SPI communication with SCC1300

1 INTRODUCTION

The objective of this document is to show how to set up SPI communication between Murata Electronics Oy SCC1300 Combined X-axis Gyroscope & 3-axis Accelerometer component and an NXP LPC1114 Cortex-M0 microcontroller. The code example contains following operations:

- The LPC1114 MCU is configured.
- SCC1300 Gyroscope and Accelerometer are configured.
- Measurement data is read from both Gyroscope and Accelerometer and sent to serial port.
- Possible Gyroscope errors are handled, accelerometer errors are not handled in this example.

2 DEVELOPMENT HARDWARE

For easy construction, a Murata prototype board SCC1300-D04 PWB was connected to NXP LPCXpresso-CN, please see Figure 1 and Figure 2 below. Depending on cable lenghts an external supply bypass capacitor may need to be added close to the PWB between power supply lines (C1 in Figure 2).

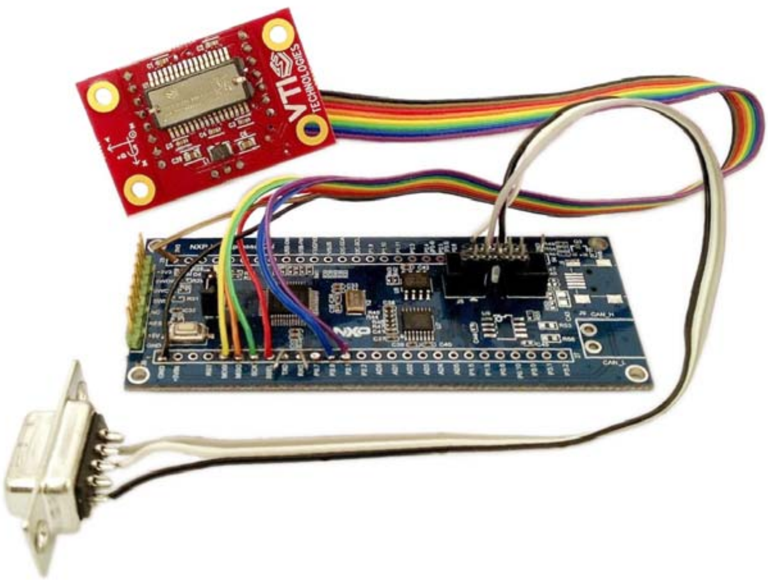

Figure 1. SCC1300 chip carrier connected to LPCXpresso-CN

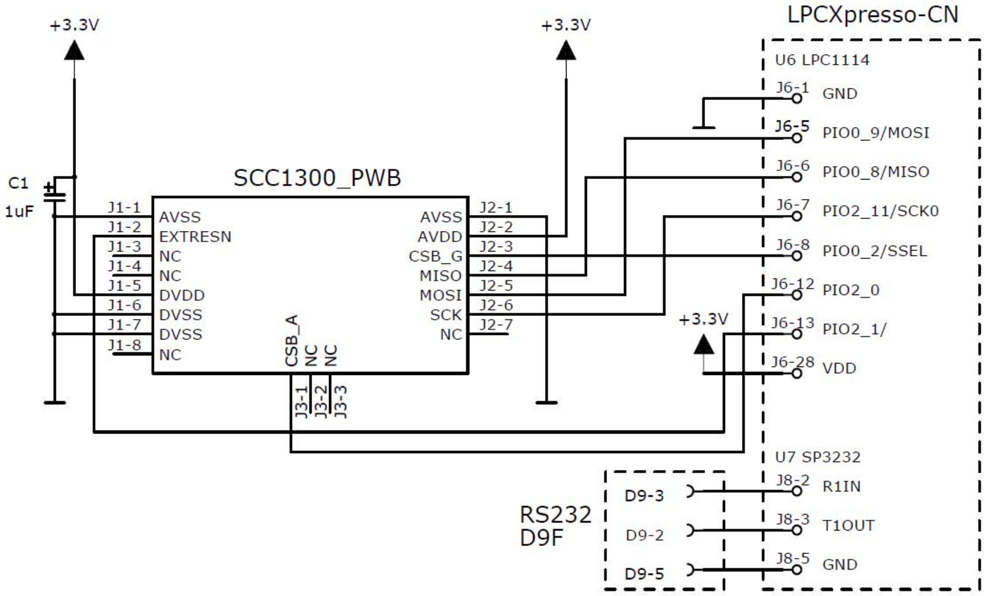

Figure 2. System schematic

3 C-CODE

Example code was created for NXP LPCXpresso-CN Development Board (LPC1114) using Keil uVision MDK-Lite Version 4.53 and Keil Ulink2 Debug Adapter.

