GNSS Tracker 2.4 - Source Code

Magnus Wasting

February 2018

Contents

1 main

```
#include "asf.h"
#include "gnss_tracker_2_4.h"
#include "system_gnss_tracker.h"
#include "uart_gnss_tracker.h"
#include "rtc_gnss_tracker.h"
#include "sim808.h"
#include "adc_gnss_tracker.h"
#include "eeprom_gnss_tracker.h"
#include "accelereometer.h"
#include "string.h"

static uint32_t more_work = 0;

/*  TO DO LIST
  eeprom error codes
*/


void EIC_Handler(void){
  // Stop and clear interrupt
  EIC->INTENCLR.reg = EIC_INTENSET_EXTINT1;
  EIC->INTFLAG.reg = EIC_INTENSET_EXTINT1;
  // Check alarm flag
  if (eeprom.alarm_active == TRUE){
    eeprom.alarm_active = FALSE;
    eeprom.alarm_triggerd = TRUE;
    write_memory();
    NVIC_SystemReset();
  }
  EIC->INTENSET.reg = EIC_INTENSET_EXTINT1;
}


///////////////////////////////////////////////
// SYSTICK
///////////////////////////////////////////////
void SysTick_Handler(void){
  mcu_status.uptime += 2; // 2 second has passed

  update_sim808_power_status();
  if(sim808.pwr_status == FALSE){
    mcu_status.error_counter++;
    print_pc_text("ERROR␣SIM808␣OFF!\n");
    print_pc_debug("pwr␣status", sim808.pwr_status);
  }

  if (mcu_status.error_counter > 20){
    eeprom.cycles++;
    eeprom.tot_errors++;
    write_memory();
    NVIC_SystemReset();
  }
}


static void wait_for_sms(uint32_t wait_time){
  print_pc_text("Waiting␣for␣SMS...");
  uint32_t start_time = RTC->MODE0.COUNT.reg;
  do {
    delay_ms(100);
    if(mcu_status.flag_unexpected_answer == TRUE){
      mcu_status.flag_unexpected_answer = FALSE;
      if((strstr(sim808.uart_buffer, "+CMTI:␣\"SM\"") != NULL)){
        more_work = TRUE;
      }
    }
    if (more_work == TRUE) {
      print_pc_text("OK!\n");
      break;
    }
  } while (RTC->MODE0.COUNT.reg < (start_time + wait_time));
  print_pc_text("NO␣SMS!\n");
}


void INIT_SLEEP(void){
  // 1022

  update_sim808_power_status();
  if (sim808.pwr_status == TRUE){
    SERCOM0->USART.INTENCLR.reg = SERCOM_USART_INTENCLR_RXS; // Stop interrupt
    PORT->Group[0].OUTCLR.reg = PORT_SIM808_PWRKEY;
    print_pc_text("-␣SIM808␣OFF\n");
    delay_ms(100);
  }

  PORT->Group[0].OUTCLR.reg = PORT_EN_4V;
  print_pc_text("-␣4V␣OFF\n");

  // Save memory;
  eeprom.cycles++;
  write_memory();

  print_status_all();
  print_pc_text("-␣Sleeping..\n");
  //delay_ms(50);
  INIT_RTC_SLEEP(60*eeprom.sleep_time); // Input in seconds
}


/////////////////////////////////////////////////
// MAIN
/////////////////////////////////////////////////
int main (void) {
  // SETUP
  INIT_MCU();

  // Update battery voltage reading
  update_bat_voltage();
  // Disable ADC (ADC draws ~1mA)
  DISABLE_ADC(); print_pc_text("-␣ADC␣DISABLE!\n");


  // Power on SIM808, if we have enough power.
  if (mcu_status.bat_volt > 3000){
    // SIM808
    INIT_UART_SIM808();
    print_pc_text("-␣UART␣SIM808␣OK!\n");
    INIT_SIM808(); print_pc_text("-␣SIM808␣OK!\n");
    uint32_t wait_for_sms_time = 35000000ul;

    // Alarm
    if (eeprom.alarm_triggerd == TRUE){
      eeprom.sleep_time = 10;
      print_pc_text("ALARM␣COMMAND!\n");
      alarm_command();

      eeprom.money = eeprom.money - (2*eeprom.cost);
      eeprom.sms_sent += 2;

      // Disable alarm
      eeprom.alarm_triggerd = FALSE;

      // Disable accelerometer
      DISABLE_ACCELEROMETER(); print_pc_text("ALARM␣DISABLED\n");
      wait_for_sms_time = 0; // no need to wait now;
    }

    // Debug
    print_status_all();

    wait_for_sms(wait_for_sms_time); // 35 s
    do_work();

    if (more_work == TRUE){
      more_work = FALSE;
      wait_for_sms(5000000); // 5 s
      do_work();
    }

    // Warn owner of low battery voltage
    if(mcu_status.bat_volt < 3400){
      if (eeprom.owner_phone_number[0] != NULL && eeprom.low_bat_flag == FALSE){
        print_pc_text("Warning␣owner␣of␣low␣battery\n");
        low_bat_command();
        eeprom.money = eeprom.money - (eeprom.cost);
        eeprom.sms_sent++;
        print_pc_text("low␣bat␣command␣OK\n");
        eeprom.low_bat_flag = TRUE;
      }
    }
    // Reset flag
    else if(mcu_status.bat_volt > 3450){
      eeprom.low_bat_flag = FALSE;
    }

  }
  else {
    eeprom.low_bat_flag = TRUE;
    print_pc_text("ERROR␣LOW␣BAT,␣CAN'T␣START␣SIM808!\n");
  }

  INIT_SLEEP();
}
```

2 system_gnss_tracker

2.1 header file

```
#ifndef SYSTEM_GNSS_TRACKER_H_
#define SYSTEM_GNSS_TRACKER_H_

#include "asf.h"

void INIT_MCU(void);
void delay_ms(volatile uint32_t time);


#endif /* SYSTEM_GNSS_TRACKER_H_ */
```

2.2 c file

```
#include "system_gnss_tracker.h"
#include "uart_gnss_tracker.h"
#include "adc_gnss_tracker.h"
#include "rtc_gnss_tracker.h"
#include "eeprom_gnss_tracker.h"
#include "gnss_tracker_2_4.h"
#include "sim808.h"

// volatile before
void delay_ms(volatile uint32_t time){
  time = 125*time; // 125 for a 1 MHz clk
  while(time--);
}

// CORE SYSTEM (CLOCKS, POWER)
static void INIT_CORE_SYSTEM(void){

  // From Reset : OSC8M @ 1MHz => Linked => GCLK0 @ 1MHz

  // Link GCLK0 to SERCOM1 and enable
  GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
  GCLK_CLKCTRL_ID_SERCOM1_CORE);

  // Link GCLK0 to SERCOM0 and enable
  GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
  GCLK_CLKCTRL_ID_SERCOM0_CORE);

  // Link GCLK0 to ADC and enable
  GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
  GCLK_CLKCTRL_ID_ADC);

  // Link GCLK0 to RTC and enable
    GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
    GCLK_CLKCTRL_ID_RTC);

  // ======= DATA BUS =========
  //PM->APBAMASK.reg &= (~PM_APBAMASK_EIC); // Disable
  //PM->AHBMASK.reg &= ~(PM_AHBMASK_DMAC);  // Disable
  PM->APBCMASK.reg |= PM_APBCMASK_SERCOM0 | PM_APBCMASK_SERCOM1 |
      PM_APBCMASK_ADC;
   // PM->APBAMASK.reg |= PM_APBAMASK_RTC;
}


static void INIT_PORT(void){
  // Switch control to multiplexer

  PORT->Group[0].PINCFG[PIN_SIM808_TX].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_SIM808_RX].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_ADC].reg |= PORT_PINCFG_PMUXEN;

  // Configure multiplexer for SERCOM1, SERCOM2 and ADC

  PORT->Group[0].PMUX[PIN_SIM808_TX/2].reg |= PORT_PMUX_PMUXE(2);   // 2 = C =
     SERCOM
  PORT->Group[0].PMUX[PIN_SIM808_RX/2].reg |= PORT_PMUX_PMUXO(2);   // 2 = C =
     SERCOM
  PORT->Group[0].PMUX[PIN_ADC/2].reg |= PORT_PMUX_PMUXO(1);   // 1 = ADC

  // Set output
  PORT->Group[0].DIRSET.reg = PORT_EN_4V;
  // Turn off 4V rail
  PORT->Group[0].OUTCLR.reg = PORT_EN_4V;
  // Input Enable
  PORT->Group[0].PINCFG[PIN_SIM808_STATUS].reg |= PORT_PINCFG_INEN;
}

static void INIT_INTERRUPT(void){

  // Setup for SysTick
  SysTick->LOAD = ((2000000ul) & SysTick_LOAD_RELOAD_Msk) - 1;
  SysTick->VAL = 0;
  SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk |
     SysTick_CTRL_TICKINT_Msk;

  // Priority
  NVIC_SetPriority(SERCOM0_IRQn, 1); // SIM808
  NVIC_SetPriority(SysTick_IRQn, 2);

  // Enable/disable interrupt
  NVIC_EnableIRQ(SysTick_IRQn);
  NVIC_DisableIRQ(RTC_IRQn);
}

// Initialize MCU
void INIT_MCU(void){
  cpu_irq_enter_critical();

  /* Various bits in the INTFLAG register can be set to one at startup.
  This will ensure that these bits are cleared */
  //SYSCTRL->INTFLAG.reg = SYSCTRL_INTFLAG_BOD33RDY | SYSCTRL_INTFLAG_BOD33DET |
      SYSCTRL_INTFLAG_DFLLRDY;

  // Wait states (flash memory)
  NVMCTRL->CTRLB.reg |= NVMCTRL_CTRLB_RWS(0); // 0 = zero wait states

  // System
  INIT_CORE_SYSTEM();
  INIT_PORT();

  INIT_INTERRUPT();

  // Other Peripherals
  INIT_RTC_1M();
  INIT_UART_PC();
  INIT_ADC();
  INIT_EEPROM();

  print_pc_text("\n\n\n==============================␣");
  print_pc_text(SOFTWARE_VERSION);
  print_pc_text("␣==============================\n");
  print_pc_text("-␣RTC,␣UART_PC,␣ADC,␣EEPROM,␣ALL_OK!\n");


  // Default state
  sim808.gnss_status = 0;
  sim808.pwr_status = 0;
  sim808.uart_sum = 0;

  cpu_irq_leave_critical();
}
```

3 uart_gnss_tracker

3.1 header file

```
#ifndef UART_GNSS_TRACKER_H_
#define UART_GNSS_TRACKER_H_

#include "asf.h"

void INIT_UART(void);

void INIT_UART_PC(void);

void INIT_UART_SIM808(void);
void DISABLE_UART_SIM808(void);


// SIM808
void print_sim808_text(const char *text);
void print_sim808_dec(uint32_t number);

// PC
void print_pc_text(const char *text);
void print_pc_dec(uint32_t number);
void print_pc_neg_dec(int32_t number);
void print_pc_debug(const char *text, uint32_t number);

void print_pc_debug_string(const char *text1, const char *text2);

#endif /* UART_GNSS_TRACKER_H_ */
```

