsif rising_edge(clk) then
    case Lon2_State is
        when idle  =>
            done_Lon_2 <= '0';
            Lon_2_sig <= "00000000";
            if Lon_GPS2_in = "00000000" then
                Lon2_State <= idle;
            else
                Lon2_State <= digit1;
                lon_data(12)<= Lon_GPS2_in;
            end if;
        when digit1 =>
            if lon2_cnt < counter - 1 then
                lon2_cnt <= lon2_cnt + 1;
                Lon2_State <= digit1;
            else
                lon2_cnt      <= 0;
                Lon2_State <= digit2;
                lon_data(13)<= Lon_GPS2_in;
            end if;
        when digit2 =>
            if lon2_cnt < counter - 1 then
                lon2_cnt <= lon2_cnt + 1;
                Lon2_State <= digit2;
            else
                lon2_cnt      <= 0;
                Lon2_State <= digit3;
                lon_data(14)<= Lon_GPS2_in;
            end if;
        when digit3 =>
            if lon2_cnt < counter - 1 then
                lon2_cnt <= lon2_cnt + 1;
                Lon2_State <= digit3;
            else
                lon2_cnt      <= 0;
                Lon2_State <= digit4;
                lon_data(15)<= Lon_GPS2_in;
            end if;
        when digit4 =>
            if lon2_cnt < counter - 1 then
                lon2_cnt <= lon2_cnt + 1;
                Lon2_State <= digit4;
            else
                lon2_cnt      <= 0;
                Lon2_State <= digit5;
                lon_data(16)<= Lon_GPS2_in;
            end if;
        when digit5 =>
            if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit5;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit6;
            lon_data(17)<= Lon_GPS2_in;
        end if;
    when digit6 =>
        if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit6;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit7;
            lon_data(18)<= Lon_GPS2_in;
        end if;
    when digit7 =>
        if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit7;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit8;
            lon_data(19)<= Lon_GPS2_in;
        end if;
    when digit8 =>
        if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit8;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit9;
            lon_data(20)<= Lon_GPS2_in;
        end if;
    when digit9 =>
        if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit9;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit10;
            lon_data(21)<= Lon_GPS2_in;
        end if;
    when digit10 =>
        if lon2_cnt < counter - 1 then
            lon2_cnt <= lon2_cnt + 1;
            Lon2_State <= digit10;
        else
            lon2_cnt      <= 0;
            Lon2_State <= digit11;
            lon_data(22)<= Lon_GPS2_in;
```

```
                    lon_data(23)<= "00101100";
                end if;
            when digit11 =>
                if lon2_cnt < counter - 1 then
                    lon2_cnt <= lon2_cnt + 1;
                    Lon2_State <= digit11;
                else
                    lon2_cnt      <= 0;
                    Lon2_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if lon_index2_done < 24 then
                        Lon2_State  <= send;
                        Lon_2_sig   <= lon_data(lon_index2_done);
                        if lon2_cnt < counter-1 then
                            lon2_cnt <= lon2_cnt + 1;
                        else
                            lon2_cnt <= 0;
                            lon_index2_done <= lon_index2_done + 1;
                        end if;
                    else
                        done_Lon_2 <= '1';
                        Lon2_State <= idle;
                        lon_index2_done <= 12;
                    end if;
                else
                    Lon2_State <= send;
                end if;
            end case;
    end if;
end process;

Longitude3 : process(clk,rst)
begin
    if rst = '0' then
        Lon_3_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lon3_State is
            when idle  =>
                done_Lon_3 <= '0';
                Lon_3_sig <= "00000000";
                if Lon_GPS3_in = "00000000" then
                    Lon3_State <= idle;
                else
                    Lon3_State <= digit1;
                    lon_data(24)<= Lon_GPS3_in;
                end if;
            when digit1 =>
                if lon3_cnt < counter - 1 then
                    lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit1;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit2;
        lon_data(25)<= Lon_GPS3_in;
    end if;
when digit2 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit2;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit3;
        lon_data(26)<= Lon_GPS3_in;
    end if;
when digit3 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit3;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit4;
        lon_data(27)<= Lon_GPS3_in;
    end if;
when digit4 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit4;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit5;
        lon_data(28)<= Lon_GPS3_in;
    end if;
when digit5 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit5;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit6;
        lon_data(29)<= Lon_GPS3_in;
    end if;
when digit6 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit6;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit7;
        lon_data(30)<= Lon_GPS3_in;
    end if;
```

```
when digit7 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit7;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit8;
        lon_data(31)<= Lon_GPS3_in;
    end if;
when digit8 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit8;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit9;
        lon_data(32)<= Lon_GPS3_in;
    end if;
when digit9 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit9;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit10;
        lon_data(33)<= Lon_GPS3_in;
    end if;
when digit10 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit10;
    else
        lon3_cnt      <= 0;
        Lon3_State <= digit11;
        lon_data(34)<= Lon_GPS3_in;
        lon_data(35)<= "00101100";
    end if;
when digit11 =>
    if lon3_cnt < counter - 1 then
        lon3_cnt <= lon3_cnt + 1;
        Lon3_State <= digit11;
    else
        lon3_cnt      <= 0;
        Lon3_State <= send;
    end if;
when send =>
    if done_altitude = '1' then
        if lon_index3_done < 36 then
            Lon3_State  <= send;
            Lon_3_sig   <= lon_data(lon_index3_done);
            if lon3_cnt < counter-1 then
```

```
                            lon3_cnt <= lon3_cnt + 1;
                        else
                            lon3_cnt <= 0;
                            lon_index3_done <= lon_index3_done + 1;
                        end if;
                    else
                        done_Lon_3 <= '1';
                        Lon3_State <= idle;
                        lon_index3_done <= 24;
                    end if;
                else
                    Lon3_State <= send;
                end if;
            end case;
    end if;
end process;

Longitude4 : process(clk,rst)
begin
    if rst = '0' then
        Lon_4_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Lon4_State is
            when idle  =>
                done_Lon_4 <= '0';
                Lon_4_sig <= "00000000";
                if Lon_GPS4_in = "00000000" then
                    Lon4_State <= idle;
                else
                    Lon4_State <= digit1;
                    lon_data(36)<= Lon_GPS4_in;
                end if;
            when digit1 =>
                if lon4_cnt < counter - 1 then
                    lon4_cnt <= lon4_cnt + 1;
                    Lon4_State <= digit1;
                else
                    lon4_cnt      <= 0;
                    Lon4_State <= digit2;
                    lon_data(37)<= Lon_GPS4_in;
                end if;
            when digit2 =>
                if lon4_cnt < counter - 1 then
                    lon4_cnt <= lon4_cnt + 1;
                    Lon4_State <= digit2;
                else
                    lon4_cnt      <= 0;
                    Lon4_State <= digit3;
                    lon_data(38)<= Lon_GPS4_in;
                end if;
            when digit3 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit3;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit4;
        lon_data(39)<= Lon_GPS4_in;
    end if;
when digit4 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit4;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit5;
        lon_data(40)<= Lon_GPS4_in;
    end if;
when digit5 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit5;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit6;
        lon_data(41)<= Lon_GPS4_in;
    end if;
when digit6 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit6;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit7;
        lon_data(42)<= Lon_GPS4_in;
    end if;
when digit7 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit7;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit8;
        lon_data(43)<= Lon_GPS4_in;
    end if;
when digit8 =>
    if lon4_cnt < counter - 1 then
        lon4_cnt <= lon4_cnt + 1;
        Lon4_State <= digit8;
    else
        lon4_cnt      <= 0;
        Lon4_State <= digit9;
```

```
                    lon_data(44)<= Lon_GPS4_in;
                end if;
            when digit9 =>
                if lon4_cnt < counter - 1 then
                    lon4_cnt <= lon4_cnt + 1;
                    Lon4_State <= digit9;
                else
                    lon4_cnt      <= 0;
                    Lon4_State <= digit10;
                    lon_data(45)<= Lon_GPS4_in;
                end if;
            when digit10 =>
                if lon4_cnt < counter - 1 then
                    lon4_cnt <= lon4_cnt + 1;
                    Lon4_State <= digit10;
                else
                    lon4_cnt      <= 0;
                    Lon4_State <= digit11;
                    lon_data(46)<= Lon_GPS4_in;
                    lon_data(47)<= "00101100";
                end if;
            when digit11 =>
                if lon4_cnt < counter - 1 then
                    lon4_cnt <= lon4_cnt + 1;
                    Lon4_State <= digit11;
                else
                    lon4_cnt      <= 0;
                    Lon4_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if lon_index4_done < 48 then
                        Lon4_State  <= send;
                        Lon_4_sig   <= lon_data(lon_index4_done);
                        if lon4_cnt < counter-1 then
                            lon4_cnt <= lon4_cnt + 1;
                        else
                            lon4_cnt <= 0;
                            lon_index4_done <= lon_index4_done + 1;
                        end if;
                    else
                        done_Lon_4 <= '1';
                        Lon4_State <= idle;
                        lon_index4_done <= 36;
                    end if;
                else
                    Lon4_State <= send;
                end if;
            end case;
    end if;
end process;
```

```
Altitude1 : process(clk,rst)
begin
    if rst = '0' then
        Alt_1_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Alt1_State is
            when idle  =>
                Alt_1_sig <= "00000000";
                if Alt_GPS1_in = "00000000" then
                    Alt1_State <= idle;
                else
                    Alt1_State <= digit1;
                    alt_data(0) <= Alt_GPS1_in;
                end if;
            when digit1 =>
                if alt1_cnt < counter - 1 then
                    alt1_cnt <= alt1_cnt + 1;
                    Alt1_State <= digit1;
                else
                    alt1_cnt     <= 0;
                    Alt1_State <= digit2;
                    alt_data(1) <= Alt_GPS1_in;
                end if;
            when digit2 =>
                if alt1_cnt < counter - 1 then
                    alt1_cnt <= alt1_cnt + 1;
                    Alt1_State <= digit2;
                else
                    alt1_cnt     <= 0;
                    Alt1_State <= digit3;
                    alt_data(2) <= Alt_GPS1_in;
                end if;
            when digit3 =>
                if alt1_cnt < counter - 1 then
                    alt1_cnt <= alt1_cnt + 1;
                    Alt1_State <= digit3;
                else
                    alt1_cnt     <= 0;
                    Alt1_State <= digit4;
                    alt_data(3) <= Alt_GPS1_in;
                end if;
            when digit4 =>
                if alt1_cnt < counter - 1 then
                    alt1_cnt <= alt1_cnt + 1;
                    Alt1_State <= digit4;
                else
                    alt1_cnt     <= 0;
                    Alt1_State <= digit5;
                    alt_data(4) <= Alt_GPS1_in;
                    alt_data(5) <= "00101100";
                end if;
            when digit5 =>
                if alt1_cnt < counter - 1 then
                    alt1_cnt <= alt1_cnt + 1;
                    Alt1_State <= digit5;
                else
                    alt1_cnt      <= 0;
                    done_Alt_1  <= '1';
                    Alt1_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if alt_index1_done < 6 then
                        Alt1_State  <= send;
                        Alt_1_sig   <= alt_data(alt_index1_done);
                        if alt1_cnt < counter-1 then
                            alt1_cnt <= alt1_cnt + 1;
                        else
                            alt1_cnt <= 0;
                            alt_index1_done <= alt_index1_done + 1;
                        end if;
                    else
                        done_Alt_1 <= '0';
                        Alt1_State <= idle;
                        alt_index1_done <= 0;
                    end if;
                else
                    Alt1_State <= send;
                end if;
            end case;
    end if;
end process;

Altitude2 : process(clk,rst)
begin
    if rst = '0' then
        Alt_2_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Alt2_State is
            when idle  =>
                Alt_2_sig <= "00000000";
                if Alt_GPS2_in = "00000000" then
                    Alt2_State <= idle;
                else
                    Alt2_State <= digit1;
                    alt_data(6) <= Alt_GPS2_in;
                end if;
            when digit1 =>
                if alt2_cnt < counter - 1 then
                    alt2_cnt <= alt2_cnt + 1;
                    Alt2_State <= digit1;
                else
                    alt2_cnt      <= 0;
                    Alt2_State <= digit2;
                    alt_data(7) <= Alt_GPS2_in;
                end if;
            when digit2 =>
                if alt2_cnt < counter - 1 then
                    alt2_cnt <= alt2_cnt + 1;
                    Alt2_State <= digit2;
                else
                    alt2_cnt      <= 0;
                    Alt2_State <= digit3;
                    alt_data(8) <= Alt_GPS2_in;
                end if;
            when digit3 =>
                if alt2_cnt < counter - 1 then
                    alt2_cnt <= alt2_cnt + 1;
                    Alt2_State <= digit3;
                else
                    alt2_cnt      <= 0;
                    Alt2_State <= digit4;
                    alt_data(9) <= Alt_GPS2_in;
                end if;
            when digit4 =>
                if alt2_cnt < counter - 1 then
                    alt2_cnt <= alt2_cnt + 1;
                    Alt2_State <= digit4;
                else
                    alt2_cnt      <= 0;
                    Alt2_State <= digit5;
                    alt_data(10)<= Alt_GPS2_in;
                    alt_data(11)<= "00101100";
                end if;
            when digit5 =>
                if alt2_cnt < counter - 1 then
                    alt2_cnt <= alt2_cnt + 1;
                    Alt2_State <= digit5;
                else
                    done_Alt_2 <= '1';
                    alt2_cnt      <= 0;
                    Alt2_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if alt_index2_done < 12  then
                        Alt2_State  <= send;
                        Alt_2_sig   <= alt_data(alt_index2_done);
                        if alt2_cnt < counter-1 then
                            alt2_cnt <= alt2_cnt + 1;
                        else
                            alt2_cnt <= 0;
```