C-language software example on the next pages shows how to implement basic communication with the SCC1300 using SPI0 block of the LPC1114 MCU. 12 MHz internal system clock is used and the SPI clock frequency is set to 500 kHz.

3.1 Code Flowchart

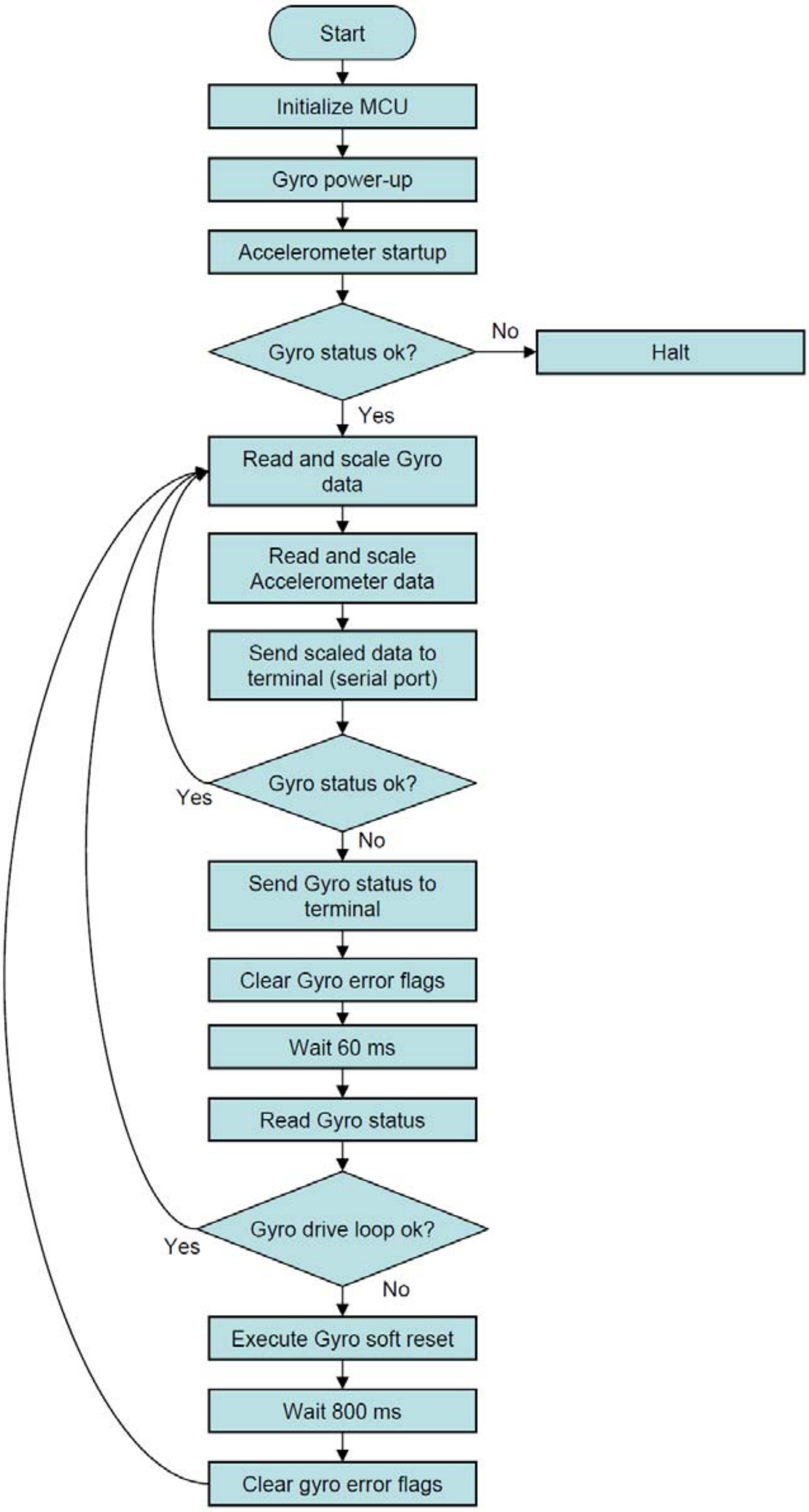

Figure 3. Example Code Flowchart

3.2 C-Code Listing

```
//********************************************************************************
// SCC1300 Demo - SPI Interface to SCC1300 Combo Sensor (Gyroscope + Accelerometer)
//
// Uses NXP LPCXpresso Development Platform LPC1114 (Cortex-M0). Measurement results
// sent to PC terminal software thru UART.
//
//**************************************************************************************
**

#include "LPC11xx.h"                      // LPC11xx definitions
#include "uart.h"
#include <stdio.h>

// SCC1300 version specific parameters, here SCC1300-D04
#define GYRO_SENSITIVITY 18               // LSB/deg
#define ACC_SENSITIVITY  650              // LSB/g

// Accelerometer Registers
#define ACC_REVID        0x00
#define ACC_CTRL         0x01
#define ACC_STATUS       0x02
#define ACC_RESET        0x03
#define ACC_X_LSB        0x04
#define ACC_X_MSB        0x05
#define ACC_Y_LSB        0x06
#define ACC_Y_MSB        0x07
#define ACC_Z_LSB        0x08
#define ACC_Z_MSB        0x09
#define ACC_TEMP_LSB     0x12
#define ACC_TEMP_MSB     0x13
#define ACC_INT_STATUS   0x16
#define ACC_ID           0x27

// Gyroscope Registers
#define GYRO_RATE_X      0x00
#define GYRO_IC_ID       0x07
#define GYRO_STATUS      0x08
#define GYRO_TEMP        0x0A

// Gyroscope status and control bits
#define BIT_S_OK         (1 << 1)         // Sensor valid flag
#define BIT_LOOPF_OK     (1 << 2)         // Loopfilter status flag
#define BIT_SOFTRESET    (1 << 1)         // Gyro soft reset

// SSP Status register
#define SSPSR_TFE        (1 << 0)
#define SSPSR_TNF        (1 << 1)
#define SSPSR_RNE        (1 << 2)
#define SSPSR_RFF        (1 << 3)
#define SSPSR_BSY        (1 << 4)

// GPIO ports
#define PORT0            0
#define PORT1            1
#define PORT2            2
#define PORT3            3

// SPI read and write buffer size
#define FIFOSIZE         8

// SSP CR0 register
#define SSPCR0_DSS       (1 << 0)
#define SSPCR0_FRF       (1 << 4)
#define SSPCR0_SPO       (1 << 6)
#define SSPCR0_SPH       (1 << 7)
#define SSPCR0_SCR       (1 << 8)
```

```
// SSP CR1 register
#define SSPCR1_LBM       (1 << 0)
#define SSPCR1_SSE       (1 << 1)
#define SSPCR1_MS        (1 << 2)
#define SSPCR1_SOD       (1 << 3)

// SSP Interrupt Mask Set/Clear register
#define SSPIMSC_RORIM    (1 << 0)
#define SSPIMSC_RTIM     (1 << 1)
#define SSPIMSC_RXIM     (1 << 2)
#define SSPIMSC_TXIM     (1 << 3)

// Pin definitions
#define PIN_CSB_GYRO     (1 << 2)
#define PIN_CSB_ACC      (1 << 0)
#define PIN_EXTRESN_GYRO (1 << 1)


// Function prototypes
void SystemInit(void);
void Main_PLL_Setup (void);
void SSP_IOConfig(void);
void SSP_Init(void);
uint8_t ACC_ReadRegister(uint8_t Address, uint8_t *Data);
uint8_t ACC_WriteRegister(uint8_t Address, uint8_t Data);
uint8_t ReadAccelerations(int16_t *Xacc, int16_t *Yacc, int16_t *Zacc);
uint16_t GYRO_ReadRegister(uint8_t Address, uint16_t *Data);
uint16_t GYRO_WriteRegister(uint8_t Address, uint16_t Data);
