Ørsted·DTU
Technical University of Denmark

Jens Kofod Hansen

GPS Synkroniseringsmodul

Bilag

Bachelorprojekt, April 2007

Jens Kofod Hansen

GPS Synkroniseringsmodul

Bilag

Bachelorprojekt, April 2007

GPS synkroniseringsmodul, bilag

Rapporten er udarbejdet af:
Jens Kofod Hansen

Vejleder(e):
Thomas Holm Hansen - DK-Technologies A/S
Gøsta Thuesen, Ørsted - DTU

Ørsted•DTU
Danmarks Tekniske Universitet
Ørsteds Plads
Bygning 348
2800 Kgs. Lyngby
Denmark

www.oersted.dtu.dk
Tel: (+45) 45 25 38 00
Fax: (+45) 45 93 16 34
E-mail: info@oersted.dtu.dk

Udgivelsesdato: 10. April, 2007

Klasse: 2 (fortrolig)

Udgave: 1. udgave

Bemærkninger: Dette dokument indeholder bilag til rapporten.

Bilag indhold

VHDL kildekode

C kildekode

GPS modul schematics

Udvalgte SMPTE LTC tidskodesider

Udvalgte NavSync databladssider

VHDL kildekode

Genlock main module

Clock kompensator

Genlock puls generator

LTC generator

I2C engine

Digital PLL

Constraint fil

```
  1: ----------------------------------------------------------------
  2: --               Main module
  3: ----------------------------------------------------------------
  4:
  5: library IEEE;
  6: use IEEE.STD_LOGIC_1164.ALL;
  7: use IEEE.STD_LOGIC_ARITH.ALL;
  8: use IEEE.STD_LOGIC_UNSIGNED.ALL;
  9:
 10: -- bibliotek til brug af IBUFGDS (clock niveauer)
 11: library UNISIM;
 12: use UNISIM.VComponents.all;
 13:
 14: entity GPS_Genlock_Module is
 15: port (
 16:     --inputs
 17:     clk_p_i     :   in std_logic;           --clock positiv
 18:    clk_n_i      :   in std_logic;           --clock negativ
 19:     scl_i           :   in std_logic;           --SCL til I2C
 20:     --reset_i       :   in std_logic;           --master reset
 21:     --dip0_i        :   in std_logic;           --dip 0 input
 22:     --dip1_i        :   in std_logic;           --dip 1 input
 23:     --dip2_i        :   in std_logic;           --dip 2 input
 24:     --dip3_i        :   in std_logic;           --dip 3 input
 25:     --freq_i        :   in std_logic;           --10 MHz input fra GPS
 26:     --led_red_i :   in std_logic;           --rød LED fra GPS
 27:     --led_green_i   :   in std_logic;       --grøn LED fra GPS
 28:     pps_i           :   in std_logic;           --1 Puls pr sekund fra GPS
 29:
 30:     --outputs
 31:     --ns_d_o        :   out std_logic;          --genlock signal output
 32:     --h_d_o         :   out std_logic;          --genlock signal output
 33:     --f_d_o         :   out std_logic;          --genlock signal output
 34:     VCXO_ctrl_o :   out std_logic;          --VCXO-control output
 35:     --event_o       :   out std_logic;          --eventpuls til GPS
 36:     --LTC_code_o    :   out std_logic;          --LTC_code output
 37:     led0_o      :   out std_logic;          --debug led
 38:     led1_o      :   out std_logic;          --debug led
 39:
 40:     --in/outputs
 41:     MCU_gpio0_io    :   out std_logic;      --general purpose io til MCU
 42:     MCU_gpio1_io    :   out std_logic;      --general purpose io til MCU
 43:     MCU_gpio2_io    :   in std_logic;       --general purpose io til MCU
 44:     MCU_gpio3_io    :   in std_logic;       --general purpose io til MCU
 45:     gpio0_io            :   in std_logic;       --general purpose io til GPS
 46:     --gpio1_io          :   inout std_logic;        --general purpose io til GPS
 47:     --gpio2_io          :   inout std_logic;        --general purpose io til GPS
 48:     --gpio3_io          :   inout std_logic;        --general purpose io til GPS
 49:     sda_io          :   inout std_logic     --SDA til I2C
 50: );
 51:
 52: end GPS_Genlock_Module;
 53:
 54: architecture Behavioral of GPS_Genlock_module is
 55:
 56: -- States til state-machine
 57: type state_type is (start_up, wait_for_GPS, wait_for_PLL, wait_for_time,
         wait_for_1pps, stable, unstable);
 58: signal kernel_state             : state_type:=start_up;
 59:
 60: -- System signals -------------------------------------------------
 61: -- master clock
 62: signal clk                     :   std_logic;      --148.5 MHz clock
 63:
 64: -- system resets
 65: signal kernel_reset         :   std_logic;
 66: signal master_reset         :   std_logic;
 67:
 68: -- system status variables
 69: signal master_status_flag   :   std_logic;
 70: signal PLL_status_flag      :   std_logic;
 71: signal PLL_locked_flag      :   std_logic;
```

```
 72: signal GPS_locked_flag      :   std_logic;
 73: signal PLL_lock_delayline   :   std_logic_vector (9 downto 0);
 74: signal sec_pulse_delayline  :   std_logic_vector (1 downto 0);
 75: signal sample_period            :   std_logic_vector (27 downto 0);
 76: signal time_recieved            :   std_logic;
 77: signal offset                   :   std_logic_vector (31 downto 0);
 78:
 79: -- I2C signaler
 80: signal I2C_read_flag            :   std_logic;
 81: signal I2C_read_delayline   :   std_logic_vector (1 downto 0);
 82: signal send_byte                :   std_logic_vector (7 downto 0);
 83: signal recieved_byte            :   std_logic_vector (7 downto 0);
 84: signal byte_count               :   std_logic_vector (2 downto 0);
 85: signal I2C_reset                :   std_logic;
 86:
 87: -- System timing ---------------------------------------------------
 88: -- GPS PLL reference clock
 89: -- 1PPS free run
 90: signal sec_pulse                :   std_logic;
 91: signal faster_tick          :   std_logic;
 92: signal slower_tick          :   std_logic;
 93:
 94: signal GPS_10_mhz               :   std_logic;
 95: signal GPS_ref_clk          :   std_logic;
 96: signal GPS_ref_count            :   std_logic_vector(4 downto 0);
 97:
 98: -- PLL decimeret output
 99: signal clk_vcxo_dec          :   std_logic;
100:
101: -- I2C interface komponent
102: component I2C_engine Port (
103:         reset_i                 : in  STD_LOGIC;
104:       clk_i                 : in  STD_LOGIC;
105:       SCL_i                 : in  STD_LOGIC;
106:       SDA_io                : inout  STD_LOGIC;
107:       send_byte_i       : in  STD_LOGIC_VECTOR (7 downto 0);
108:         recieved_byte_o     : out STD_LOGIC_VECTOR (7 downto 0);
109:         address_i           : in    STD_LOGIC_VECTOR (6 downto 0);
110:         byte_read_o         : out STD_LOGIC
111:     );
112: end component;
113:
114: -- Digital PLL komponent
115: component digital_pll is
116:    generic (
117:       vcxo_div_per         : integer         -- decimation of clk_i to pll, (594)
118:    );
119:    port (
120:       clk_vcxo_i           : in std_logic;
121:       clk_ref_decimated_i  : in std_logic;
122:       clk_vcxo_dec_tick_o  : out std_logic;  -- tick at falling edge of decimated
          vcxo to phase comparator
123:       pdm_o                : out std_logic;  -- pulse density modulated signal
124:       clk_vcxo_decimated_o : out std_logic;  -- for debug
125:       lock_warning_o       : out std_logic;
126:       lock_error_o         : out std_logic;
127:         ext_make_shorter        : in std_logic;
128:         ext_make_longer     : in std_logic
129:    );
130: end component;
131:
132: component clock_compensate is
133:    port (
134:       rst_i             :   in  std_logic;
135:         clk_i               :   in std_logic;
136:         pps_i               :   in  std_logic;
137:         synth_pps_o     :   out std_logic;
138:         phase_diff_o    :   out std_logic_vector(7 downto 0);
139:         faster_tick     :   out std_logic;
140:         slower_tick     :   out std_logic
141:    );
142: end component;
```

```
143:
144: begin
145:
146:
147: -------------------------------------------------------------------------
148: --------------    Instansering af komponenter    ---------------------
149: -------------------------------------------------------------------------
150:     diff_buf_clk : IBUFGDS
151:     generic map (IOSTANDARD => "LVDS_25_DT")
152:     port map (
153:         O  => clk,
154:         I  => clk_p_i,
155:         IB => clk_n_i );
156:
157:     -- I2C interface til LTC encoder
158:     I2C_interface: I2C_engine port map (
159:         reset_i => I2C_reset,
160:         clk_i => clk,
161:         SCL_i => SCL_i,
162:         SDA_io => SDA_io,
163:         send_byte_i => send_byte,
164:         recieved_byte_o => recieved_byte,
165:         address_i => "1010000",
166:         byte_read_o => I2C_read_flag
167:         );
168:
169:     -- PLL til VCXO med GPS 0.25 MHz som reference
170:     pll : digital_pll
171:     generic map(
172:         vcxo_div_per    =>  594
173:     )
174:     port map (
175:         clk_vcxo_i => clk,
176:       clk_ref_decimated_i => GPS_ref_clk,
177:         pdm_o => VCXO_ctrl_o,
178:         clk_vcxo_decimated_o => clk_vcxo_dec,
179:         --lock_error_o => led0_o,
180:         lock_warning_o => PLL_locked_flag,
181:         ext_make_longer => slower_tick,
182:         ext_make_shorter => faster_tick
183:     );
184:
185:     pps_synth : clock_compensate port map (
186:         rst_i => kernel_reset,
187:         clk_i => clk,
188:         pps_i => pps_i,
189:         synth_pps_o => sec_pulse,
190:         --phase_diff_o => send_byte,
191:         faster_tick => faster_tick,
192:         slower_tick => slower_tick
193:     );
194:
195:
196: -------------------------------------------------------------------------
197: --------------   hovedkerne statemachine styring   ---------------------
198: -------------------------------------------------------------------------
199:     kernel_machine : process(master_reset, clk)
200:     begin
201:         if clk'event and clk='1' then
202:             if master_reset='1' then
203:                 kernel_state <= start_up;
204:             else
205:                 if kernel_state=start_up then
206:                     kernel_state <= wait_for_GPS;
207:                 elsif kernel_state=wait_for_GPS and GPS_locked_flag='1' then
208:                     kernel_state <= wait_for_PLL;
209:                 elsif kernel_state=wait_for_PLL and PLL_status_flag='1' then
210:                     kernel_state <= wait_for_time;
211:                 elsif kernel_state=wait_for_time and time_recieved='1' then
212:                     kernel_state <= wait_for_1pps;
213:                 elsif kernel_state=wait_for_1pps and sec_pulse_delayline = "01"
                        then
```

```
214:                     kernel_state <= stable;
215:                 elsif kernel_state=stable and GPS_locked_flag='0' then
216:                     kernel_state <= unstable;
217:                 elsif kernel_state=unstable and GPS_locked_flag='1' then
218:                     kernel_state <= stable;
219:                 end if;
220:             end if;
221:         end if;
222:     end process;
223: 
224:     kernel_machine_interpret : process(kernel_state)
225:     begin
226:         if kernel_state = start_up then
227:             kernel_reset <= '1';
228:             I2C_reset <= '1';
229:             send_byte <= "00000001";
230:         elsif kernel_state = wait_for_GPS then
231:             kernel_reset <= '1';
232:             I2C_reset <= '1';
233:             send_byte <= "00000010";
234:         elsif kernel_state = wait_for_PLL then
235:             kernel_reset <= '1';
236:             I2C_reset <= '1';
237:             send_byte <= "00000011";
238:         elsif kernel_state = wait_for_time then
239:             mcu_gpio1_io <= PLL_status_flag;
240:             kernel_reset <= '1';
241:             I2C_reset <= '0';
242:             send_byte <= "00000100";
243:         elsif kernel_state = wait_for_1pps then