```
                            alt_index2_done <= alt_index2_done + 1;
                        end if;
                    else
                        done_Alt_2 <= '0';
                        Alt2_State <= idle;
                        alt_index2_done <= 6;
                    end if;
                else
                    Alt2_State <= send;
                end if;
            end case;
    end if;
end process;

Altitude3 : process(clk,rst)
begin
    if rst = '0' then
        Alt_3_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Alt3_State is
            when idle  =>
                Alt_3_sig <= "00000000";
                if Alt_GPS3_in = "00000000" then
                    Alt3_State <= idle;
                else
                    Alt3_State <= digit1;
                    alt_data(12)<= Alt_GPS3_in;
                end if;
            when digit1 =>
                if alt3_cnt < counter - 1 then
                    alt3_cnt <= alt3_cnt + 1;
                    Alt3_State <= digit1;
                else
                    alt3_cnt      <= 0;
                    Alt3_State <= digit2;
                    alt_data(13)<= Alt_GPS3_in;
                end if;
            when digit2 =>
                if alt3_cnt < counter - 1 then
                    alt3_cnt <= alt3_cnt + 1;
                    Alt3_State <= digit2;
                else
                    alt3_cnt      <= 0;
                    Alt3_State <= digit3;
                    alt_data(14)<= Alt_GPS3_in;
                end if;
            when digit3 =>
                if alt3_cnt < counter - 1 then
                    alt3_cnt <= alt3_cnt + 1;
                    Alt3_State <= digit3;
                else
```

```
                    alt3_cnt      <= 0;
                    Alt3_State <= digit4;
                    alt_data(15)<= Alt_GPS3_in;
                end if;
            when digit4 =>
                if alt3_cnt < counter - 1 then
                    alt3_cnt <= alt3_cnt + 1;
                    Alt3_State <= digit4;
                else
                    alt3_cnt      <= 0;
                    Alt3_State <= digit5;
                    alt_data(16)<= Alt_GPS3_in;
                    alt_data(17)<= "00101100";
                end if;
            when digit5 =>
                if alt3_cnt < counter - 1 then
                    alt3_cnt <= alt3_cnt + 1;
                    Alt3_State <= digit5;
                else
                    done_Alt_3 <= '1';
                    alt3_cnt      <= 0;
                    Alt3_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if alt_index3_done < 18 then
                        Alt3_State  <= send;
                        Alt_3_sig   <= alt_data(alt_index3_done);
                        if alt3_cnt < counter-1 then
                            alt3_cnt <= alt3_cnt + 1;
                        else
                            alt3_cnt <= 0;
                            alt_index3_done <= alt_index3_done + 1;
                        end if;
                    else
                        done_Alt_3 <= '0';
                        Alt3_State <= idle;
                        alt_index3_done <= 12;
                    end if;
                else
                    Alt3_State <= send;
                end if;
            end case;
    end if;
end process;

Altitude4 : process(clk,rst)
begin
    if rst = '0' then
        Alt_4_sig <= (others => '0');
    elsif rising_edge(clk) then
```

```
case Alt4_State is
    when idle  =>
        Alt_4_sig <= "00000000";
        if Alt_GPS4_in = "00000000" then
            Alt4_State <= idle;
        else
            Alt4_State <= digit1;
            alt_data(18)<= Alt_GPS4_in;
        end if;
    when digit1 =>
        if alt4_cnt < counter - 1 then
            alt4_cnt <= alt4_cnt + 1;
            Alt4_State <= digit1;
        else
            alt4_cnt     <= 0;
            Alt4_State <= digit2;
            alt_data(19)<= Alt_GPS4_in;
        end if;
    when digit2 =>
        if alt4_cnt < counter - 1 then
            alt4_cnt <= alt4_cnt + 1;
            Alt4_State <= digit2;
        else
            alt4_cnt     <= 0;
            Alt4_State <= digit3;
            alt_data(20)<= Alt_GPS4_in;
        end if;
    when digit3 =>
        if alt4_cnt < counter - 1 then
            alt4_cnt <= alt4_cnt + 1;
            Alt4_State <= digit3;
        else
            alt4_cnt     <= 0;
            Alt4_State <= digit4;
            alt_data(21)<= Alt_GPS4_in;
        end if;
    when digit4 =>
        if alt4_cnt < counter - 1 then
            alt4_cnt <= alt4_cnt + 1;
            Alt4_State <= digit4;
        else
            alt4_cnt     <= 0;
            Alt4_State <= digit5;
            alt_data(22)<= Alt_GPS4_in;
            alt_data(23)<= "00101100";
        end if;
    when digit5 =>
        if alt4_cnt < counter - 1 then
            alt4_cnt <= alt4_cnt + 1;
            Alt4_State <= digit5;
        else
```

```
                    done_Alt_4 <= '1';
                    alt4_cnt      <= 0;
                    Alt4_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if alt_index4_done < 24 then
                        Alt4_State  <= send;
                        Alt_4_sig   <= alt_data(alt_index4_done);
                        if alt4_cnt < counter-1 then
                            alt4_cnt <= alt4_cnt + 1;
                        else
                            alt4_cnt <= 0;
                            alt_index4_done <= alt_index4_done + 1;
                        end if;
                    else
                        done_Alt_4 <= '0';
                        Alt4_State <= idle;
                        alt_index4_done <= 18;
                    end if;
                else
                    Alt4_State <= send;
                end if;
            end case;
    end if;
end process;

Satellite1 : process(clk,rst)
begin
    if rst = '0' then
        Sat_1_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Sat1_State is
            when idle  =>
                Sat_1_sig <= "00000000";
                if Sat_GPS1_in = "00000000" then
                    Sat1_State <= idle;
                else
                    Sat1_State <= digit1;
                    sat_data(0) <= Sat_GPS1_in;
                end if;
            when digit1 =>
                if sat1_cnt < counter - 1 then
                    sat1_cnt <= sat1_cnt + 1;
                    Sat1_State <= digit1;
                else
                    sat1_cnt      <= 0;
                    Sat1_State <= digit2;
                    sat_data(1) <= Sat_GPS1_in;
                    sat_data(2) <= "00101100";
                end if;
            when digit2 =>
                if sat1_cnt < counter - 1 then
                    sat1_cnt <= sat1_cnt + 1;
                    Sat1_State <= digit2;
                else
                    sat1_cnt      <= 0;
                    Sat1_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if sat_index1_done < 3 then
                        Sat1_State  <= send;
                        Sat_1_sig   <= sat_data(sat_index1_done);
                        if sat1_cnt < counter-1 then
                            sat1_cnt <= sat1_cnt + 1;
                        else
                            sat1_cnt <= 0;
                            sat_index1_done <= sat_index1_done + 1;
                        end if;
                    else
                        Sat1_State  <= idle;
                        sat_index1_done <= 0;
                    end if;
                else
                    Sat1_State <= send;
                end if;
            end case;
    end if;
end process;

Satellite2 : process(clk,rst)
begin
    if rst = '0' then
        Sat_2_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Sat2_State is
            when idle  =>
                Sat_2_sig <= "00000000";
                if Sat_GPS2_in = "00000000" then
                    Sat2_State <= idle;
                else
                    Sat2_State <= digit1;
                    sat_data(3) <= Sat_GPS2_in;
                end if;
            when digit1 =>
                if sat2_cnt < counter - 1 then
                    sat2_cnt <= sat2_cnt + 1;
                    Sat2_State <= digit1;
                else
                    sat2_cnt      <= 0;
                    Sat2_State <= digit2;
                    sat_data(4) <= Sat_GPS2_in;
                    sat_data(5) <= "00101100";
                end if;
            when digit2 =>
                if sat2_cnt < counter - 1 then
                    sat2_cnt <= sat2_cnt + 1;
                    Sat2_State <= digit2;
                else
                    sat2_cnt      <= 0;
                    Sat2_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if sat_index2_done < 6 then
                        Sat2_State  <= send;
                        Sat_2_sig   <= sat_data(sat_index2_done);
                        if sat2_cnt < counter-1 then
                            sat2_cnt <= sat2_cnt + 1;
                        else
                            sat2_cnt <= 0;
                            sat_index2_done <= sat_index2_done + 1;
                        end if;
                    else
                        Sat2_State  <= idle;
                        sat_index2_done <= 3;
                    end if;
                else
                    Sat2_State <= send;
                end if;
            end case;
    end if;
end process;

Satellite3 : process(clk,rst)
begin
    if rst = '0' then
        Sat_3_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Sat3_State is
            when idle  =>
                Sat_3_sig <= "00000000";
                if Sat_GPS3_in = "00000000" then
                    Sat3_State <= idle;
                else
                    Sat3_State <= digit1;
                    sat_data(6) <= Sat_GPS3_in;
                end if;
            when digit1 =>
                if sat3_cnt < counter - 1 then
                    sat3_cnt <= sat3_cnt + 1;
                    Sat3_State <= digit1;
                else
                    sat3_cnt      <= 0;
                    Sat3_State <= digit2;
                    sat_data(7) <= Sat_GPS3_in;
                    sat_data(8) <= "00101100";
                end if;
            when digit2 =>
                if sat3_cnt < counter - 1 then
                    sat3_cnt <= sat3_cnt + 1;
                    Sat3_State <= digit2;
                else
                    sat3_cnt      <= 0;
                    Sat3_State <= send;
                end if;
            when send =>
                if done_altitude = '1' then
                    if sat_index3_done < 9 then
                        Sat3_State  <= send;
                        Sat_3_sig   <= sat_data(sat_index3_done);
                        if sat3_cnt < counter-1 then
                            sat3_cnt <= sat3_cnt + 1;
                        else
                            sat3_cnt <= 0;
                            sat_index3_done <= sat_index3_done + 1;
                        end if;
                    else
                        Sat3_State  <= idle;
                        sat_index3_done <= 6;
                    end if;
                else
                    Sat3_State <= send;
                end if;
            end case;
    end if;
end process;

Satellite4 : process(clk,rst)
begin
    if rst = '0' then
        Sat_4_sig <= (others => '0');
    elsif rising_edge(clk) then
        case Sat4_State is
            when idle  =>
                Sat_4_sig <= "00000000";
                if Sat_GPS4_in = "00000000" then
                    Sat4_State <= idle;
                else
                    Sat4_State <= digit1;
                    sat_data(9) <= Sat_GPS4_in;
                end if;
            when digit1 =>
```

```
                    if sat4_cnt < counter - 1 then
                        sat4_cnt <= sat4_cnt + 1;
                        Sat4_State <= digit1;
                    else
                        sat4_cnt      <= 0;
                        Sat4_State <= digit2;
                        sat_data(10)<= Sat_GPS4_in;
                        sat_data(11)<= "00101100";
                    end if;
                when digit2 =>
                    if sat4_cnt < counter - 1 then
                        sat4_cnt <= sat4_cnt + 1;
                        Sat4_State <= digit2;
                    else
                        sat4_cnt      <= 0;
                        Sat4_State <= send;
                    end if;
                when send =>
                    if done_altitude = '1' then
                        if sat_index4_done < 12 then
                            Sat4_State  <= send;
                            Sat_4_sig   <= sat_data(sat_index4_done);
                            if sat4_cnt < counter-1 then
                                sat4_cnt <= sat4_cnt + 1;
                            else
                                sat4_cnt <= 0;
                                sat_index4_done <= sat_index4_done + 1;
                            end if;
                        else
                            Sat4_State  <= idle;
                            sat_index4_done <= 9;
                        end if;
                    else
                        Sat4_State <= send;
                    end if;
                end case;
        end if;
    end process;


-- SEND TO TRANSMIT
Send_Latitude : process(clk, rst)
begin
    if rising_edge(clk) then
    if keluarkan_data = '1' then
        if lat_index1 < 11 and lat_index2 < 22 and lat_index3 < 33 and lat_index4 < 44 then
            Lat_1_out_sig   <= lat_data(lat_index1);
            Lat_2_out_sig   <= lat_data(lat_index2);
            Lat_3_out_sig   <= lat_data(lat_index3);
            Lat_4_out_sig   <= lat_data(lat_index4);
            if cnt_lat < counter-1 then
```

```
                cnt_lat <= cnt_lat + 1;
            else
                cnt_lat <= 0;
                lat_index1 <= lat_index1 + 1;
                lat_index2 <= lat_index2 + 1;
                lat_index3 <= lat_index3 + 1;
                lat_index4 <= lat_index4 + 1;
            end if;
        else
            lat_index1 <= 0;
            lat_index2 <= 11;
            lat_index3 <= 22;
            lat_index4 <= 33;
        end if;
    end if;
    end if;
end process;