void Wait_us(uint16_t us);
void send_string_2_serial(char *str);


// Wait us, depends on clock frequency so adjust accordingly
void Wait_us(uint16_t us)
{
  uint16_t a;

  if (us > 1) us -= 2;
  for (a = us; a > 0; a--)
  {
      __NOP();
      __NOP();
  }
}


void SSP_Init(void)
{
  uint8_t i, Dummy=Dummy;

  // Set DSS data to 8-bit, Frame format SPI, CPOL = 0, CPHA = 0, and SCR is 5 (serial clock divisor = 6)
  LPC_SSP0->CR0 = 0x0507;

  // SSPCPSR clock prescale register, master mode, minimum divisor is 0x02
  LPC_SSP0->CPSR = 0x2;

  for ( i = 0; i < FIFOSIZE; i++ )
  {
    Dummy = LPC_SSP0->DR;  // clear the RxFIFO
  }

  // Master mode
  LPC_SSP0->CR1 = SSPCR1_SSE;

  // Set SSPINMS registers to enable interrupts
  // Enable all error related interrupts
  LPC_SSP0->IMSC = SSPIMSC_RORIM | SSPIMSC_RTIM;

  return;
}
```

```
void SSP_IOConfig(void)
{
  LPC_SYSCON->PRESETCTRL     |= (1 << 0);
  LPC_SYSCON->SYSAHBCLKCTRL  |= (1 << 11);
  LPC_SYSCON->SSP0CLKDIV      = 0x02;        // Divided by 2
  LPC_IOCON->PIO0_8           &= ~0x07;      // SSP I/O config
  LPC_IOCON->PIO0_8           |= 0x01;       // SSP MISO
  LPC_IOCON->PIO0_9           &= ~0x07;
  LPC_IOCON->PIO0_9           |= 0x01;       // SSP MOSI

  LPC_IOCON->SCK_LOC = 0x01;
  LPC_IOCON->PIO2_11 = 0x01;                 // P2.11 function 1 is SSP clock, need to
                                             // combine with IOCONSCKLOC register setting
  // Enable AHB clock to the GPIO domain.
  LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 6);

  LPC_IOCON->PIO0_2 &= ~0x07;                // SSP SSEL is a GPIO pin GYRO
  LPC_IOCON->PIO2_0 &= ~0x07;                // SSP SSEL is a GPIO pin ACC

  // Port0, bit 2 is set to GPIO output and high, GYRO CSB
  LPC_GPIO0->DIR |= PIN_CSB_GYRO;
  LPC_GPIO0->DATA |= PIN_CSB_GYRO;
  // Port2, bit 0 is set to GPIO output and high, ACC CSB
  LPC_GPIO2->DIR |= PIN_CSB_ACC;
  LPC_GPIO2->DATA |= PIN_CSB_ACC;
  // Port2, bit 0 is set to GPIO output and high, GYRO EXTRESN
  LPC_GPIO2->DIR |= PIN_EXTRESN_GYRO;
  LPC_GPIO2->DATA |= PIN_EXTRESN_GYRO;

  return;
}


void Main_PLL_Setup ( void )
{

  LPC_SYSCON->SYSPLLCLKSEL = 0x01;           // Select system OSC
  LPC_SYSCON->MAINCLKSEL   = 0x01;           // Main clock source = Input clock to system PLL
  LPC_SYSCON->MAINCLKUEN   = 0x01;           // Update MCLK clock source
  LPC_SYSCON->MAINCLKUEN   = 0x00;           // Toggle update register once
  LPC_SYSCON->MAINCLKUEN   = 0x01;
  while (!(LPC_SYSCON->MAINCLKUEN & 0x01)); // Wait until updated

  LPC_SYSCON->SYSAHBCLKDIV = 0x01;           // SYS AHB clock

  return;
}


void SystemInit(void)
{
  uint32_t i;

  // Bit 0 default is crystal bypass, bit1 0=0~20Mhz crystal input, 1=15~50Mhz crystal input.
  LPC_SYSCON->SYSOSCCTRL = 0x00;

  // Main system OSC run is cleared, bit 5 in PDRUNCFG register
  LPC_SYSCON->PDRUNCFG &= ~(1 << 5);
  // Wait 200us for OSC to be stablized, no status indication, dummy wait.
  for ( i = 0; i < 0x100; i++ );

  Main_PLL_Setup();

  // Enable USB clock. USB clock bit 8 and 10 in PDRUNCFG.
  LPC_SYSCON->PDRUNCFG &= ~((1 << 8)|(1 << 10));

  // System clock to the IOCON needs to be enabled or most of the I/O related peripherals won't work.
  LPC_SYSCON->SYSAHBCLKCTRL |= (1 << 16);

  return;
}
```

```
uint8_t CalcParity(uint16_t Data)