244:             kernel_reset <= '1';
245:             I2C_reset <= '0';
246:             send_byte <= "00000101";
247:         elsif kernel_state = stable then
248:             kernel_reset <= '0';
249:             I2C_reset <= '0';
250:             send_byte <= "00000110";
251:         elsif kernel_state = unstable then
252:             kernel_reset <= '0';
253:             I2C_reset <= '0';
254:             send_byte <= "00000111";
255:         end if;
256:     end process;
257: 
258: ------------------------------------------------------------------------
259: ----------------------    I2C-håndtering    ----------------------------
260: ------------------------------------------------------------------------
261: 
262:     I2C_handling : process(clk, I2C_reset)
263:     begin
264:         if clk'event and clk= '1' then
265:             if I2C_reset = '1' then
266:                 byte_count <= "000";
267:                 time_recieved <= '0';
268:             else
269:                 I2C_read_delayline <= I2C_read_delayline(0) & I2C_read_flag;
270: 
271:                 if I2C_read_delayline = "01" then
272:                     if recieved_byte = "10111011" then
273:                         byte_count <= "000";
274:                     else
275:                         byte_count <= byte_count + 1;
276:                         case byte_count is
277:                             when "001" =>
278:                                 offset(7 downto 0) <= recieved_byte;
279:                             when "010" =>
280:                                 offset(15 downto 8) <= recieved_byte;
281:                             when "011" =>
282:                                 offset(23 downto 16) <= recieved_byte;
283:                             when "100" =>
284:                                 offset(31 downto 24) <= recieved_byte;
285:                             when others =>
```

```
286:                                 offset <= "00000000000000000000000000000000";
287:                             end case;
288:                         end if;
289:                     end if;
290:
291:                     if byte_count = "011" then
292:                         time_recieved <= '1';
293:                     else
294:                         time_recieved <= '0';
295:                     end if;
296:
297:                 end if;
298:             end if;
299:         end process;
300:
301: -------------------------------------------------------------------------
302: ------------------   master-signaler og clocks   ----------------------
303: -------------------------------------------------------------------------
304:         --generer 250 KHz til PLL fra 10 MHz GPS clock
305:         GPS_ref_clk_gen : process(GPS_10_mhz, master_reset)
306:         begin
307:             if master_reset = '1' then
308:                 GPS_ref_count <= conv_std_logic_vector(19, 5);
309:                 GPS_ref_clk <= '0';
310:             elsif GPS_10_mhz'event and GPS_10_mhz='1' then
311:                 GPS_ref_count <= GPS_ref_count-1;
312:                 if GPS_ref_count = 0 then
313:                     GPS_ref_count <= conv_std_logic_vector(19, 5);
314:                     GPS_ref_clk <= NOT GPS_ref_clk;
315:                 end if;
316:             end if;
317:         end process;
318:
319:         --delay_line generator for div signaler
320:         delayline_gen : process(clk, master_reset)
321:         begin
322:             if clk'event and clk='1' then
323:                 if master_reset='1' then
324:                     PLL_lock_delayline <= "0000000000";
325:                     sample_period <= conv_std_logic_vector(148499999, 28);
326:                     sec_pulse_delayline <= "00";
327:                 else
328:                     --intern 1 PPS delayline
329:                     sec_pulse_delayline <= sec_pulse_delayline(0) & pps_i;
330:
331:                     --PLL lock line
332:                     if sample_period = 0 then
333:                         sample_period <= conv_std_logic_vector(148499999, 28);
334:                         PLL_lock_delayline <= PLL_lock_delayline(8 downto 0) & not
                                PLL_locked_flag;
335:
336:                         if PLL_lock_delayline="1111111111" then
337:                             PLL_status_flag <= '1';
338:                         else
339:                             PLL_status_flag <= '0';
340:                         end if;
341:                     else
342:                         sample_period <= sample_period-1;
343:                     end if;
344:                 end if;
345:             end if;
346:         end process;
347:
348:
349: -------------------------------------------------------------------------
350: -----------------   hovedkerne signal-routing   ----------------------
351: -------------------------------------------------------------------------
352:         --output for LEDs (debuggig)
353:         led0_o <= sec_pulse;
354:         led1_o <= master_reset;--pps_i;
355:
356:         --master signaler
```

```
357:     master_reset <= mcu_gpio3_io;--reset_i;
358:
359:     --in/outs
360:     mcu_gpio0_io <= sec_pulse;
361:     GPS_10_mhz <= gpio0_io;
362:     GPS_locked_flag <= mcu_gpio2_io;
363: end Behavioral;
364:
365:
```

```
  1: ----------------------------------------------------------------
  2: --                Clock kompensator
  3: ----------------------------------------------------------------
  4:
  5: library IEEE;
  6: use IEEE.STD_LOGIC_1164.ALL;
  7: use IEEE.STD_LOGIC_ARITH.ALL;
  8: use IEEE.STD_LOGIC_SIGNED.ALL;
  9:
 10: entity clock_compensate is
 11:    port (
 12:       rst_i             :   in  std_logic;
 13:         clk_i               :   in std_logic;
 14:         pps_i               :   in  std_logic;
 15:         synth_pps_o     :   out std_logic;
 16:         phase_diff_o    :   out std_logic_vector(7 downto 0);
 17:         faster_tick     :   out std_logic;
 18:         slower_tick     :   out std_logic
 19:    );
 20: end clock_compensate;
 21:
 22: architecture Behavioral of clock_compensate is
 23: -- 1 pps syntese
 24: signal sec_pulse                :   std_logic;
 25: signal pps_count                :   std_logic_vector (28 downto 0);
 26: signal phase_count          :   std_logic_vector (28 downto 0);
 27: signal pps_i_delayline      :   std_logic_vector (1 downto 0);
 28: signal pps_phase_diff       :   std_logic_vector (7 downto 0);
 29: signal shorter_pers         :   std_logic_vector (7 downto 0);
 30: signal longer_pers          :   std_logic_vector (7 downto 0);
 31: signal make_longer          :   std_logic;
 32: signal make_shorter         :   std_logic;
 33: signal pps_first_time_flag  :   std_logic;
 34:
 35: begin
 36:     --generer 1 Hz 50:50 puls i sync med 1PPS fra GPS
 37:     pps_gen : process(clk_i, rst_i)
 38:     begin
 39:         if clk_i'event and clk_i='1' then
 40:             if rst_i='1' then
 41:                 pps_count <= "11111111111111111111111111111";
 42:                 phase_count <= conv_std_logic_vector(74250000, 29);
 43:                 sec_pulse <= '1';
 44:                 pps_first_time_flag <= '1';
 45:                 pps_i_delayline <= "00";
 46:                 shorter_pers <= "00000000";
 47:                 longer_pers <= "00000000";
 48:             else
 49:                 --generer 50:50 duty cycle
 50:                 if pps_count = 0 and sec_pulse = '1' then
 51:                     phase_count <= conv_std_logic_vector(74250000, 29);
 52:                     pps_count <= conv_std_logic_vector(74249999, 29);
 53:                     sec_pulse <= '0';
 54:                 elsif pps_count = 0 and sec_pulse = '0' then
 55:                     pps_count <= conv_std_logic_vector(74249999, 29);
 56:                     sec_pulse <= '1';
 57:                 else
 58:                     phase_count <= phase_count-1;
 59:                     pps_count <= pps_count-1;
 60:                 end if;
 61:
 62:                 -- synkroniser med GPS 1PPS, efter stabilisering ved opstart
 63:                 pps_i_delayline(1) <= pps_i_delayline(0);
 64:                 pps_i_delayline(0) <= pps_i;
 65:
 66:                 if pps_i_delayline = "01" and pps_first_time_flag = '1' then
 67:                     pps_count <= conv_std_logic_vector(74249999, 29);
 68:                     pps_first_time_flag <= '0';
 69:
 70:                 -- mål faseforskydning på free-run pps og GPS PPS
 71:                 elsif pps_i_delayline = "01" and pps_first_time_flag = '0' then
 72:                     pps_phase_diff <= phase_count(7 downto 0);
```

```
 73:
 74:                     --positiv forskydning
 75:                     if phase_count > 0 then
 76:                         shorter_pers <= shorter_pers + 1;
 77:                         longer_pers <= "00000000";
 78:
 79:                         if shorter_pers > "00000011" then
 80:                             slower_tick <= '1';
 81:                             shorter_pers <= "00000000";
 82:                         end if;
 83:
 84:                     --negativ forskydning
 85:                     elsif phase_count < 0 then
 86:                         longer_pers <= longer_pers + 1;
 87:                         shorter_pers <= "00000000";
 88:
 89:                         if longer_pers > "00000011" then
 90:                             faster_tick <= '1';
 91:                             longer_pers <= "00000000";
 92:                         end if;
 93:                     --ingen forskydning
 94:                     else
 95:                         longer_pers <= "00000000";
 96:                         shorter_pers <= "00000000";
 97:                     end if;
 98:                 else
 99:                     faster_tick <= '0';
100:                     slower_tick <= '0';
101:                 end if;
102:
103:             end if; --reset = '1'
104:         end if; -- clk'event
105:     end process;
106:
107:     synth_pps_o <= sec_pulse;
108:     phase_diff_o <= pps_phase_diff;
109:
110: end Behavioral;
111:
112:
```

```vhdl
----------------------------------------------------------------
--               Pulse generator
----------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity pulse_gen is
    Port ( clk_i : in  STD_LOGIC;
           rst_i : in  STD_LOGIC;
           per_hi_i : in  STD_LOGIC_VECTOR (19 downto 0);
           per_lo_i : in  STD_LOGIC_VECTOR (19 downto 0);
           per_offset_i : in  STD_LOGIC_VECTOR (7 downto 0);
           pulse_o : out  STD_LOGIC;
           start_state_i : in  STD_LOGIC);
end pulse_gen;

architecture Behavioral of pulse_gen is

signal pulse_state  :   std_logic;
signal count            :   std_logic_vector(19 downto 0);

begin

    count_period : process(clk_i, rst_i, start_state_i, per_offset_i )
    begin
        --reset
        if rst_i = '1' then
            --set start state, og count for reset
            pulse_state <= start_state_i;
            count <= "000000000000" & per_offset_i;

        elsif clk_i'event and clk_i='1' then
            if count = 0 then
                if pulse_state = '0' then
                    pulse_state <= '1';
                    count <= per_hi_i;
                else
                    pulse_state <='0';
                    count <= per_lo_i;
                end if;
            else
                count <= count - 1;
            end if;
        end if;
    end process count_period;

    pulse_o <= pulse_state;

end Behavioral;


