1. Introduction of SCH1600 basic demo

Purpose of the demo is to help customers in developing application code for the Murata SCH1600 series sensors. Demo utilizes STM32 Nucleo board with SMT32F401RE MCU. The application is built with Keil uVision IDE version 5.39, with Keil STM32F4xx_DFP pack version 2.17.1 applied.

By using this demo the user application can be developed faster and most common pitfalls can be avoided. Demo has the following functionalities:

- Demo board hardware initialization, including system clock, SPI, UART and timers. UART baud rate is 460800.

- SCH1600 initialization

- Reading the sensor with 48-bit SPI frames and 20-bit data precision at 1000 Hz and averaging the data to get 100 Hz Output Data Rate (ODR).

- Reading temperature from the sensor.

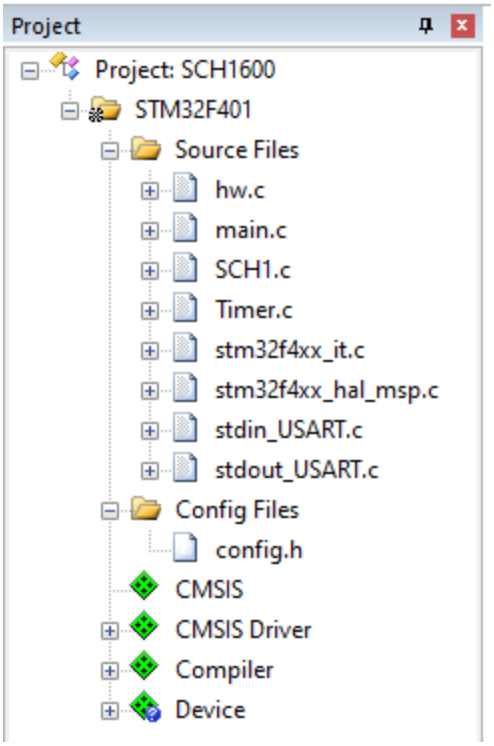
- Send averaged data via UART to Client application.

- Checking possible sensor errors.

2. Demo project structure

To be able to run Murata example code at customer target system, the hardware dependent routines in the demo are placed in file *hw.c* for easier portability.

Keil uVision project view:



3. Functionality at Main level

Main loop of the SCH1600 basic demo.

```
int main(void)
{
    char serial_num[15];
    int  init_status;
    SCH1_filter         Filter;
    SCH1_sensitivity    Sensitivity;
    SCH1_decimation     Decimation;

    // STM32F4xx HAL initialization
    HAL_Init();

    // Configure system clock to 80 MHz.
    SystemClock_Config();
    SystemCoreClockUpdate();

    // Initialize IO-redirection
    stdout_init();

    // Initialize demo board HW (STM Nucleo-F401RE)
    hw_init();


    // SCH1600 settings and initialization
    //------------------------------------

    // SCH1600 filter settings
    Filter.Rate12 = FILTER_RATE;
    Filter.Acc12  = FILTER_ACC12;
    Filter.Acc3   = FILTER_ACC3;

    // SCH1600 sensitivity settings
    Sensitivity.Rate1 = SENSITIVITY_RATE1;
    Sensitivity.Rate2 = SENSITIVITY_RATE2;
    Sensitivity.Acc1  = SENSITIVITY_ACC1;
    Sensitivity.Acc2  = SENSITIVITY_ACC2;
    Sensitivity.Acc3  = SENSITIVITY_ACC3;

    // SCH1600 decimation settings (for Rate2 and Acc2 channels).
    Decimation.Rate2 = DECIMATION_RATE;
    Decimation.Acc2  = DECIMATION_ACC;

    // Initialize the sensor
    init_status = SCH1_init(Filter, Sensitivity, Decimation, false);
    if (init_status != SCH1_OK) {
        printf("ERROR: SCH1_init failed with code: %d\r\nApplication halted\r\n", init_status);
        HAL_Delay(10);
        while (true);
    }

    // Read serial number from the sensor.
    strcpy(serial_num, SCH1_getSnbr());
    printf("Serial number: %s\r\n\r\n", serial_num);


    // Start sampling timer at 1000 Hz
    //--------------------------------

    // With 1000 Hz sample rate and 10x averaging we get Output Data Rate (ODR) of 100 Hz.
    sample_timer_init(sampling_callback, 1000);
```

```
    // Main loop
    while (true)
    {
        // Check if sampling_callback has new summed raw data for us to process.
        if (new_summed_data_available()) {
            // Start critical section

            // Convert summed raw data from the ring buffer & calculate averages.
            SCH1_result Data;
            SCH1_convert_data(&SCH1_summed_data_buffer[ring_buffer_idx_rd], &Data);

            // End critical section
            inc_ring_buffer_rd_idx();

            // Send converted data to UART
            printf("GYRO(X,Y,Z)[dps]:%+.3f,%+.3f,%+.3f\t"
                   "ACC(X,Y,Z)[m/s2]:%+.3f,%+.3f,%+.3f\t"
                   "T[C]:%+.1f\r\n",
                   Data.Rate1[AXIS_X], Data.Rate1[AXIS_Y], Data.Rate1[AXIS_Z],
                   Data.Acc1[AXIS_X], Data.Acc1[AXIS_Y], Data.Acc1[AXIS_Z],
                   Data.Temp);
        }

        // Handle possible sensor errors
        if (SCH1_error_available) {

            // Error from sensor. Stop sample timer to avoid race condition when reading status.
            sample_timer_stop();
            SCH1_error_available = false;

            // Read SCH1600 status registers
            SCH1_status Status;
            SCH1_getStatus(&Status);

            // Send status data to UART. Status registers are read in sampling_callback()
            printf("STATUS:\tSUM:%4x\tSUM_SAT:%4x\tCOM:%4x\t"
                   "RATE_COM:%4x\tRATE_X:%4x\tRATE_Y:%4x\tRATE_Z:%4x\t"
                   "ACC_X:%4x\tACC_Y:%4x\tACC_Z:%4x\r\n",
                Status.Summary, Status.Summary_Sat, Status.Common,
                Status.Rate_Common, Status.Rate_X, Status.Rate_Y, Status.Rate_Z,
                Status.Acc_X, Status.Acc_Y, Status.Acc_Z);

            // Restart sampling timer
            sample_timer_start();
        }
    } // while (true)
}
```

4. Functional structure

File *config.h* is used for setting demo project and SCH1600 sensor operating parameters.

Main loop of the demo is placed in file *main.c*. At the beginning, the demo performs mandatory initializations before entering the eternal loop.

hw_init() function provides initialization of low level hardware of the demo board. It initializes system clocks, SPI, UART, timers and I/O. Clients may need to modify this code to adapt for their specific hardware, please refer to file *hw.c*.