{
  // Source: svicent http://www.edaboard.com/thread46732.html
  uint8_t NoOfOnes = 0;

  while(Data != 0)
  {
    NoOfOnes++;
    Data &= (Data - 1); // Loop will execute once for each bit of Data set
  }

  // If NoOfOnes is odd, least significant bit will be 1
  return (~NoOfOnes & 1);
}


uint8_t ACC_ReadRegister(uint8_t Address, uint8_t *Data)
{
  uint8_t Status;

  Address <<= 2;                                              // Address to be shifted left by 2 and
                                                              // RW bit to be reset
  Address += CalcParity(Address);                             // Add parity bit

  LPC_GPIO2->DATA &= ~PIN_CSB_ACC;                            // Select acceleration sensor

  Status = LPC_SSP0->DR;                                      // Read RX buffer just to clear
                                                              // interrupt flag

  while ((LPC_SSP0->SR & (SSPSR_TNF|SSPSR_BSY)) != SSPSR_TNF);// Move on only if NOT busy and TX FIFO
                                                              // not full
  LPC_SSP0->DR = Address;                                     // Write address to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Status = LPC_SSP0->DR;                                      // Read RX buffer (status byte)

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  *Data = LPC_SSP0->DR;                                       // Read RX buffer (data byte)

  LPC_GPIO2->DATA |= PIN_CSB_ACC;                             // Deselect acceleration sensor

  return Status;
}


uint8_t ACC_WriteRegister(uint8_t Address, uint8_t Data)
{
  uint8_t Status;

  Address <<= 2;                                              // Address to be shifted left by 2
  Address |= 2;                                               // Set RW bit
  Address += CalcParity(Address);                             // Add parity bit

  LPC_GPIO2->DATA &= ~PIN_CSB_ACC;                            // Select acceleration sensor

  Status = LPC_SSP0->DR;                                      // Read RX buffer just to clear
                                                              // interrupt flag

  while ((LPC_SSP0->SR & (SSPSR_TNF|SSPSR_BSY)) != SSPSR_TNF);// Move on only if NOT busy and TX FIFO
                                                              // not full
  LPC_SSP0->DR = Address;                                     // Write address to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Status = LPC_SSP0->DR;                                      // Read RX buffer (status byte)

  LPC_SSP0->DR = Data;                                        // Write data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Data = LPC_SSP0->DR;                                        // Read RX buffer (dummy)

  LPC_GPIO2->DATA |= PIN_CSB_ACC;                             // Deselect acceleration sensor

  return Status;
}
```

```
// Read all 6 acceleration data registers using the decrement reading feature
// and convert to 16-bit signed values
uint8_t ReadAccelerations(int16_t *Xacc, int16_t *Yacc, int16_t *Zacc)
{
  uint8_t Status;
  uint8_t Result;
  uint8_t Address = ACC_Z_MSB;

  Address <<= 2;                                              // Address shifted left by 2