3.2 c file

```
#include "uart_gnss_tracker.h"
#include "gnss_tracker_2_4.h" // IO PINS


#include "stdlib.h"



// UART PC
void INIT_UART_PC(void){
  /* The TX pin on the "flower led" is connected to PA24
    SERCOM1 - PAD2
  */


  // PORT
  PORT->Group[0].PINCFG[PIN_PC_TX].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PMUX[PIN_PC_TX/2].reg |= PORT_PMUX_PMUXE(2);   // 2 = C =
      SERCOM

  // Settings
  SERCOM1->USART.CTRLA.reg |= SERCOM_USART_CTRLA_MODE(1) // UART with internal
      clock
  | SERCOM_USART_CTRLA_TXPO(1) // 0 = PAD0, 1 = PAD2
  | SERCOM_USART_CTRLA_DORD;    // LSB is transmitted first.

  // Enable TX
  SERCOM1->USART.CTRLB.reg |= SERCOM_USART_CTRLB_TXEN;
  // Baud rate, baud = 65536*(1-s*f_buad/f_clk), s = 16 from reset
  SERCOM1->USART.BAUD.reg = (uint16_t)45403; // 45403 : baud = 19200 with a clk
      = 1MHz

  // Enable UART
  while( SERCOM1->USART.SYNCBUSY.reg & (SERCOM_USART_CTRLA_ENABLE |
      SERCOM_USART_SYNCBUSY_SWRST)); // waiting for sync
  SERCOM1->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE; // Module on



}


void DISABLE_UART_PC(void){

}

// UART SIM808
void INIT_UART_SIM808(void){
  // PORT_SIM808_TX = PA22 = SERCOM0 PAD0
  // PORT_SIM808_RX = PA17 = SERCOM0 PAD3



    // Settings
  SERCOM0->USART.CTRLA.reg |= SERCOM_USART_CTRLA_MODE(1) // UART with internal
      clock
  | SERCOM_USART_CTRLA_TXPO(1) // 0 = PAD0, 1 = PAD2
  | SERCOM_USART_CTRLA_RXPO(3) // 3 = PAD3
  | SERCOM_USART_CTRLA_DORD;    // LSB is transmitted first.


  // Clear flag
  SERCOM0->USART.INTFLAG.reg = 0xFF;

  // Enable TX and RX
  SERCOM0->USART.CTRLB.reg |= SERCOM_USART_CTRLB_TXEN | SERCOM_USART_CTRLB_RXEN
      | SERCOM_USART_CTRLB_SFDE;

  // Baud rate, baud = 65536*(1-s*f_buad/f_clk), s = 16 from reset
  SERCOM0->USART.BAUD.reg = (uint16_t)45403; // 45403 : baud = 19200 with a clk
      = 1MHz

  // Enable UART
  while( SERCOM0->USART.SYNCBUSY.reg & (SERCOM_USART_CTRLA_ENABLE |
      SERCOM_USART_SYNCBUSY_SWRST)); // waiting for sync
  SERCOM0->USART.CTRLA.reg |= SERCOM_USART_CTRLA_ENABLE; // Module on

}



void DISABLE_UART_SIM808(void){
  NVIC_DisableIRQ(SERCOM0_IRQn);
  SERCOM0->USART.CTRLA.reg &= ~SERCOM_USART_CTRLA_ENABLE;
  PM->APBCMASK.reg &= ~(PM_APBCMASK_SERCOM0);
}
```

```
void print_sim808_text(const char *text){

  uint32_t i = 0;
  do {
    while( !((SERCOM0->USART.INTFLAG.reg) & SERCOM_USART_INTFLAG_DRE) ); //
       Check if buffer is empty
    SERCOM0->USART.DATA.reg = text[i++];
  } while (text[i] != 0);
}


// Print string
void print_pc_text(const char *text){

  uint32_t i = 0;
  do {
    while( !((SERCOM1->USART.INTFLAG.reg) & SERCOM_USART_INTFLAG_DRE) ); //
       Check if buffer is empty
    SERCOM1->USART.DATA.reg = text[i++];
  } while (text[i] != 0);
}

void print_pc_debug(const char *text, uint32_t number){
  print_pc_text(text);
  print_pc_text("␣:␣");
  print_pc_dec(number);
  print_pc_text("\n");
}

void print_pc_debug_string(const char *text1, const char *text2){
  print_pc_text(text1);
  print_pc_text("␣:␣");
  print_pc_text(text2);
  print_pc_text("\n");
}

// Print a integer to a string, etc 87 to "87"
void print_pc_dec(uint32_t number){

  uint8_t text[32];
  itoa(number, text, 10);
  uint32_t i = 0;
  do {
    while( !((SERCOM1->USART.INTFLAG.reg) & SERCOM_USART_INTFLAG_DRE) ); //
       Check if buffer is empty
    SERCOM1->USART.DATA.reg = text[i++];
  } while (text[i] != NULL);
}


// Print a integer to a string, etc 87 to "87"
void print_sim808_dec(uint32_t number){

  uint8_t text[32];
  itoa(number, text, 10);
  uint32_t i = 0;
  do {
    while( !((SERCOM0->USART.INTFLAG.reg) & SERCOM_USART_INTFLAG_DRE) ); //
        Check if buffer is empty
    SERCOM0->USART.DATA.reg = text[i++];
  } while (text[i] != NULL);
}



void print_pc_neg_dec(int32_t number){
  uint8_t text[32];
  itoa(number, text, 10);
  uint32_t i = 0;
  do {
    while( !((SERCOM1->USART.INTFLAG.reg) & SERCOM_USART_INTFLAG_DRE) ); //
        Check if buffer is empty
    SERCOM1->USART.DATA.reg = text[i++];
  } while (text[i] != NULL);
}
```

4 rtc_gnss_tracker

4.1 header file

```
#ifndef RTC_GNSS_TRACKER_H_
#define RTC_GNSS_TRACKER_H_

#include "asf.h"

void INIT_RTC_1M(void);
void delay_rtc_us(uint32_t delayTime);
void INIT_RTC_SLEEP(uint32_t sleep_time);

#endif /* RTC_GNSS_TRACKER_H_ */
```

4.2 c file

```
#include "rtc_gnss_tracker.h"
#include "gnss_tracker_2_4.h"
#include "uart_gnss_tracker.h"


void RTC_Handler(void){
  NVIC_SystemReset();
}


// THIS NEEDS A 1MHZ CLK!
void INIT_RTC_1M(void){
  RTC->MODE0.CTRL.reg = RTC_MODE0_CTRL_SWRST;
  while((RTC->MODE0.STATUS.reg));
  // Enable
  RTC->MODE0.CTRL.reg |= RTC_MODE0_CTRL_ENABLE;
  RTC->MODE0.COUNT.reg = 0;
}


static void INIT_RTC_1K(uint32_t sleep_time){
  // rst RTC
  RTC->MODE0.CTRL.reg = RTC_MODE0_CTRL_SWRST;
  while((RTC->MODE0.STATUS.reg));

  // RST ALL CLOCKS
  //GCLK->CTRL.reg = GCLK_CTRL_SWRST;
  RTC->MODE0.COUNT.reg = 0;
  RTC->MODE0.COMP->reg = 1000*sleep_time; // 1000 => seconds

  // settings for sleep
  RTC->MODE0.CTRL.reg |= RTC_MODE0_CTRL_ENABLE | RTC_MODE0_CTRL_MATCHCLR;
  RTC->MODE0.INTENSET.reg = RTC_MODE0_INTENSET_CMP0;
}


// Sleep, input is ms
void INIT_RTC_SLEEP(uint32_t sleep_time){

  SysTick->CTRL &= ~SysTick_CTRL_TICKINT_Msk;
  NVIC_DisableIRQ(SysTick_IRQn);
  NVIC_DisableIRQ(SERCOM1_IRQn);
  NVIC_DisableIRQ(SERCOM0_IRQn);
  // Change main clock to 1kHz!
  // Divide GCLK2 by 2^(4 + 1) to get 1024Hz
  // The generic clock generator equals the clock source divided by 2^(GENDIV.
      DIV+1).
  GCLK->GENDIV.reg = GCLK_GENDIV_DIV(4) | GCLK_GENDIV_ID(0); // 0 = GCLK[0]
  // Link OSCULP32K to GCLK2 and enable
   GCLK->GENCTRL.reg = (GCLK_GENCTRL_GENEN | GCLK_GENCTRL_SRC_OSCULP32K |
       GCLK_GENCTRL_DIVSEL
   | GCLK_GENCTRL_ID(0) | GCLK_GENCTRL_RUNSTDBY); // 0 = GCLK[0]

  INIT_RTC_1K(sleep_time);

  NVIC_SetPriority(RTC_IRQn, 3);
  NVIC_EnableIRQ(RTC_IRQn);
  //ADC->CTRLA.reg |= ADC_CTRLA_SWRST;

  SERCOM0->USART.CTRLA.reg = SERCOM_USART_CTRLA_SWRST;
  while(SERCOM0->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_SWRST);

  SERCOM1->USART.CTRLA.reg = SERCOM_USART_CTRLA_SWRST;
  while(SERCOM1->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_SWRST);

  SERCOM2->SPI.CTRLA.reg = SERCOM_USART_CTRLA_SWRST;
  while(SERCOM2->USART.SYNCBUSY.reg & SERCOM_USART_SYNCBUSY_SWRST);

  PM->APBCMASK.reg &= ~(PM_APBCMASK_SERCOM0 | PM_APBCMASK_SERCOM1 |
      PM_APBCMASK_SERCOM2 | PM_APBCMASK_ADC);
  PM->APBAMASK.reg &= ~(PM_APBAMASK_PAC0 | PM_APBAMASK_WDT);
  PM->APBBMASK.reg &= ~(PM_APBBMASK_PAC1 | PM_APBBMASK_DMAC);

  SYSCTRL->OSC8M.reg &= ~(SYSCTRL_OSC8M_ENABLE);
  PM->SLEEP.reg |= 0x2;;

  // The following peripheral clocks running: PM, SYSCTRL, RTC

  // Sleep
  SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
  __DSB();
  __WFI();
}


// ========================= delay =========================
// Actual delay @ 1 MHz CLK : value + 34
void delay_rtc_us(uint32_t delayTime){
  uint32_t startTime = RTC->MODE0.COUNT.reg;
  while(RTC->MODE0.COUNT.reg < (startTime + delayTime));
}
```

5 eeprom_gnss_tracker

5.1 header file

```
#ifndef EEPROM_GNSS_TRACKER_H_
#define EEPROM_GNSS_TRACKER_H_

void INIT_EEPROM(void);
void write_memory(void);
void read_memory(void);

#endif /* EEPROM_GNSS_TRACKER_H_ */
```