```

```
  1: ----------------------------------------------------------------
  2: --               LTC generator
  3: ----------------------------------------------------------------
  4:
  5: library IEEE;
  6: use IEEE.STD_LOGIC_1164.ALL;
  7: use IEEE.STD_LOGIC_ARITH.ALL;
  8: use IEEE.STD_LOGIC_UNSIGNED.ALL;
  9:
 10: entity LTC_generator is
 11:     Port ( hours_i  :   in STD_LOGIC_VECTOR (7 downto 0);
 12:                mins_i    :   in STD_LOGIC_VECTOR (7 downto 0);
 13:                secs_i    :   in STD_LOGIC_VECTOR (7 downto 0);
 14:                format_select_i   :   in STD_LOGIC_VECTOR (1 downto 0);
 15:            rst_i : in  STD_LOGIC;
 16:            clk_i : in  STD_LOGIC;    -- 148.5 MHz master clock
 17:            ltc_code_o : out  STD_LOGIC);
 18: end LTC_generator;
 19:
 20: architecture Behavioral of LTC_generator is
 21:
 22: --main signals
 23: signal format_select    : STD_LOGIC_VECTOR(1 downto 0); --valg af 24, 25 og 30/1,
         001 FPS
 24:
 25: --modulation clock signals
 26: signal ltc_clock_tick   : STD_LOGIC;
 27: signal period_count     : STD_LOGIC_VECTOR (15 downto 0); --downclock tæller
 28: signal period_num_count : STD_LOGIC_VECTOR (3 downto 0); --tæller for lange/korte
         perioder
 29: signal long_short_select: STD_LOGIC:='0'; --skifter mellem at tælle korte='0'/lange
         ='1' perioder
 30:
 31: --24 FPS
 32: constant long_per_24_FPS    : STD_LOGIC_VECTOR (15 downto 0):="1001011100010000";
 33: constant short_per_24_FPS   : STD_LOGIC_VECTOR (15 downto 0):="1001011100001111";
 34: constant long_per_24_numof  : STD_LOGIC_VECTOR (3 downto 0):="0111";
 35: constant short_per_24_numof: STD_LOGIC_VECTOR (3 downto 0):="0001";
 36:
 37: --25 FPS
 38: constant long_per_25_FPS    : STD_LOGIC_VECTOR (15 downto 0):="1001000100000101";
 39: constant short_per_25_FPS   : STD_LOGIC_VECTOR (15 downto 0):="1001000100000101";
 40: constant long_per_25_numof  : STD_LOGIC_VECTOR (3 downto 0):="0011";
 41: constant short_per_25_numof: STD_LOGIC_VECTOR (3 downto 0):="0011";
 42:
 43: --30/1,001 FPS
 44: constant long_per_30_FPS    : STD_LOGIC_VECTOR (15 downto 0):="0111100011111111";
 45: constant short_per_30_FPS   : STD_LOGIC_VECTOR (15 downto 0):="0111100011111000";
 46: constant long_per_30_numof  : STD_LOGIC_VECTOR (3 downto 0):="0001";
 47: constant short_per_30_numof: STD_LOGIC_VECTOR (3 downto 0):="1111";
 48:
 49:
 50: --biphase modulated frame signals
 51: signal bit_count            : STD_LOGIC_VECTOR (7 downto 0);
 52: signal biphase_code     : STD_LOGIC;
 53: signal ltc_frame            : STD_LOGIC_VECTOR (79 downto 0);
 54:
 55: --timing frame signal
 56: signal hours_tens           : STD_LOGIC_VECTOR (1 downto 0);
 57: signal hours_units      : STD_LOGIC_VECTOR (3 downto 0);
 58: signal mins_tens            : STD_LOGIC_VECTOR (2 downto 0);
 59: signal mins_units           : STD_LOGIC_VECTOR (3 downto 0);
 60: signal secs_tens            : STD_LOGIC_VECTOR (2 downto 0);
 61: signal secs_units           : STD_LOGIC_VECTOR (3 downto 0);
 62: signal frame_tens           : STD_LOGIC_VECTOR (0 to 1);
 63: signal frame_units      : STD_LOGIC_VECTOR (0 to 3);
 64:
 65: constant sync_word      : STD_LOGIC_VECTOR (15 downto 0):="0011111111111101";
 66: constant BG                 : STD_LOGIC_VECTOR (3 downto 0):="0000";
 67:
 68: signal down_clk : STD_LOGIC_VECTOR(3 downto 0);
 69:
```

```
 70: begin
 71:     format_select <= format_select_i;
 72:
 73:     code_generator : process(clk_i, rst_i)
 74:     begin
 75: ---------------------------------------------------------------------------------
 76:         --LTC reset
 77:         if rst_i = '1' then
 78:             bit_count <= conv_std_logic_vector(0, 8);
 79:             hours_tens <= hours_i(4) & hours_i(5);
 80:             hours_units <= hours_i(0) & hours_i(1) & hours_i(2) & hours_i(3);
 81:             mins_tens <= mins_i(4) & mins_i(5) & mins_i(6);
 82:             mins_units <= mins_i(0) & mins_i(1) & mins_i(2) & mins_i (3);
 83:             secs_tens <= secs_i(4) & secs_i(5) & secs_i(6);
 84:             secs_units <= secs_i(0) & secs_i(1) & secs_i(2) & secs_i(3);
 85:             frame_tens <= "00";
 86:             frame_units <= "0000";
 87:
 88:             ltc_clock_tick <= '0';
 89:             long_short_select <= '0';
 90:             period_count <= "0000000000000001";
 91:             period_num_count <= "0001";
 92:
 93: ---------------------------------------------------------------------------------
 94:         --LTC clock gen
 95:         elsif clk_i'event and clk_i='1' then
 96:             period_count <= period_count-1;
 97:
 98:             case format_select is
 99:                 --generate 24 FPS clock
100:                 when "00" =>
101:                     if period_count = 1 then
102:                         ltc_clock_tick <= '1';
103:                         period_num_count <= period_num_count-1;
104:
105:                         if period_num_count = 1 and long_short_select = '1' then
106:                             long_short_select <= '0';
107:                             period_num_count <= long_per_24_numof;
108:                         elsif period_num_count = 1 and long_short_select = '0' then
109:                             long_short_select <= '1';
110:                             period_num_count <= short_per_24_numof;
111:                         end if;
112:
113:                         if long_short_select = '0' then
114:                             period_count <= long_per_24_FPS;
115:                         else
116:                             period_count <= short_per_24_FPS;
117:                         end if;
118:                     else
119:                         ltc_clock_tick <= '0';
120:                     end if;
121:                 --generate 25 FPS clock
122:                 when "01" =>
123:                     if period_count = 1 then
124:                         ltc_clock_tick <= '1';
125:                         period_num_count <= period_num_count-1;
126:
127:                         if period_num_count = 1 and long_short_select = '1' then
128:                             long_short_select <= '0';
129:                             period_num_count <= long_per_25_numof;
130:                         elsif period_num_count = 1 and long_short_select = '0' then
131:                             long_short_select <= '1';
132:                             period_num_count <= short_per_25_numof;
133:                         end if;
134:
135:                         if long_short_select = '0' then
136:                             period_count <= long_per_25_FPS;
137:                         else
138:                             period_count <= short_per_25_FPS;
139:                         end if;
140:                     else
141:                         ltc_clock_tick <= '0';
```

```
142:                     end if;
143:                 --generate 30 FPS clock
144:                 when "10" =>
145:                     if period_count = 1 then
146:                         ltc_clock_tick <= '1';
147:                         period_num_count <= period_num_count-1;
148:
149:                         if period_num_count = 1 and long_short_select = '1' then
150:                             long_short_select <= '0';
151:                             period_num_count <= long_per_30_numof;
152:                         elsif period_num_count = 1 and long_short_select = '0' then
153:                             long_short_select <= '1';
154:                             period_num_count <= short_per_30_numof;
155:                         end if;
156:
157:                         if long_short_select = '0' then
158:                             period_count <= long_per_30_FPS;
159:                         else
160:                             period_count <= short_per_30_FPS;
161:                         end if;
162:                     else
163:                         ltc_clock_tick <= '0';
164:                     end if;
165:                 when others =>
166:                     null;
167:             end case;
168:
169: ---------------------------------------------------------------------------
170:             if ltc_clock_tick = '1' then
171:                 down_clk <= down_clk +1 ;
172:
173:                 bit_count <= bit_count-1;
174:                 --for hver frame
175:                 if bit_count = 0 then
176:                     bit_count <= conv_std_logic_vector(159, 8);
177:                     hours_tens <= hours_i(4) & hours_i(5);
178:                     hours_units <= hours_i(0) & hours_i(1) & hours_i(2) & hours_i(3
                            );
179:                     mins_tens <= mins_i(4) & mins_i(5) & mins_i(6);
180:                     mins_units <= mins_i(0) & mins_i(1) & mins_i(2) & mins_i (3);
181:                     secs_tens <= secs_i(4) & secs_i(5) & secs_i(6);
182:                     secs_units <= secs_i(0) & secs_i(1) & secs_i(2) & secs_i(3);
183:
184:                     --opdater 80 bits i LTC frame
185:                     ltc_frame <= frame_units & BG & frame_tens & "00" & BG &
                            secs_units & BG & secs_tens & "0" & BG & mins_units & BG &
                            mins_tens & "0" & BG & hours_units & BG & hours_tens & "00"
                            & BG & sync_word;
186:
187:                     if down_clk = "1111" then
188:                         --tæl frames
189:                         frame_units <= frame_units + 1;
190:
191:                         if frame_units = "1001" then
192:                             frame_tens <= frame_tens + 1;
193:                             frame_units <= "0000";
194:                         end if;
195:                     end if;
196:
197:                     --reset frame tæller
198:                     case format_select is
199:                         --24 FPS
200:                         when "00" =>
201:                             if frame_tens = "10" and frame_units = "0011" then
202:                                 frame_tens <= "00";
203:                                 frame_units <= "0000";
204:                             end if;
205:                         --25 FPS
206:                         when "01" =>
207:                             if frame_tens = "01" and frame_units = "0010" then
208:                                 frame_tens <= "00";
209:                                 frame_units <= "0000";
```

```
210:                             end if;
211:                         --30 FPS
212:                         when "10" =>
213:                             if frame_tens = "01" and frame_units = "1001" then
214:                                 frame_tens <= "00";
215:                                 frame_units <= "0000";
216:                             end if;
217:                         --others = 25 FPS
218:                         when others =>
219:                             if frame_tens = "01" and frame_units = "0010" then
220:                                 frame_tens <= "00";
221:                                 frame_units <= "0000";
222:                             end if;
223:                     end case;
224:                 end if;
225:
226:                 --for hver bit
227:                 if bit_count(0) = '0' then
228:                     biphase_code <= not biphase_code;
229:
230:                 elsif bit_count(0)='1' and ltc_frame(conv_integer(bit_count(7
                        downto 1)))='1' then
231:                     biphase_code <= not biphase_code;
232:                 end if;
233:             end if;
234:         end if; -- ltc_clock_tick = '1'
235:     end process code_generator;
236:
237:     ltc_code_o <= biphase_code;
238: end Behavioral;
239:
240:
```

```
  1: ----------------------------------------------------------------
  2: --                I2C engine
  3: ----------------------------------------------------------------
  4:
  5: library IEEE;
  6: use IEEE.STD_LOGIC_1164.ALL;
  7: use IEEE.STD_LOGIC_ARITH.ALL;
  8: use IEEE.STD_LOGIC_UNSIGNED.ALL;
  9:
 10: entity I2C_engine is
 11:     Port ( reset_i          : in  STD_LOGIC;
 12:            clk_i                : in  STD_LOGIC;
 13:            SCL_i                : in  STD_LOGIC;
 14:            SDA_io           : inout  STD_LOGIC;
 15:            send_byte_i      : in  STD_LOGIC_VECTOR (7 downto 0);
 16:            recieved_byte_o : out    STD_LOGIC_VECTOR (7 downto 0);
 17:               address_i         : in    STD_LOGIC_VECTOR (6 downto 0);
 18:               byte_read_o       : out STD_LOGIC
 19:               );
 20: end I2C_engine;
 21:
 22: architecture Behavioral of I2C_engine is
 23: signal SDA                   : STD_LOGIC;
 24: signal SCL                   : STD_LOGIC;
 25: signal SDA_out               : STD_LOGIC;
 26: signal SDA_in                : STD_LOGIC;
 27: signal SDA_delay             : STD_LOGIC;
 28: signal SCL_delay             : STD_LOGIC;
 29: signal SDA_rise_flag    : STD_LOGIC;
 30: signal SDA_fall_flag    : STD_LOGIC;
 31: signal SCL_rise_flag    : STD_LOGIC;
 32: signal SCL_fall_flag    : STD_LOGIC;
 33:
 34: signal bit_count             : integer range 0 to 8:=0;
 35: signal command               : STD_LOGIC_VECTOR(7 downto 0);
 36: signal recieved_byte         : STD_LOGIC_VECTOR(7 downto 0);
 37: signal time_out          : STD_LOGIC_VECTOR(21 downto 0);
 38: signal RW_dir                : STD_LOGIC;
 39:
 40: -- States til state-machine
 41: type state_type is (idle, read_command, read_byte, write_byte, give_ack_command,
         give_ack_byte);
 42: signal state                 : state_type:=idle;
 43:
 44: begin
 45:     --SDA og SCL delay line
 46:     signal_reclocking: process (reset_i, clk_i)
 47:     begin
 48:         if reset_i = '1' then
 49:             SDA <= '1';
 50:             SDA_delay <= '1';
 51:             SCL <= '1';
 52:             SCL_delay <= '1';
 53:         elsif clk_i'event and clk_i = '1' then
 54:             SDA <= SDA_in;
 55:             SDA_delay <= SDA;
 56:             SCL <= SCL_i;
 57:             SCL_delay <= SCL;
 58:         end if;
 59:     end process signal_reclocking;
 60:
 61:
 62:     --SDA og SCL event-monitor. Sætter flag højt ved stigende eller faldende flanke
 63:     I2C_event: process (reset_i, clk_i)
 64:     begin
 65:         if reset_i = '1' then
 66:             SDA_rise_flag <= '0';
 67:             SDA_fall_flag <= '0';
 68:             SCL_rise_flag <= '0';
 69:             SCL_fall_flag <= '0';
 70:         elsif clk_i'event and clk_i = '1' then
 71:             --SDA events
```

```
 72:             if SDA = '1' and SDA_delay = '0' then
 73:                 SDA_fall_flag <= '0';
 74:                 SDA_rise_flag <= '1';
 75:             elsif SDA = '0' and SDA_delay = '1' then
 76:                 SDA_rise_flag <= '0';
 77:                 SDA_fall_flag <= '1';
 78:             else
 79:                 SDA_rise_flag <= '0';
 80:                 SDA_fall_flag <= '0';
 81:             end if;
 82:             --SCL events
 83:             if SCL = '1' and SCL_delay = '0' then
 84:                 SCL_fall_flag <= '0';
 85:                 SCL_rise_flag <= '1';
 86:             elsif SCL = '0' and SCL_delay = '1' then
 87:                 SCL_rise_flag <= '0';
 88:                 SCL_fall_flag <= '1';
 89:             else
 90:                 SCL_rise_flag <= '0';
 91:                 SCL_fall_flag <= '0';
 92:             end if;
 93:         end if;
 94:     end process I2C_event;
 95:
 96:
 97:     --I2C maskine, fortolker SDA og SCL og sætter pågældende states
 98:     I2C_machine: process (reset_i, clk_i)
 99:     begin
100:         if reset_i = '1' then
101:             state <= idle;
102:             bit_count <= 0;
103:             time_out <= conv_std_logic_vector(0, 22);
104:             RW_dir <= '0';
105:
106:         elsif clk_i'event and clk_i='1' then
107:         --reset timer
108:         time_out <= time_out + 1;
109:
110:         --I2C-start
111:             if SCL = '1' and SDA_fall_flag = '1' and state = idle then
112:                 bit_count <= 0;
113:                 time_out <= conv_std_logic_vector(0, 22);
114:                 state <= read_command;
115:
116:         --I2C-stop
117:             elsif SCL = '1' and SDA_rise_flag = '1' then
118:                 state <= idle;
119:
120:         --læs kommando+adresse/byte ind bitvist
121:             elsif SCL_rise_flag = '1' then
122:                 time_out <= conv_std_logic_vector(0, 22);
123:                 --command
124:                 if state = read_command then
125:                     if bit_count/=8 then
126:                         command(7-bit_count) <= SDA;
127:                         bit_count <= bit_count + 1;
128:                     end if;
129:
130:                 --byte