Send_Longitude : process(clk,rst)
begin
    if rising_edge(clk) then
    if keluarkan_data = '1' then
        if lon_index1 < 12 and lon_index2 < 24 and lon_index3 < 36 and lon_index4 < 48 then
            Lon_1_out_sig   <= lon_data(lon_index1);
            Lon_2_out_sig   <= lon_data(lon_index2);
            Lon_3_out_sig   <= lon_data(lon_index3);
            Lon_4_out_sig   <= lon_data(lon_index4);
            if cnt_lon < counter-1 then
                cnt_lon <= cnt_lon + 1;
            else
                cnt_lon <= 0;
                lon_index1 <= lon_index1 + 1;
                lon_index2 <= lon_index2 + 1;
                lon_index3 <= lon_index3 + 1;
                lon_index4 <= lon_index4 + 1;
            end if;
        else
            lon_index1 <= 0;
            lon_index2 <= 12;
            lon_index3 <= 24;
            lon_index4 <= 36;
        end if;
    end if;
    end if;
end process;

Send_Altitude : process(clk,rst)
begin
    if rising_edge(clk) then
    if keluarkan_data = '1' then
        if alt_index1 < 6 and alt_index2 < 12 and alt_index3 < 18 and alt_index4 < 24 then
```

```
            Alt_1_out_sig   <= alt_data(alt_index1);
            Alt_2_out_sig   <= alt_data(alt_index2);
            Alt_3_out_sig   <= alt_data(alt_index3);
            Alt_4_out_sig   <= alt_data(alt_index4);
            if cnt_alt < counter-1 then
                cnt_alt <= cnt_alt + 1;
            else
                cnt_alt <= 0;
                alt_index1 <= alt_index1 + 1;
                alt_index2 <= alt_index2 + 1;
                alt_index3 <= alt_index3 + 1;
                alt_index4 <= alt_index4 + 1;
            end if;
        else
            alt_index1 <= 0;
            alt_index2 <= 6;
            alt_index3 <= 12;
            alt_index4 <= 18;
        end if;
    end if;
    end if;
end process;

Send_Satellite : process(clk,rst)
begin
    if rising_edge(clk) then
    if keluarkan_data = '1' then
        if sat_index1 < 3 and sat_index2 < 6 and sat_index3 < 9 and sat_index4 < 12 then
            Sat_1_out_sig   <= sat_data(sat_index1);
            Sat_2_out_sig   <= sat_data(sat_index2);
            Sat_3_out_sig   <= sat_data(sat_index3);
            Sat_4_out_sig   <= sat_data(sat_index4);
            if cnt_sat < counter-1 then
                cnt_sat <= cnt_sat + 1;
            else
                cnt_sat <= 0;
                sat_index1 <= sat_index1 + 1;
                sat_index2 <= sat_index2 + 1;
                sat_index3 <= sat_index3 + 1;
                sat_index4 <= sat_index4 + 1;
            end if;
        else
            sat_index1 <= 0;
            sat_index2 <= 3;
            sat_index3 <= 6;
            sat_index4 <= 9;
        end if;
    end if;
    end if;
end process;
end architecture;
```

List VHDL program of ASCII_to_int.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ASCII_to_int is
    port(
        clk     : in std_logic;
        rst     : in std_logic;
        Lat_GPS1    : in std_logic_vector(7 downto 0);
        Lat_GPS2    : in std_logic_vector(7 downto 0);
        Lat_GPS3    : in std_logic_vector(7 downto 0);
        Lat_GPS4    : in std_logic_vector(7 downto 0);
        Lon_GPS1    : in std_logic_vector(7 downto 0);
        Lon_GPS2    : in std_logic_vector(7 downto 0);
        Lon_GPS3    : in std_logic_vector(7 downto 0);
        Lon_GPS4    : in std_logic_vector(7 downto 0);
        Alt_GPS1    : in std_logic_vector(7 downto 0);
        Alt_GPS2    : in std_logic_vector(7 downto 0);
        Alt_GPS3    : in std_logic_vector(7 downto 0);
        Alt_GPS4    : in std_logic_vector(7 downto 0);
        Sat_GPS1    : in std_logic_vector(7 downto 0);
        Sat_GPS2    : in std_logic_vector(7 downto 0);
        Sat_GPS3    : in std_logic_vector(7 downto 0);
        Sat_GPS4    : in std_logic_vector(7 downto 0);
        Lat_int1    : out integer;
        Lat_int2    : out integer;
        Lat_int3    : out integer;
        Lat_int4    : out integer;
        Lon_int1    : out integer;
        Lon_int2    : out integer;
        Lon_int3    : out integer;
        Lon_int4    : out integer;
        Alt_int1    : out integer;
        Alt_int2    : out integer;
        Alt_int3    : out integer;
        Alt_int4    : out integer;
        Sat_int1    : out integer;
        Sat_int2    : out integer;
        Sat_int3    : out integer;
        Sat_int4    : out integer;
        done_lon    : out std_logic
    );
end entity;

architecture rtl of ASCII_to_int is
    signal done_lon1 : std_logic := '0';
    signal done_lon2 : std_logic := '0';
    signal done_lon3 : std_logic := '0';
    signal done_lon4 : std_logic := '0';
```

```
begin
done_sig : process(clk) is
begin
    if rising_edge(clk) then
        if done_lon1 = '1' and done_lon2 = '1' and done_lon3 = '1' and done_lon4 = '1' then
            done_lon <= '1';
        else
            done_lon <= '0';
        end if;
    end if;
end process;

-- Lattitude instantiation
Lat1 : entity work.conv_lat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lat_GPS1,
        int   => Lat_int1
    );


Lat2 : entity work.conv_lat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lat_GPS2,
        int   => Lat_int2
    );


Lat3 : entity work.conv_lat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lat_GPS3,
        int   => Lat_int3
    );


Lat4 : entity work.conv_lat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lat_GPS4,
        int   => Lat_int4
    );

-- Longitude instantiation
Lon1 : entity work.conv_lon(rtl)
```

```vhdl
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lon_GPS1,
        done  => done_lon1,
        int   => Lon_int1
    );

Lon2 : entity work.conv_lon(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lon_GPS2,
        done  => done_lon2,
        int   => Lon_int2
    );

Lon3 : entity work.conv_lon(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lon_GPS3,
        done  => done_lon3,
        int   => Lon_int3
    );

Lon4 : entity work.conv_lon(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Lon_GPS4,
        done  => done_lon4,
        int   => Lon_int4
    );

-- Satelite instantiation
Sat1 : entity work.conv_sat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Sat_GPS1,
        int   => Sat_int1
    );

Sat2 : entity work.conv_sat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Sat_GPS2,
        int   => Sat_int2
    );
```

```
Sat3 : entity work.conv_sat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Sat_GPS3,
        int   => Sat_int3
    );

Sat4 : entity work.conv_sat(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII => Sat_GPS4,
        int   => Sat_int4
    );

-- Altitude instantiation
Alt1 : entity work.conv_alt(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII   => Alt_GPS1,
        int   => Alt_int1
    );


Alt2 : entity work.conv_alt(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII   => Alt_GPS2,
        int   => Alt_int2
    );

Alt3 : entity work.conv_alt(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII   => Alt_GPS3,
        int   => Alt_int3
    );

Alt4 : entity work.conv_alt(rtl)
    port map(
        clk   => clk,
        rst   => rst,
        ASCII   => Alt_GPS4,
        int   => Alt_int4
    );
```

```
end architecture;
```

List VHDL program of conv_lat.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity conv_lat is
    port(
        clk     : in std_logic;
        rst     : in std_logic;
        ASCII   : in std_logic_vector(7 downto 0);
        int     : out integer
    );
end entity;

architecture rtl of conv_lat is
    constant  counter_byte : integer := 57290;
    type ASCIIrom is array (0 to 9) of std_logic_vector(7 downto 0);
    type digit is (idle, digit1, digit2, digit3, digit4, digit5, digit6,
                   digit7, digit8, digit9, digit10, send_data);
    signal state: digit := idle;
    signal rom  : ASCIIrom;
    signal clk_count : integer := 0;
    signal int_a: integer := 0;
begin
    process(clk,rst) is
    begin
        if rst = '0' then
            int_a <= 0;
        elsif rising_edge(clk) then
            case state is
                when idle   =>
                    rom(0)  <= "00110000";
                    rom(1)  <= "00110000";
                    rom(2)  <= "00110000";
                    rom(3)  <= "00110000";
                    rom(4)  <= "00101110";
                    rom(5)  <= "00110000";
                    rom(6)  <= "00110000";
                    rom(7)  <= "00110000";
                    rom(8)  <= "00110000";
                    rom(9)  <= "00110000";
                    if ASCII /= "00000000" then
                        state   <= digit1;
                    else
                        state <= idle;
                    end if;
                when digit1 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit1;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(0)      <= ASCII;
                            state       <= digit2;
                        end if;
                    end if;
                when digit2 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit2;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(1)      <= ASCII;
                            state       <= digit3;
                        end if;
                    end if;
                when digit3 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit3;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(2)      <= ASCII;
                            state       <= digit4;
                        end if;
                    end if;
                when digit4 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit4;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(3)      <= ASCII;
                            state       <= digit5;
                        end if;
                    end if;
                when digit5 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit5;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(4)      <= ASCII;
                            state       <= digit6;
                        end if;
                    end if;
                when digit6 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit6;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(5)      <= ASCII;
                            state       <= digit7;
                        end if;
                    end if;
                when digit7 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit7;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(6)      <= ASCII;
                            state       <= digit8;
                        end if;
                    end if;
                when digit8 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit8;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(7)      <= ASCII;
                            state       <= digit9;
                        end if;
                    end if;
                when digit9 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit9;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(8)      <= ASCII;
                            state       <= digit10;
                        end if;
                    end if;
                when digit10 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit10;
                    else
                        clk_count   <= 0;
```

```
                        if ASCII /= "00000000" then
                            rom(9)      <= ASCII;
                            state       <= send_data;
                        end if;
                    end if;
                when send_data =>
                    if clk_count    < 2 then
                        clk_count   <= clk_count + 1;
                        state       <= send_data;
                    else
                        int_a       <=(to_integer(unsigned(rom(0)) - 48)*10_000_000) +
                                       (to_integer(unsigned(rom(1)) - 48)*1_000_000) +
                                       (to_integer(unsigned(rom(2)) - 48)*100_000)+
                                       (to_integer(unsigned(rom(3)) - 48)*10_000) +
                                       (to_integer(unsigned(rom(4)) - 48)*0) +
                                       (to_integer(unsigned(rom(5)) - 48)*1000) +
                                       (to_integer(unsigned(rom(6))-48)*100) +
                                       (to_integer(unsigned(rom(7))-48)*10) +
                                       (to_integer(unsigned(rom(8))-48)) +
                                       (to_integer(unsigned(rom(9))-48)*0);
                        state       <= idle;
                        clk_count   <= 0;
                    end if;
            end case;
        end if;
    end process;
    int <= int_a;