5.2 c file

```
#include "eeprom_gnss_tracker.h"
#include "gnss_tracker_2_4.h"
#include "uart_gnss_tracker.h"  // Debug

#include "asf.h"

/*
  Embedded Flash starts at : 0x00000000

  Flash size (FLASH_PM)     : 16Kbytes
  Number of pages (FLASH_P)    : 256
  Page size (FLASH_W)       : 64 (bytes)

  256*64 = 16384 (16kB)

  From datasheet; eeprom size is 256 bytes (1 row)
  so, do we have 4 pages then? yes!

  The NVM is organized into rows, where each row contains four pages,
  as shown in Figure 21-2. The NVM has a rowerase granularity, while the
  write granularity is by page. In other words, a single row erase will erase
  all four pages in the row, while four write operations are used to write the
  complete row.

  From rst : CLK_NVMCTRL_APB is enable


*/


#define DATA_KEY      (216) // RANDOM NUMBER, MAX 255
#define NVM_MEMORY16      ((volatile uint16_t *)FLASH_ADDR )
#define NVM_MEMORY32      ((volatile uint32_t *)FLASH_ADDR )
#define EEPROM_ADDRESS      ((uint32_t)0x00003F00)

/*
  How do we know which address to write to.. brute force?
*/


// FLASH_ADDR; write data to this address, then write command and address in
    memory
// => NVMCTRL will move the data to the correct address in memmory...

void INIT_EEPROM(void){
  // Manual write
  NVMCTRL->CTRLB.reg |= NVMCTRL_CTRLB_MANW;
  // Read data, if data is currupt rst to defaults values
  read_memory();
  if (eeprom.data_key != DATA_KEY){
    eeprom.money = 100000; // 100 kr
    eeprom.sms_sent = 0;
    eeprom.data_key = DATA_KEY;
    eeprom.cycles = 0;
    eeprom.sleep_time = 30;
    eeprom.tot_errors = 0;
    eeprom.cost = 690; // 0.68 kr
    eeprom.flag_low_bat = 0;
    eeprom.owner_phone_number[0] = NULL;
    eeprom.owner_phone_number[1] = NULL;
    eeprom.alarm_active = 0;
    eeprom.alarm_triggerd = 0;
    eeprom.low_bat_flag = 0;

    for (int i = 0; i < (sizeof(eeprom.dummy_mem) >> 2); i++){
      eeprom.dummy_mem[i] = 0;
    }
    write_memory();
    read_memory();
  }
}


void write_memory(void){
  cpu_irq_enter_critical();
  /*
    1. Write address and erase row
    2. Clear page buffer
    3. Write data to page buffer (flash base address)
    4. Write address
    5. Execute write command
  */

  uint32_t bytes = sizeof(eeprom) >> 2; // dive by 4 to get number of "32bits"
  uint32_t *p = &eeprom;

  // Erase row (4 pages!)
  NVMCTRL->ADDR.reg = EEPROM_ADDRESS >> 1;
  NVMCTRL->CTRLA.reg = NVMCTRL_CTRLA_CMD_ER | NVMCTRL_CTRLA_CMDEX_KEY;
  while (!(NVMCTRL->INTFLAG.reg & NVMCTRL_INTFLAG_READY));

  // Clear page buffer.
  NVMCTRL->CTRLA.reg = NVMCTRL_CTRLA_CMD_PBC | NVMCTRL_CTRLA_CMDEX_KEY; // PBC;
      page buffer clear
  while (!(NVMCTRL->INTFLAG.reg & NVMCTRL_INTFLAG_READY));

  // Write data to PAGE buffer.
  //while (!(NVMCTRL->INTFLAG.reg & NVMCTRL_INTFLAG_READY));
  for (int i = 0; i < bytes; i++){ // length; how many 4 bytes to write (uin32_t
      )
    NVM_MEMORY32[i] = *p++; // buffer; data to be writen
  }

  // Execute write command with the given address.
  NVMCTRL->ADDR.reg = (EEPROM_ADDRESS >> 1); // page is 64 bytes but addressed
      as 16bits
  NVMCTRL->CTRLA.reg = NVMCTRL_CTRLA_CMD_WP | NVMCTRL_CTRLA_CMDEX_KEY;
  while (!(NVMCTRL->INTFLAG.reg & NVMCTRL_INTFLAG_READY));

  cpu_irq_leave_critical();
}


void read_memory(void){
  uint32_t page = 0;
  uint32_t *p = &eeprom;
  uint32_t bytes = sizeof(eeprom) >> 2; // Divide by 4

  while (!(NVMCTRL->INTFLAG.reg & NVMCTRL_INTFLAG_READY));
  for (int i = 0; i < bytes; i++){
    // divide address by 4: memory is indexed with 1 byte, we're reading 4 bytes
    // 16*page: page*64/4 = page*16
    *p++ = NVM_MEMORY32[(EEPROM_ADDRESS >> 2) + i];
  }
}
```

6 accelereometer

6.1 header file

```
#ifndef ACCELEREOMETER_H_
#define ACCELEREOMETER_H_

#include "asf.h"

void INIT_SPI(void);
void INIT_EIC(void);
void INIT_ACCELEROMETER(void);
void DISABLE_ACCELEROMETER(void);
uint8_t read_accelerometer(uint8_t address);
void write_accelerometer(uint8_t address, uint8_t data);

// COMMANDS
#define ENABLE_LP_1HZ_MODE          (0b0011101)
#define ENABLE_LP_10HZ_MODE         (0b0101101)
#define ENABLE_LP_25HZ_MODE         (0b0111101)
#define ENABLE_LP_100HZ_MODE        (0b1011101)
#define ENABLE_INT_FUNC_1_2         (0b1100000)
#define ENABLE_INT2             (0b0100000)
#define ENABLE_XYZ              (0b111111)
#define POWER_DOWN_MODE             (0b0)
#define INT2_CFG_ENABLE_Z_HIGHER     (0b100000)
#define INT2_CFG_ENABLE_Z_LOWER      (0b010000)
#define INT2_CFG_ENABLE_X_HIGHER     (0b000010)
#define INT2_CFG_ENABLE_X_LOWER      (0b100001)
#define ENABLE_XYZ_6D           (0b1111111)
#define ENABLE_XYZ_AOI              (0b10111111)
#define ENABLE_X_6D             (0b1000011)
#define ENABLE_Y_6D             (0b1001100)
#define ENABLE_Z_6D             (0b1110000)

// ADDRESS
#define OUT_X_L_ADDRESS             (0b0101000)
#define OUT_X_H_ADDRESS             (0b0101001)
#define OUT_Y_H_ADDRESS             (0b0101011)
#define OUT_Z_H_ADDRESS             (0b0101101)
#define INT1_CFG_ADDRESS            (0b0110000)
#define INT2_CFG_ADDRESS            (0b0110100)
#define WHO_AM_I_ADDRESS            (0b0001111)
#define CTRL_REG1_ADDRESS           (0b0100000)
#define CTRL_REG6_ADDRESS           (0b0100101)
#define INT1_THS_ADDRESS            (0b0110010)
#define INT2_THS_ADDRESS            (0b0110110)

#endif /* ACCELEREOMETER_H_ */
```

6.2 c file

```
#include "accelereometer.h"
#include "gnss_tracker_2_4.h"
#include "uart_gnss_tracker.h"





void INIT_SPI(void){
  cpu_irq_enter_critical();
  // SERCOM2

  // Link GCLK0 to SERCOM2 and enable
  GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
  GCLK_CLKCTRL_ID_SERCOM2_CORE);

  PM->APBCMASK.reg |= PM_APBCMASK_SERCOM2;

  //PORT->Group[0].PINCFG[PIN_ACC_CS].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_ACC_DATA_IN].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_ACC_DATA_OUT].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_ACC_CLK].reg |= PORT_PINCFG_PMUXEN;
  //PORT->Group[0].PINCFG[PIN_ACC_INT1].reg |= PORT_PINCFG_PMUXEN;
  //PORT->Group[0].PINCFG[PIN_ACC_INT2].reg |= PORT_PINCFG_PMUXEN;

  //PORT->Group[0].PMUX[PIN_ACC_CS/2].reg |= PORT_PMUX_PMUXE(3); // 3 = D =
      SERCOM-ALT
  PORT->Group[0].PMUX[PIN_ACC_DATA_IN/2].reg |= PORT_PMUX_PMUXO(3); // 3 = D =
      SERCOM-ALT
  PORT->Group[0].PMUX[PIN_ACC_DATA_OUT/2].reg |= PORT_PMUX_PMUXE(3); // 3 = D =
      SERCOM-ALT
  PORT->Group[0].PMUX[PIN_ACC_CLK/2].reg |= PORT_PMUX_PMUXO(3); // 3 = D =
      SERCOM-ALT

  PORT->Group[0].DIRSET.reg = PORT_ACC_CS;
  PORT->Group[0].OUTSET.reg = PORT_ACC_CS;



  // Those lines are driven at the
  // falling edge of SPC and should be captured at the rising edge of SPC.

  SERCOM2->SPI.CTRLA.reg |= SERCOM_SPI_CTRLA_CPOL | SERCOM_SPI_CTRLA_CPHA |
      SERCOM_SPI_CTRLA_DIPO(3) |
  SERCOM_SPI_CTRLA_DIPO(0) | SERCOM_SPI_CTRLA_MODE(3);

  SERCOM2->SPI.CTRLB.reg |= SERCOM_SPI_CTRLB_RXEN | SERCOM_SPI_CTRLB_MSSEN;

  SERCOM2->SPI.BAUD.reg = 0;  // 0 = 500kHz @ 1MHz


  //SERCOM2->SPI.INTENSET.reg = SERCOM_SPI_INTENSET_RXC;


  SERCOM2->SPI.CTRLA.reg |= SERCOM_SPI_CTRLA_ENABLE;

  // 1 = Enable synchronization is busy.
  while(SERCOM2->SPI.SYNCBUSY.reg & SERCOM_SPI_SYNCBUSY_ENABLE);



  //NVIC_SetPriority(SERCOM2_IRQn, 3);
  //NVIC_EnableIRQ(SERCOM2_IRQn);

  print_pc_text("INIT_SPI!\n");

  cpu_irq_leave_critical();
}
```

```
uint8_t read_accelerometer(uint8_t address){

  // RW bit. When 0, the data DI(7:0) is written into the device. When 1, the
      data DO(7:0)
  // from the device is read. In latter case, the chip will drive SDO at the
      start of bit 8.

  address |= (1 << 7);
  uint8_t data;
  // Dummy read
  //while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_RXC));
  //uint8_t data = SERCOM2->SPI.DATA.reg;


  PORT_IOBUS->Group[0].OUTCLR.reg = PORT_ACC_CS;
  // Send address
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_DRE));
  SERCOM2->SPI.DATA.reg = address;
  // Dummy read
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_RXC));
  data = SERCOM2->SPI.DATA.reg;
  // Dummy send
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_DRE));
  SERCOM2->SPI.DATA.reg = 0;

  // Read data
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_RXC));
  data = SERCOM2->SPI.DATA.reg;


  //delay_ms(10);
  PORT_IOBUS->Group[0].OUTSET.reg = PORT_ACC_CS;

  return data;
}

void write_accelerometer(uint8_t address, uint8_t data){
  PORT_IOBUS->Group[0].OUTCLR.reg = PORT_ACC_CS;

  // Send address
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_DRE));
  SERCOM2->SPI.DATA.reg = address;
  // Dummy read
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_RXC));
  uint8_t data2 = SERCOM2->SPI.DATA.reg;
  // Write data
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_DRE));
  SERCOM2->SPI.DATA.reg = data;
  // dummy read
  while(!(SERCOM2->SPI.INTFLAG.reg & SERCOM_SPI_INTFLAG_RXC));
  data2 = SERCOM2->SPI.DATA.reg;

  PORT_IOBUS->Group[0].OUTSET.reg = PORT_ACC_CS;
}


void INIT_EIC(void){

  // Link GCLK0 to EIC and enable
  GCLK->CLKCTRL.reg = (uint16_t)(GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_CLKEN |
  GCLK_CLKCTRL_ID_EIC); // EIC
  PM->APBAMASK.reg |= PM_APBAMASK_EIC;

  /*
    PA27 = INT1 = EXTINT7
    PA15 = INT2 = EXTINT1
  */

  //PORT->Group[0].PINCFG[PIN_ACC_INT1].reg |= PORT_PINCFG_PMUXEN;
  PORT->Group[0].PINCFG[PIN_ACC_INT2].reg |= PORT_PINCFG_PMUXEN;

  //PORT->Group[0].PMUX[PIN_ACC_INT1/2].reg |= PORT_PMUX_PMUXO(0); // 0 = A =
     EIC
  PORT->Group[0].PMUX[PIN_ACC_INT2/2].reg |= PORT_PMUX_PMUXO(0); // 0 = A = EIC

  EIC->CONFIG[0].reg |= EIC_CONFIG_FILTEN1 | EIC_CONFIG_SENSE1_RISE;


  EIC->WAKEUP.reg |= EIC_WAKEUP_WAKEUPEN1;

  // Clear flag and enable interrupt.
  EIC->INTFLAG.reg = EIC_INTENSET_EXTINT1;
  EIC->INTENSET.reg = EIC_INTENSET_EXTINT1;

  // Enable
  EIC->CTRL.reg |= EIC_CTRL_ENABLE;
  while((EIC->STATUS.reg)); // Wait for sync
  // This bit is set when the synchronization of registers between clock domains
      is started.

  NVIC_SetPriority(EIC_IRQn, 4);
  NVIC_EnableIRQ(EIC_IRQn);
}


void INIT_ACCELEROMETER(void){

  INIT_SPI();
  delay_ms(400);

  // Update speed
  write_accelerometer(CTRL_REG1_ADDRESS, ENABLE_LP_25HZ_MODE);

  delay_ms(20);

  // Read location
  uint8_t x,z;
  x = read_accelerometer(OUT_X_H_ADDRESS); // tilt
  z = read_accelerometer(OUT_Z_H_ADDRESS); // up/down

  // Debug
  print_pc_text("[X,Z]␣:␣");
  print_pc_dec(x);
  print_pc_text(",␣");
  print_pc_dec(z);
  print_pc_text("\n");

  // X
  if(x < 240){
    write_accelerometer(INT2_CFG_ADDRESS, INT2_CFG_ENABLE_X_HIGHER);
    write_accelerometer(INT2_THS_ADDRESS, x + 10);
    print_pc_debug("X␣higher␣than␣", x + 10);
  }
  else {
    // 240 < x < 255
    write_accelerometer(INT2_CFG_ADDRESS, INT2_CFG_ENABLE_X_LOWER);
    write_accelerometer(INT2_THS_ADDRESS, x - 10);
    print_pc_debug("X␣lower␣than␣", x - 10);
  }

  // Z
  if(z < 240){
    write_accelerometer(INT2_CFG_ADDRESS, INT2_CFG_ENABLE_Z_HIGHER);
    write_accelerometer(INT2_THS_ADDRESS, z + 10);
    print_pc_debug("Z␣higher␣than␣", z + 10);
  }
  else {
    // 240 < x < 255
    write_accelerometer(INT2_CFG_ADDRESS, INT2_CFG_ENABLE_Z_LOWER);
    write_accelerometer(INT2_THS_ADDRESS, z - 10);
    print_pc_debug("Z␣lower␣than␣", z - 10);
  }

  // Enable INT2
  write_accelerometer(CTRL_REG6_ADDRESS, ENABLE_INT2);

  INIT_EIC();
}


void DISABLE_ACCELEROMETER(void){
  print_pc_text("disable␣EIC\n");
  EIC->CTRL.reg = EIC_CTRL_SWRST;
  while (EIC->STATUS.reg & EIC_STATUS_SYNCBUSY);
  INIT_SPI();
  write_accelerometer(CTRL_REG1_ADDRESS, POWER_DOWN_MODE);
}
```