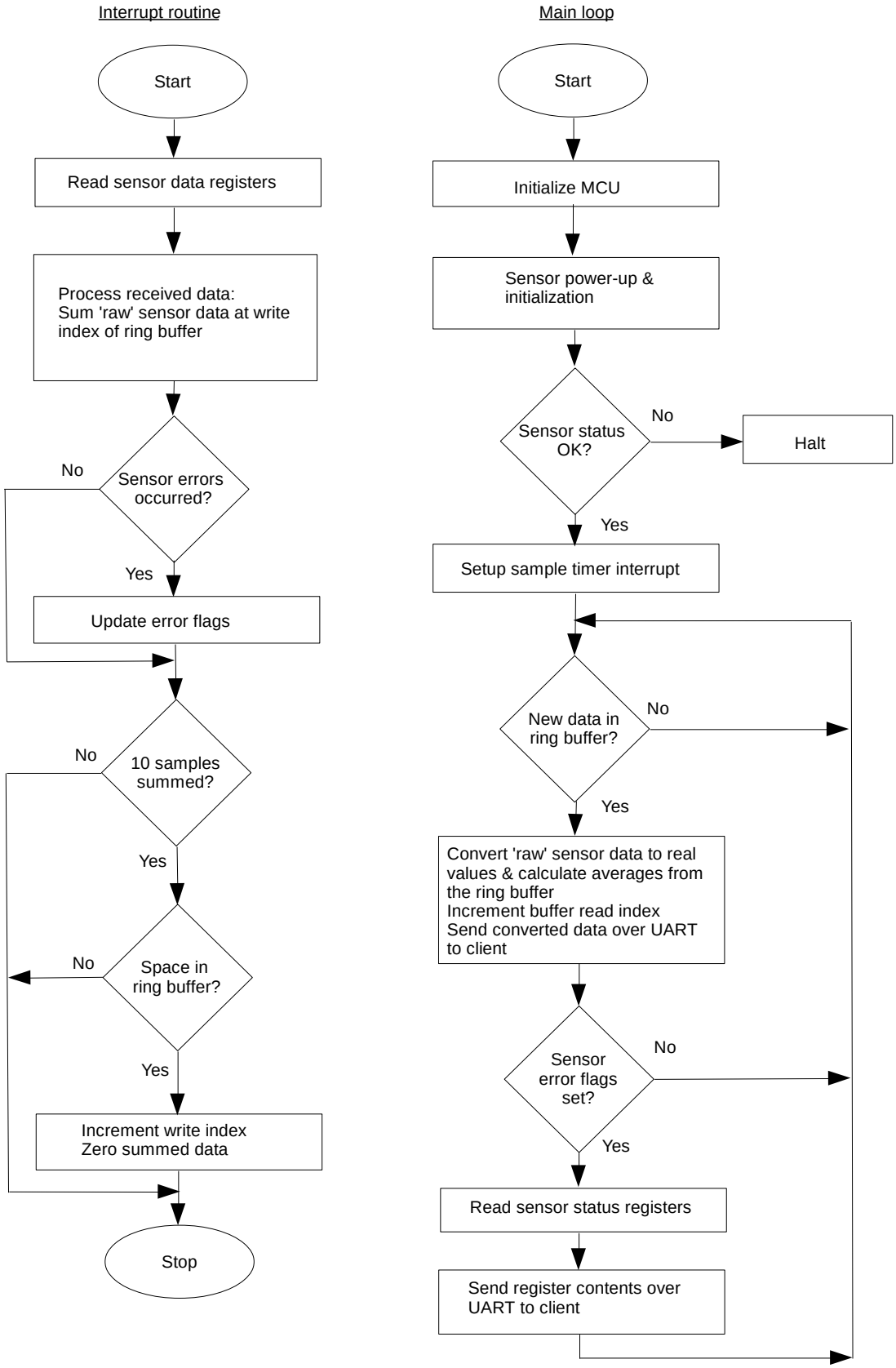
SCH1600 initialization is done in function *sch1_init().* File *sch1.c* contains this function as well as other sensor related operations.

Function *sample_timer_init()* sets up TIM2 timer for sampling at 1000 Hz. It also sets a callback function for data sampling, to be called from TIM2 timer interrupt service routine. All timer related functions are placed in file *Timer.c*. Data rate is reduced to 100 Hz with 10x client averaging.

Sending data to UART is done using I/O Retargeting. This allows easy printf-style formatted data printout from application level.