end architecture;
```

List VHDL program of conv_lon.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity conv_lon is
    port(
        clk     : in std_logic;
        rst     : in std_logic;
        ASCII   : in std_logic_vector(7 downto 0);
        done    : out std_logic;
        int     : out integer
    );
end entity;

architecture rtl of conv_lon is
    constant  counter_byte : integer := 57290;
    type ASCIIrom is array (0 to 10) of std_logic_vector(7 downto 0); -- rom nya 11 slot,
    untuk konversi longitude
    type digit is (idle, digit1, digit2, digit3, digit4, digit5,
                   digit6, digit7, digit8, digit9, digit10, digit11, send_data);
    signal state: digit := idle;
    signal rom  : ASCIIrom;
    signal clk_count : integer := 0;
    signal int_a: integer := 0;
    signal done_sig : std_logic := '0';
begin
    done <= done_sig;
    process(clk,rst) is
    begin
        if rst = '0' then
            int_a <= 0;
        elsif rising_edge(clk) then
            case state is
                when idle   =>
                    rom(0)  <= "00110000";
                    rom(1)  <= "00110000";
                    rom(2)  <= "00110000";
                    rom(3)  <= "00110000";
                    rom(4)  <= "00110000";
                    rom(5)  <= "00101110";
                    rom(6)  <= "00110000";
                    rom(7)  <= "00110000";
                    rom(8)  <= "00110000";
                    rom(9)  <= "00110000";
                    rom(10) <= "00110000";
                    if ASCII /= "00000000" then
                        state   <= digit1;
                    else
                        state <= idle;
                    end if;
                when digit1 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit1;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(0)      <= ASCII;
                            state       <= digit2;
                        end if;
                    end if;
                when digit2 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit2;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(1)      <= ASCII;
                            state       <= digit3;
                        else
                            state       <= digit3;
                        end if;
                    end if;
                when digit3 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit3;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(2)      <= ASCII;
                            state       <= digit4;
                        end if;
                    end if;
                when digit4 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit4;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(3)      <= ASCII;
                            state       <= digit5;
                        end if;
                    end if;
                when digit5 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit5;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(4)      <= ASCII;
                            state       <= digit6;
                        end if;
                    end if;
                when digit6 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit6;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(5)      <= ASCII;
                            state       <= digit7;
                        end if;
                    end if;
                when digit7 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit7;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(6)      <= ASCII;
                            state       <= digit8;
                        end if;
                    end if;
                when digit8 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit8;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(7)      <= ASCII;
                            state       <= digit9;
                        end if;
                    end if;
                when digit9 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit9;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(8)      <= ASCII;
                            state       <= digit10;
                        end if;
```

```vhdl
                    end if;
                when digit10 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit10;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(9)      <= ASCII;
                            state       <= digit11;
                        end if;
                    end if;
                when digit11 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit11;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            done_sig    <= '1';
                            rom(10)     <= ASCII;
                            state       <= send_data;
                        end if;
                    end if;
                when send_data =>
                    if clk_count    < 2 then
                        clk_count   <= clk_count + 1;
                        state       <= send_data;
                    else
                        done_sig <= '0';
                        int_a       <= (to_integer(unsigned(rom(0)) - 48)*100_000_000) +
                                       (to_integer(unsigned(rom(1)) - 48)*10_000_000) +
                                       (to_integer(unsigned(rom(2)) - 48)*1_000_000)+
                                       (to_integer(unsigned(rom(3)) - 48)*100_000) +
                                       (to_integer(unsigned(rom(4)) - 48)*10_000) +
                                       (to_integer(unsigned(rom(5)) - 48)*0) +
                                       (to_integer(unsigned(rom(6))-48)*1000) +
                                       (to_integer(unsigned(rom(7))-48)*100) +
                                       (to_integer(unsigned(rom(8))-48)*10) +
                                       (to_integer(unsigned(rom(9))-48)) +
                                       (to_integer(unsigned(rom(10))-48)*0);
                        state       <= idle;
                        clk_count   <= 0;
                    end if;
            end case;
        end if;
    end process;
    int <= int_a;

end architecture;
```

List VHDL program of conv_sat.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity conv_sat is
    port(
        clk     : in std_logic;
        rst     : in std_logic;
        ASCII   : in std_logic_vector(7 downto 0);
        int     : out integer
    );
end entity;

architecture rtl of conv_sat is
    constant  counter_byte : integer := 57290;
    type ASCIIrom is array (0 to 1) of std_logic_vector(7 downto 0); -- rom nya 2 slot,
untuk konversi satelit
    type digit is (idle, digit1, digit2, send_data);
    signal state: digit := idle;
    signal rom  : ASCIIrom;
    signal clk_count : integer := 0;
    signal int_a: integer := 0;
begin
    process(clk,rst) is
    begin
        if rst = '0' then
            int_a    <= 0;
        elsif rising_edge(clk) then
            case state is
                when idle   =>
                    rom(0)  <= "00110000";
                    rom(1)  <= "00110000";
                    if ASCII /= "00000000" then
                        state   <= digit1;
                    else
                        state <= idle;
                    end if;
                when digit1 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit1;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(0)      <= ASCII;
                            state       <= digit2;
                        end if;
                    end if;
                when digit2 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit2;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(1)      <= ASCII;
                            state       <= send_data;
                        end if;
                    end if;
                when send_data =>
                    if clk_count    < 2 then
                        clk_count   <= clk_count + 1;
                        state       <= send_data;
                    else
                        int_a       <= (to_integer(unsigned(rom(0)) - 48)*10) +
                                      (to_integer(unsigned(rom(1)) - 48));
                        state       <= idle;
                        clk_count   <= 0;
                    end if;
            end case;
        end if;
    end process;
    int <= int_a;

end architecture;
```

List VHDL program of conv_alt.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity conv_alt is
    port(
        clk     : in std_logic;
        rst : in std_logic;
        done    : out std_logic;
        ASCII   : in std_logic_vector(7 downto 0);
        int : out integer
    );
end entity;

architecture rtl of conv_alt is
    constant  counter_byte : integer := 57290;
    type ASCIIrom is array (0 to 4) of std_logic_vector(7 downto 0); -- rom nya 5 slot,
untuk konversi altitude
    type digit is (idle, digit1, digit2, digit3, digit4, digit5, send_data);
    signal state: digit := idle;
    signal rom  : ASCIIrom;
    signal clk_count : integer := 0;
    signal int_a: integer := 0;
    signal done_sig : std_logic := '0';
begin
    done <= done_sig;

    process(clk,rst) is
    begin
        if rst = '0' then
            int_a<= 0;
        elsif rising_edge(clk) then
            case state is
                when idle   =>
                    rom(0)  <= "00110000";
                    rom(1)  <= "00110000";
                    rom(2)  <= "00110000";
                    rom(3)  <= "00101110";
                    rom(4)  <= "00110000";
                    if ASCII /= "00000000" then
                        state   <= digit1;
                    else
                        state <= idle;
                    end if;
                when digit1 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit1;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(0)      <= ASCII;
                            state       <= digit2;
                        else
                            state   <= digit2;
                        end if;
                    end if;
                when digit2 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit2;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(1)  <= ASCII;
                            state       <= digit3;
                        end if;
                    end if;
                when digit3 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit3;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(2)      <= ASCII;
                            state       <= digit4;
                        end if;
                    end if;
                when digit4 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit4;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            rom(3)      <= ASCII;
                            state       <= digit5;
                        end if;
                    end if;
                when digit5 =>
                    if clk_count < counter_byte - 1 then
                        clk_count   <= clk_count + 1;
                        state       <= digit5;
                    else
                        clk_count   <= 0;
                        if ASCII /= "00000000" then
                            done_sig    <= '1';
                            rom(4)      <= ASCII;
                            state       <= send_data;
```

```
                        end if;
                    end if;
                when send_data =>
                    if clk_count    < 2 then
                        clk_count   <= clk_count + 1;
                        state       <= send_data;
                    else
                        done_sig    <= '0';
                        int_a       <= (to_integer(unsigned(rom(0)) - 48)*1000) +
                                       (to_integer(unsigned(rom(1)) - 48)*100) +
                                       (to_integer(unsigned(rom(2)) - 48)*10)+
                                       (to_integer(unsigned(rom(3)) - 48)*0) +
                                       (to_integer(unsigned(rom(4)) - 48));
                        state       <= idle;
                        clk_count   <= 0;
                    end if;
            end case;
        end if;
    end process;
    int <= int_a;

end architecture;
```

List VHDL program of validator.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Validator IS
    PORT(
    Clk          : in std_logic;
    rst          : in std_logic;
    start_val    : in std_logic;
    satelit_gps1: in integer;
    satelit_gps2: in integer;
    satelit_gps3: in integer;
    satelit_gps4: in integer;
    done_val         : out std_logic;
    Data_valid   : out std_logic_vector(3 downto 0)
    );
END entity;

ARCHITECTURE rtl OF Validator IS
    signal gps_valid : std_logic_vector(3 downto 0) := (others => '0');
    signal start_sig : std_logic := '0';
    signal done_sig  : std_logic_vector(3 downto 0) := (others => '0');
    signal done_gps1 : std_logic := '0';
    signal done_gps2 : std_logic := '0';
    signal done_gps3 : std_logic := '0';
    signal done_gps4 : std_logic := '0';

BEGIN

    start_sig  <= start_val;
    Data_valid <= gps_valid;
    done_sig(0)<= done_gps1;
    done_sig(1)<= done_gps2;
    done_sig(2)<= done_gps3;
    done_sig(3)<= done_gps4;

    done_flag : process(clk,rst) is
    begin
        if rst = '0' then
            done_val <= '0';
        elsif rising_edge(clk) then
            if done_sig = "1111" then
                done_val <= '1';
            elsif done_sig = "0000" then
                done_val <= '0';
            end if;
        end if;
    end process;
```

```
GPS1 : PROCESS(Clk,rst)
BEGIN
    IF rst = '0' then
        gps_valid(0) <= '0';
        done_gps1    <= '0';
    ELSIF RISING_EDGE(Clk) THEN
        if start_sig = '1' then
            IF satelit_gps1 >= 3 THEN
                gps_valid(0) <= '1';
                done_gps1    <= '1';
            ELSE
                gps_valid(0) <= '0';
                done_gps1    <= '1';
            END IF;
        elsif start_sig = '0' then
            done_gps1    <= '0';
        end if;
    END IF;
END PROCESS;



GPS2 : PROCESS(Clk,rst)
BEGIN
    IF rst = '0' then
        gps_valid(1) <= '0';
        done_gps2    <= '0';
    ELSIF RISING_EDGE(Clk) THEN
        if start_sig = '1' then
            IF satelit_gps2 >= 3 THEN
                gps_valid(1) <= '1';
                done_gps2    <= '1';
            ELSE
                gps_valid(1) <= '0';
                done_gps2    <= '1';
            END IF;
        elsif start_sig = '0' then
            done_gps2    <= '0';
        end if;
    END IF;
END PROCESS;

GPS3 : PROCESS(Clk,rst)
BEGIN
    IF rst = '0' then
        gps_valid(2) <= '0';
        done_gps3    <= '0';
    ELSIF RISING_EDGE(Clk) THEN
        if start_sig = '1' then
            IF satelit_gps3 >= 3 THEN
                gps_valid(2) <= '1';
```

```vhdl
                    done_gps3    <= '1';
                ELSE
                    gps_valid(2) <= '0';
                    done_gps3    <= '1';
                END IF;
            elsif start_sig = '0' then
                done_gps3    <= '0';
            end if;
        END IF;
    END PROCESS;

    GPS4 : PROCESS(Clk,rst)
    BEGIN
        IF rst = '0' then
            gps_valid(3) <= '0';
            done_gps4    <= '0';
        ELSIF RISING_EDGE(Clk) THEN
            if start_sig = '1' then
                IF satelit_gps4 >= 3 THEN
                    gps_valid(3) <= '1';
                    done_gps4    <= '1';
                ELSE
                    gps_valid(3) <= '0';
                    done_gps4    <= '1';
                END IF;
            elsif start_sig = '0' then
                done_gps4    <= '0';
            end if;
        END IF;
    END PROCESS;