7 adc_gnss_tracker

7.1 header file

```c
#ifndef ADC_GNSS_TRACKER_H_
#define ADC_GNSS_TRACKER_H_

#include "asf.h"

void INIT_ADC(void);
void DISABLE_ADC(void);
uint32_t adc_read(void);

#endif /* ADC_GNSS_TRACKER_H_ */
```

7.2 c file

```c
#include "adc_gnss_tracker.h"



void INIT_ADC(void){

  // Input
  ADC->INPUTCTRL.reg |= ADC_INPUTCTRL_MUXNEG_IOGND | ADC_INPUTCTRL_MUXPOS_PIN1; // Internal GND

  ADC->REFCTRL.reg |= ADC_REFCTRL_REFSEL_INTVCC0; // Vdd/1.48 => 1.7567 @ 2.6V Vdd
  ADC->SAMPCTRL.reg = ADC_SAMPCTRL_SAMPLEN(0); // 10 = 80us, max 5 bits!
  ADC->AVGCTRL.reg |= ADC_AVGCTRL_SAMPLENUM_16; // Number of samples
  ADC->AVGCTRL.reg |= ADC_AVGCTRL_ADJRES(4); // 4? => 2^4 = 16 = samples
  ADC->CTRLB.reg |= ADC_CTRLB_RESSEL_16BIT;
  ADC->CALIB.reg = ADC_CALIB_BIAS_CAL(0x03) | ADC_CALIB_LINEARITY_CAL(0x0b | (0x04 << 5));

  ADC->CTRLA.reg = ADC_CTRLA_ENABLE;
  while(ADC->STATUS.reg);
}


void DISABLE_ADC(void){
  ADC->REFCTRL.reg = 0;
  ADC->CTRLA.reg |= ADC_CTRLA_SWRST;
  ADC->CTRLA.reg &= ~ADC_CTRLA_ENABLE;
  PM->APBCMASK.reg &= ~(PM_APBCMASK_ADC);
}

uint32_t adc_read(void){
  ADC->SWTRIG.reg |= ADC_SWTRIG_START;
  while( !(ADC->INTFLAG.reg & ADC_INTFLAG_RESRDY));
  return (ADC->RESULT.reg);
}
```

8 sim808

8.1 header file

```
#ifndef SIM808_H_
#define SIM808_H_

#include "asf.h"

#define FLAG_PHONE_NUMBER ("+46")
#define PHONE_NUMBER_LENGTH 15 // Max length
#define COMMAND_START_FLAG ("--")
#define COMMAND_START_FLAG_LENGTH 2
#define COMMAND_END_FLAG ('\x3B') // x3B = ;
#define ANSWER_HI ("Hi!␣:)")

// STRING SUMS
#define CFUN_READY_SUM  813
#define CPIN_READY_SUM  850
#define CALL_READY_SUM  959
#define SMS_READY_SUM 822


// AT COMMANDS FOR SIM808
#define AT_DELETE_ALL_SMS ("AT+CMGDA=\"DEL␣ALL\"\r")
#define AT_SMS_MSG_FORMAT_TEXT ("AT+CMGF=1\r")
#define AT_SIGNAL_STRENGTH ("AT+CSQ\r")

void INIT_SIM808(void);

uint32_t send_at_command(const char *command, const char *response, uint32_t
    timeout);
uint32_t send_fast_at_command(const char *command, const char *response,
    uint32_t timeout);
uint32_t read_sms(uint32_t index);
void convert_gnss_string(uint8_t *p);
void print_gnss_string(void);
uint32_t update_location(void);
void update_csq(void);
void update_sim808_power_status(void);

struct GNSS {
  uint8_t buffer[32];
  uint8_t date[20];
  int32_t longitude;
  int32_t latitude;
  int32_t elevation;
  uint32_t speed;
  uint32_t direction;
  uint32_t hdop;
  uint32_t pdop;
  uint32_t vdop;
  uint32_t gps_sats_in_view;
  uint32_t gnss_sats_used;
  uint32_t glonass_sats_in_view; // never seen this..
  uint32_t cn_ratio;

  uint32_t tfff; // time for first fix

  uint8_t longitude_string[16];
  uint8_t latitude_string[16];
```

```
} volatile gnss;


struct SIM808{

  // UART buffer for receiving data from sim808
  uint8_t uart_buffer[256];
  uint32_t uart_sum;
  uint8_t google_link[128];
  uint8_t sms_buffer[64]; // length of one sms = 160
  uint32_t pwr_status; // power status of sim808
  uint32_t gnss_status; // power status of the inbuilt gnss module

  uint32_t csq; // carrier signal quality
  uint8_t csq_string[8];
  uint32_t csq_dbm;

} volatile sim808;


#endif /* SIM808_H_ */
```