                                                              // and RW bit to be reset
  Address += CalcParity(Address);                             // Add parity bit

  LPC_GPIO2->DATA &= ~PIN_CSB_ACC;                            // Select acceleration sensor

  Result = LPC_SSP0->DR;                                      // Read RX buffer just to clear
                                                              // interrupt flag

  while ((LPC_SSP0->SR & (SSPSR_TNF|SSPSR_BSY)) != SSPSR_TNF);// Move on only if NOT busy and TX FIFO
                                                              // not full
  LPC_SSP0->DR = Address;                                     // Write address to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Status = LPC_SSP0->DR;                                      // Read RX buffer (status byte)

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Zacc = (int16_t)(Result << 8);                             // Store Z MSByte

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Zacc |= Result;                                            // Store Z LSByte

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Yacc = (int16_t)(Result << 8);                             // Store Y MSByte

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Yacc |= Result;                                            // Store Y LSByte

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Xacc = (int16_t)(Result << 8);                             // Store X MSByte

  LPC_SSP0->DR = 0;                                           // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                           // Wait until the Busy bit is cleared
  Result = LPC_SSP0->DR;                                      // Read RX buffer (data byte)
  *Xacc |= Result;                                            // Store X LSByte

  LPC_GPIO2->DATA |= PIN_CSB_ACC;                             // Deselect acceleration sensor

  return Status;
}
```

```
uint16_t GYRO_ReadRegister(uint8_t Address, uint16_t *Data)
{
  uint16_t Status;
  uint16_t ui16Address;

  // Build address transfer frame
  ui16Address = Address;
  ui16Address <<= 3;                                            // Address to be shifted left by 3
                                                                // (and RW = 0)
  ui16Address += CalcParity(ui16Address);                       // Add parity bit

  LPC_GPIO0->DATA &= ~PIN_CSB_GYRO;                             // Select gyro sensor

  Status = LPC_SSP0->DR;                                        // Read RX buffer just to clear
                                                                // interrupt flag

  while ((LPC_SSP0->SR & (SSPSR_TNF|SSPSR_BSY)) != SSPSR_TNF);// Move on only if NOT busy and TX FIFO
                                                                // not full
  LPC_SSP0->DR = ui16Address >> 8;                              // Write address MSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Status = LPC_SSP0->DR;                                        // Read RX buffer (status byte MSB)
  Status <<= 8;

  LPC_SSP0->DR = ui16Address & 0x00FF;                          // Write address LSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Status |= LPC_SSP0->DR;                                       // Read RX buffer (status byte LSB)