END architecture;
```

List VHDL program of Buffer_module.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY Buffer_Module IS
    PORT(
        Clk         : in std_logic;
        Reset       : in std_logic;
        Gps_Valid   : in std_logic_vector(3 downto 0);
        Start_buffer: in std_logic;
        Lat_GPS1_in : in integer;
        Lat_GPS2_in : in integer;
        Lat_GPS3_in : in integer;
        Lat_GPS4_in : in integer;
        Lon_GPS1_in : in integer;
        Lon_GPS2_in : in integer;
        Lon_GPS3_in : in integer;
        Lon_GPS4_in : in integer;
        Alt_GPS1_in : in integer;
        Alt_GPS2_in : in integer;
        Alt_GPS3_in : in integer;
        Alt_GPS4_in : in integer;
        Sat_GPS1_in : in integer;
        Sat_GPS2_in : in integer;
        Sat_GPS3_in : in integer;
        Sat_GPS4_in : in integer;
        done_buffer : out std_logic;
        Lat_Val_1   : out integer;
        Lat_Val_2   : out integer;
        Lat_Val_3   : out integer;
        Lat_Val_4   : out integer;
        Lon_Val_1   : out integer;
        Lon_Val_2   : out integer;
        Lon_Val_3   : out integer;
        Lon_Val_4   : out integer;
        Sat_Val_1   : out integer;
        Sat_Val_2   : out integer;
        Sat_Val_3   : out integer;
        Sat_Val_4   : out integer;
        Alt_Val_1   : out integer;
        Alt_Val_2   : out integer;
        Alt_Val_3   : out integer;
        Alt_Val_4   : out integer
    );
END Entity;

ARCHITECTURE rtl OF Buffer_Module IS
    constant buffer_done_cnt : integer := 41664;
    type rom is array (0 to 15) of integer;
    type state is (idle, lat_store, lon_store, sat_store, alt_store, done_state);
    signal rom_buffer : rom :=(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
    signal store         : state := idle;
    signal done_store    : std_logic := '0';
    signal cnt           : integer := 0;

    signal Lat_Val_1_sig: integer := 0;
    signal Lat_Val_2_sig: integer := 0;
    signal Lat_Val_3_sig: integer := 0;
    signal Lat_Val_4_sig: integer := 0;
    signal Lon_Val_1_sig: integer := 0;
    signal Lon_Val_2_sig: integer := 0;
    signal Lon_Val_3_sig: integer := 0;
    signal Lon_Val_4_sig: integer := 0;
    signal Sat_Val_1_sig: integer := 0;
    signal Sat_Val_2_sig: integer := 0;
    signal Sat_Val_3_sig: integer := 0;
    signal Sat_Val_4_sig: integer := 0;
    signal Alt_Val_1_sig: integer := 0;
    signal Alt_Val_2_sig: integer := 0;
    signal Alt_Val_3_sig: integer := 0;
    signal Alt_Val_4_sig: integer := 0;
BEGIN
    done_buffer <= done_store;

    Lat_Val_1   <= Lat_Val_1_sig;
    Lat_Val_2   <= Lat_Val_2_sig;
    Lat_Val_3   <= Lat_Val_3_sig;
    Lat_Val_4   <= Lat_Val_4_sig;
    Lon_Val_1   <= Lon_Val_1_sig;
    Lon_Val_2   <= Lon_Val_2_sig;
    Lon_Val_3   <= Lon_Val_3_sig;
    Lon_Val_4   <= Lon_Val_4_sig;
    Sat_Val_1   <= Sat_Val_1_sig;
    Sat_Val_2   <= Sat_Val_2_sig;
    Sat_Val_3   <= Sat_Val_3_sig;
    Sat_Val_4   <= Sat_Val_4_sig;
    Alt_Val_1   <= Alt_Val_1_sig;
    Alt_Val_2   <= Alt_Val_2_sig;
    Alt_Val_3   <= Alt_Val_3_sig;
    Alt_Val_4   <= Alt_Val_4_sig;

    buffer_process: process(clk,Reset)
    begin
        if Reset = '0' then
            rom_buffer(0)   <= 0;
            rom_buffer(1)   <= 0;
            rom_buffer(2)   <= 0;
            rom_buffer(3)   <= 0;
            rom_buffer(4)   <= 0;
            rom_buffer(5)   <= 0;
        rom_buffer(6)   <= 0;
        rom_buffer(7)   <= 0;
        rom_buffer(8)   <= 0;
        rom_buffer(9)   <= 0;
        rom_buffer(10)  <= 0;
        rom_buffer(11)  <= 0;
        rom_buffer(12)  <= 0;
        rom_buffer(13)  <= 0;
        rom_buffer(14)  <= 0;
        rom_buffer(15)  <= 0;
    elsif rising_edge(clk) then
        case store is
            when idle =>
                done_store    <= '0';
                if Start_buffer = '1' then
                    store   <= lat_store;
                else
                    store   <= idle;
                end if;
            when lat_store =>
                if Start_buffer = '1' then
                    rom_buffer(0) <= Lat_GPS1_in;
                    rom_buffer(1) <= Lat_GPS2_in;
                    rom_buffer(2) <= Lat_GPS3_in;
                    rom_buffer(3) <= Lat_GPS4_in;
                    store   <= lon_store;
                else
                    store   <= lat_store;
                end if;
            when lon_store =>
                if Start_buffer = '1' then
                    rom_buffer(4) <= Lon_GPS1_in;
                    rom_buffer(5) <= Lon_GPS2_in;
                    rom_buffer(6) <= Lon_GPS3_in;
                    rom_buffer(7) <= Lon_GPS4_in;
                    store   <= sat_store;
                else
                    store   <= lon_store;
                end if;
            when sat_store =>
                if Start_buffer = '1' then
                    rom_buffer(12)<= Sat_GPS1_in;
                    rom_buffer(13)<= Sat_GPS2_in;
                    rom_buffer(14)<= Sat_GPS3_in;
                    rom_buffer(15)<= Sat_GPS4_in;
                    store   <= alt_store;
                else
                    store   <= sat_store;
                end if;
            when alt_store =>
                if Start_buffer = '1' then
```

```vhdl
                    rom_buffer(8) <= Alt_GPS1_in;
                    rom_buffer(9) <= Alt_GPS2_in;
                    rom_buffer(10)<= Alt_GPS3_in;
                    rom_buffer(11)<= Alt_GPS4_in;
                    store   <= done_state;
                else
                    store   <= alt_store;
                end if;
            when done_state =>
                if cnt < buffer_done_cnt - 1 then
                    done_store  <= '1';
                    cnt <= cnt + 1;
                    store   <= done_state;
                else
                    store   <= idle;
                    done_store  <= '0';
                    cnt <= 0;
                end if;
            end case;
    end if;
end process;

send_data : process(clk)
begin
    if rising_edge(clk) then
        if done_store = '1' then
            Sat_Val_1_sig <= rom_buffer(12);
            Sat_Val_2_sig <= rom_buffer(13);
            Sat_Val_3_sig <= rom_buffer(14);
            Sat_Val_4_sig <= rom_buffer(15);
            IF Gps_Valid = "0000" THEN
                Lat_Val_1_sig <= 0;
                Lat_Val_2_sig <= 0;
                Lat_Val_3_sig <= 0;
                Lat_Val_4_sig <= 0;
                Lon_Val_1_sig <= 0;
                Lon_Val_2_sig <= 0;
                Lon_Val_3_sig <= 0;
                Lon_Val_4_sig <= 0;
                Alt_Val_1_sig <= 0;
                Alt_Val_2_sig <= 0;
                Alt_Val_3_sig <= 0;
                Alt_Val_4_sig <= 0;
            ELSIF Gps_Valid = "0001" THEN
                Lat_Val_1_sig <= rom_buffer(0);
                Lat_Val_2_sig <= 0;
                Lat_Val_3_sig <= 0;
                Lat_Val_4_sig <= 0;
                Lon_Val_1_sig <= rom_buffer(4);
                Lon_Val_2_sig <= 0;
                Lon_Val_3_sig <= 0;
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= rom_buffer(8);
    Alt_Val_2_sig <= 0;
    Alt_Val_3_sig <= 0;
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0010" THEN
    Lat_Val_1_sig <= 0;
    Lat_Val_2_sig <= rom_buffer(1);
    Lat_Val_3_sig <= 0;
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= 0;
    Lon_Val_2_sig <= rom_buffer(5);
    Lon_Val_3_sig <= 0;
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= 0;
    Alt_Val_2_sig <= rom_buffer(9);
    Alt_Val_3_sig <= 0;
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0011" THEN
    Lat_Val_1_sig <= rom_buffer(0);
    Lat_Val_2_sig <= rom_buffer(1);
    Lat_Val_3_sig <= 0;
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= rom_buffer(4);
    Lon_Val_2_sig <= rom_buffer(5);
    Lon_Val_3_sig <= 0;
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= rom_buffer(8);
    Alt_Val_2_sig <= rom_buffer(9);
    Alt_Val_3_sig <= 0;
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0100" THEN
    Lat_Val_1_sig <= 0;
    Lat_Val_2_sig <= 0;
    Lat_Val_3_sig <= rom_buffer(2);
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= 0;
    Lon_Val_2_sig <= 0;
    Lon_Val_3_sig <= rom_buffer(6);
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= 0;
    Alt_Val_2_sig <= 0;
    Alt_Val_3_sig <= rom_buffer(10);
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0101" THEN
    Lat_Val_1_sig <= rom_buffer(0);
    Lat_Val_2_sig <= 0;
    Lat_Val_3_sig <= rom_buffer(2);
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= rom_buffer(4);
    Lon_Val_2_sig <= 0;
    Lon_Val_3_sig <= rom_buffer(6);
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= rom_buffer(8);
    Alt_Val_2_sig <= 0;
    Alt_Val_3_sig <= rom_buffer(10);
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0110" THEN
    Lat_Val_1_sig <= 0;
    Lat_Val_2_sig <= rom_buffer(1);
    Lat_Val_3_sig <= rom_buffer(2);
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= 0;
    Lon_Val_2_sig <= rom_buffer(5);
    Lon_Val_3_sig <= rom_buffer(6);
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= 0;
    Alt_Val_2_sig <= rom_buffer(9);
    Alt_Val_3_sig <= rom_buffer(10);
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "0111" THEN
    Lat_Val_1_sig <= rom_buffer(0);
    Lat_Val_2_sig <= rom_buffer(1);
    Lat_Val_3_sig <= rom_buffer(2);
    Lat_Val_4_sig <= 0;
    Lon_Val_1_sig <= rom_buffer(4);
    Lon_Val_2_sig <= rom_buffer(5);
    Lon_Val_3_sig <= rom_buffer(6);
    Lon_Val_4_sig <= 0;
    Alt_Val_1_sig <= rom_buffer(8);
    Alt_Val_2_sig <= rom_buffer(9);
    Alt_Val_3_sig <= rom_buffer(10);
    Alt_Val_4_sig <= 0;
ELSIF Gps_Valid = "1000" THEN
    Lat_Val_1_sig <= 0;
    Lat_Val_2_sig <= 0;
    Lat_Val_3_sig <= 0;
    Lat_Val_4_sig <= rom_buffer(3);
    Lon_Val_1_sig <= 0;
    Lon_Val_2_sig <= 0;
    Lon_Val_3_sig <= 0;
    Lon_Val_4_sig <= rom_buffer(7);
    Alt_Val_1_sig <= 0;
    Alt_Val_2_sig <= 0;
    Alt_Val_3_sig <= 0;
    Alt_Val_4_sig <= rom_buffer(11);
ELSIF GPS_Valid = "1001" THEN
    Lat_Val_1_sig <= rom_buffer(0);
    Lat_Val_2_sig <= 0;
    Lat_Val_3_sig <= 0;
    Lat_Val_4_sig <= rom_buffer(3);
    Lon_Val_1_sig <= rom_buffer(4);
```

```
        Lon_Val_2_sig <= 0;
        Lon_Val_3_sig <= 0;
        Lon_Val_4_sig <= rom_buffer(7);
        Alt_Val_1_sig <= rom_buffer(8);
        Alt_Val_2_sig <= 0;
        Alt_Val_3_sig <= 0;
        Alt_Val_4_sig <= rom_buffer(11);
    ELSIF Gps_Valid = "1010" THEN
        Lat_Val_1_sig <= 0;
        Lat_Val_2_sig <= rom_buffer(1);
        Lat_Val_3_sig <= 0;
        Lat_Val_4_sig <= rom_buffer(3);
        Lon_Val_1_sig <= 0;
        Lon_Val_2_sig <= rom_buffer(5);
        Lon_Val_3_sig <= 0;
        Lon_Val_4_sig <= rom_buffer(7);
        Alt_Val_1_sig <= 0;
        Alt_Val_2_sig <= rom_buffer(9);
        Alt_Val_3_sig <= 0;
        Alt_Val_4_sig <= rom_buffer(11);
    ELSIF Gps_Valid = "1011" THEN
        Lat_Val_1_sig <= rom_buffer(0);
        Lat_Val_2_sig <= rom_buffer(1);
        Lat_Val_3_sig <= 0;
        Lat_Val_4_sig <= rom_buffer(3);
        Lon_Val_1_sig <= rom_buffer(4);
        Lon_Val_2_sig <= rom_buffer(5);
        Lon_Val_3_sig <= 0;
        Lon_Val_4_sig <= rom_buffer(7);
        Alt_Val_1_sig <= rom_buffer(8);
        Alt_Val_2_sig <= rom_buffer(9);
        Alt_Val_3_sig <= 0;
        Alt_Val_4_sig <= rom_buffer(11);
    ELSIF Gps_Valid = "1100" THEN
        Lat_Val_1_sig <= 0;
        Lat_Val_2_sig <= 0;
        Lat_Val_3_sig <= rom_buffer(2);
        Lat_Val_4_sig <= rom_buffer(3);
        Lon_Val_1_sig <= 0;
        Lon_Val_2_sig <= 0;
        Lon_Val_3_sig <= rom_buffer(6);
        Lon_Val_4_sig <= rom_buffer(7);
        Alt_Val_1_sig <= 0;
        Alt_Val_2_sig <= 0;
        Alt_Val_3_sig <= rom_buffer(10);
        Alt_Val_4_sig <= rom_buffer(11);
    ELSIF Gps_Valid = "1101" THEN
        Lat_Val_1_sig <= rom_buffer(0);
        Lat_Val_2_sig <= 0;
        Lat_Val_3_sig <= rom_buffer(2);
        Lat_Val_4_sig <= rom_buffer(3);
```

```
                    Lon_Val_1_sig <= rom_buffer(4);
                    Lon_Val_2_sig <= 0;
                    Lon_Val_3_sig <= rom_buffer(6);
                    Lon_Val_4_sig <= rom_buffer(7);
                    Alt_Val_1_sig <= rom_buffer(8);
                    Alt_Val_2_sig <= 0;
                    Alt_Val_3_sig <= rom_buffer(10);
                    Alt_Val_4_sig <= rom_buffer(11);
                ELSIF Gps_Valid = "1110" THEN
                    Lat_Val_1_sig <= 0;
                    Lat_Val_2_sig <= rom_buffer(1);
                    Lat_Val_3_sig <= rom_buffer(2);
                    Lat_Val_4_sig <= rom_buffer(3);
                    Lon_Val_1_sig <= 0;
                    Lon_Val_2_sig <= rom_buffer(5);
                    Lon_Val_3_sig <= rom_buffer(6);
                    Lon_Val_4_sig <= rom_buffer(7);
                    Alt_Val_1_sig <= 0;
                    Alt_Val_2_sig <= rom_buffer(9);
                    Alt_Val_3_sig <= rom_buffer(10);
                    Alt_Val_4_sig <= rom_buffer(11);
                ELSIF Gps_Valid = "1111" THEN
                    Lat_Val_1_sig <= rom_buffer(0);
                    Lat_Val_2_sig <= rom_buffer(1);
                    Lat_Val_3_sig <= rom_buffer(2);
                    Lat_Val_4_sig <= rom_buffer(3);
                    Lon_Val_1_sig <= rom_buffer(4);
                    Lon_Val_2_sig <= rom_buffer(5);
                    Lon_Val_3_sig <= rom_buffer(6);
                    Lon_Val_4_sig <= rom_buffer(7);
                    Alt_Val_1_sig <= rom_buffer(8);
                    Alt_Val_2_sig <= rom_buffer(9);
                    Alt_Val_3_sig <= rom_buffer(10);
                    Alt_Val_4_sig <= rom_buffer(11);
                END IF;
            end if;
        end if;
    end process;
END Architecture;
```