8.2 c file

```
#include "sim808.h"
#include "uart_gnss_tracker.h" // send data to sim808 and debug
#include "gnss_tracker_2_4.h" // define
#include "rtc_gnss_tracker.h" // rtc delay
#include "accelereometer.h"
#include "string.h" // strstr()

static void convert_location_to_string(int32_t longitude, int32_t latitude);

void update_csq(void){
  while(!send_fast_at_command("AT+CSQ\r", "OK", 1000));
  // Search in buffer, set pointer if found
  uint8_t *p = strstr(sim808.uart_buffer, "CSQ:"); // this took 265 ticks

  if(p != NULL){
    p += 5; // move pointer where number starts

    // String
    sim808.csq_string[0] = p[0];
    sim808.csq_string[1] = p[1];
    sim808.csq_string[2] = NULL;

    // Integer
    sim808.csq = atoi(p); // 590 ticks
    sim808.csq_dbm = 113 - 2*sim808.csq;
  }
}

void update_sim808_power_status(void){
  sim808.pwr_status = ((PORT->Group[0].IN.reg & PORT_SIM808_STATUS) >>
      PIN_SIM808_STATUS);
}


//////////////////////////////////////////////////////////////
// SERCOM0_Handler
//////////////////////////////////////////////////////////////
```

```c
void SERCOM0_Handler(void){

  SERCOM0->USART.INTENCLR.reg = SERCOM_USART_INTENCLR_RXS; // Stop interrupt
  mcu_status.flag_new_data_in_buffer = TRUE;
  sim808.uart_sum = 0;

  uint32_t timeout_us;
  // timeout, f = 19200 bits/s => T = 52us
  if(mcu_status.flag_command_active == TRUE){
    timeout_us = 100000;
  } else{
    uint32_t timeout_us = 3000;
  }

  uint32_t index = 0;
  do {
    uint32_t t1 = RTC->MODE0.COUNT.reg; // Log start time

    // Wait for a character
    while( !(SERCOM0->USART.INTFLAG.reg & SERCOM_USART_INTFLAG_RXC) ){

      // Check if timeout
      if(RTC->MODE0.COUNT.reg > (t1 + timeout_us)){

        sim808.uart_buffer[index] = NULL; // End string

        mcu_status.flag_unexpected_answer = TRUE;

        // Check if the response was one of the "expected"
        uint32_t t = sim808.uart_sum;
        if (t == CFUN_READY_SUM || t == CPIN_READY_SUM || t == CALL_READY_SUM ||
            t == SMS_READY_SUM){
          // Clear flag
          SERCOM0->USART.INTFLAG.reg |= SERCOM_USART_INTFLAG_RXS;
          // Start interrupt
          SERCOM0->USART.INTENSET.reg = SERCOM_USART_INTENSET_RXS;
          return;
        }

        // we want to print this because: sim808 has sent an msg
        // to us! it's not an answer to a command that we're sending!
        print_pc_text("\n- flag_unexpected_answer\n");
        print_pc_text("=========================\n");
        print_pc_text(sim808.uart_buffer);
        print_pc_text("\n=========================\n");

        //print_pc_debug("UART SUM", sim808.uart_sum);


        mcu_status.error_sercom0_handler++;

        // Clear flag
        SERCOM0->USART.INTFLAG.reg |= SERCOM_USART_INTFLAG_RXS;
        // Start interrupt
        SERCOM0->USART.INTENSET.reg = SERCOM_USART_INTENSET_RXS;
        return;
      }
    }

    // Save data from sim808
    uint32_t temp = (SERCOM0->USART.DATA.reg) & (0xFF);
    sim808.uart_buffer[index] = temp;
    sim808.uart_sum += temp;


    if(index > 3){
      uint32_t a = sim808.uart_buffer[index] + sim808.uart_buffer[index - 1] +
         sim808.uart_buffer[index - 2];
      // CHECK FOR END FLAG "K\r\n" (OK<ENTER><NEW LINE>)
      if( a == ('\n' + '\r' + 'K') ){

        sim808.uart_buffer[index] = NULL;
        mcu_status.flag_command_answer = TRUE;

        // Clear flag
        SERCOM0->USART.INTFLAG.reg |= SERCOM_USART_INTFLAG_RXS;
        // Start interrupt
        SERCOM0->USART.INTENSET.reg = SERCOM_USART_INTENSET_RXS;
        return;
      }

    }
    // No end flag and there's time left.. one more run!
    index++;
  } while(1);
}

////////////////////////////////////////////////////////////
// INIT_SIM808
////////////////////////////////////////////////////////////
void INIT_SIM808(void){

  //  PORT settings
  PORT->Group[0].DIRSET.reg = PORT_SIM808_PWRKEY;
  PORT->Group[0].OUTCLR.reg = PORT_SIM808_PWRKEY;

  // Turn on 4V rail
  print_pc_text("-␣TURNING␣4V␣ON..");
  PORT->Group[0].OUTSET.reg = PORT_EN_4V;
  print_pc_text("OK!\n");

  // Wait until 4V is stable... which is?
  //delay_ms(100);

  // Starting SIM808
  print_pc_text("-␣STARTING␣SIM808..");
  while (!(PORT->Group[0].IN.reg & PORT_SIM808_STATUS)){
    print_pc_text("..toggle....");
    // Toggle PWRKEY for a second.
    PORT->Group[0].OUTSET.reg = PORT_SIM808_PWRKEY;
    delay_ms(1000);
    PORT->Group[0].OUTCLR.reg = PORT_SIM808_PWRKEY;
    delay_ms(1100);
  }
  print_pc_text("OK!\n");

  SERCOM0->USART.INTFLAG.reg |= SERCOM_USART_INTFLAG_RXS;
  NVIC_EnableIRQ(SERCOM0_IRQn);
  SERCOM0->USART.INTENSET.reg = SERCOM_USART_INTENSET_RXS; // Start of a frame
     detection

  delay_ms(700);
```

```
// Auto baud
while( !send_fast_at_command("AT\r", "OK", 1000) );
print_pc_text("-␣BAUD␣OK!\n");

uint32_t start_time = RTC->MODE0.COUNT.reg;
uint32_t break_true = 0;

// CFUN READY
do {
  if(mcu_status.flag_unexpected_answer){
    mcu_status.flag_unexpected_answer = FALSE;

    if (sim808.uart_sum == CFUN_READY_SUM){
      print_pc_text("-␣CFUN␣OK!\n");
      break_true = 1;
    }
  }
  if (break_true) break;
} while (RTC->MODE0.COUNT.reg < (start_time + 10000000ul));

// CPIN READY
break_true = 0;
do {
  if(mcu_status.flag_unexpected_answer){
    mcu_status.flag_unexpected_answer = FALSE;

    if (sim808.uart_sum == CPIN_READY_SUM){
      print_pc_text("-␣CPIN␣OK!\n");
      break_true = 1;
    }
  }
  if (break_true) break;
} while (RTC->MODE0.COUNT.reg < (start_time + 10000000ul));

// CALL READY
break_true = 0;
do {
  if(mcu_status.flag_unexpected_answer){
    mcu_status.flag_unexpected_answer = FALSE;

    if (sim808.uart_sum == CALL_READY_SUM){
      print_pc_text("-␣CALL␣OK!\n");
      break_true = 1;
    }
  }
  if (break_true) break;
} while (RTC->MODE0.COUNT.reg < (start_time + 10000000ul));

// SMS READY
break_true = 0;
do {
  if(mcu_status.flag_unexpected_answer){
    mcu_status.flag_unexpected_answer = FALSE;
    if (sim808.uart_sum == SMS_READY_SUM){
      print_pc_text("-␣SMS␣OK!\n");
      break_true = 1;
    }
  }
  if (break_true) break;
} while (RTC->MODE0.COUNT.reg < (start_time + 10000000ul));
```

```
  // sms format: text
  while(!send_fast_at_command(AT_SMS_MSG_FORMAT_TEXT, "OK", 1000));

  // Wait until we have a signal from the base station
  print_pc_text("Waiting␣for␣signal..");
  do {
    print_pc_text(".");
    delay_ms(1000);
    update_csq();
  } while (sim808.csq < 1);
  print_pc_text("OK!\n");
}

///////////////////////////////////////////////////////////////
// SEND AT COMMAND TO SIM808
///////////////////////////////////////////////////////////////
uint32_t send_at_command(const char *command, const char *response, uint32_t
    timeout){
  timeout = timeout*1000; //us to ms
  mcu_status.flag_command_active = TRUE;

  // DEBUG
  print_pc_text("\n-␣SENDING␣COMMAND...");

  // Send command to sim808
  print_sim808_text(command);

  // Log current time
  uint32_t log_time = RTC->MODE0.COUNT.reg;
  do
  {
    if (mcu_status.flag_new_data_in_buffer){
      mcu_status.flag_new_data_in_buffer = FALSE;

      // Debug Print
      print_pc_text("..␣FROM␣SIM808\n");
      print_pc_text("================================================================\
          n");
      print_pc_text(sim808.uart_buffer);
      print_pc_text("\n
          ================================================================\n\n\n");

      // How long does this actually take??
      if (strstr(sim808.uart_buffer, response) != NULL){
        mcu_status.flag_command_active = FALSE;

        return TRUE;
      }
    }
  } while (RTC->MODE0.COUNT.reg < (log_time + timeout));

  print_pc_text("=================================\n");
  print_pc_text("-␣ERROR␣NO␣RESPONCE\n");
  print_pc_text("=================================\n");
  mcu_status.error_no_responce++;
  mcu_status.error_counter++;
  mcu_status.flag_command_active = FALSE;
  return FALSE;
}
```

```
uint32_t send_fast_at_command(const char *command, const char *response,
    uint32_t timeout){
  timeout = timeout*1000; //us to ms

  mcu_status.flag_command_active = TRUE;

  // Send command to sim808
  print_sim808_text(command);

  // Log current time
  uint32_t log_time = RTC->MODE0.COUNT.reg;
  do
  {
    if (mcu_status.flag_new_data_in_buffer){
      mcu_status.flag_new_data_in_buffer = FALSE;

      if (strstr(sim808.uart_buffer, response) != NULL){
        mcu_status.flag_command_active = FALSE;
        return TRUE;
      }
    }
  } while (RTC->MODE0.COUNT.reg < (log_time + timeout));

  print_pc_text("========================================\n");
  print_pc_text("-_ERROR_NO_RESPONCE\n");
  print_pc_text("========================================\n");

  mcu_status.error_no_responce++;
  mcu_status.error_counter++;
  mcu_status.flag_command_active = FALSE;

  return FALSE;
}


////////////////////////////////////////////////////////////
// READ SMS AT INDEX
////////////////////////////////////////////////////////////
uint32_t read_sms(uint32_t index){

  print_pc_debug("\n\n-_READING_SMS_AT_INDEX", index);

  // Init stuff that we need
  uint32_t t1, t2;
  uint8_t temp_buffer[20];
  uint8_t temp_command[40];

  // Create AT Command with index and send it to SIM808.
  sprintf(temp_buffer, "AT+CMGR=%d\r", index);
  while (!send_at_command(temp_buffer, "OK", 1000));


  //////////////////////////////////////
  // 1. SEARCH FOR PHONE NUMBER
  //////////////////////////////////////
  // At the moment only Swedish phone numbers work... fix?
  uint8_t *p = strstr(sim808.uart_buffer, FLAG_PHONE_NUMBER);
  if(p == NULL){
    print_pc_text("-_NO_PHONE_NUMBER_FOUND!\n");
    delete_sms_at_index(index); // if we have a sms number but with no phone number, etc from the operator

    return FALSE;
  }

  // Copy phone number
  uint32_t a = 0;
  do {
    //sim808.phone_number[a] = *p;
    work[index].phone_number[a++] = *p++;

    if(a > 15){ // phone number length
      print_pc_text("- PHONE NUMBER TO BIG!\n");
      delete_sms_at_index(index);
      return FALSE;
    }

  } while (work[index].phone_number[a - 1] != '\"'); // phone number is always "xxx.."

   // Remove the " sign
  work[index].phone_number[a - 1] = NULL;


  ////////////////////////////////////
  // 2. SEARCH - START FLAG
  ////////////////////////////////////
  p = strstr(p, COMMAND_START_FLAG);
  if(p == NULL){
    print_pc_text(" - NO START FLAG FOUND!\n");
    delete_sms_at_index(index);
    return FALSE;
  }
  // Jump over start flag
  p += COMMAND_START_FLAG_LENGTH;

  ////////////////////////////////////
  // 3. SEARCH - END FLAG
  ////////////////////////////////////
  uint32_t i = 0;
  while(i != 40){ // Search length
    // Save data to temp buffer
    temp_command[i++] = *p++;
    // Check if the last character was the end flag.
    if (temp_command[i - 1] == COMMAND_END_FLAG){
      temp_command[i - 1] = 0; // Remove the end flag.
      break;
    }
    if(i > 39){ // Max length of a command
      print_pc_text("NO END FLAG!\n");
      delete_sms_at_index(index);
      return FALSE;
    }
  }

  // If we're here; phone number, start and end flag was found

  // DEBUG
  //print_pc_debug_string("phone number", sim808.phone_number);
```

```
//print_pc_debug_string("command (original)", sim808.command);

// Convert command to lower case
strlwr(temp_command);

// MONEY COMMAND
p = strstr(temp_command, "money");
if(p != NULL){
  p += 5; // 5 = length of "money"
  uint32_t var = atoi(p);  print_pc_debug("\n\n\nMONEY_COMMAND", var);
  eeprom.money = var*1000;
  eeprom.sms_sent = 0;
}

// COST COMMAND
p = strstr(temp_command, "cost");
if(p != NULL){
  p += 4; // 4 = length of "cost"
  uint32_t var = atoi(p);  print_pc_debug("\n\n\nCOST_COMMAND", var);
  eeprom.cost = var;
}

// SLEEP COMMAND
p = strstr(temp_command, "sleep");
if(p != NULL){
  p += 5; // 5 = length of "sleep"
  uint32_t var = atoi(p); print_pc_debug("\n\n\nSLEEP_COMMAND", var);
  if(var < 601) eeprom.sleep_time = var;
}

// OWNER COMMAND
p = strstr(temp_command, "owner");
if(p != NULL){
  //p += 5; // 5 = length of "owner"
  // Copy string
  memcpy(eeprom.owner_phone_number, work[index].phone_number, 16);
  print_pc_debug_string("\n\n\nOWNER_COMMAND", eeprom.owner_phone_number);
}

// ALARM COMMAND
p = strstr(temp_command, "alarm");
if(p != NULL){
  print_pc_text("\n\n\nALARM_COMMAND\n");
  if (eeprom.owner_phone_number[0] != NULL){