131:                 elsif state = read_byte then
132:                     if bit_count/=8 then
133:                         recieved_byte(7-bit_count) <= SDA;
134:                         bit_count <= bit_count + 1;
135:                     end if;
136:                 end if; -- State = read_command/read_byte
137:
138:             elsif SCL_fall_flag = '1' then
139:                 if bit_count /= 8 and state = write_byte then
140:                     SDA_out <= send_byte_i(7-bit_count);
141:                     bit_count <= bit_count + 1;
142:                 end if;
143:
```

```
144:                 if bit_count = 8 and (state = write_byte or state = read_byte) then
145:                     rw_dir<='1';
146:                     SDA_out <= '0';
147:                     recieved_byte_o <= recieved_byte;
148:                     state<=give_ack_byte;
149:                     byte_read_o <= '1';
150:
151:                 elsif bit_count = 8 and state = read_command then
152:                     if command(7 downto 1) = address_i then
153:                         rw_dir <= '1';
154:                         SDA_out <= '0';
155:                         state <= give_ack_command;
156:                     --forkert adresse -> idle
157:                     else
158:                         state <= idle;
159:                     end if; --endif command = adresse
160:
161:                 elsif state = give_ack_command then
162:                     if command(0) = '0' then
163:                         rw_dir <= '0';
164:                         bit_count <= 0;
165:                         state <= read_byte;
166:                     else
167:                         rw_dir <= '1';
168:                         bit_count <= 1;
169:                         SDA_out <= send_byte_i(7);
170:                         state <= write_byte;
171:                     end if; --endif command(0)
172:
173:                 elsif state = give_ack_byte then
174:                     rw_dir <= '0';
175:                     byte_read_o <= '0';
176:                     state<=idle;
177:                 end if;
178:
179:             elsif state = idle then
180:                 rw_dir <= '0';
181:                 byte_read_o <= '0';
182:
183:         --Ellers gør intet, indtil der er time-out (25 ms)
184:             else
185:                 if time_out = conv_std_logic_vector(3712500, 22) then
186:                     state <= idle;
187:                 end if;
188:             end if;
189:         end if; -- clk_i'event
190:     end process I2C_machine;
191:
192:     --tristate controller for I2C SDA IO
193:     with RW_dir select
194:         SDA_io <= SDA_out when '1',
195:                   'Z' when others;
196:
197:     SDA_in <= SDA_io;
198:
199: end Behavioral;
200:
201:
```

```
  1: ------------------------------------------------------------
  2: --               Digital PLL
  3: ------------------------------------------------------------
  4:
  5: library ieee;
  6: USE ieee.STD_LOGIC_1164.ALL;
  7: USE ieee.std_logic_arith.all;
  8: USE ieee.std_logic_signed.all;
  9:
 10: entity digital_pll is
 11:    generic (
 12:       vcxo_div_per         : integer         -- decimation of clk_i to pll, (495/
               500)
 13:    );
 14:    port (
 15:       clk_vcxo_i           : in std_logic;
 16:       clk_ref_decimated_i  : in std_logic;
 17:       clk_vcxo_dec_tick_o  : out std_logic;  -- tick at falling edge of decimated
               vcxo to phase comparator
 18:       pdm_o                : out std_logic;  -- pulse density modulated signal
 19:       clk_vcxo_decimated_o : out std_logic;  -- for debug
 20:       lock_warning_o       : out std_logic;
 21:       lock_error_o         : out std_logic;
 22:         ext_make_shorter        : in std_logic;
 23:         ext_make_longer     : in std_logic
 24:    );
 25: end digital_pll;
 26:
 27: architecture behavioral of digital_pll is
 28:
 29: -- phase detector:
 30: signal clk_ref_dec_delayed          : std_logic_vector(2 downto 0);
 31: signal clk_ref_dec_dac_en_delayed   : std_logic_vector(2 downto 0);
 32:
 33: signal clk_ref_dec_rising           : std_logic;
 34: signal per_end_load_pos_index       : std_logic_vector(6 downto 0);
 35: signal make_shorter                 : std_logic;
 36: signal make_longer                  : std_logic;
 37: signal phase_count                  : std_logic_vector(9 downto 0);
 38: signal period_end_delayed           : std_logic_vector(3 downto 0);
 39: signal phase_diff                   : std_logic_vector(10 downto 0);
 40: signal phase_diff_limit             : std_logic_vector(10 downto 0);
 41: signal ext_make_shorter_flag           : std_logic;
 42: signal ext_make_longer_flag         : std_logic;
 43: signal ext_make_longer_delayline       : std_logic_vector(1 downto 0);
 44: signal ext_make_shorter_delayline   : std_logic_vector(1 downto 0);
 45: signal clear_flag                          : std_logic;
 46:
 47: -- dac:
 48: signal dac_input                    : std_logic_vector(15 downto 0);
 49: signal dac_enable                   : std_logic;
 50: signal pdm_low                      : std_logic;
 51: signal dac_enable_count             : std_logic_vector(4 downto 0);
 52: signal dac_int_reg                  : std_logic_vector(16 downto 0);
 53:
 54: --loop filter:
 55: signal phase_diff_sign_vector       : std_logic_vector(20 downto 0);
 56: signal phase_diff_extended          : std_logic_vector(31 downto 0);
 57: signal lf_input_sum                 : std_logic_vector(31 downto 0);
 58: signal lf_int_limit                 : std_logic_vector(30 downto 0);
 59: signal lf_int_reg                   : std_logic_vector(30 downto 0);
 60: signal lf_output_sum                : std_logic_vector(16 downto 0);
 61: signal lf_output                    : std_logic_vector(15 downto 0);
 62:
 63: constant zeros                      : std_logic_vector(21 downto 0) := "
         0000000000000000000000";
 64: constant alpha                      : integer := -9;  --[-15;0]
 65: constant beta                       : integer := 5;   --[1;6]
 66:
 67:
 68: begin
 69:
```

```
 70:
 71: ------------------------------------------------------------------------
 72: -- Phase detector :
 73: ------------------------------------------------------------------------
 74: ref_rising_generation : process (clk_vcxo_i)
 75: begin
 76:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
 77:       clk_ref_dec_delayed <= clk_ref_dec_delayed(1 downto 0) & clk_ref_decimated_i;
 78:       if clk_ref_dec_delayed(2 downto 1) = "01" then
 79:          clk_ref_dec_rising <= '1';
 80:       else
 81:          clk_ref_dec_rising <= '0';
 82:       end if;
 83:    end if;
 84: end process;
 85:
 86: per_end_load_pos_index <= period_end_delayed(2 downto 0) & make_shorter &
        make_longer & ext_make_shorter_flag & ext_make_longer_flag;
 87:
 88: vcxo_div_counting : process (clk_vcxo_i)
 89: begin
 90:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
 91:         ext_make_shorter_delayline(1) <= ext_make_shorter_delayline(0);
 92:         ext_make_shorter_delayline(0) <= ext_make_shorter;
 93:
 94:         ext_make_longer_delayline(1) <= ext_make_longer_delayline(0);
 95:         ext_make_longer_delayline(0) <= ext_make_longer;
 96:
 97:         if ext_make_longer_delayline = "01" then
 98:             ext_make_longer_flag <= '1';
 99:         elsif ext_make_shorter_delayline = "01" then
100:             ext_make_shorter_flag <= '1';
101:         elsif clear_flag = '1' then
102:             ext_make_shorter_flag <= '0';
103:             ext_make_longer_flag <= '0';
104:         end if;
105:
106:       case per_end_load_pos_index is
107:             when "0100000" | "0101100" => -- normal count load position, normal
                      period:
108:                 phase_count <= conv_std_logic_vector((vcxo_div_per-1)/2,10);
109:             when "0011000" | "0010010" => -- 1 before normal count load position,
                      shorter period:
110:                 phase_count <= conv_std_logic_vector((vcxo_div_per-1)/2,10);
111:                 clear_flag <= '1';
112:             when "1000100" | "1000001" => -- 1 after normal count load position,
                      longer period:
113:                 phase_count <= conv_std_logic_vector((vcxo_div_per-1)/2,10);
114:                 clear_flag <= '1';
115:             when others =>
116:                 phase_count <= phase_count - 1;
117:                 clear_flag <= '0';
118:       end case;
119:    end if;
120: end process;
121:
122: make_longer_generation : process (clk_vcxo_i)
123: begin
124:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
125:       if period_end_delayed(0) = '1' and clk_ref_dec_rising = '1' then
126:          make_longer <= '1';
127:       elsif period_end_delayed(2) = '1' then
128:          make_longer <= '0';
129:       end if;
130:    end if;
131: end process;
132:
133: make_shorter_generation : process (clk_vcxo_i)
134: begin
135:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
136:       if period_end_delayed(3) = '1' and clk_ref_dec_rising = '1' then
137:          make_shorter <= '1';
```

```
138:       elsif period_end_delayed(2) = '1' then
139:          make_shorter <= '0';
140:       end if;
141:    end if;
142: end process;
143:
144: period_end_delayed_generation : process (phase_count, clk_vcxo_i)
145: begin
146:    if phase_count = conv_std_logic_vector((2-vcxo_div_per)/2,10) then
147:       period_end_delayed(0) <= '1';
148:    else
149:       period_end_delayed(0) <= '0';
150:    end if;
151:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
152:       period_end_delayed(3 downto 1) <= period_end_delayed(2 downto 0);
153:    end if;
154: end process;
155:
156: phase_diff_generation : process (clk_vcxo_i)
157: begin
158:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
159:       if clk_ref_dec_rising = '1' then
160:          phase_diff <= phase_count & '1';
161:       end if;
162:    end if;
163: end process;
164:
165: clk_vcxo_decimated_generation : process (clk_vcxo_i)
166: begin
167:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
168:       clk_vcxo_decimated_o <= phase_count(9);
169:    end if;
170: end process;
171:
172: clk_vcxo_dec_tick_o <= period_end_delayed(1);
173:
174: ------------------------------------------------------------------------
175: -- Loop filter :
176: ------------------------------------------------------------------------
177: phase_diff_sign_vector_generation : process (phase_diff)
178: begin
179:    for i in 0 to 20 loop
180:       phase_diff_sign_vector(i) <= phase_diff(10);
181:    end loop;
182: end process;
183: --      32                          21                  10
184: phase_diff_extended <= phase_diff_sign_vector & phase_diff;       -- phase_diff
         represented by 32 length vector in 2's compliment
185:
186: lf_input_summing : process (lf_int_reg, phase_diff_extended)
187: begin
188:    lf_input_sum(16-alpha downto 0) <= ('0' & lf_int_reg(15-alpha downto 0)) +
             phase_diff_extended(16-alpha downto 0);     -- multiply the input to the
             integrator by 2^alpha using oner more msb to prevent overflow
189: end process;
190:
191: lf_int_limiter : process (lf_input_sum)                            -- limit
         integrator register input so no overflow occurs in integrator:
192: begin
193:    if lf_input_sum(16-alpha downto 15-alpha) = "11" then          -- lower limit:
194:       lf_int_limit(15-alpha downto 0) <= (others => '0');
195:    elsif lf_input_sum(16-alpha downto 15-alpha) = "10" then       -- upper limit:
196:       lf_int_limit(15-alpha downto 0) <= (others => '1');
197:    else                                                           -- normal range:
198:       lf_int_limit(15-alpha downto 0) <= lf_input_sum(15-alpha downto 0);
199:    end if;
200: end process;
201:
202: lf_integrator_reg : process (clk_vcxo_i)
203: begin
204:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
205:       if period_end_delayed(1) = '1' then
```

```
206:          lf_int_reg(15-alpha downto 0) <= lf_int_limit(15-alpha downto 0);
207:       end if;
208:    end if;
209: end process;
210:
211: lf_output_summing : process (lf_int_reg, phase_diff_extended)  -- LF output sum
        using one more msb than the DAC to prevent from overflow
212: begin
213:    --
          beta zeros
214:    lf_output_sum <= ('0' & lf_int_reg(15-alpha downto -alpha)) + (
          phase_diff_extended(16-beta downto 0) & zeros(beta-1 downto 0));
215: end process;
216:
217: lf_output_limiter : process (lf_output_sum)           -- limit the lf_output_sum
      and take out the lsb's to DAC (16)
218: begin
219:    if lf_output_sum(16 downto 15) = "11" then         -- lower limit:
220:       lf_output <= (others => '0');
221:    elsif lf_output_sum(16 downto 15) = "10" then      -- upper limit:
222:       lf_output <= (others => '1');
223:    else
224:       lf_output <= lf_output_sum(15 downto 0);        -- normal range:
225:    end if;
226: end process;
227:
228:
229: ----------------------------------------------------------------------
230: -- Single bit DAC :
231: -- 16-bit resolution 1st order single bit modulator with return to zero pulses
232: ----------------------------------------------------------------------
233: dac_input_register : process (clk_vcxo_i)
234: begin
235:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
236:       if dac_enable = '1' then
237:          clk_ref_dec_dac_en_delayed <= clk_ref_dec_dac_en_delayed(1 downto 0) &
                clk_ref_dec_delayed(2);
238:          if clk_ref_dec_dac_en_delayed(2 downto 1) = "01" then
239:             dac_input <= lf_output;
240:          end if;
241:       end if;
242:    end if;
243: end process;
244:
245: dac_enable_generation : process (clk_vcxo_i)
246: begin
247:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
248:       if dac_enable_count = 0 then
249:          dac_enable_count <= "11111";
250:       else
251:          dac_enable_count <= dac_enable_count - 1;
252:       end if;
253:       if dac_enable_count = "00001" then
254:          dac_enable <= '1';
255:       else
256:          dac_enable <= '0';
257:       end if;
258:       if dac_enable_count = 0 then
259:          pdm_low <= '1';
260:       elsif dac_enable_count = "11000" then
261:          pdm_low <= '0';
262:       end if;
263:    end if;
264: end process;
265:
266: dac_int_reg_generation : process (clk_vcxo_i)
267: begin
268:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
269:       if dac_enable = '1' then
270:          dac_int_reg <= dac_int_reg + (not dac_int_reg(16) & dac_input);
271:       end if;
272:    end if;
```

```
273: end process;
274:
275: pdm_generation : process (clk_vcxo_i)
276: begin
277:    if clk_vcxo_i'event and clk_vcxo_i = '1' then
278:       if pdm_low = '0' and dac_int_reg(16) = '0' then
279:          pdm_o <= '1';
280:       else
281:          pdm_o <= '0';
282:       end if;
283:    end if;
284: end process;
285:
286: ------------------------------------------------------------------------
287: -- PLL status:
288: ------------------------------------------------------------------------
289: lock_warning_generation : process (phase_diff)
290: begin
291:    if phase_diff > "1111111000" and phase_diff < "0000000111" then
292:       lock_warning_o <= '0';
293:    else
294:       lock_warning_o <= '1';
295:    end if;
296: end process;
297:
298: lock_error_generation : process (make_longer, make_shorter)
299: begin
300:    if make_shorter = '1' or make_longer = '1' then
301:       lock_error_o <= '1';
302:    else
303:       lock_error_o <= '0';
304:    end if;
305: end process;
306:
307:
308: end behavioral;
```

```
#inputs
NET "clk_p_i"           LOC = "p83";
NET "clk_n_i"           LOC = "p84";
NET "scl_i"         LOC = "p40";
#NET "reset_i"      LOC = "p22";
#NET "dip0_i"           LOC = "p53";
#NET "dip1_i"           LOC = "p54";
#NET "dip2_i"           LOC = "p57";
#NET "dip3_i"           LOC = "p58";
#NET "freq_i"           LOC = "p24";
#NET "led_red_i"        LOC = "p3";
#NET "led_green_i"  LOC = "p2";
NET "pps_i"         LOC = "p18";

#outputs
#NET "ns_d_o"           LOC = "p61";
#NET "h_d_o"            LOC = "p60";
#NET "f_d_o"            LOC = "p63";
NET "VCXO_ctrl_o"   LOC = "p62";
#NET "event_o"          LOC = "p27";
#NET "LTC_code_o"       LOC = "p26";
NET "led0_o"            LOC = "p12";
NET "led1_o"            LOC = "p11";

#in/outputs
NET "MCU_gpio0_io"  LOC = "p65";
NET "MCU_gpio1_io"  LOC = "p68";
NET "MCU_gpio2_io"  LOC = "p67";
NET "MCU_gpio3_io"  LOC = "p66";
NET "gpio0_io"      LOC = "p33";
#NET "gpio1_io"     LOC = "p32";
#NET "gpio2_io"     LOC = "p36";
#NET "gpio3_io"     LOC = "p35";
NET "sda_io"            LOC = "p41";
```