List VHDL program of average.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Average is
    port(
        clk : in std_logic;
        rst : in std_logic;
        Lat1: in integer;
        Lat2: in integer;
        Lat3: in integer;
        Lat4: in integer;
        Lon1: in integer;
        Lon2: in integer;
        Lon3: in integer;
        Lon4: in integer;
        Alt1: in integer;
        Alt2: in integer;
        Alt3: in integer;
        Alt4: in integer;
        data_valid : in std_logic_vector(3 downto 0);
        noDivisor : out std_logic;
        done: out std_logic;
        Lat_ave : out integer;
        Lon_ave : out integer;
        Alt_ave : out integer
    );
end entity;

architecture rtl of Average is
    type state is (idle, total, calculate);
    constant pembagi3   : unsigned(31 downto 0) := to_unsigned(1431655765,32);
                              -- (2^32 = 4294967296) -> 4294967296/3 = 1431655765
    signal noDivisor_sig : std_logic := '1';
    signal done_sig  : std_logic := '0';
    signal Lat_total : integer := 0;
    signal Lon_total : integer := 0;
    signal Alt_total : integer := 0;
    signal Lat_sig   : unsigned(31 downto 0) := (others => '0');
    signal Lon_sig    : unsigned(31 downto 0) := (others => '0');
    signal Alt_sig    : unsigned(31 downto 0) := (others => '0');
    signal Lat_out    : unsigned(31 downto 0) := (others => '0');
    signal Lon_out    : unsigned(31 downto 0) := (others => '0');
    signal Alt_out    : unsigned(31 downto 0) := (others => '0');
    signal Lat_temp  : unsigned(63 downto 0) := (others => '0');
    signal Lon_temp  : unsigned(63 downto 0) := (others => '0');
    signal Alt_temp  : unsigned(63 downto 0) := (others => '0');
begin
```

```
noDivisor <= noDivisor_sig;
done      <= done_sig;
Lat_ave <= to_integer(Lat_out);
Lon_ave <= to_integer(Lon_out);
Alt_ave <= to_integer(Alt_out);
process(clk,rst)
begin
    if rst = '0' then
        Lat_out <= (others => '0');
        Lon_out <= (others => '0');
        Alt_out <= (others => '0');
        noDivisor_sig <= '1';
    elsif rising_edge(clk) then
            done_sig <= '0';
            noDivisor_sig <= '0';
            Lat_total <= Lat1 + Lat2 + Lat3 + Lat4;
            Lon_total <= Lon1 + Lon2 + Lon3 + Lon4;
            Alt_total <= Alt1 + Alt2 + Alt3 + Alt4;
            Lat_sig <= to_unsigned(Lat_total,32);
            Lon_sig <= to_unsigned(Lon_total,32);
            Alt_sig <= to_unsigned(Alt_total,32);
            if data_valid = "0000" then
                Lat_out <= (others => '0');
                Lon_out <= (others => '0');
                Alt_out <= (others => '0');
                noDivisor_sig <= '1';
                done_sig <= '1';
            elsif data_valid = "0001" OR data_valid = "0010" OR data_valid = "0100" OR
            data_valid = "1000" then
                Lat_out <= Lat_sig;
                Lon_out <= Lon_sig;
                Alt_out <= Alt_sig;
                done_sig <= '1';
            elsif data_valid = "0011" OR data_valid = "0101" OR data_valid = "0110" OR
            data_valid = "1001" OR data_valid = "1010" OR data_valid = "1100" then
                Lat_out <= shift_right(Lat_sig,1);
                Lon_out <= shift_right(Lon_sig,1);
                Alt_out <= shift_right(Alt_sig,1);
                done_sig <= '1';
            elsif data_valid = "0111" OR data_valid = "1011" OR data_valid = "1101" OR
            data_valid = "1110" then
                Lat_temp <= Lat_sig * pembagi3;
                Lon_temp <= Lon_sig * pembagi3;
                Alt_temp <= Alt_sig * pembagi3;
                Lat_out  <= Lat_temp(63 downto 32);
                Lon_out  <= Lon_temp(63 downto 32);
                Alt_out  <= Alt_temp(63 downto 32);
                done_sig <= '1';
            elsif data_valid = "1111" then
                Lat_out <= shift_right(Lat_sig,2);
                Lon_out <= shift_right(Lon_sig,2);
```