    INIT_ACCELEROMETER(); print_pc_text("INIT_ACCELEROMETER!\n");
    eeprom.alarm_active = 1;
  }
  else {
    print_pc_text("ERROR_NO_OWNER!\n");
  }
}

uint32_t command_length = strlen(temp_command);

// Take the sum of the command string and save it in WORK
for (uint32_t i = 0; i < command_length; i++){
  work[index].command += (uint32_t)temp_command[i];  // NEW
}

// DEBUG
print_pc_debug_string("␣-␣Command␣found", temp_command);
```

```
  print_pc_debug("␣-␣Command␣value", work[index].command);
  print_pc_debug("␣-␣Index", index);
  print_pc_debug_string("␣-␣Phone␣number", work[index].phone_number);

  return TRUE;
}


////////////////////////////////////////////////////////
// PRINT GNSS STRING (for debug)
////////////////////////////////////////////////////////
void print_gnss_string(void){
  print_pc_text("==================\n");
  print_pc_debug_string("-␣DATE", gnss.date);
  print_pc_debug("-␣LATITUDE", gnss.latitude);
  print_pc_debug("-␣LONGITUDE", gnss.longitude);
  print_pc_debug("-␣ELEVATION", gnss.elevation);
  print_pc_debug("-␣SPEED", gnss.speed);
  print_pc_debug("-␣DIRECTION", gnss.direction);
  print_pc_debug("-␣HDOP", gnss.hdop);
  print_pc_debug("-␣PDOP", gnss.pdop);
  print_pc_debug("-␣VDOP", gnss.vdop);
  print_pc_debug("-␣GPS␣SATS␣IN␣VIEW", gnss.gps_sats_in_view);
  print_pc_debug("-␣GNSS␣SATS␣USED", gnss.gnss_sats_used);
  print_pc_debug("-␣CN␣RATIO", gnss.cn_ratio);
  print_pc_text("==================\n");
}


////////////////////////////////////////////////////////
// CONVERT GNSS STRING
////////////////////////////////////////////////////////
void convert_gnss_string(uint8_t *p){
  // This functions breaks down the complex string we get from SIM808 when
     requesting location information.
  // This function took 8200 clk cycles

  uint8_t *p2; // temp pointer

  while(*p++ != ',');
  while(*p++ != ',');

  // Date
  p2 = gnss.date;
  do {
    *p2++ = *p;
  } while (*p++ != ',');
  *(--p2) = NULL; // Remove the ',' character.

  // Latitude
  p2 = gnss.buffer;
  do {
    if(*p != '.') *p2++ = *p;
  } while (*p++ != ',');
  *(--p2) = NULL; // Remove the ',' character.

  gnss.latitude = atoi(gnss.buffer);

  // Longitude
  p2 = gnss.buffer;
  do {
```

```
  if(*p != '.') *p2++ = *p;
} while (*p++ != ',');
*(--p2) = NULL; // Remove the ',' character.
gnss.longitude = atoi(gnss.buffer);

// Elevation [m]
p2 = gnss.buffer;
do {
  if(*p == '.') break;
  *p2++ = *p;
} while (*p++ != ',');
*(p2) = NULL; // End string
while(*p++ != ',');
gnss.elevation = atoi(gnss.buffer);

// Speed [m/s]
p2 = gnss.buffer;
do {
  if(*p == '.') break;
  *p2++ = *p;
} while (*p++ != ',');
*(p2) = NULL; // End string
while(*p++ != ',');
gnss.speed = atoi(gnss.buffer);


// Direction
p2 = gnss.buffer;
do {
  if(*p == '.') break;
  *p2++ = *p;
} while (*p++ != ',');
*(p2) = NULL; // End string
while(*p++ != ',');
gnss.direction = atoi(gnss.buffer);


// Skip over <fix mode> and <reserved1>
while(*p++ != ',');
while(*p++ != ',');

// HDOP
p2 = gnss.buffer;
if (*p == '0'){ // 0.x
  p += 2; // Skip the zero and dot
  *p2++ = *p;
  p += 2; //skip the end ','
}
else{ // x.xx
  do {
    if(*p != '.') *p2++ = *p;
  } while (*p++ != ',');
}
*p2 = NULL; // End string
gnss.hdop = atoi(gnss.buffer);

// PDOP
p2 = gnss.buffer;
if (*p == '0'){ // 0.x
  p += 2; // Skip the zero and dot
  *p2++ = *p;
```

```
    p += 2; //skip the end ','
  }
  else{ // x.xx
    do {
      if(*p != '.') *p2++ = *p;
    } while (*p++ != ',');
  }
  *p2 = NULL; // End string
  gnss.pdop = atoi(gnss.buffer);


  // VDOP
  p2 = gnss.buffer;
  if (*p == '0'){ // 0.x
    p += 2; // Skip the zero and dot
    *p2++ = *p;
    p += 2; //skip the end ','
  }
  else{ // x.xx
    do {
      if(*p != '.') *p2++ = *p;
    } while (*p++ != ',');
  }
  *p2 = NULL; // End string
  gnss.vdop = atoi(gnss.buffer);

  p++; // Skip the <Reserved2>

  // GPS SATS IN VIEW
  p2 = gnss.buffer;
  do {
    *p2++ = *p;
  } while (*p++ != ',');
  *p2 = NULL; // Remove the ',' character.
  gnss.gps_sats_in_view = atoi(gnss.buffer);

  // GNSS SATS USED
  p2 = gnss.buffer;
  do {
    *p2++ = *p;
  } while (*p++ != ',');
  *p2 = NULL; // Remove the ',' character.
  gnss.gnss_sats_used = atoi(gnss.buffer);


  while(*p++ != ',');
  while(*p++ != ',');

  // CN RATIO
  p2 = gnss.buffer;
  do {
    *p2++ = *p;
  } while (*p++ != ',');
  *p2 = NULL; // Remove the ',' character.
  gnss.cn_ratio = atoi(gnss.buffer);

}


////////////////////////////////////////////////////////////
// UPDATE LOCATION
```

```
////////////////////////////////////////////////////////////
uint32_t update_location(void){
  // Temp variables
  int32_t latitude_avg = 0;
  int32_t longitude_avg = 0;
  int32_t elevation_avg = 0;
  uint32_t speed_avg = 0;
  uint32_t hdop_avg = 0;
  uint32_t pdop_avg = 0;
  uint32_t vdop_avg = 0;
  uint32_t cn_ratio_avg = 0;
  uint32_t location_sum = 0;
  uint32_t location_counter = 0;

  // As with latitude and longitude, the values are bounded by 90 and 180
     respectively.
  // Latitude dd.dddddd   +-90
  // Longitude ddd.dddddd  +-180

  // Power on GNSS if it's off
  if (!sim808.gnss_status){
    while(!send_at_command("AT+CGNSPWR=1\r", "OK", 1000));
    sim808.gnss_status = TRUE;
    delay_ms(1000);
  }

  // Logging current time.
  uint32_t t0 = RTC->MODE0.COUNT.reg;

  while(!send_fast_at_command("AT+CGNSSEQ=\"RMC\"\r", "OK", 1000));
  uint8_t *p;
  while(location_counter != COOR_LENGTH){
    do{
      //delay_ms(100);
      while(!send_fast_at_command("AT+CGNSINF\r", "OK", 2000));

      // Debug
      print_pc_text(".");

      // check if we got a fix on the location.
      p = strstr(sim808.uart_buffer, "CGNSINF:␣1,1"); // how long does this take
          ?

      // Extra safety check, if this happens then something is fucked up!
      if (strstr(sim808.uart_buffer, "CGNSINF:␣0") != NULL){
        print_pc_text("\n\n\n\n\nERROR:␣GPS␣IS␣OFF!\n\n\n\n\n\n\n\n\n\n");
        sim808.gnss_status = FALSE;
        mcu_status.error_counter += 1000;
        return FALSE;
      }

      // Check for timeout
      if ( (RTC->MODE0.COUNT.reg - t0) > ( 8*60*1000000 )){ // 300000000ul = 5
          min
        return FALSE;
      }
    } while (p == NULL);

    convert_gnss_string(sim808.uart_buffer);

    uint32_t temp0 = gnss.latitude + gnss.longitude;
    if( temp0 != location_sum){
      location_counter++;
      location_sum = temp0;
      print_pc_dec(location_counter);

      // Take the sum of everything!
      latitude_avg += gnss.latitude;
      longitude_avg += gnss.longitude;
      elevation_avg += gnss.elevation;
      speed_avg += gnss.speed;
      cn_ratio_avg += gnss.cn_ratio;

      hdop_avg += gnss.hdop;
      pdop_avg += gnss.pdop;
      vdop_avg += gnss.vdop;
    }
    else {
      location_sum = 0;
      print_pc_text("x");
    }
  } // End while loop
  print_pc_text("\n");

  // If we're here; we got 10 fixes!
  gnss.tfff = ((RTC->MODE0.COUNT.reg-t0)/1000000);

  // Average
  latitude_avg = latitude_avg/COOR_LENGTH;
  longitude_avg = longitude_avg/COOR_LENGTH;
  elevation_avg = elevation_avg/COOR_LENGTH;
  speed_avg = speed_avg/COOR_LENGTH;
  cn_ratio_avg = cn_ratio_avg/COOR_LENGTH;
  hdop_avg = hdop_avg/COOR_LENGTH;
  pdop_avg = pdop_avg/COOR_LENGTH;
  vdop_avg = vdop_avg/COOR_LENGTH;

  // DEBUG; print time
  print_pc_text("\n");
  print_pc_debug("-␣TIME_FOR_FIX␣[s]", gnss.tfff);
  print_pc_text("\n");

  print_gnss_string();

  // Convert the average coordinates to a  string
  convert_location_to_string(latitude_avg, longitude_avg);

  // Creating google maps link.
  sprintf(sim808.google_link, "\rhttps://www.google.com/maps/search/%s,%s", gnss
     .latitude_string, gnss.longitude_string);

  // Debug
  print_pc_text(gnss.latitude_string);
  print_pc_text(",");
  print_pc_text(gnss.longitude_string);
  print_pc_text("\n");
  print_pc_text(sim808.google_link);
  print_pc_text("\n\n");

  return TRUE;
}
```

```
static void convert_location_to_string(int32_t latitude, int32_t longitude){
  // CONVERT TO STRING
  itoa(latitude, gnss.latitude_string, 10);
  itoa(longitude, gnss.longitude_string, 10);

  // LATITUDE
  if(latitude > 0 ){ // Positive degrees
    if (latitude < 10000000){ // x.yyyyyy
      // 1223344 => 1.223344
      gnss.latitude_string[7] = gnss.latitude_string[6];
      gnss.latitude_string[6] = gnss.latitude_string[5];
      gnss.latitude_string[5] = gnss.latitude_string[4];
      gnss.latitude_string[4] = gnss.latitude_string[3];
      gnss.latitude_string[3] = gnss.latitude_string[2];
      gnss.latitude_string[2] = gnss.latitude_string[1];
      gnss.latitude_string[1] = '.';
      gnss.latitude_string[8] = NULL;
    }
    else { // xx.yyyyyy
      gnss.latitude_string[8] = gnss.latitude_string[7];
      gnss.latitude_string[7] = gnss.latitude_string[6];
      gnss.latitude_string[6] = gnss.latitude_string[5];
      gnss.latitude_string[5] = gnss.latitude_string[4];
      gnss.latitude_string[4] = gnss.latitude_string[3];
      gnss.latitude_string[3] = gnss.latitude_string[2];
      gnss.latitude_string[2] = '.';
      gnss.latitude_string[9] = NULL;
    }
  }
  else { // Negative degrees
    if (latitude < -10000000){ // -x.yyyyyy
      gnss.latitude_string[8] = gnss.latitude_string[7];
      gnss.latitude_string[7] = gnss.latitude_string[6];
      gnss.latitude_string[6] = gnss.latitude_string[5];
      gnss.latitude_string[5] = gnss.latitude_string[4];
      gnss.latitude_string[4] = gnss.latitude_string[3];
      gnss.latitude_string[3] = gnss.latitude_string[2];
      gnss.latitude_string[2] = '.';
      gnss.latitude_string[9] = NULL;
    }
    else { // -xx.yyyyyy
      gnss.latitude_string[9] = gnss.latitude_string[8];
      gnss.latitude_string[8] = gnss.latitude_string[7];
      gnss.latitude_string[7] = gnss.latitude_string[6];
      gnss.latitude_string[6] = gnss.latitude_string[5];
      gnss.latitude_string[5] = gnss.latitude_string[4];
      gnss.latitude_string[4] = gnss.latitude_string[3];
      gnss.latitude_string[3] = '.';
      gnss.latitude_string[10] = NULL;
    }
  } // End latitude

  // LONGITUDE
  if(longitude > 0 ){ // Positive degrees
    if (longitude < 10000000){ // x.yyyyyy
      // 1223344 => 1.223344
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = gnss.longitude_string[3];
      gnss.longitude_string[3] = gnss.longitude_string[2];
      gnss.longitude_string[2] = gnss.longitude_string[1];
      gnss.longitude_string[1] = '.';
      gnss.longitude_string[8] = NULL;
    }
    else if (longitude > 100000000){ // xxx.yyyyyy
      // 111223344 => 111.223344
      gnss.longitude_string[9] = gnss.longitude_string[8];
      gnss.longitude_string[8] = gnss.longitude_string[7];
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = gnss.longitude_string[3];
      gnss.longitude_string[3] = '.';
      gnss.longitude_string[10] = NULL;
    }
    else { // xx.yyyyyy
      gnss.longitude_string[8] = gnss.longitude_string[7];
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = gnss.longitude_string[3];
      gnss.longitude_string[3] = gnss.longitude_string[2];
      gnss.longitude_string[2] = '.';
      gnss.longitude_string[9] = NULL;
    }
  }
  else { // Negative degrees
    if (longitude < -10000000){ // -x.yyyyyy
      gnss.longitude_string[8] = gnss.longitude_string[7];
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = gnss.longitude_string[3];
      gnss.longitude_string[3] = gnss.longitude_string[2];
      gnss.longitude_string[2] = '.';
      gnss.longitude_string[9] = NULL;
    }
    else if (longitude < -100000000){ // x-xx.yyyyyy
      // -111223344 => -111.223344
      gnss.longitude_string[10] = gnss.longitude_string[9];
      gnss.longitude_string[9] = gnss.longitude_string[8];
      gnss.longitude_string[8] = gnss.longitude_string[7];
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = '.';
      gnss.longitude_string[11] = NULL;
    }
    else { // -xx.yyyyyy
      gnss.longitude_string[9] = gnss.longitude_string[8];
      gnss.longitude_string[8] = gnss.longitude_string[7];
      gnss.longitude_string[7] = gnss.longitude_string[6];
      gnss.longitude_string[6] = gnss.longitude_string[5];
      gnss.longitude_string[5] = gnss.longitude_string[4];
      gnss.longitude_string[4] = gnss.longitude_string[3];
      gnss.longitude_string[3] = '.';
      gnss.longitude_string[10] = NULL;
    }
  } // End longitude
}
```