  LPC_SSP0->DR = 0;                                             // Write dummy data to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  *Data = LPC_SSP0->DR;                                         // Read RX buffer (data byte MSB)
  *Data <<= 8;

  LPC_SSP0->DR = 0x01;                                          // Write dummy data to TX buffer; NOTE:
                                                                // even if the data is dummy it still
                                                                // needs to have correct parity so
                                                                // force odd parity
                                                                // (TX 0x0000 -> 0x0001)
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  *Data |= LPC_SSP0->DR;                                        // Read RX buffer (data byte LSB)

  LPC_GPIO0->DATA |= PIN_CSB_GYRO;                              // Deselect gyro sensor

  return Status;
}
```

```
uint16_t GYRO_WriteRegister(uint8_t Address, uint16_t Data)
{
  uint16_t Status;
  uint16_t ui16Address;
  uint8_t Dummy;

  // Build address transfer frame
  ui16Address = Address;
  ui16Address <<= 3;                                            // Address to be shifted left by 3
  ui16Address |= 0x04;                                          // Set RW bit
  ui16Address += CalcParity(ui16Address);                       // Add parity bit

  // Add parity to data
  Data <<= 1;
  Data += CalcParity(Data);

  LPC_GPIO0->DATA &= ~PIN_CSB_GYRO;                             // Select gyro sensor

  Status = LPC_SSP0->DR;                                        // Read RX buffer just to clear
                                                                // interrupt flag

  while ((LPC_SSP0->SR & (SSPSR_TNF|SSPSR_BSY)) != SSPSR_TNF);// Move on only if NOT busy and TX FIFO
                                                                // not full
  LPC_SSP0->DR = ui16Address >> 8;                              // Write address MSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Status = LPC_SSP0->DR;                                        // Read RX buffer (status byte MSB)
  Status <<= 8;

  LPC_SSP0->DR = ui16Address & 0x00FF;                          // Write address LSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Status |= LPC_SSP0->DR;                                       // Read RX buffer (status byte LSB)

  LPC_SSP0->DR = Data >> 8;                                     // Write data MSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Dummy = LPC_SSP0->DR;                                         // Read RX buffer (dummy data byte MSB)

  LPC_SSP0->DR = Data & 0x00FF;                                 // Write data LSB to TX buffer
  while (LPC_SSP0->SR & SSPSR_BSY);                             // Wait until the Busy bit is cleared
  Dummy = LPC_SSP0->DR;                                         // Read RX buffer (dummy data byte LSB)
  Dummy--;                                                      // Just to suppress compiler warning

  LPC_GPIO0->DATA |= PIN_CSB_GYRO;                              // Deselect gyro sensor

  return Status;
}


void send_string_2_serial(char *str)
{
  while ( *str != '\0' )
  {
    UARTSend((uint8_t *)str++, 1);
  }
}
```

```
int main(void)