```
                    Alt_out <= shift_right(Alt_sig,2);
                    done_sig <= '1';
                end if;
        end if;
    end process;
end architecture;
```

List VHDL program of int_to_ASCII.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity int_to_ASCII is
    generic(
        cnt_byte : integer := 57292; -- untuk counter transmisi 8 bit data baudrate 9600
        input_width     : integer := 32; -- lebar data integer;
        lattitude_digits: integer := 7; -- jumlah digit data dalam integer
        longitude_digits: integer := 9; -- jumlah digit data dalam integer
        altitude_digits : integer := 4  -- jumlah digit data dalam integer
    );
    port(
        clk     : in std_logic;
        rst     : in std_logic;
        start   : in std_logic;
        lat_ave : in integer;
        lon_ave : in integer;
        alt_ave : in integer;

        done    : out std_logic;
        out_lat : out std_logic_vector(7 downto 0);
        out_lon : out std_logic_vector(7 downto 0);
        out_alt : out std_logic_vector(7 downto 0)
    );
end entity;

architecture rtl of int_to_ASCII is
    type BCD_State is (idle, shift, check_shift_index, add, check_digit_index, bcd_done,
send);
    signal lat_state    : BCD_State := idle;
    signal lon_state    : BCD_State := idle;
    signal alt_state    : BCD_State := idle;

    type lat_array is array (0 to 8) of std_logic_vector(7 downto 0);
    type lon_array is array (0 to 10) of std_logic_vector(7 downto 0);
    type alt_array is array (0 to 5) of std_logic_vector(7 downto 0);
    signal lat_data : lat_array;
    signal lon_data : lon_array;
    signal alt_data : alt_array;
    -- Inputs
    signal mulai_konversi : std_logic := '0';
    -- Outputs
    signal out_lat_sig  : std_logic_vector(7 downto 0) := (others => '0');
    signal out_lon_sig  : std_logic_vector(7 downto 0) := (others => '0');
    signal out_alt_sig  : std_logic_vector(7 downto 0) := (others => '0');
    signal bcd_lat_sig  : std_logic_vector(lattitude_digits*4-1 downto 0) := (others =>
    '0');
    signal bcd_lon_sig  : std_logic_vector(longitude_digits*4-1 downto 0) := (others =>
    '0');
    signal bcd_alt_sig  : std_logic_vector(altitude_digits*4-1 downto 0) := (others =>
    '0');
    -- Temporary Data
    signal lat_temp     : std_logic_vector(input_width-1 downto 0) := (others => '0');
    signal lon_temp     : std_logic_vector(input_width-1 downto 0) := (others => '0');
    signal alt_temp     : std_logic_vector(input_width-1 downto 0) := (others => '0');
    -- Counters
    signal index_lat    : natural range 0 to lattitude_digits-1 := 0;
    signal index_lon    : natural range 0 to longitude_digits-1 := 0;
    signal index_alt    : natural range 0 to altitude_digits-1 := 0;
    signal loop_cnt_lat : natural range 0 to input_width-1 := 0;
    signal loop_cnt_lon : natural range 0 to input_width-1 := 0;
    signal loop_cnt_alt : natural range 0 to input_width-1 := 0;
    signal cnt_lat      : integer := 0;
    signal cnt_lon      : integer := 0;
    signal cnt_alt      : integer := 0;
    signal cnt_lat_byte : integer range 0 to cnt_byte-1 := 0;
    signal cnt_lon_byte : integer range 0 to cnt_byte-1 := 0;
    signal cnt_alt_byte : integer range 0 to cnt_byte-1 := 0;
    -- Flags
    signal lon_done     : std_logic := '0';
begin
    mulai_konversi  <= start;
    done        <= lon_done;
    lat_data(3) <= "00101110"; -- untuk karakter titik "."
    lat_data(8) <= "00101100"; -- untuk karakter koma ","
    lon_data(5) <= "00101110";
    lon_data(10)<= "00101100";
    alt_data(3) <= "00101110";
    alt_data(5) <= "00001010"; -- untuk karakter new line

    lattitude_process : process(clk,rst)
        variable bcd_digit_lat : unsigned(3 downto 0);
        variable cnt2clk    : integer := 0;
    begin
        if rst = '0' then
            lat_data(0) <= (others => '0');
            lat_data(1) <= (others => '0');
            lat_data(2) <= (others => '0');
            lat_data(4) <= (others => '0');
            lat_data(5) <= (others => '0');
            lat_data(6) <= (others => '0');
            lat_data(7) <= (others => '0');
            out_lat_sig  <= (others => '0');
        elsif rising_edge(clk) then
            case lat_state is
                when idle =>
                    cnt_lat     <= 0;
                    lat_data(0) <= (others => '0');
    lat_data(1) <= (others => '0');
    lat_data(2) <= (others => '0');
    lat_data(4) <= (others => '0');
    lat_data(5) <= (others => '0');
    lat_data(6) <= (others => '0');
    lat_data(7) <= (others => '0');
    out_lat_sig  <= (others => '0');
    if mulai_konversi = '1' then
        if cnt2clk = 4 then
            cnt2clk := 0;
            bcd_lat_sig <= (others => '0');
            lat_temp    <= std_logic_vector(to_unsigned(lat_ave,32));
            lat_state   <= shift;
        else
            cnt2clk := cnt2clk + 1;
        end if;
    else
        lat_state   <= idle;
    end if;
when shift =>
    bcd_lat_sig <= bcd_lat_sig(bcd_lat_sig'left-1 downto 0) &
    lat_temp(lat_temp'left);
    lat_temp    <= lat_temp(lat_temp'left-1 downto 0) & '0';
    lat_state   <= check_shift_index;
when check_shift_index =>
    if loop_cnt_lat = input_width-1 then
        loop_cnt_lat <= 0;
        lat_state    <= bcd_done;
    else
        loop_cnt_lat <= loop_cnt_lat + 1;
        lat_state    <= add;
    end if;
when add =>
    if index_lat = 0 then
        bcd_digit_lat := unsigned(bcd_lat_sig(3 downto 0));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(3 downto 0) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 1 then
        bcd_digit_lat := unsigned(bcd_lat_sig(7 downto 4));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;

        bcd_lat_sig(7 downto 4) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 2 then
        bcd_digit_lat := unsigned(bcd_lat_sig(11 downto 8));
        if bcd_digit_lat > 4 then
```

```
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(11 downto 8) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 3 then
        bcd_digit_lat := unsigned(bcd_lat_sig(15 downto 12));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(15 downto 12) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 4 then
        bcd_digit_lat := unsigned(bcd_lat_sig(19 downto 16));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(19 downto 16) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 5 then
        bcd_digit_lat := unsigned(bcd_lat_sig(23 downto 20));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(23 downto 20) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    elsif index_lat = 6 then
        bcd_digit_lat := unsigned(bcd_lat_sig(27 downto 24));
        if bcd_digit_lat > 4 then
            bcd_digit_lat := bcd_digit_lat + 3;
        end if;
        bcd_lat_sig(27 downto 24) <= std_logic_vector(bcd_digit_lat);
        lat_state <= check_digit_index;
    end if;

when check_digit_index =>
    if index_lat = lattitude_digits-1 then
        index_lat <= 0;
        lat_state <= shift;
    else
        index_lat <= index_lat + 1;
        lat_state <= add;
    end if;
when bcd_done =>
    lat_data(0) <= "0011" & bcd_lat_sig(27 downto 24);
    lat_data(1) <= "0011" & bcd_lat_sig(23 downto 20);
    lat_data(2) <= "0011" & bcd_lat_sig(19 downto 16);
    lat_data(4) <= "0011" & bcd_lat_sig(15 downto 12);
    lat_data(5) <= "0011" & bcd_lat_sig(11 downto 8);
    lat_data(6) <= "0011" & bcd_lat_sig(7 downto 4);
    lat_data(7) <= "0011" & bcd_lat_sig(3 downto 0);
    lat_state   <= send;
```

```
            when send =>
                if lon_done = '1' then
                    if cnt_lat < 9 then
                        lat_state   <= send;
                        out_lat_sig <= lat_data(cnt_lat);
                        if cnt_lat_byte < cnt_byte-1 then
                            cnt_lat_byte <= cnt_lat_byte + 1;
                        else
                            cnt_lat_byte <= 0;
                            cnt_lat     <= cnt_lat + 1;
                        end if;
                    else
                        cnt_lat     <= 0;
                        lat_state   <= idle;
                    end if;
                else
                    out_lat_sig <= "00000000";
                end if;
        end case;
    end if;
end process;

out_lat  <= out_lat_sig;

longitude_process : process(clk,rst)
    variable bcd_digit_lon : unsigned(3 downto 0);
    variable cnt2clk : integer := 0;
begin
    if rst = '0' then
        lon_data(0) <= (others => '0');
        lon_data(1) <= (others => '0');
        lon_data(2) <= (others => '0');
        lon_data(3) <= (others => '0');
        lon_data(4) <= (others => '0');
        lon_data(6) <= (others => '0');
        lon_data(7) <= (others => '0');
        lon_data(8) <= (others => '0');
        lon_data(9) <= (others => '0');
        out_lon_sig  <= (others => '0');
    elsif rising_edge(clk) then
        case lon_state is
            when idle =>
                cnt_lon     <= 0;
                lon_data(0) <= (others => '0');
                lon_data(1) <= (others => '0');
                lon_data(2) <= (others => '0');
                lon_data(3) <= (others => '0');
                lon_data(4) <= (others => '0');
                lon_data(6) <= (others => '0');
                lon_data(7) <= (others => '0');
                lon_data(8) <= (others => '0');
    lon_data(9) <= (others => '0');
    out_lon_sig  <= (others => '0');
    if mulai_konversi = '1' then
        if cnt2clk = 4 then
            cnt2clk := 0;
            bcd_lon_sig <= (others => '0');
            lon_temp    <= std_logic_vector(to_unsigned(lon_ave,32));
            lon_state   <= shift;
        else
            cnt2clk := cnt2clk + 1;
        end if;
    else
        lon_state   <= idle;
    end if;
when shift =>
    bcd_lon_sig <= bcd_lon_sig(bcd_lon_sig'left-1 downto 0) &
    lon_temp(lon_temp'left);
    lon_temp    <= lon_temp(lon_temp'left-1 downto 0) & '0';
    lon_state   <= check_shift_index;
when check_shift_index =>
    if loop_cnt_lon = input_width-1 then
        loop_cnt_lon <= 0;
        lon_state    <= bcd_done;
    else
        loop_cnt_lon <= loop_cnt_lon + 1;
        lon_state    <= add;
    end if;
when add =>
    if index_lon = 0 then
        bcd_digit_lon := unsigned(bcd_lon_sig(3 downto 0));
        if bcd_digit_lon > 4 then
            bcd_digit_lon := bcd_digit_lon + 3;
        end if;

        bcd_lon_sig(3 downto 0) <= std_logic_vector(bcd_digit_lon);
        lon_state <= check_digit_index;
    elsif index_lon = 1 then
        bcd_digit_lon := unsigned(bcd_lon_sig(7 downto 4));
        if bcd_digit_lon > 4 then
            bcd_digit_lon := bcd_digit_lon + 3;
        end if;

        bcd_lon_sig(7 downto 4) <= std_logic_vector(bcd_digit_lon);
        lon_state <= check_digit_index;
    elsif index_lon = 2 then
        bcd_digit_lon := unsigned(bcd_lon_sig(11 downto 8));
        if bcd_digit_lon > 4 then
            bcd_digit_lon := bcd_digit_lon + 3;
        end if;

        bcd_lon_sig(11 downto 8) <= std_logic_vector(bcd_digit_lon);
```

```vhdl
    lon_state <= check_digit_index;
elsif index_lon = 3 then
    bcd_digit_lon := unsigned(bcd_lon_sig(15 downto 12));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(15 downto 12) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
elsif index_lon = 4 then
    bcd_digit_lon := unsigned(bcd_lon_sig(19 downto 16));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(19 downto 16) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
elsif index_lon = 5 then
    bcd_digit_lon := unsigned(bcd_lon_sig(23 downto 20));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(23 downto 20) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
elsif index_lon = 6 then
    bcd_digit_lon := unsigned(bcd_lon_sig(27 downto 24));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(27 downto 24) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
elsif index_lon = 7 then
    bcd_digit_lon := unsigned(bcd_lon_sig(31 downto 28));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(31 downto 28) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
elsif index_lon = 8 then
    bcd_digit_lon := unsigned(bcd_lon_sig(35 downto 32));
    if bcd_digit_lon > 4 then
        bcd_digit_lon := bcd_digit_lon + 3;
    end if;

    bcd_lon_sig(35 downto 32) <= std_logic_vector(bcd_digit_lon);
    lon_state <= check_digit_index;
end if;
```

```
            when check_digit_index =>
                if index_lon = longitude_digits-1 then
                    index_lon <= 0;
                    lon_state <= shift;
                else
                    index_lon <= index_lon + 1;
                    lon_state <= add;
                end if;
            when bcd_done =>
                lon_data(0) <= "0011" & bcd_lon_sig(35 downto 32);
                lon_data(1) <= "0011" & bcd_lon_sig(31 downto 28);
                lon_data(2) <= "0011" & bcd_lon_sig(27 downto 24);
                lon_data(3) <= "0011" & bcd_lon_sig(23 downto 20);
                lon_data(4) <= "0011" & bcd_lon_sig(19 downto 16);
                lon_data(6) <= "0011" & bcd_lon_sig(15 downto 12);
                lon_data(7) <= "0011" & bcd_lon_sig(11 downto 8);
                lon_data(8) <= "0011" & bcd_lon_sig(7 downto 4);
                lon_data(9) <= "0011" & bcd_lon_sig(3 downto 0);
                lon_state   <= send;
            when send =>
                if cnt_lon < 11 then
                    lon_state   <= send;
                    out_lon_sig <= lon_data(cnt_lon);
                    if cnt_lon_byte < cnt_byte-1 then
                        cnt_lon_byte <= cnt_lon_byte + 1;
                    else
                        cnt_lon_byte <= 0;
                        cnt_lon      <= cnt_lon + 1;
                    end if;
                else
                    cnt_lon     <= 0;
                    lon_state   <= idle;
                end if;
        end case;
    end if;
end process;

lon_done <= '1' when lon_state = send else '0';
out_lon  <= out_lon_sig;

altitude_process : process(clk,rst)
    variable bcd_digit_alt : unsigned(3 downto 0);
    variable cnt2clk : integer := 0;
begin
    if rst = '0' then
        alt_data(0) <= (others => '0');
        alt_data(1) <= (others => '0');
        alt_data(2) <= (others => '0');
        alt_data(4) <= (others => '0');
        out_alt_sig  <= (others => '0');
    elsif rising_edge(clk) then
```

```
case alt_state is
    when idle =>
        cnt_alt     <= 0;
        alt_data(0) <= (others => '0');
        alt_data(1) <= (others => '0');
        alt_data(2) <= (others => '0');
        alt_data(4) <= (others => '0');
        out_alt_sig  <= (others => '0');
        if mulai_konversi = '1' then
            if cnt2clk = 4 then
                cnt2clk := 0;
                bcd_alt_sig <= (others => '0');
                alt_temp    <= std_logic_vector(to_unsigned(alt_ave,32));
                alt_state   <= shift;
            else
                cnt2clk := cnt2clk + 1;
            end if;
        else
            alt_state   <= idle;
        end if;
    when shift =>
        bcd_alt_sig <= bcd_alt_sig(bcd_alt_sig'left-1 downto 0) &
        alt_temp(alt_temp'left);
        alt_temp    <= alt_temp(alt_temp'left-1 downto 0) & '0';
        alt_state   <= check_shift_index;
    when check_shift_index =>
        if loop_cnt_alt = input_width-1 then
            loop_cnt_alt <= 0;
            alt_state    <= bcd_done;
        else
            loop_cnt_alt <= loop_cnt_alt + 1;
            alt_state    <= add;
        end if;
    when add =>
            if index_alt = 0 then
                bcd_digit_alt := unsigned(bcd_alt_sig(3 downto 0));
                if bcd_digit_alt > 4 then
                    bcd_digit_alt := bcd_digit_alt + 3;
                end if;

                bcd_alt_sig(3 downto 0) <= std_logic_vector(bcd_digit_alt);
                alt_state <= check_digit_index;
            elsif index_alt = 1 then
                bcd_digit_alt := unsigned(bcd_alt_sig(7 downto 4));
                if bcd_digit_alt > 4 then
                    bcd_digit_alt := bcd_digit_alt + 3;
                end if;

                bcd_alt_sig(7 downto 4) <= std_logic_vector(bcd_digit_alt);
                alt_state <= check_digit_index;
            elsif index_alt = 2 then
```

```
                    bcd_digit_alt := unsigned(bcd_alt_sig(11 downto 8));
                    if bcd_digit_alt > 4 then
                        bcd_digit_alt := bcd_digit_alt + 3;
                    end if;

                    bcd_alt_sig(11 downto 8) <= std_logic_vector(bcd_digit_alt);
                    alt_state <= check_digit_index;
                elsif index_alt = 3 then
                    bcd_digit_alt := unsigned(bcd_alt_sig(15 downto 12));
                    if bcd_digit_alt > 4 then
                        bcd_digit_alt := bcd_digit_alt + 3;
                    end if;

                    bcd_alt_sig(15 downto 12) <= std_logic_vector(bcd_digit_alt);
                    alt_state <= check_digit_index;
                end if;

        when check_digit_index =>
            if index_alt = altitude_digits-1 then
                index_alt <= 0;
                alt_state <= shift;
            else
                index_alt <= index_alt + 1;
                alt_state <= add;
            end if;
        when bcd_done =>
            alt_data(0) <= "0011" & bcd_alt_sig(15 downto 12);
            alt_data(1) <= "0011" & bcd_alt_sig(11 downto 8);
            alt_data(2) <= "0011" & bcd_alt_sig(7 downto 4);
            alt_data(4) <= "0011" & bcd_alt_sig(3 downto 0);
            alt_state   <= send;
        when send =>
            if lon_done = '1' then
                if cnt_alt < 6 then
                    alt_state   <= send;
                    out_alt_sig <= alt_data(cnt_alt);
                    if cnt_alt_byte < cnt_byte-1 then
                        cnt_alt_byte <= cnt_alt_byte + 1;
                    else
                        cnt_alt_byte <= 0;
                        cnt_alt      <= cnt_alt + 1;
                    end if;
                else
                    cnt_alt     <= 0;
                    alt_state   <= idle;
                end if;
            else
                out_alt_sig <= "00000000";
            end if;
    end case;
end if;
```

```
    end process;

    out_alt  <= out_alt_sig;

end architecture;
```