9 gnss_tracker_2_4

9.1 header file

```
#ifndef GNSS_TRACKER_2_4_H_
#define GNSS_TRACKER_2_4_H_

#include "asf.h"

void delay_ms(volatile uint32_t time);
void delay_10us(volatile uint32_t time);

void print_status_all(void);
void update_bat_voltage(void);

void delete_sms_at_index(uint32_t index);
void print_sim808_sms_debug(void);

void hi_command(uint32_t index);
void gps_command(uint32_t index);
void taken_command(uint32_t index);
void help_command(uint32_t index);
void taken_command(uint32_t index);
void alarm_command(void);
void low_bat_command(void);

void do_work(void);
void get_work(void);


#define  SOFTWARE_VERSION ("20180218")
#define COOR_LENGTH  10
#define  WORK_SIZE  20

struct EEPROM {
  uint16_t sms_sent;
  uint8_t data_key;
  uint8_t flag_low_bat;
  uint16_t tot_errors;
  uint16_t cycles;
  uint16_t sleep_time;
  uint16_t cost;
  uint32_t money;
  uint8_t owner_phone_number[16];
  uint8_t alarm_active;
  uint8_t alarm_triggerd;
  uint8_t low_bat_flag;
  uint8_t dummy3;
  uint32_t dummy_mem[7];
} volatile eeprom;


struct MCU_STATUS {
  // FLAGS
  uint8_t flag_command_active;
  uint8_t flag_new_data_in_buffer;
  uint8_t flag_unexpected_answer;
  uint8_t flag_command_answer;

  uint32_t uptime;  // uptime in seconds
  // BATTERY
  uint32_t bat_volt;
  uint8_t bat_volt_string[10];
  uint8_t bat_voltage_procent;

  // SOFTWARE ERROR
  uint32_t error_no_responce;
  uint32_t error_counter;
  uint32_t error_gps_glitch;
  uint32_t error_sercom0_handler;

  uint32_t command_input_variable;
}  volatile mcu_status;



struct WORK {
  uint8_t phone_number[16];
  uint32_t index;
  uint32_t command;
} volatile work[WORK_SIZE];



// ==========================================
// ================= DEFINE =================
// ==========================================

// COMMANDS
#define HI_COMMAND     (   ((uint32_t)'h') + ((uint32_t)'i')   )
#define GPS_COMMAND    (   ((uint32_t)'g') + ((uint32_t)'p') + ((uint32_t)'s')   )
#define HELP_COMMAND   (   ((uint32_t)'h') + ((uint32_t)'e') + ((uint32_t)'l') +
    ((uint32_t)'p')  )
#define TAKEN_COMMAND (   ((uint32_t)'t') + ((uint32_t)'a') + ((uint32_t)'k') +
    ((uint32_t)'e') + ((uint32_t)'n')   )
#define DEBUG_COMMAND (   ((uint32_t)'d') + ((uint32_t)'e') + ((uint32_t)'b') +
    ((uint32_t)'u') + ((uint32_t)'g')   )
#define ALARM_COMMAND (   ((uint32_t)'a') + ((uint32_t)'l') + ((uint32_t)'a') +
    ((uint32_t)'r') + ((uint32_t)'m')   )


// FLAGS
#define FALSE       (0)
#define TRUE        (1)


// ==========================================
// ================= IO  PINS ===============
// ==========================================

// ACCELEROMETER
#define PORT_ACC_CLK          (PORT_PA23)
#define PIN_ACC_CLK           (PIN_PA23)
#define PORT_ACC_DATA_OUT       (PORT_PA22)
#define PIN_ACC_DATA_OUT        (PIN_PA22)
#define PORT_ACC_DATA_IN        (PORT_PA17)
#define PIN_ACC_DATA_IN         (PIN_PA17)
#define PORT_ACC_CS           (PORT_PA16)
#define PIN_ACC_CS            (PIN_PA16)
#define PORT_ACC_INT1         (PORT_PA27)
#define PIN_ACC_INT1          (PIN_PA27)
```

```
#define PORT_ACC_INT2       (PORT_PA15)
#define PIN_ACC_INT2        (PIN_PA15)

// 4V EN
#define PORT_EN_4V          (PORT_PA05)
#define PIN_EN_4V       (PIN_PA05)

// ADC - Analog input 5
#define PORT_ADC        (PORT_PA03)
#define PIN_ADC         (PIN_PA03)

// SIM808
#define PORT_SIM808_TX          (PORT_PA10)
#define PIN_SIM808_TX       (PIN_PA10)

#define PORT_SIM808_RX          (PORT_PA11)
#define PIN_SIM808_RX       (PIN_PA11)

#define PORT_PC_TX          (PORT_PA24)
#define PIN_PC_TX       (PIN_PA24)

#define PORT_SIM808_RESET     (PORT_PA08)
#define PIN_SIM808_RESET      (PIN_PA08)

#define PORT_SIM808_RTS       (PORT_PA09)
#define PIN_SIM808_RTS        (PIN_PA09)

#define PORT_SIM808_STATUS      (PORT_PA07)
#define PIN_SIM808_STATUS     (PIN_PA07)

#define PORT_SIM808_PWRKEY      (PORT_PA14)
#define PIN_SIM808_PWRKEY     (PIN_PA14)


#endif /* GNSS_TRACKER_2_4_H_ */
```