{
  char Buffer[80];
  uint16_t Rate_X;
  int16_t i16Rate_X;
  uint16_t Status_Gyro;
  uint8_t Status_Acc;
  uint8_t Int_Status_Acc;
  int16_t Acc_X;
  int16_t Acc_Y;
  int16_t Acc_Z;
  uint16_t Count;

  SystemInit();

  SSP_IOConfig();       // initialize SSP port, share pins with SPI1 on port2(p2.0-3)
  SSP_Init();

  UARTInit(115200);     // Initialize serial port
  send_string_2_serial("\f");

  // Gyro power-up
  //---------------------------------------------------------------------------------------------
  for(Count = 0; Count < 800; Count++) Wait_us(1000);           // Wait 800 ms
  // Read status register twice to clear error flags
  GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);
  GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);
  //---------------------------------------------------------------------------------------------

  // Accelerometer start-up
  //---------------------------------------------------------------------------------------------
  Status_Acc = ACC_ReadRegister(ACC_INT_STATUS, &Int_Status_Acc); // Acknowledge for possible
                                                                  // saturation (SAT-bit)
  Status_Acc = ACC_WriteRegister(ACC_CTRL, 0);                    // Set PORST to 0
  for(Count = 0; Count < 10; Count++) Wait_us(1000);              // Wait 10 ms
  //---------------------------------------------------------------------------------------------

  // If gyro is ok start reading the sensors
  if(Status_Gyro & BIT_S_OK)
  {
    while(1)
    {
      Status_Gyro = GYRO_ReadRegister(GYRO_RATE_X, &Rate_X);      // Read angular rate output register
      i16Rate_X = Rate_X;                                         // Convert and scale gyroscope data
      i16Rate_X >>= 2;
      Status_Acc = ReadAccelerations(&Acc_X, &Acc_Y, &Acc_Z);
      Acc_X >>= 2;                                                // Scale accelerometer data
      Acc_Y >>= 2;
      Acc_Z >>= 2;
      sprintf(Buffer, "G:%6d  GS:%X  AX:%5d  AY:%5d  AZ:%5d  AS:%X\r\n", i16Rate_X, Status_Gyro, Acc_X,
Acc_Y, Acc_Z, Status_Acc);
      send_string_2_serial(Buffer);
      Wait_us(60000);
      if(!(Status_Gyro & BIT_S_OK))
      {
        // Gyro error, show status and recover
        sprintf(Buffer, "Gyro s_ok fail: Status = %X\r\n", Status_Gyro);
        send_string_2_serial(Buffer);
        GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);   // Read status register to clear error flags
        Wait_us(60000);
        GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);   // Read status register to clear flags again
        if(!(Status_Gyro & BIT_LOOPF_OK))
        {
          // If LOOPF still fails, reset gyroscope
          Status_Gyro = GYRO_WriteRegister(GYRO_IC_ID, BIT_SOFTRESET);  // Perform soft reset
          for(Count = 0; Count < 800; Count++) Wait_us(1000);           // Wait 800 ms
          // Read status register twice to clear error flags
          GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);
          GYRO_ReadRegister(GYRO_STATUS, &Status_Gyro);
        }
      }
    }
  }
```

```
  else
  {
    // Gyro error, show status and stop operation
    sprintf(Buffer, "Gyro s_ok fail: Status = %X\r\n", Status_Gyro);
    send_string_2_serial(Buffer);
    while(1) Count++;
  }
}
```