C kildekode

GPS main module

In/ud definering (header fil)

GPS modul initialisering

GPS modul initialisering (header fil)

NavSync protokol parser

NavSync protokol parser (header fil)

UART

UART (header fil)

I2C software bus

I2C software bus (header fil)

I2C hardware bus, slave

I2C hardware bus, slave (header fil)

```
#include <C8051F320.h>
#include <string.h>
#include "inouts.h"
#include "UART.h"
#include "GPS_init.h"
#include "NavSync_prot.h"
#include "i2c_bus.h"
#include "i2c_slave.h"

char reset_flag=1;
char rise_flag=0;
char fall_flag=0;

unsigned char hours;
unsigned char mins;
unsigned char secs;

unsigned long GPS_week;
unsigned long GPS_tow;
unsigned long clock_offset;
unsigned long temp;
unsigned char send_byte;

sbit PLL_lock_flag_i = P2^3;
sbit GPS_lock_flag_o = P2^4;
sbit FPGA_reset_o = P2^5;

void int_fall() interrupt 0
{
    fall_flag=1;
}

void int_rise() interrupt 2
{
    rise_flag=1;
}

char in_byte=0;
unsigned long xdata buffer[20];
unsigned char index=0;

void main()
{
    //disable watchdog timer
    PCA0MD &= ~0x40;

    //Setup sekvens
    setup_ports();
    setup_osc();
    setup_timer1();
    setup_timer3();
    setup_I2C();
    setup_UART();

    FPGA_reset_o=1;

    EA=1;
    ES0=0;
    //send kun polyt-beskeder
    setup_msg("$PRTHS,U1OP,NMEA,ALL=0,PLT=1,GLL=1");
    //dynamic = stationary, but unknown position
    setup_msg("$PRTHS,DYNA,1");
    //output 250 KHz
    setup_msg("$PRTHS,FRQD,10");
    ES0=1;

    FPGA_reset_o=0;
    memset(buffer, 0, 256);

    //main løkke
    while(1)
    {
```

```
 73:         //output GPS lås
 74:         LED1_o = !gps_valid;
 75:
 76:         //hvis hel NMEA besked indlæst, opdater tid/GPS status
 77:         if(update_flag)
 78:         {
 79:             get_info();
 80:             update_flag=0;
 81:         }
 82:
 83:         if(reset_flag==1 && PLL_lock_flag_i==1 && gps_valid==1)
 84:         {
 85:             GPS_week = timing_info[5];
 86:             GPS_tow = timing_info[6];
 87:
 88:             clock_offset = GPS_week%143;
 89:             clock_offset = clock_offset*604800+GPS_tow+1;
 90:             clock_offset = clock_offset%1001;
 91:             clock_offset = 148500000 - ((clock_offset*4450549)%148500000);
 92:
 93:             //send til FPGA (0xBB = modtage offset)
 94:             i2c_start();
 95:             i2c_sendbyte(0xA0);
 96:             i2c_giveack();
 97:             i2c_sendbyte(0xBB);
 98:             i2c_giveack();
 99:             i2c_stop();
100:
101:             send_byte = (unsigned char) clock_offset&0xff;
102:             clock_offset = clock_offset << 8;
103:
104:             //send MSB byte
105:             i2c_start();
106:             i2c_sendbyte(0xA0);
107:             i2c_giveack();
108:             i2c_sendbyte(send_byte);
109:             i2c_giveack();
110:             i2c_stop();
111:
112:             send_byte = (unsigned char) clock_offset&0xff;
113:             clock_offset = clock_offset << 8;
114:
115:             //2. byte
116:             i2c_start();
117:             i2c_sendbyte(send_byte);
118:             i2c_giveack();
119:             i2c_sendbyte(0xAA);
120:             i2c_giveack();
121:             i2c_stop();
122:
123:             send_byte = (unsigned char) clock_offset&0xff;
124:             clock_offset = clock_offset << 8;
125:
126:             //3. byte
127:             i2c_start();
128:             i2c_sendbyte(0xA0);
129:             i2c_giveack();
130:             i2c_sendbyte(send_byte);
131:             i2c_giveack();
132:             i2c_stop();
133:
134:             send_byte = (unsigned char) clock_offset&0xff;
135:             clock_offset = clock_offset << 8;
136:
137:             //LSB byte
138:             i2c_start();
139:             i2c_sendbyte(0xA0);
140:             i2c_giveack();
141:             i2c_sendbyte(send_byte);
142:             i2c_giveack();
143:             i2c_stop();
144:
```

```
145:             reset_flag = 0;
146:         }
147:
148:         //opdater tid til LTC i FPGA
149:         if(rise_flag==1 && reset_flag!=1)
150:         {
151:             GPS_lock_flag_o=gps_valid;
152:             LED0_o=1;
153:             temp=timing_info[0];
154:
155:             secs=temp%100; temp=temp/100;
156:             mins=temp%100; temp=temp/100;
157:             hours=temp%100;
158:
159:             i2c_start();
160:             i2c_sendbyte(0xA0);
161:             i2c_giveack();
162:             i2c_sendbyte(0xAA);
163:             i2c_giveack();
164:             i2c_stop();
165:
166:             i2c_start();
167:             i2c_sendbyte(0xA0);
168:             i2c_giveack();
169:             i2c_sendbyte(hours);
170:             i2c_giveack();
171:             i2c_stop();
172:
173:             i2c_start();
174:             i2c_sendbyte(0xA0);
175:             i2c_giveack();
176:             i2c_sendbyte(mins);
177:             i2c_giveack();
178:             i2c_stop();
179:
180:             i2c_start();
181:             i2c_sendbyte(0xA0);
182:             i2c_giveack();
183:             i2c_sendbyte(secs);
184:             i2c_giveack();
185:             i2c_stop();
186:
187:             rise_flag=0;
188:             LED0_o=0;
189:         }
190:
191:         //læs output fra FPGA
192:         /*  i2c_start();
193:             i2c_sendbyte(0xA1);
194:             i2c_giveack();
195:             in_byte=i2c_readbyte();
196:             buffer[index]=in_byte;
197:             index++;
198:             i2c_giveack();
199:             i2c_stop();*/
200:     }
201: }
```

```
#include <c8051f320.h>

//defination af inputs og outputs

//I2C til mainframe
sbit sda_io = P1^0;
sbit scl_io = P1^1;

//intern I2C til FPGA
sbit sda_int_io = P1^3;
sbit scl_int_io = P1^2;

//Debug LEDs
sbit LED0_o = P2^0;
sbit LED1_o = P2^1;

//DIP
sbit DIP_i = P2^6;

//General purpose IOs
sbit gpio0_io = P2^2;
sbit gpio1_io = P2^3;
sbit gpio2_io = P2^4;
sbit gpio3_io = P2^5;

```

```c
#include <C8051F320.h>
//#include <string.h>

void setup_ports()
{
    IT01CF=0x91;
    EX1=1;
    EX0=1;

    XBR0 = 0x05;    // Crossbar Register 1 (enable UART + SMBus)
    XBR1 = 0x40;    // Crossbar Register 2 (weak pull-ups globalt disabled)

    //sæt ubrugte inputs til analog = formindsk strømforbrug+støj
    P0MDIN = 0x33;  // Input configuration for P0 (ext xtal)
    P1MDIN = 0x0F;  // Input configuration for P1 (0-3 = digitale (I2C))
    P2MDIN = 0x7F;  // Input configuration for P2 (0-6 = digitale (gpio, LEDS, dip)
        )
    P3MDIN = 0x00;  // Input configuration for P3 (alle analoge)

    //sæt I2C=open-drain, resten push-pull
    P0MDOUT = 0xC3; // Output configuration for P0
    P1MDOUT = 0xF0; // Output configuration for P1 (0-3 I2C = opendrain)
    P2MDOUT = 0xFF; // Output configuration for P2
    P3MDOUT = 0xFF; // Output configuration for P3

    //brug ekstern oscillator, SDA og SCL på P1.0 og P1.1
    P0SKIP = 0xCF;  // Port 0 Crossbar Skip Register (skip ext xtal)
    P1SKIP = 0x00;  // Port 1 Crossbar Skip Register
    P2SKIP = 0x00;  // Port 2 Crossbar Skip Register
}

void setup_timer1()
{
    CKCON = 0x08;     // Timer1 bruger SYSCLK
    TL1   = 0x00;     // Timer 1 Low Byte
    TH1   = 0x70;     // BaudRate = 38400
    TMOD  = 0x21;     // Timer Mode Register
    TCON  = 0x45;     // Timer Control Register, timer1 enabled
    TR1 = 1;          // Start timer1
}

void setup_timer3()
{
    TMR3RLH=0xA6; TMR3RLL=0x00; //Timer3 reload => 25 ms overflow
    TMR3H=TMR3RLH; TMR3L=TMR3RLL;
    TMR3CN=0x04;                //16 bit autoreload, sysclk/12, timer enabled
    EIE1|=0x80;                 //enable timer3 interrupt
}

void setup_osc()
{
    int n=0;
    //Oscillator setup
    OSCXCN = 0x67;          // EXTERNAL Oscillator Control Register >10 MHz

    // vent til krystal er stabiliseret
    while ( (OSCXCN & 0x80) == 0 )  ;

    CLKSEL = 0x01;          // = Ekstern Oscillator
}

```

```
1: void setup_ports();
2: void setup_osc();
3: void setup_timer1();
4: void setup_timer3();
```

```c
#include <string.h>
#include <stdlib.h>
#include "inouts.h"
#include "NavSync_prot.h"
#include "UART.h"

//struct time_info GPS_time;
long xdata timing_info[16];
char xdata in_string[115];
char gps_valid;

void get_info()
{
    char index, offset, string_count=0;
    long temp_val=0;
    char temp_string[8]="00000000";
    char set[3]={'.',',',0x0D};

    //hent buffer fra UART
    read_buf(&in_string);

    //hvis headeren er POLYT (returnere 0 hvis sandt!)
    if(strncmp(in_string, "POLYT", 5)==0)
    {
        //index efter "$polyt,"
        index=6; string_count=0;

        while (string_count!=15 && index<100)
        {
            //reset midlertidig streng
            memset(temp_string, '\0', sizeof(temp_string));
            //søg efter næste komma, punktum eller end-line
            offset = strcspn(&in_string[index], set);
            //kopier fra index og længden af næste streng
            memcpy(temp_string, &in_string[index], offset);
            //opdater index
            index=index+offset+1;

            timing_info[string_count++]=atol(temp_string);
        }
    }

    //hvis headeren er GPGLL
    if (strncmp(in_string, "GPGLL", 5)==0)
    {
        //'A' på plads 46 i strengen betyder valid GPS lock
        if(in_string[46]=='A')
            gps_valid=1;
        else
            gps_valid=0;
    }
}

void setup_msg(char *str)
{
    int n=0;

    for(n; n<strlen(str); n++)
        putch(str[n]);

    putch(0x0D);
    putch(0x0A);
}


```

```
1: extern long xdata timing_info[16];
2: extern char gps_valid;
3:
4: void get_info();
5: void setup_msg(char *str);
```

```
#include <C8051F320.h>
#include <string.h>
#include "inouts.h"
#include "NavSync_prot.h"
#include "UART.h"

#define RX_SERBUFLEN  115

unsigned char idata rx_serbuf[RX_SERBUFLEN];
unsigned char rx_inptr;
char update_flag;


void setup_UART()
{
    //UART setup
    SCON0 = 0x30;        // 8 bit med interurrupt
    PCON = 0x00;         // ikke i idle eller stop-mode

    RI0 = 0;         //clear receive interrupt bit
    TI0 = 0;         //clear transmit interrupt bit

    rx_inptr  = 0;
    ES0=1;           //Enable serial interrupt
}

void putch(unsigned char c)
{
  SBUF0 = c;
  while (!TI0)
    ;
  TI0   = 0;
}

void serint0(void) interrupt 4 using 1
{
  if (RI0)
  {
    //'$' = start på en NMEA besked
    if(SBUF0=='$')
        rx_inptr=0;

    //'*' = sidste karakter før checksum
    else if(SBUF0=='*')
        update_flag=1;

    else
    {
        rx_serbuf[rx_inptr] = SBUF0;
        rx_inptr++;

        if(rx_inptr>RX_SERBUFLEN)
            rx_inptr=0;
    }

    RI0 = 0;
  }
}

void read_buf(char *out_buffer)
{
    memcpy(out_buffer, rx_serbuf, RX_SERBUFLEN);
}
```

```
1: extern char update_flag;
2:
3: void setup_UART();
4: void putch(unsigned char c);
5: void serint0(void);
6: void read_buf(char * out_buffer);
```

```
#include <C8051F320.h>
#include "inouts.h"
#include "i2c_bus.h"

void tmr1_isr() interrupt 3
{
}

//short delay
void sdelay()
{
    TCON &= 0xCF;   //stop timer, clear OF flag
    TL0 = 0x00;     //0xE8
    TH0 = 0xFF;
    TCON |= 0x10;   //start timer
    while(! (TCON&0x20))
    ;
}

void i2c_start()
{
    scl_int_io = 1;
    sdelay();
    sda_int_io = 0;
    sdelay();
    scl_int_io = 0;
    sdelay();
    sda_int_io = 1;
    sdelay();
}

void i2c_stop()
{
    sda_int_io = 0;
    scl_int_io = 1;
    sdelay();
    sda_int_io = 1;
    sdelay();
}

void i2c_sendbyte(char byte)
{
    char count=0;

    for(count; count<8; count++)
    {
        sda_int_io=byte&128;
        sdelay();
        scl_int_io=1;
        sdelay();
        scl_int_io=0;
        byte= byte << 1;
        sdelay();
    }

    //end transmission
    sda_int_io=1;
    //scl=1;

    sdelay();
}

unsigned char i2c_readbyte()
{
    unsigned char inbyte=0;
    char count=0;

    for(count; count<8; count++)
    {
        inbyte=inbyte<<1;
        scl_int_io=1;
        sdelay();
```