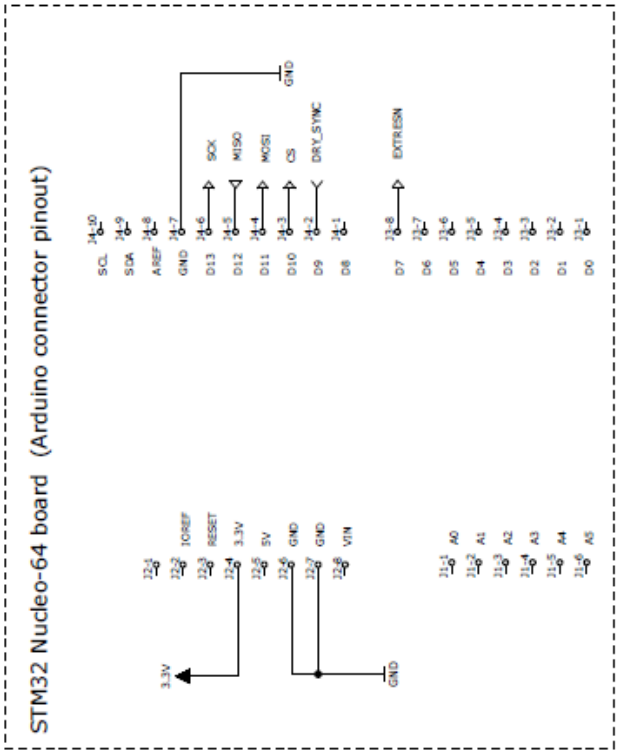
5. Main function flowchart

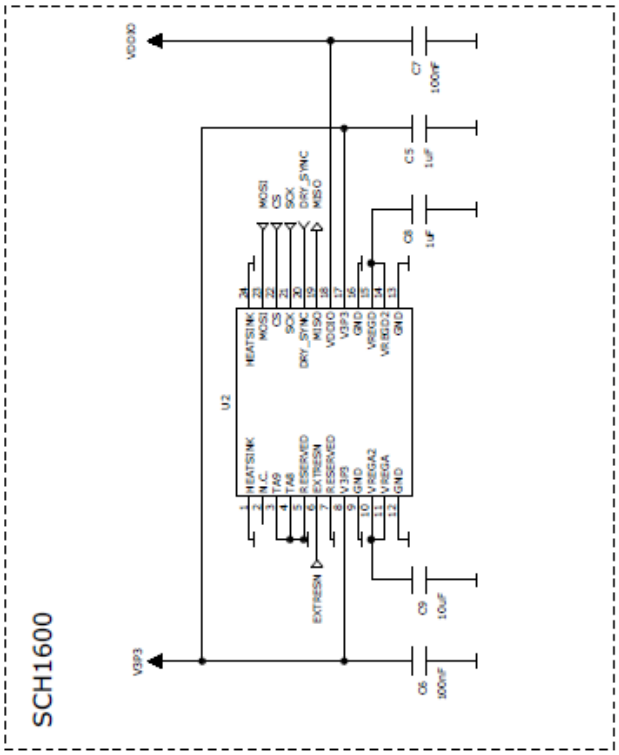
Please see the next page for basic demo software flowchart.


Interrupt routine
Start
Read sensor data registers
Process received data:
Sum 'raw' sensor data at write index of ring buffer
Sensor errors occurred?
No
Yes
Update error flags
10 samples summed?
No
Yes
Space in ring buffer?
No
Yes
Increment write index
Zero summed data
Stop
Main loop
Start
Initialize MCU
Sensor power-up & initialization
Sensor status OK?
No
Halt
Yes
Setup sample timer interrupt
New data in ring buffer?
No
Yes
Convert 'raw' sensor data to real values & calculate averages from the ring buffer
Increment buffer read index
Send converted data over UART to client
Sensor error flags set?
No
Yes
Read sensor status registers
Send register contents over UART to client

6. Schematic

Example schematic for STM32 Nucleo-F401RE board.





7. Troubleshooting

If the output data stream is slow or data corruption appears (extra or missing characters), changing the PC serial port latency timer setting to 2 ms may help. This setting is found from (Windows 7): Control Panel - System - Device Manager - Ports - Right-click your serial port - Properties - Port Settings - Advanced... - Latency Timer.