List program of Arduino Master.

```
#include <Wire.h>

char ram[156];
char a,b,c,d;

void setup() {
  Wire.begin();
  Serial.begin(9600);
  pinMode(13, OUTPUT);
}

void loop() {
  readData();
  calculation();
  for(int n = 0; n<156; n++){
    Serial.print(ram[n]);
  }
  delay(500);
}

void readData(){
  const char new_line = '\n';
  const char koma = ',';
  const char titik = '.';
  ram[1]   = koma;
  ram[133] = titik;
  ram[138] = koma;
  ram[144] = titik;
  ram[149] = koma;
  ram[153] = titik;
  ram[155] = new_line;

  Wire.requestFrom(1,32);
  while (Wire.available()) {
    for(int i = 2; i<34; i++){
      a = Wire.read();
      ram[i] = a;
    }
  }
  Wire.requestFrom(2,32);
  while (Wire.available()) {
    for(int i = 34; i<66; i++){
      b = Wire.read();
      ram[i] = b;
    }
  }
  Wire.requestFrom(3,32);
  while (Wire.available()) {
    for(int i = 66; i<98; i++){
      c = Wire.read();
      ram[i] = c;
    }
  }
  Wire.requestFrom(4,32);
  while (Wire.available()) {
    for(int i = 98; i<130; i++){
      d = Wire.read();
      ram[i] = d;
    }
  }
}

void calculation(){
  double latInt1, latInt2, latInt3, latInt4, result_lat;
  double lonInt1, lonInt2, lonInt3, lonInt4, result_lon;
  double altInt1, altInt2, altInt3, altInt4, result_alt;
  long satInt1, satInt2, satInt3, satInt4;

  latInt1 = (((long)ram[2] - 48)*1000)+ (((long)ram[3]-48)*100) + (((long)ram[4]-48)*10) +
  (((long)ram[5]-48)*1) + (((long)ram[6]-48)*0) + (((long)ram[7]-48)*0.1) +
  (((long)ram[8]-48)*0.01) +  (((long)ram[9]-48)*0.001) + (((long)ram[10]-48)*0.0001) +
  (((long)ram[11]-48)*0.00001);
  latInt2 = (((long)ram[34] - 48)*1000)+(((long)ram[35]-48)*100)+ (((long)ram[36]-48)*10) +
  (((long)ram[37]-48)*1) + (((long)ram[38]-48)*0) + (((long)ram[39]-48)*0.1) +
  (((long)ram[40]-48)*0.01) +  (((long)ram[41]-48)*0.001) + (((long)ram[42]-48)*0.0001) +
  (((long)ram[43]-48)*0.00001);
  latInt3 = (((long)ram[66] - 48)*1000)+ (((long)ram[67]-48)*100)+(((long)ram[68]-48)*10) +
  (((long)ram[69]-48)*1) + (((long)ram[70]-48)*0) + (((long)ram[71]-48)*0.1) +
  (((long)ram[72]-48)*0.01) +  (((long)ram[73]-48)*0.001) + (((long)ram[74]-48)*0.0001) +
  (((long)ram[75]-48)*0.00001);
  latInt4 = (((long)ram[98] - 48)*1000)+(((long)ram[99]-48)*100)+(((long)ram[100]-48)*10) +
  (((long)ram[101]-48)*1) + (((long)ram[102]-48)*0) + (((long)ram[103]-48)*0.1) +
  (((long)ram[104]-48)*0.01) + (((long)ram[105]-48)*0.001) + (((long)ram[106]-48)*0.0001) +
  (((long)ram[107]-48)*0.00001);

  lonInt1 = (((long)ram[13]-48)*10000)+ ((long)ram[14]-48)*1000)+(((long)ram[15]-48)*100) +
  (((long)ram[16]-48)*10) + (((long)ram[17]-48)*1) + (((long)ram[18]-48)*0) +
  (((long)ram[19]-48)*0.1) + (((long)ram[20]-48)*0.01) +  (((long)ram[21]-48)*0.001) +
  (((long)ram[22]-48)*0.0001) + (((long)ram[23]-48)*0.00001);
  lonInt2 = (((long)ram[45]-48)*10000)+(((long)ram[46]-48)*1000)+(((long)ram[47]-48)*100) +
  (((long)ram[48]-48)*10) + (((long)ram[49]-48)*1) + (((long)ram[50]-48)*0) +
  (((long)ram[51]-48)*0.1) + (((long)ram[52]-48)*0.01) +  (((long)ram[53]-48)*0.001) +
  (((long)ram[54]-48)*0.0001) + (((long)ram[55]-48)*0.00001);
  lonInt3 = (((long)ram[77]-48)*10000)+(((long)ram[78]-48)*1000)+(((long)ram[79]-48)*100) +
  (((long)ram[80]-48)*10) + (((long)ram[81]-48)*1) + (((long)ram[82]-48)*0) +
  (((long)ram[83]-48)*0.1) + (((long)ram[84]-48)*0.01) +  (((long)ram[85]-48)*0.001) +
  (((long)ram[86]-48)*0.0001) + (((long)ram[87]-48)*0.00001);
  lonInt4 = (((long)ram[109]-48)*10000)+(((long)ram[110]-48)*1000)+
  (((long)ram[111]-48)*100) + (((long)ram[112]-48)*10) + (((long)ram[113]-48)*1) +
  (((long)ram[114]-48)*0) + (((long)ram[115]-48)*0.1) + (((long)ram[116]-48)*0.01) +
  (((long)ram[117]-48)*0.001) + (((long)ram[118]-48)*0.0001) +
  (((long)ram[119]-48)*0.00001);

  altInt1 = (((long)ram[28]-48)*100) + (((long)ram[29]-48)*10) + (((long)ram[30]-48)*1) +
  (((long)ram[31]-48)*0) + (((long)ram[32]-48)*0.1);
  altInt2 = (((long)ram[60]-48)*100) + (((long)ram[61]-48)*10) + (((long)ram[62]-48)*1) +
  (((long)ram[63]-48)*0) + (((long)ram[64]-48)*0.1);
  altInt3 = (((long)ram[92]-48)*100) + (((long)ram[93]-48)*10) + (((long)ram[94]-48)*1) +
  (((long)ram[95]-48)*0) + (((long)ram[96]-48)*0.1);
  altInt4 = (((long)ram[124]-48)*100) + (((long)ram[125]-48)*10)+ (((long)ram[126]-48)*1) +
  (((long)ram[127]-48)*0) + (((long)ram[128]-48)*0.1);

  satInt1 = (((long)ram[25]-48)*10) + ((long)ram[26]-48);
  satInt2 = (((long)ram[57]-48)*10) + ((long)ram[58]-48);
  satInt3 = (((long)ram[89]-48)*10) + ((long)ram[90]-48);
  satInt4 = (((long)ram[121]-48)*10) + ((long)ram[122]-48);

  // 4 Satellites valid
  if(satInt1 > 3 && satInt2 > 3 && satInt3 > 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt1 + latInt2 + latInt3 + latInt4)/4;
    lat_store(result_lat);
    result_lon = (lonInt1 + lonInt2 + lonInt3 + lonInt4)/4;
    lon_store(result_lon);
    result_alt = (altInt1 + altInt2 + altInt3 + altInt4)/4;
    alt_store(result_alt);
  }
  // 3 Satellites valid
  else if(satInt1 <= 3 && satInt2 > 3 && satInt3 > 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt2 + latInt3 + latInt4)/3;
    lat_store(result_lat);
    result_lon = (lonInt2 + lonInt3 + lonInt4)/3;
    lon_store(result_lon);
    result_alt = (altInt2 + altInt3 + altInt4)/3;
    alt_store(result_alt);
  }
  else if(satInt1 > 3 && satInt2 <= 3 && satInt3 > 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt1 + latInt3 + latInt4)/3;
    lat_store(result_lat);
    result_lon = (lonInt1 + lonInt3 + lonInt4)/3;
    lon_store(result_lon);
    result_alt = (altInt1 + altInt3 + altInt4)/3;
    alt_store(result_alt);
  }
  else if(satInt1 > 3 && satInt2 > 3 && satInt3 <= 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt1 + latInt2 + latInt4)/3;
    lat_store(result_lat);
    result_lon = (lonInt1 + lonInt2 + lonInt4)/3;
    lon_store(result_lon);
    result_alt = (altInt1 + altInt2 + altInt4)/3;
    alt_store(result_alt);
  }
  else if(satInt1 > 3 && satInt2 > 3 && satInt3 > 3 && satInt4 <= 3){
    ram[0] = '0';
    result_lat = (latInt2 + latInt3 + latInt4)/3;
    lat_store(result_lat);
    result_lon = (lonInt2 + lonInt3 + lonInt4)/3;
    lon_store(result_lon);
    result_alt = (altInt2 + altInt3 + altInt4)/3;
    alt_store(result_alt);
  }
  // 2 Satellites valid
  else if(satInt1 <= 3 && satInt2 <= 3 && satInt3 > 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt3 + latInt4)/2;
    lat_store(result_lat);
    result_lon = (lonInt3 + lonInt4)/2;
    lon_store(result_lon);
    result_alt = (altInt3 + altInt4)/2;
    alt_store(result_alt);
  }
  else if(satInt1 <= 3 && satInt2 > 3 && satInt3 <= 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt2 + latInt4)/2;
    lat_store(result_lat);
    result_lon = (lonInt2 + lonInt4)/2;
    lon_store(result_lon);
    result_alt = (altInt2 + altInt4)/2;
    alt_store(result_alt);
  }
  else if(satInt1 <= 3 && satInt2 > 3 && satInt3 > 3 && satInt4 <= 3){
    ram[0] = '0';
    result_lat = (latInt2 + latInt3)/2;
    lat_store(result_lat);
    result_lon = (lonInt2 + lonInt3)/2;
    lon_store(result_lon);
    result_alt = (altInt2 + altInt3)/2;
    alt_store(result_alt);
  }
  else if(satInt1 > 3 && satInt2 <= 3 && satInt3 <= 3 && satInt4 > 3){
    ram[0] = '0';
    result_lat = (latInt1 + latInt4)/2;
    lat_store(result_lat);
    result_lon = (lonInt1 + lonInt4)/2;
    lon_store(result_lon);
    result_alt = (altInt1 + altInt4)/2;
    alt_store(result_alt);
  }
```

```
else if(satInt1 > 3 && satInt2 <= 3 && satInt3 > 3 && satInt4 <= 3){
  ram[0] = '0';
  result_lat = (latInt1 + latInt3)/2;
  lat_store(result_lat);
  result_lon = (lonInt1 + lonInt3)/2;
  lon_store(result_lon);
  result_alt = (altInt1 + altInt3)/2;
  alt_store(result_alt);
}
else if(satInt1 > 3 && satInt2 > 3 && satInt3 <= 3 && satInt4 <= 3){
  ram[0] = '0';
  result_lat = (latInt1 + latInt2)/2;
  lat_store(result_lat);
  result_lon = (lonInt1 + lonInt2)/2;
  lon_store(result_lon);
  result_alt = (altInt1 + altInt2)/2;
  alt_store(result_alt);
}
// 1 Satellites valid
else if(satInt1 <= 3 && satInt2 <= 3 && satInt3 <= 3 && satInt4 > 3){
  ram[0] = '0';
  result_lat = latInt4;
  lat_store(result_lat);
  result_lon = lonInt4;
  lon_store(result_lon);
  result_alt = altInt4;
  alt_store(result_alt);
}
else if(satInt1 <= 3 && satInt2 <= 3 && satInt3 > 3 && satInt4 <= 3){
  ram[0] = '0';
  result_lat = latInt3;
  lat_store(result_lat);
  result_lon = lonInt3;
  lon_store(result_lon);
  result_alt = altInt3;
  alt_store(result_alt);
}
else if(satInt1 <= 3 && satInt2 > 3 && satInt3 <= 3 && satInt4 <= 3){
  ram[0] = '0';
  result_lat = latInt2;
  lat_store(result_lat);
  result_lon = lonInt2;
  lon_store(result_lon);
  result_alt = altInt2;
  alt_store(result_alt);
}
else if(satInt1 > 3 && satInt2 <= 3 && satInt3 <= 3 && satInt4 <= 3){
  ram[0] = '0';
  result_lat = latInt1;
  lat_store(result_lat);
  result_lon = lonInt1;
```

```
    lon_store(result_lon);
    result_alt = altInt1;
    alt_store(result_alt);
  }
  else{
    ram[0] = '1';
    result_lat = 0;
    lat_store(result_lat);
    result_lon = 0;
    lon_store(result_lon);
    result_alt = 0;
    alt_store(result_alt);
  }
}

void lat_store(double result_o){
  long result_n;

  result_n = result_o/100;
  ram[130] = result_n + 48;
  result_o = result_o - (result_n*100);

  result_n = result_o/10;
  ram[131] = result_n + 48;
  result_o = result_o - (result_n*10);

  result_n = result_o;
  ram[132] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[135] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[135] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[136] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[137] = result_n + 48;
}

void lon_store(double result_o){
  long result_n;

  result_n = result_o/10000;
  ram[139] = result_n + 48;
  result_o = result_o - (result_n*10000);

  result_n = result_o/1000;
  ram[140] = result_n + 48;
  result_o = result_o - (result_n*1000);

  result_n = result_o/100;
  ram[141] = result_n + 48;
  result_o = result_o - (result_n*100);

  result_n = result_o/10;
  ram[142] = result_n + 48;
  result_o = result_o - (result_n*10);

  result_n = result_o;
  ram[143] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[145] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[146] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[147] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[148] = result_n + 48;
}

void alt_store(double result_o){
  long result_n;

  result_n = result_o/100;
  ram[150] = result_n + 48;
  result_o = result_o - (result_n*100);

  result_n = result_o/10;
  ram[151] = result_n + 48;
  result_o = result_o - (result_n*10);

  result_n = result_o;
  ram[152] = result_n + 48;
  result_o = result_o - result_n;

  result_o = result_o*10;
  result_n = result_o;
  ram[154] = result_n + 48;
}
```

List program of Arduino Slave.

```
#include <Wire.h>
#include <SoftwareSerial.h>

//Uncomment chosen address
#define address 1
//#define address 2
//#define address 3
//#define address 4
SoftwareSerial GPSModule(3, 4); // RX, TX

int updates;
int failedUpdates;
int pos;
int stringplace = 0;

String timeUp;
String nmea[15];

void setup() {
  Wire.begin(address);
  Wire.onRequest(requestEvent);
  Serial.begin(9600);
  GPSModule.begin(9600);
}

void loop() {
  while (GPSModule.available() > 0)
  {
    GPSModule.read();
  }
  if (GPSModule.find("$GPGGA,")) {
    String tempMsg = GPSModule.readStringUntil('\n');
    for (int i = 0; i < tempMsg.length(); i++) {
      if (tempMsg.substring(i, i + 1) == ",") {
        nmea[pos] = tempMsg.substring(stringplace, i);
        stringplace = i + 1;
        pos++;
      }
      if (i == tempMsg.length() - 1) {
        nmea[pos] = tempMsg.substring(stringplace, i);
      }
    }
    updates++;
 }
  else {
    failedUpdates++;
  }
  stringplace = 0;
  pos = 0;

}

void requestEvent(){
  Wire.write(nmea[1].c_str());
  Wire.write(",");
  Wire.write(nmea[3].c_str());
  Wire.write(",");
  Wire.write(nmea[6].c_str());
  Wire.write(",");
  Wire.write(nmea[8].c_str());
  Wire.write(",");
}
```