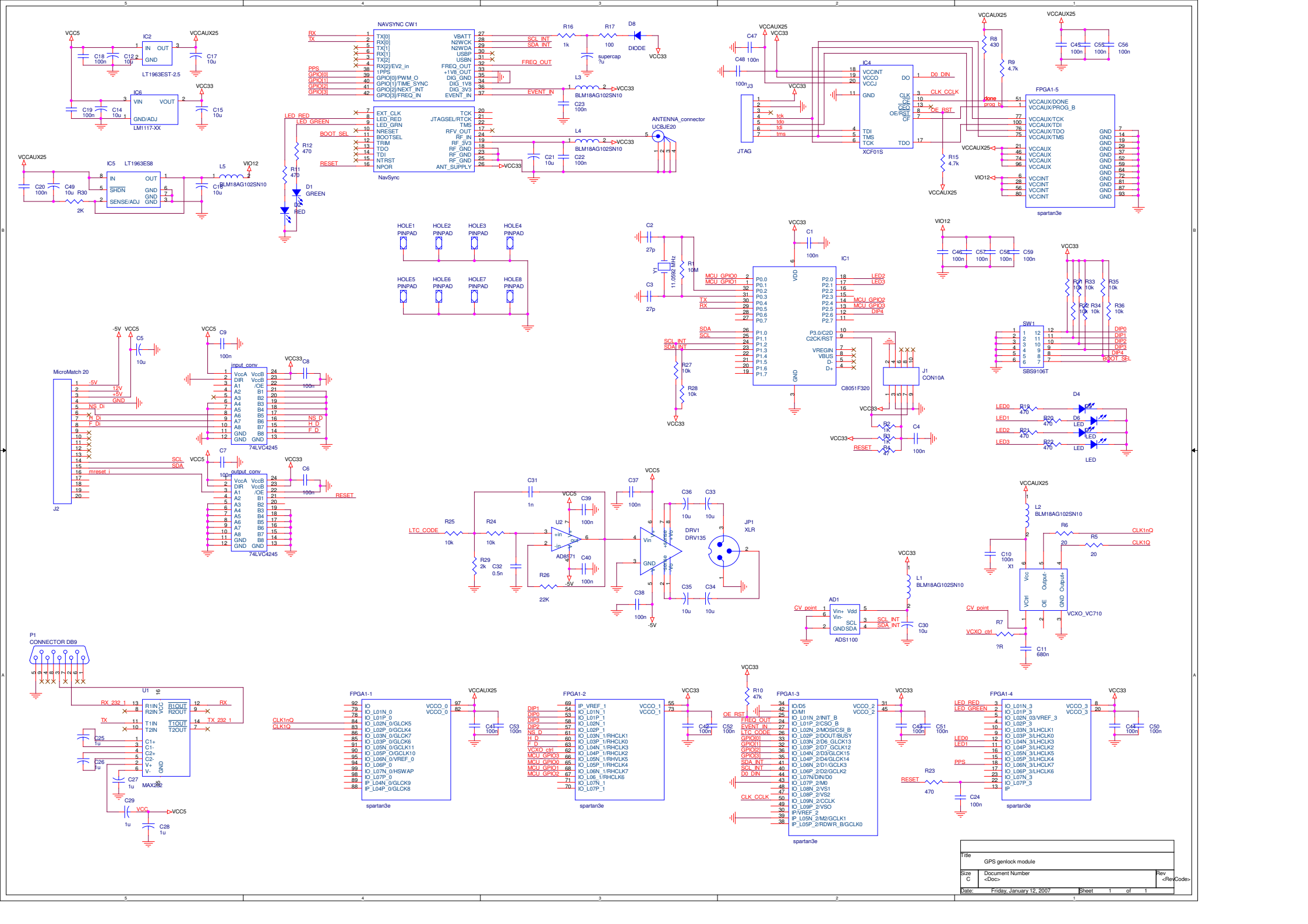
```
 73:         inbyte|=sda_int_io;
 74:         scl_int_io=0;
 75:         sdelay();
 76:     }
 77:
 78:     sda_int_io=1;
 79:     //scl=1;
 80:
 81:     sdelay();
 82:
 83:     return inbyte;
 84: }
 85:
 86: char i2c_getack()
 87: {
 88:     //get acknowledge
 89:     unsigned char count=0;
 90:     sda_int_io=1;
 91:     //scl=1;
 92:     while(sda_int_io)   //while sda != low
 93:     {
 94:         count++;
 95:         if(count==255)  //if timeout
 96:             return 0;
 97:     }
 98:
 99:     scl_int_io=1;
100:     sdelay();
101:     scl_int_io=0;
102:     sdelay();
103:
104:     return 1;
105: }
106:
107: void i2c_giveack()
108: {
109:     scl_int_io=0;
110:     sda_int_io=0;
111:     sdelay();
112:     scl_int_io=1;
113:     sdelay();
114:     scl_int_io=0;
115:     sdelay();
116: }
117:
```

```
1: void delay();
2: void sdelay();
3: void i2c_start();
4: void i2c_stop();
5: void i2c_sendbyte(char byte);
6: unsigned char i2c_readbyte();
7: char i2c_getack();
8: void i2c_giveack();
```

```c
#include<c8051f320.h>

sbit LED1 = P2^0;

#define WRITE 0x00          // SMBus WRITE command
#define READ 0x01           // SMBus READ command
#define SLAVE_ADDR 0xA0     // Device addresses (7 bits)
#define SMB_SRADD 0x20      // (SR) slave address received
                            // (also could be a lost
                            // arbitration)
#define SMB_SRSTO 0x10      // (SR) STOP detected while SR or ST,
                            // or lost arbitration
#define SMB_SRDB 0x00       // (SR) data byte received, or
                            // lost arbitration
#define SMB_STDB 0x40       // (ST) data byte transmitted
#define SMB_STSTO 0x50      // (ST) STOP detected during a
                            // transaction; bus error
                            // End status vector definition

//-----------------------------------------------------------------------------
// Global VARIABLES
//-----------------------------------------------------------------------------
unsigned char SMB_DATA; // Global holder for SMBus data.

bit DATA_READY = 0; // Set to '1' by the SMBus ISR

//interrupt ved overflow af lav SCL
void timer3_ISR() interrupt 14
{
    SMB0CF &= ~0x80; // Disable SMBus
    SMB0CF |= 0x80;  // Re-enable SMBus
    TMR3CN &= ~0x80; // Clear Timer3 interrupt-pending flag
}

void SMBus_ISR() interrupt 7
{
    if (ARBLOST == 0)
    {
        switch (SMB0CN & 0xF0) // Decode the SMBus status vector
        {
            // Slave Receiver: Start+Address received
            case SMB_SRADD:
                STA = 0; // Clear STA bit
                if((SMB0DAT&0xFE) == (SLAVE_ADDR&0xFE)) // Decode address
                {                                       // If the received address
                    matches,
                    ACK = 1;                            // ACK the received slave
                        address
                    if((SMB0DAT&0x01) == READ)          // If the transfer is a
                        master READ,
                        SMB0DAT = SMB_DATA;             // Prepare outgoing byte
                }
                else                    // If received slave address does not match
                    ,
                    ACK = 0;            // NACK received address
            break;

            // Slave Receiver: Data received
            case SMB_SRDB:
                SMB_DATA = SMB0DAT;     // Store incoming data
                DATA_READY = 1;         // Indicate new data received
                ACK = 1;                // ACK received data
            break;

            // Slave Receiver: Stop received while either a Slave Receiver or
            // Slave Transmitter
            case SMB_SRSTO:
                STO = 0;                // STO must be cleared by software when
                                        // a STOP is detected as a slave
            break;

            // Slave Transmitter: Data byte transmitted
```

```
 69:             case SMB_STDB:
 70:             // No action required;
 71:             // one-byte transfers
 72:             // only for this example
 73:             break;
 74:             // Slave Transmitter: Arbitration lost, Stop detected.
 75:             // This state will only be entered on a bus error condition.
 76:             // In normal operation, the slave is no longer sending data or has
 77:             // data pending when a STOP is received from the master, so the TXMODE
 78:             // bit is cleared and the slave goes to the SRSTO state.
 79:             case SMB_STSTO:
 80:                 STO = 0;        // STO must be cleared by software when
 81:                                 // a STOP is detected as a slave
 82:
 83:             break;
 84:             // Default: all other cases undefined
 85:             default:
 86:                 SMB0CF &= ~0x80; // Reset communication
 87:                 SMB0CF |= 0x80;
 88:                 STA = 0;
 89:                 STO = 0;
 90:                 ACK = 0;
 91:             break;
 92:         }
 93:     }
 94:
 95:     // ARBLOST = 1, Abort failed transfer
 96:     else
 97:     {
 98:         STA = 0;
 99:         STO = 0;
100:         ACK = 0;
101:     }
102:     SI = 0; // Clear SMBus interrupt flag
103: }
104:
105: void setup_I2C()
106: {
107:     //I2C setup
108:     SMB0CF|=0x89;               //SMBus enabled, Slave mode, SCL low timeout, Bus
         free detect, timer 1 bitrate
109:     EIE1|=0x01;                 //enable SMBus interrupt
110: }
```

```
1: void timer3_ISR();
2: void SMBus_ISR();
3: void setup_I2C();
4:
```


NAVSYNC CW1
NavSync
IC1
IC2
LT1963EST-2.5
IC4
XCF01S
JTAG
FPGA1-5
spartan3e
IC5
LT1963ES8
IC6
LM1117-XX
ANTENNA_connector
UCBJE20
MicroMatch 20
input_conv
output_conv
74LVC4245
SBS9106T
C8051F320
CON10A
P1
CONNECTOR DB9
MAX232
AD8571
DRV135
ADS1100
VCXO_VC710
FPGA1-1
FPGA1-2
FPGA1-3
FPGA1-4
GPS genlock module

Udvalgte SMPTE LTC tidskodesider

8.2.5 Biphase mark polarity correction

This flag bit is specific to the LTC modulation method described in 8.3. The position of this flag is listed in table 3.

Because of the nature of the modulation method, the polarity of the first clock transition of the first bit of the synchronization word may differ from code word to code word depending on the number of logical zeros in the data.

Applications which switch between two sources of time and control code require the polarity of the two sources to be stable during the synchronization word. In order to stabilize the polarity of the sync word, the biphase mark polarity correction bit shall be put in a state so that every 80-bit word will contain an even number of logical zeros. This requirement is summarized as follows:

If polarity correction of the code word is desired and the number of logical “zeros” in bit positions 0 through 63 (exclusive of the polarity correction bit itself) is odd, then the polarity correction bit shall be set to “one,” else the polarity correction bit shall be set to “zero.”

8.3 Modulation method

The NRZ unmodulated signal is biphase mark encoded according to the following coding rules (see figure 1):

1) A transition occurs at each bit cell boundary, regardless of the value of the bit.

2) A logic one is represented by an additional transition occurring at the bit cell midpoint.

3) A logic zero is represented by having no additional transitions within the bit cell.

The biphase mark encoded signal has no dc component, is amplitude and polarity insensitive, and includes transitions at every bit cell boundary from which the clock may be extracted.