9.2 c file

```
#include "gnss_tracker_2_4.h"
#include "sim808.h"
#include "uart_gnss_tracker.h"
#include "adc_gnss_tracker.h"

#include "string.h" // strstr()


void debug_command(uint32_t index);




//////////////////////////////////////////////
// UPDATE BATTARY VOLTAGE
//////////////////////////////////////////////
void update_bat_voltage(void){
  // Read battery voltage
  mcu_status.bat_volt = (adc_read()*5270)/4095;

  // Convert to % scale..
  if (mcu_status.bat_volt > 3300){
    mcu_status.bat_voltage_procent = (mcu_status.bat_volt - 3300)/9; // 3300;
        lower limit. 4200; upper limit. divide by 9 to get %
  } else {
    mcu_status.bat_voltage_procent = 0;
  }

  // Convert voltage from int to string
  uint8_t *text = mcu_status.bat_volt_string;
  itoa(mcu_status.bat_volt, text, 10);
  text[3] = text[2];
  text[2] = text[1];
  text[1] = '.';
  text[4] = 'V';
  text[5] = NULL;

  print_pc_text("- ADC READ OK!\n");
}



///////////////////////////////////////////////
// DEBUG PRINT, EVERYTHING FROM STATUS
///////////////////////////////////////////////
void print_status_all(void){
  update_sim808_power_status();

  if (sim808.pwr_status)  update_csq();

  print_pc_text("\n==================== STATUS ==================\n");
  print_pc_debug("- Uptime [s]", mcu_status.uptime);
  print_pc_debug("- Battery", mcu_status.bat_volt);
  print_pc_debug("- SIM808 status", sim808.pwr_status);
  print_pc_debug("- Error no response", mcu_status.error_no_responce);
  print_pc_debug("- Error GNSS glitch", mcu_status.error_gps_glitch);
  print_pc_debug("- Error error_sercom0_handler", mcu_status.
     error_sercom0_handler);
  print_pc_debug("- Error counter", mcu_status.error_counter);
  print_pc_debug("- Signal CSQ (-dBm)", sim808.csq_dbm);

  print_pc_text("\n===== EEPROM =====\n");
  // State/errors/flags
  print_pc_debug("- CYCLES", eeprom.cycles);
  print_pc_debug("- SLEEP", eeprom.sleep_time);
  print_pc_debug("- DATA KEY", eeprom.data_key);
  print_pc_debug("- TOT ERRORS", eeprom.tot_errors);
  print_pc_debug("- ALARM ACTIVE", eeprom.alarm_active);
  print_pc_debug("- ALARM TRIGGERD", eeprom.alarm_triggerd);
  print_pc_debug("- LOW BAT", eeprom.low_bat_flag);
  // User config
  print_pc_debug_string("\n- OWNER", eeprom.owner_phone_number);
  print_pc_debug("- MONEY", eeprom.money);
  print_pc_debug("- COST", eeprom.cost);
  print_pc_debug("- SMS SENT", eeprom.sms_sent);

  print_pc_text("=================== END =================\n\n");
}


///////////////////////////////////////////////
// GET WORK
///////////////////////////////////////////////
void get_work(void){
  uint32_t extra_sms = 4;
  for (uint32_t i = 1; i < extra_sms; i++){
    if(read_sms(i)){
      extra_sms++;
    }
  }
}


////////////////////////////////////////////////
// DO WORK
////////////////////////////////////////////////
void do_work(void){

  get_work();
  print_pc_text("\n\n==========␣DOING␣WORK␣==========\n\n");

  // Go trough work list.
  for (int i = 1; i < WORK_SIZE; i++){

    uint32_t command = work[i].command;

    if (command == HI_COMMAND){
      print_pc_debug("−␣HI␣COMMAND␣FOUND,␣INDEX", i);
      eeprom.money = eeprom.money − eeprom.cost;
      eeprom.sms_sent++;
      hi_command(i);
    }
    else if (command == GPS_COMMAND){
      print_pc_debug("−␣GPS␣COMMAND␣FOUND,␣INDEX", i);
      eeprom.money = eeprom.money − eeprom.cost;
      eeprom.sms_sent++;
      gps_command(i);
    }
    else if (command == HELP_COMMAND){
      print_pc_debug("−␣HELP␣COMMAND␣FOUND,␣INDEX", i);
      eeprom.money = eeprom.money − eeprom.cost;
      eeprom.sms_sent++;
      help_command(i);
    }
    else if (command == TAKEN_COMMAND){
      print_pc_debug("−␣TAKEN␣COMMAND␣FOUND,␣INDEX", i);
      eeprom.money = eeprom.money − eeprom.cost;
      eeprom.sms_sent++;
      taken_command(i);
    }
    else if (command == DEBUG_COMMAND){
      print_pc_debug("−␣DEBUG␣COMMAND␣FOUND,␣INDEX", i);
      debug_command(i);
    }


    // True if start and end flag was found but corrupt command value
    if(command > 0){
      delete_sms_at_index(i);
    }
    work[i].command = 0; // clear command
  }
  print_pc_text("\n\n==========␣WORK␣DONE␣==========\n\n");
}
```

```
////////////////////////////////////////////////
// HI COMMAND
////////////////////////////////////////////////
void hi_command(uint32_t index){
  update_csq();

  sprintf(sim808.sms_buffer, "AT+CMGS=\"%s\"\r", work[index].phone_number);
  uint32_t ret;
  mcu_status.flag_command_active = TRUE;


  do {
    // Part 1: AT+CMGS=<PhoneNumber>
    print_sim808_text(sim808.sms_buffer);
    delay_ms(100);
    // Part 2: <SMS text>
    print_sim808_text("Hi!␣:)");
    print_sim808_sms_debug();

    delay_ms(100);
    // Part 3: <Send SMS>
    ret = send_at_command("\x1A\r", "OK", 10000);

    if (ret) {
      mcu_status.flag_command_active = FALSE;
      return;
    }

    mcu_status.error_counter++;
  } while(!ret);
}


////////////////////////////////////////////////
// GPS COMMAND
////////////////////////////////////////////////
void gps_command(uint32_t index){

  sprintf(sim808.sms_buffer, "AT+CMGS=\"%s\"\r", work[index].phone_number);
  uint32_t ret;

  print_pc_debug_string("work_number", work[index].phone_number);


  // UPDATE LOCATION (if we fail; send error sms to caller)
  if(update_location() == FALSE){
    mcu_status.flag_command_active = TRUE;
    // If we're here; timeout in update location, we can't get a position fix!
    do {
      // Part 1 : AT+CMGS=<currentPhoneNumber>
      print_sim808_text(sim808.sms_buffer);
      delay_ms(100);

      // Part 2 : <SMS text>
      print_sim808_text("ERROR!␣Can't␣get␣position!␣(timeout)\n");
      print_sim808_sms_debug();

      // Part 3 : <Send SMS>
      delay_ms(100);
      ret = send_at_command("\x1A\r", "OK", 5000);
      if (ret){
        mcu_status.flag_command_active = FALSE;
        return;
      }
      mcu_status.error_counter++;
    } while (1);
  }

  update_csq();

  // SEND LOCATION TO CALLER
  mcu_status.flag_command_active = TRUE;
  do {
    // Part 1: AT+CMGS=<currentPhoneNumber>
    print_sim808_text(sim808.sms_buffer);
    delay_ms(100);
    // Part 2: <SMS text>
    print_sim808_text(sim808.google_link);
    print_sim808_sms_debug();

    // Part 3: <Send SMS>
    delay_ms(100);
    ret = send_at_command("\x1A\r", "OK", 5000);

    if (ret) {
      mcu_status.flag_command_active = FALSE;
      return;
    }
    mcu_status.error_counter++;
  } while(1);
}

//////////////////////////////////////////////
// HELP COMMAND
//////////////////////////////////////////////
void help_command(uint32_t index){
  sprintf(sim808.sms_buffer, "AT+CMGS=\"%s\"\r", work[index].phone_number);

  update_csq();

  // Send a sms to caller
  uint32_t ret;
  mcu_status.flag_command_active = TRUE;
  do {
    // Part 1: AT+CMGS=<PhoneNumber>
    print_sim808_text(sim808.sms_buffer);
    delay_ms(100);
    // Part 2: <SMS text>
    print_sim808_text("Commands:␣\"hi\"␣\"help\"␣\"gps\"␣\n\n");

    // Battery
    print_sim808_text("\nBattery␣voltage:␣");
    print_sim808_text(mcu_status.bat_volt_string);
    print_sim808_text("␣(");
    print_sim808_dec(mcu_status.bat_voltage_procent);
    print_sim808_text("\%)");

    // CSQ
    print_sim808_text("\nCSQ:␣");
    print_sim808_text(sim808.csq_string);
    print_sim808_text("␣(-");
    print_sim808_dec(sim808.csq_dbm);
    print_sim808_text("dBm)");

    // Part 3: <Send SMS>
    delay_ms(100);
    ret = send_at_command("\x1A\r", "OK", 10000);

    if (ret){
      mcu_status.flag_command_active = FALSE;
      return;
    }
    mcu_status.error_counter++;
  } while(1);
}



///////////////////////////////////////////////
// TAKEN COMMAND
///////////////////////////////////////////////
void taken_command(uint32_t index){

}


///////////////////////////////////////////////
// ALARM COMMAND
///////////////////////////////////////////////
void alarm_command(void){
  update_csq();

  sprintf(sim808.sms_buffer, "AT+CMGS=\"%s\"\r", eeprom.owner_phone_number);
  uint32_t ret;
  mcu_status.flag_command_active = TRUE;


  do {
    // Part 1: AT+CMGS=<PhoneNumber>
    print_sim808_text(sim808.sms_buffer);
    delay_ms(100);
    // Part 2: <SMS text>
    print_sim808_text("Alarm!");
    print_sim808_sms_debug();

    delay_ms(100);
    // Part 3: <Send SMS>
    ret = send_at_command("\x1A\r", "OK", 10000);

    if (ret) {

      // now send locations
      print_pc_text("SMS␣sent,␣getting␣location..\n");

      // Convert number int to string.
      memcpy(work[WORK_SIZE - 1].phone_number, eeprom.owner_phone_number, 16);
      gps_command(WORK_SIZE - 1);

      print_pc_text("Location␣done..\n");

      // stuck here...
      mcu_status.flag_command_active = FALSE;
      return;
    }

    mcu_status.error_counter++;
  } while(!ret);
}



/////////////////////////////////////////////////
// LOW BATTERY COMMAND
/////////////////////////////////////////////////
void low_bat_command(void){
  update_csq();

  sprintf(sim808.sms_buffer, "AT+CMGS=\"%s\"\r", eeprom.owner_phone_number);
  uint32_t ret;
  mcu_status.flag_command_active = TRUE;


  do {
    // Part 1: AT+CMGS=<PhoneNumber>
    print_sim808_text(sim808.sms_buffer);
    delay_ms(100);
    // Part 2: <SMS text>
    print_sim808_text("Low␣battery!");
    print_sim808_sms_debug();

    delay_ms(100);
    // Part 3: <Send SMS>
    ret = send_at_command("\x1A\r", "OK", 10000);

    if (ret) {
      mcu_status.flag_command_active = FALSE;
      return;
    }

    mcu_status.error_counter++;
  } while(!ret);
}



/////////////////////////////////////////////////
// DEBUG COMMAND
/////////////////////////////////////////////////
void debug_command(uint32_t index){
  print_pc_text("\n\n\n\n\n\nDEBUG␣COMMAND\n\n\n\n\n\n\n\n");
}




void delete_sms_at_index(uint32_t index){
  // AT+CMGD=<INDEX> // Delete sms at index

  print_pc_debug("Delete␣sms␣at", index);

  uint8_t temp_buffer[20];
  // Create AT command with given index
  sprintf(temp_buffer, "AT+CMGD=%d\r", index);

  // Send command
  while(!send_fast_at_command(temp_buffer, "OK", 10000));
}


void print_sim808_sms_debug(void){

  // Battery
  print_sim808_text("\nBat:␣");
  print_sim808_text(mcu_status.bat_volt_string);
  print_sim808_text("␣(");
  print_sim808_dec(mcu_status.bat_voltage_procent);
  print_sim808_text("\%)");

  // CSQ
  print_sim808_text("\nCSQ:␣-");
  print_sim808_dec(sim808.csq_dbm);
  print_sim808_text("␣dBm");

  // CN Ratio
  print_sim808_text("\nCN:␣");
  print_sim808_dec(gnss.cn_ratio);

  // Elevation
  print_sim808_text("\nZ:␣");
  print_sim808_dec(gnss.elevation);
  print_sim808_text("␣m");

  // Time for first fix
  print_sim808_text("\nT:␣");
  print_sim808_dec(gnss.tfff);
  print_sim808_text("␣s\n");

  // money / sms
  print_sim808_text("Money:␣");
  print_sim808_dec((eeprom.money/1000));
  print_sim808_text("\n");
  print_sim808_text("SMS:␣");
  print_sim808_dec(eeprom.sms_sent);
  print_sim808_text("\n");
  print_sim808_text("Sleep:␣");
  print_sim808_dec(eeprom.sleep_time);
  print_sim808_text("\n");
}
```