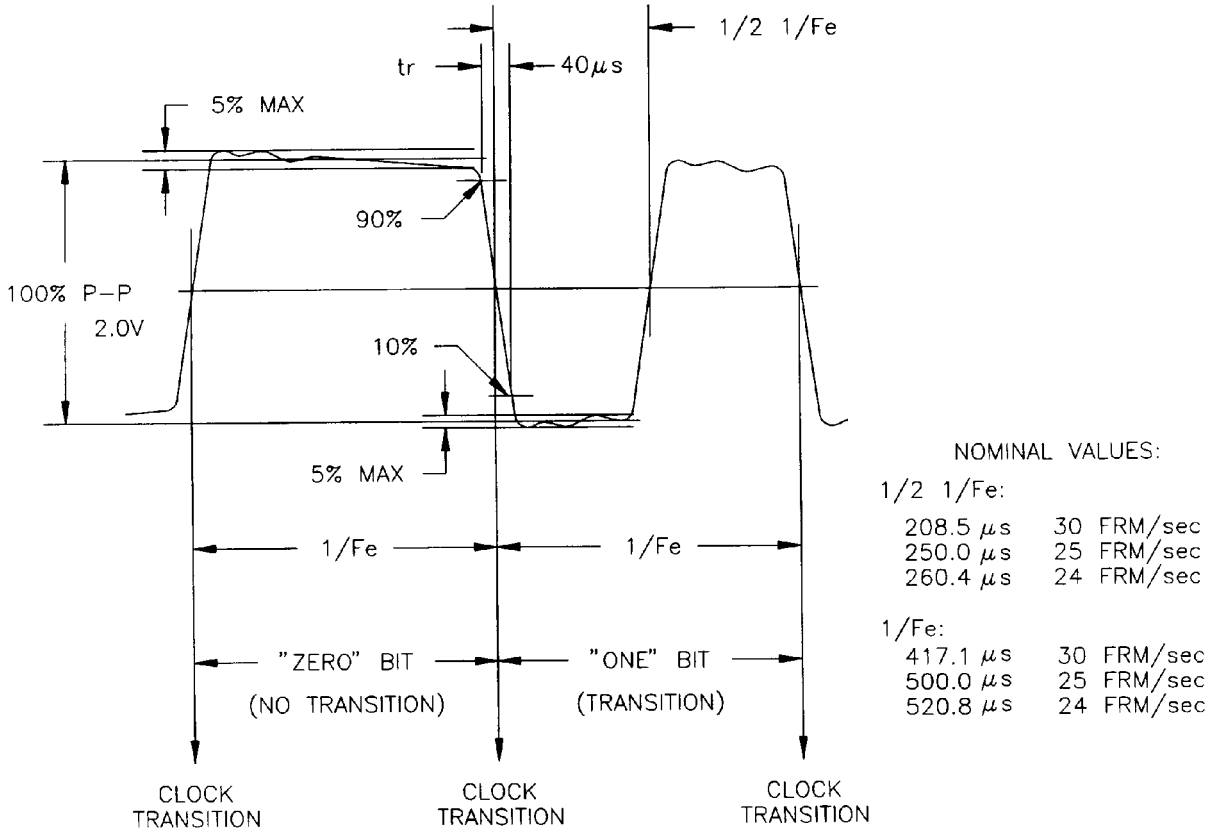


Figure 1 – Linear time code source output waveform

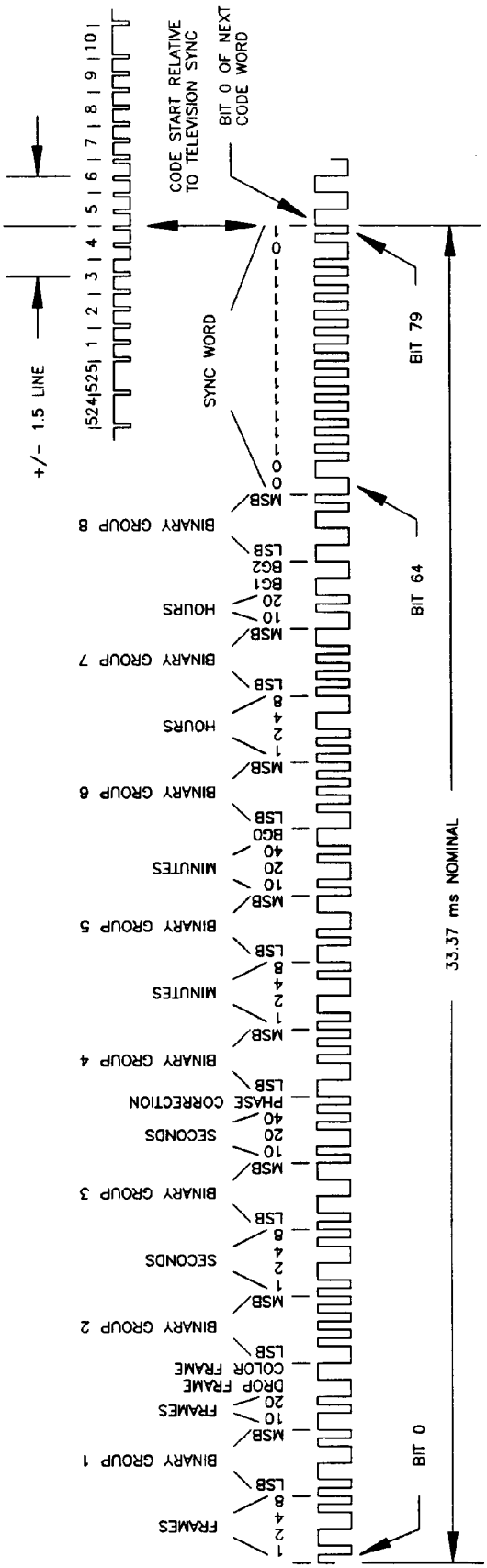


Figure 2a - 30-frame linear time code example

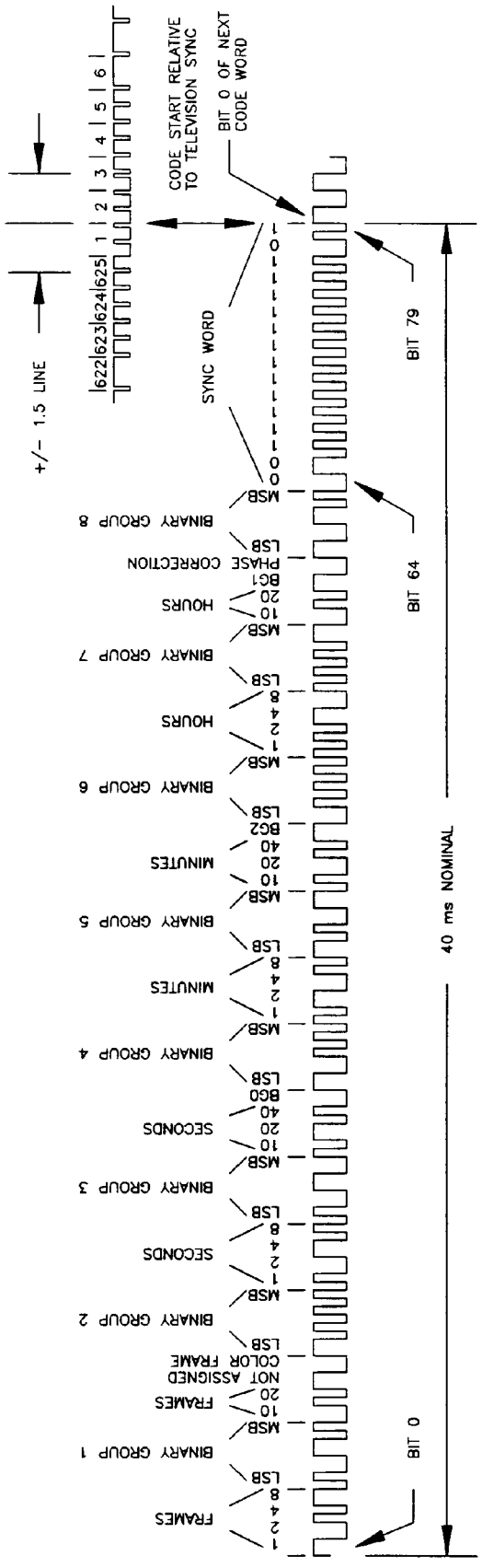


Figure 2c - -25-frame linear time code example

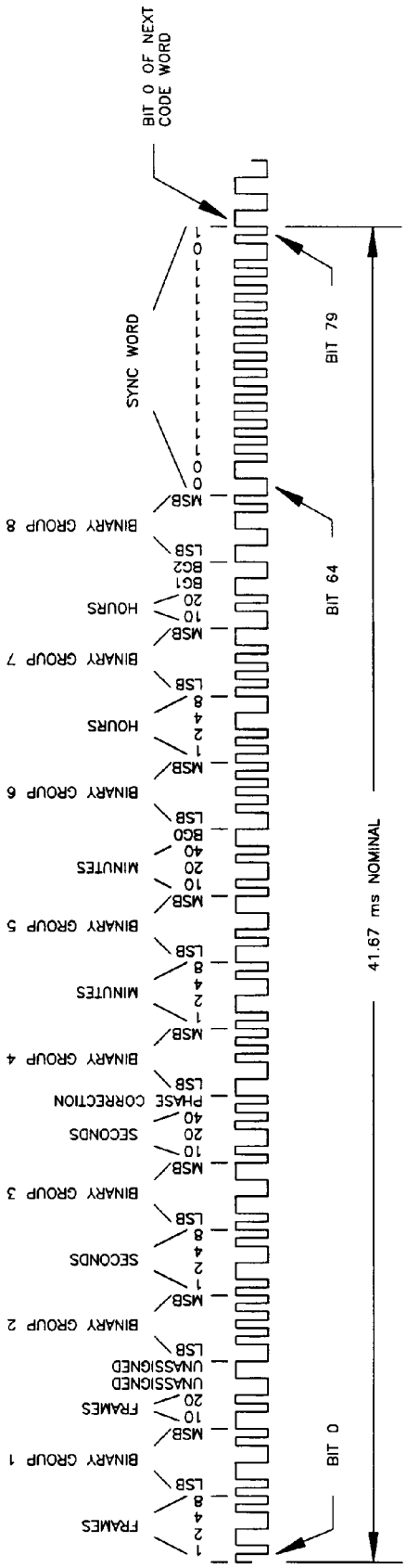


Figure 2d - -24-frame linear time code example

Table 11 – Summation of VITC and LTC code word bit definitions

VITC BIT NO.	*VALUE* (WEIGHT)	COMMON ASSIGNMENT	60–FIELD TELEVISION	50–FIELD TELEVISION	24–FRAME FILM	LTC BIT NO.
0	*1*	VITC SYNC BIT				
1	*0*	VITC SYNC BIT				
2	(1)	TV FRAME UNITS				0
3	(2)	TV FRAME UNITS				1
4	(4)	TV FRAME UNITS				2
5	(8)	TV FRAME UNITS				3
6	(LSB)	FIRST BINARY GROUP				4
7	|	FIRST BINARY GROUP				5
8	|	FIRST BINARY GROUP				6
9	(MSB)	FIRST BINARY GROUP				7
10	*1*	VITC SYNC BIT				
11	*0*	VITC SYNC BIT				
12	(1)	TV FRAME TENS				8
13	(2)	TV FRAME TENS				9
14	FLAG	FLAG	DROP FRAME FLAG	UNUSED BIT	UNUSED BIT	10
15	FLAG	FLAG	COLOR FRAME FLAG	COLOR FRAME FLAG	UNUSED BIT	11
16	(LSB)	SECOND BINARY GROUP				12
17	|	SECOND BINARY GROUP				13
18	|	SECOND BINARY GROUP				14
19	(MSB)	SECOND BINARY GROUP				15
20	*1*	VITC SYNC BIT				
21	*0*	VITC SYNC BIT				
22	(1)	TV SECONDS UNITS				16
23	(2)	TV SECONDS UNITS				17
24	(4)	TV SECONDS UNITS				18
25	(8)	TV SECONDS UNITS				19
26	(LSB)	THIRD BINARY GROUP				20
27	|	THIRD BINARY GROUP				21
28	|	THIRD BINARY GROUP				22
29	(MSB)	THIRD BINARY GROUP				23
30	*1*	VITC SYNC BIT				
31	*0*	VITC SYNC BIT				
32	(1)	TV SECONDS TENS				24
33	(2)	TV SECONDS TENS				25
34	(4)	TV SECONDS TENS				26
35	FLAG	FLAG	FIELD/PHASE	BINARY GROUP FLAG 0	PHASE	27
36	(LSB)	FOURTH BINARY GROUP				28
37	|	FOURTH BINARY GROUP				29
38	|	FOURTH BINARY GROUP				30
39	(MSB)	FOURTH BINARY GROUP				31
40	*1*	VITC SYNC BIT				
41	*0*	VITC SYNC BIT				
42	(1)	TV MINUTES UNITS				32
43	(2)	TV MINUTES UNITS				33
44	(4)	TV MINUTES UNITS				34
45	(8)	TV MINUTES UNITS				35
46	(LSB)	FIFTH BINARY GROUP				36
47	|	FIFTH BINARY GROUP				37
48	|	FIFTH BINARY GROUP				38
49	(MSB)	FIFTH BINARY GROUP				39
50	*1*	VITC SYNC BIT				
51	*0*	VITC SYNC BIT				
52	(1)	TV MINUTES TENS				40
53	(2)	TV MINUTES TENS				41
54	(4)	TV MINUTES TENS				42
55	FLAG	FLAG	BINARY GROUP FLAG 0	BINARY GROUP FLAG 2	BINARY GROUP FLAG 0	43
56	(LSB)	SIXTH BINARY GROUP				44
57	|	SIXTH BINARY GROUP				45
58	|	SIXTH BINARY GROUP				46
59	(MSB)	SIXTH BINARY GROUP				47
60	*1*	VITC SYNC BIT				
61	*0*	VITC SYNC BIT				
62	(1)	TV HOURS UNITS				48
63	(2)	TV HOURS UNITS				49
64	(4)	TV HOURS UNITS				50
65	(8)	TV HOURS UNITS				51
66	(LSB)	SEVENTH BINARY GROUP				52
67	|	SEVENTH BINARY GROUP				53
68	|	SEVENTH BINARY GROUP				54
69	(MSB)	SEVENTH BINARY GROUP				55
70	*1*	VITC SYNC BIT				
71	*0*	VITC SYNC BIT				
72	(1)	TV HOURS TENS				56
73	(2)	TV HOURS TENS				57
74	FLAG	FLAG	BINARY GROUP FLAG 1	BINARY GROUP FLAG 1	BINARY GROUP FLAG 1	58
75	FLAG	FLAG	BINARY GROUP FLAG 2	FIELD/PHASE	BINARY GROUP FLAG 2	59
76	(LSB)	EIGTH BINARY GROUP				60
77	|	EIGTH BINARY GROUP				61
78	|	EIGTH BINARY GROUP				62
79	(MSB)	EIGTH BINARY GROUP				63
80	*1*	VITC SYNC BIT				
81	*0*	VITC SYNC BIT				
82–99		VITC CRC CODE				
		LTC SYNC WORD				64–79

Udvalgte NavSync databladsider

Chipset specifikationer

Fysisk opbygning/pinouts

POLYT besked data

POLYS besked data

2. SPECIFICATION

2.1 Performance

CW25 GPS RECEIVER SPECIFICATIONS[1]

Physical	Module dimensions	25mm (D) x 27mm (W) x 4.2mm (H)
	Supply voltages	3V3 (Digital I/O), 3V3 (RF), 1V8 (Core option), 3V (Standby Battery)
	Operating Temp	-40°C to +75°C
	Storage Temp	-55°C to +125°C
	Humidity	5% to 95% non-condensing
	Max Velocity / Altitude	515ms^{-1} / 18,000m (increased rating version available subject to export license)
	Max Acceleration / Jerk	4g / 1gs^{-1} (sustained for less than 5 seconds)
Sensitivity	Acquisition/Tracking	-185dBW / -186dBW
Acquisition Time	Hot Start with network assist	Outdoor: <2s Indoor (-178dBW): <5s
	Stand Alone (Outdoor)	Cold: <45s Warm: <38s Hot: <5s Re-acquisition: <1s (90%confidence)
Accuracy	Position: Outdoor / Indoor	<5m rms / <50m rms
	Velocity	<0.05ms^{-1}
	Latency	<200ms
	Raw Measurement Accuracy	Pseudorange <0.3m rms, Carrier phase <5mm rms
	Tracking	Code and carrier coherent
Power	1 fix per second	0.25W typically
	Coma Mode Current	<10mA
Interfaces	Serial	3 ports, CMOS levels; USB v1.1
	Multi function I/O	1PPS and Frequency Output

		Event Counter/Timer Input
		4 x GPIO (multi-function)
		2 x LED Status Drive
		I^2C, External Clock (on special build)
	Protocols	Network Assist, NMEA 0183,Proprietary ASCII and binary message formats
	1pps Timing Output	30ns rms accuracy, <5ns resolution
		User selectable pulse width
	Event Input	30ns rms accuracy, <10ns resolution
	Frequency Output	10Hz to 10MHz
	Receiver Type	12 parallel channel x 32 taps up to 32 point FFT.
General	Processor	ARM 966E-S on a 0.18µ process at up to 120MHz.

Table 1 CW25 Specification

Footnotes

1. The features listed above may require specific software builds and may not all be available in the initial release.

2.2 Recommended Ratings

Symbol	Parameter	Min	Max	Units
RF_3V3	RF Supply Voltage	+3.0	+3.6	Volts
DIG_3V3	Digital Supply Voltage	+3.0	+3.6	Volts
DIG_1V8	Digital Supply Voltage	+1.65	+1.95	Volts
VBATT	Battery Backup Voltage	+2.7	+3.5	Volts
ANT_SUPPLY	Antenna Supply Voltage	+3.0	+12	Volts

Table 2 Absolute Maximum Ratings

2.3 Absolute Maximum Ratings

Symbol	Parameter	Min	Max	Units
RF_3V3	RF Supply Voltage	-0.3	+6.5	Volts
DIG_1V8	Digital Supply Voltage	-0.3	+2.0	Volts
DIG_3V3	Digital Supply Voltage	-0.3	+3.7	Volts
VBATT	Battery Backup Voltage	-0.5	+7.0	Volts

ANT_SUPPLY	Antenna Supply Voltage	-15	+15	Volts
DIG_SIG_IN	Any Digital Input Signal	-0.3	+5.5	Volts
RF_IN	RF Input	-15	+15	Volts
T_{STORE}	Storage temperature	-55	+150	℃
T_{BIAS}	Temperature under bias	-40	+100	℃
I_{OUT}	Digital Signal Output Current	-6	+6	mA

Table 3 Absolute Maximum Ratings

2.4 Block Diagram

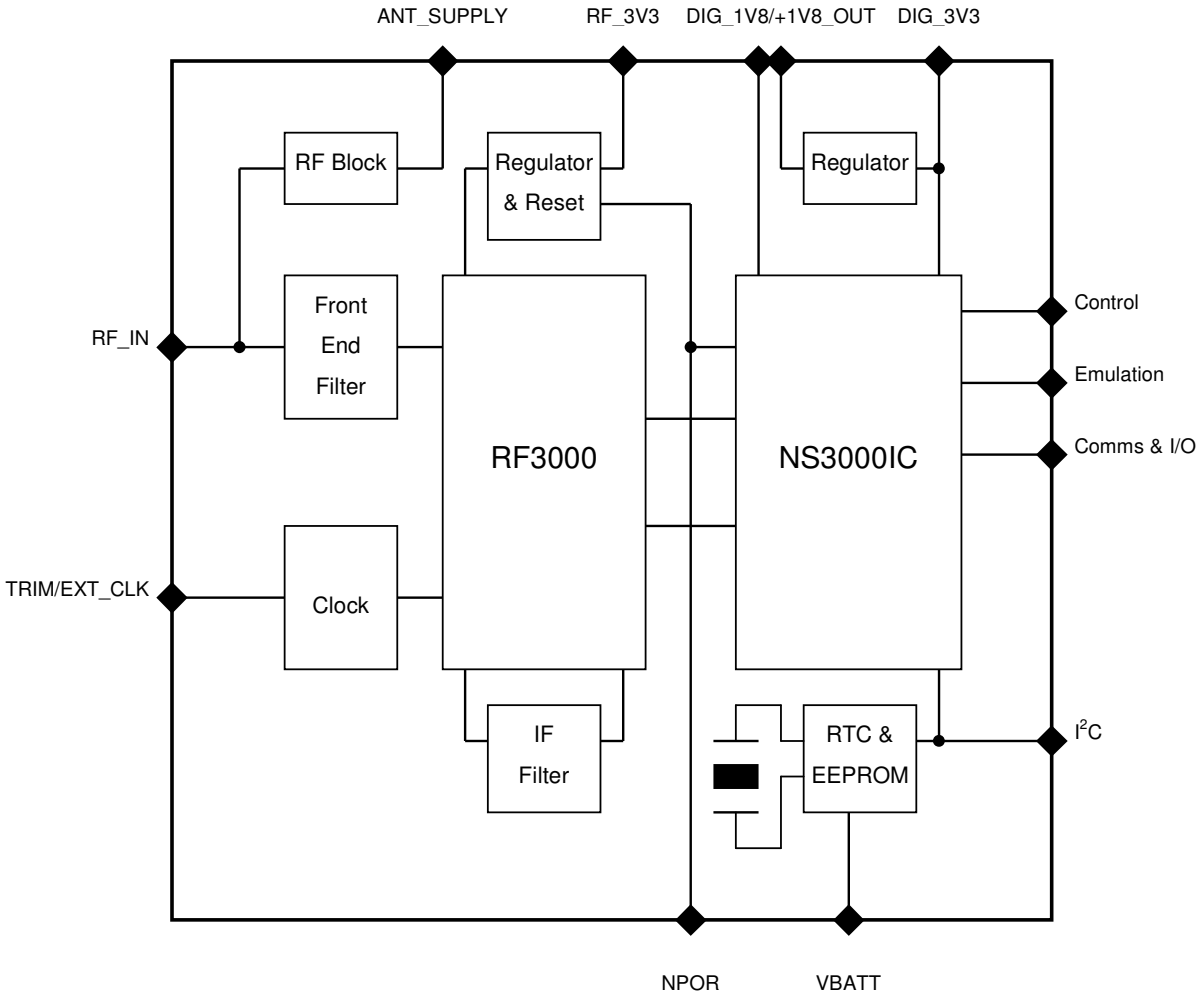


Figure 1 CW25 Block Diagram

3. PHYSICAL CHARACTERISTICS

The CW25 is a multi-chip module built on an FR4 fibreglass PCB. All digital and power connections to the MCM are via castellations on the 25 x 27 mm PCB. The RF connection is via castellations or an RF connector. The general arrangement of the CW25 is shown in the diagram below. Dimensions in mm (inches/1000).

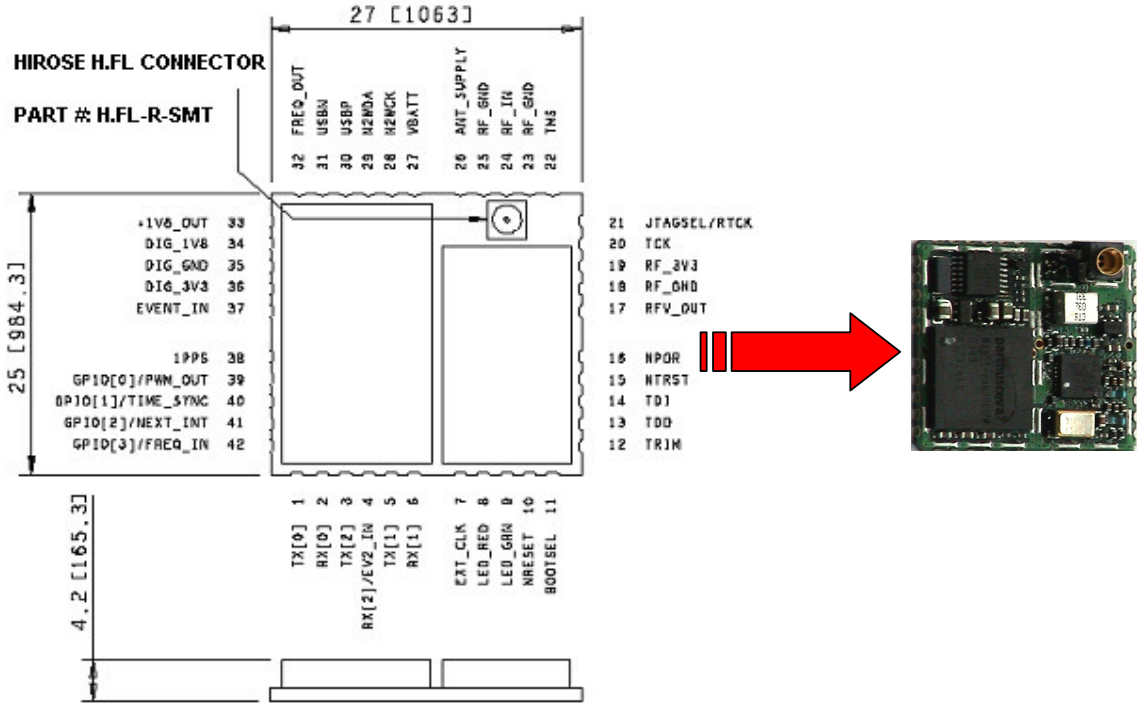


Figure 2 CW25 Form and Size

3.1 Physical Interface Details

The interface to the MCM is via 1mm castellation on a 2mm pitch. There are 42 connections in all. There is also an RF connector for connecting to the GPS antenna. The details of the interface connections are given below.

Pin	Function	Pin	Function
1	TX[0]	22	TMS
2	RX[0]	23	RF_GND
3	TX[2]	24	RF_IN
4	RX[2]/EV2_IN	25	RF_GND
5	TX[1]	26	ANT_SUPPLY
6	RX[1]	27	VBATT
7	EXT_CLK	28	N2WCK
8	LED_RED	29	N2WDA
9	LED_GRN	30	USBP
10	NRESET	31	USBN

11	BOOTSEL	32	FREQ_OUT
12	TRIM	33	+1V8_OUT
13	TDO	34	DIG_1V8
14	TDI	35	DIG_GND
15	NTRST	36	DIG_3V3
16	NPOR	37	EVENT_IN
17	RFV_OUT	38	1PPS
18	RF_GND	39	GPIO[0]/PWM_OUT
19	RF_3V3	40	GPIO[1]/TIME_SYNC
20	TCK	41	GPIO[2]/NEXT_INT
21	JTAGSEL/RTCK	42	GPIO[3]/FREQ_IN

Table 4 CW25 Signal List

3.2 MCM Dimensions

The figure below provides the dimensions of the positioning of the CW25 castellations. Dimensions in mm (inches/1000).

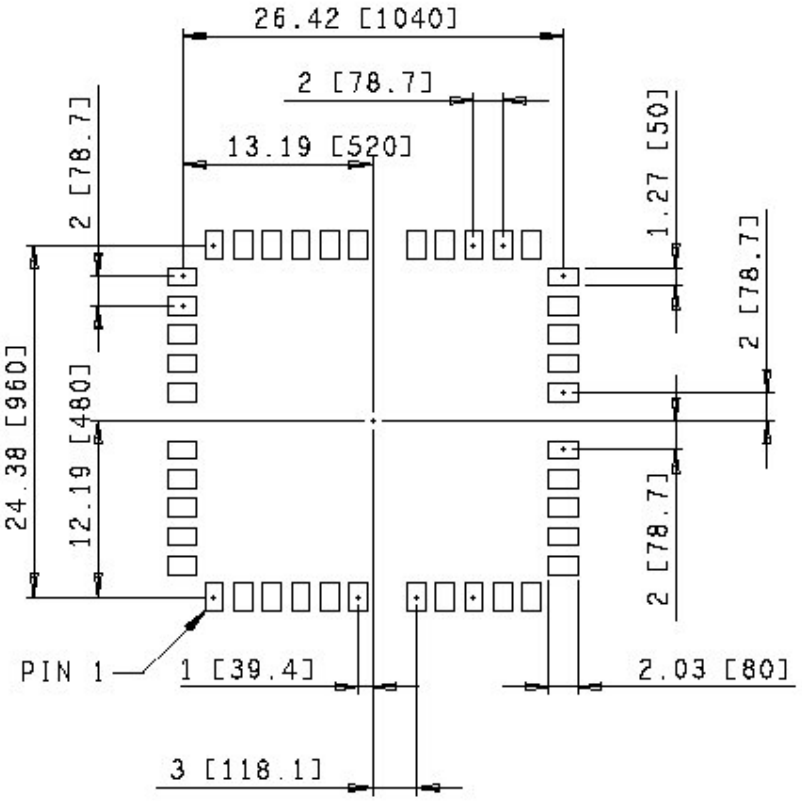


Figure 3 MCM Dimensions

cs	Message checksum in hexadecimal

7.2.1.7 POLYT, Time of Day

$POLYT,hhmmss.ss,ddmmyy, UTC_TOW ,week, GPS_TOW ,Clk_B , Clk_D ,PG,cs

```
$POLYT,123456.00,250299,123456.00,0978,123456.00,123456,123.456,28,cs
```

Name	Description
$POLYT	Navsync Proprietary NMEA sentence header (Position Data)
hhmmss.ss	UTC Time in hours, minutes, seconds. and decimal seconds format.
ddmmyy	Date in day, month, year format.
UTC_TOW	UTC Time of Week (seconds)
week	GPS week number (continues beyond 1023)
GPS_TOW	GPS Time of Week (seconds)
Clk_B	Receiver clock Bias (nanoseconds)
Clk_D	Receiver clock Drift (nanoseconds/second)
PG	1PPS Granularity (nanoseconds)
cs	Message checksum in hexadecimal

7.2.2 POLYP, Position Data

$POLYP,hhmmss.ss, Latitude ,N, Longitude ,E, AltRef ,FS,Hacc,Vacc, SOG , COG , Vvel ,ageC,HDOP,VDOP,PDOP,TDOP,GU,RU,DR,cs

```
$POLYP,123456.00,5214.12345,N,00056.12345,W,00138.80,G3,0002,0002,021.21,1
80.00,+003.96,99.9,01.1,01.6,01.9,01.7,07,00,00,cs
```

Name	Description
$POLYP	Navsync Proprietary NMEA sentence header (Position Data)
hhmmss.ss	UTC Time in hours, minutes, seconds.

The $POLYG is the same as the $POLYU sentence, except that the UTM position has been shifted to a Local Grid position by applying a Easting, Northing and Height offsets. The position output will be exactly the same as the UTM position if the Local Grid has not been defined.

7.2.2.3 POLYS, Satellite Status

$POLYS,GT,ID,s,AZM,EL,SN,LK,ID,s,AZM,EL,SN,LK,ID,

s,AZM,EL,SN,LK,ID,s,AZM,EL,SN,LK,ID,s,AZM,EL,SN,

LK,...,cs

```
$POLYS,05,03,U,103,56,48,99,23,U,225,61,39,99,16,
U,045,02,41,99,26,U,160,46,50,32,30,-,340,04,50,
00,cs
```

Name	Description
$POLYS	Navsync Proprietary NMEA sentence header (Satellite Data)
GT	Number of GPS satellites tracked
ID	Satellite PRN number (1-32)
s	Satellite status - = not used U = used in solution e = available for use, but no ephemeris
AZM	Satellite azimuth angle (range 000 - 359 degrees)
EL	Satellite elevation angle (range 00 - 90 degrees)
SN	Signal to noise ratio in (range 0 - 55 dBHz)
LK	Satellite carrier lock count (range 0 - 255 seconds) 0 = code lock only 255 = lock for 255 or more seconds
cs	Message checksum in hexadecimal

7.2.2.4 POLYI, Additional Information Message

$POLYI,JN,jammer,EXT,efields,INT,ifields,cs

$POLYI,JN,12,EXT,HPOS,VPOS,INT,CLKB

Name	Description
$POLYI	Navsync Proprietary NMEA sentence header