4 RESULT DATA OUTPUT

In this example system the measurement and status data are sent to PC terminal program (HyperTerminal) for easier examination. The serial communication parameters are: bps - 115200, Data - 8, Parity - None.

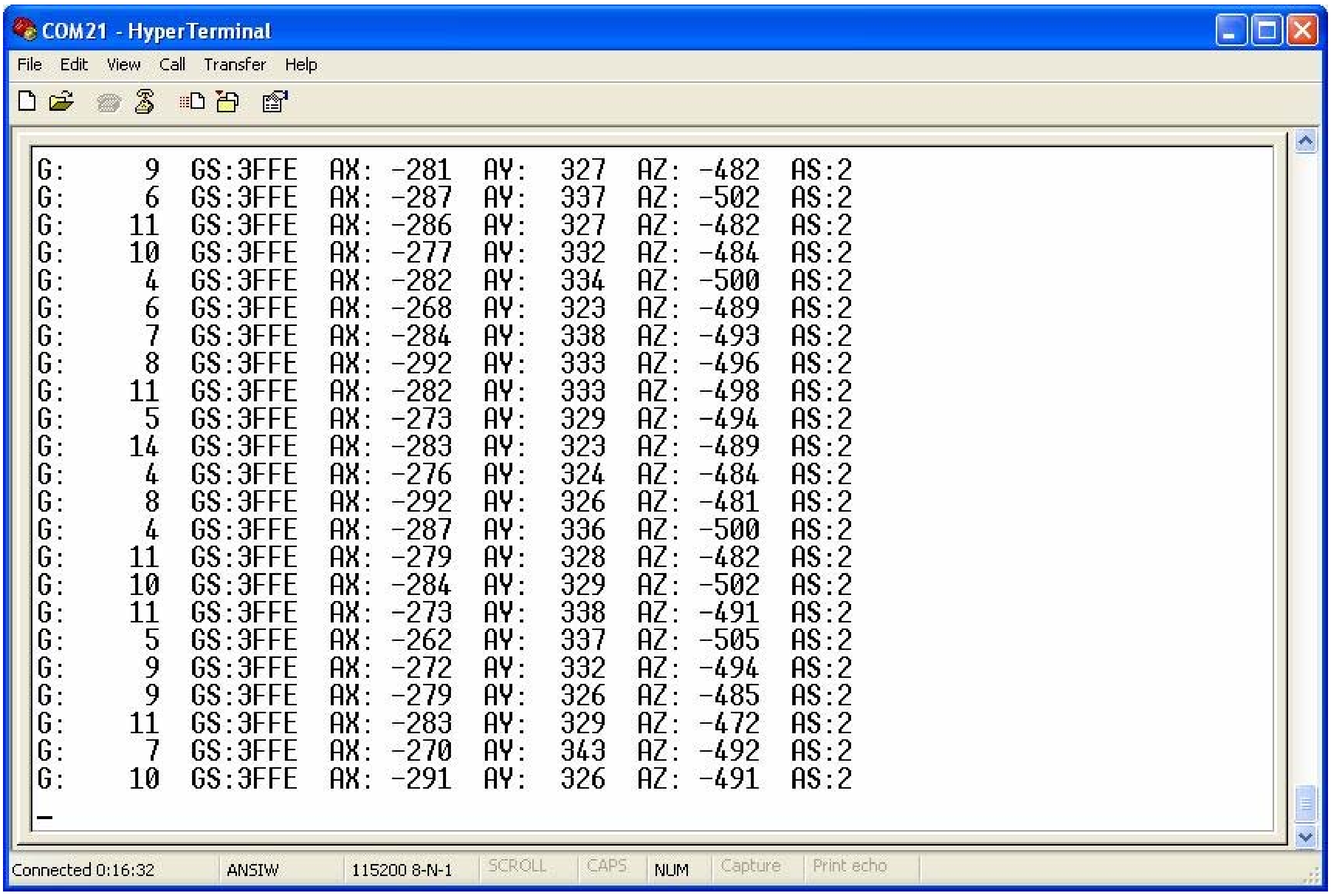

Figure 4. Sample result data captured from PC screen

Logic analyzer waveforms are shown on Figure 5 below:

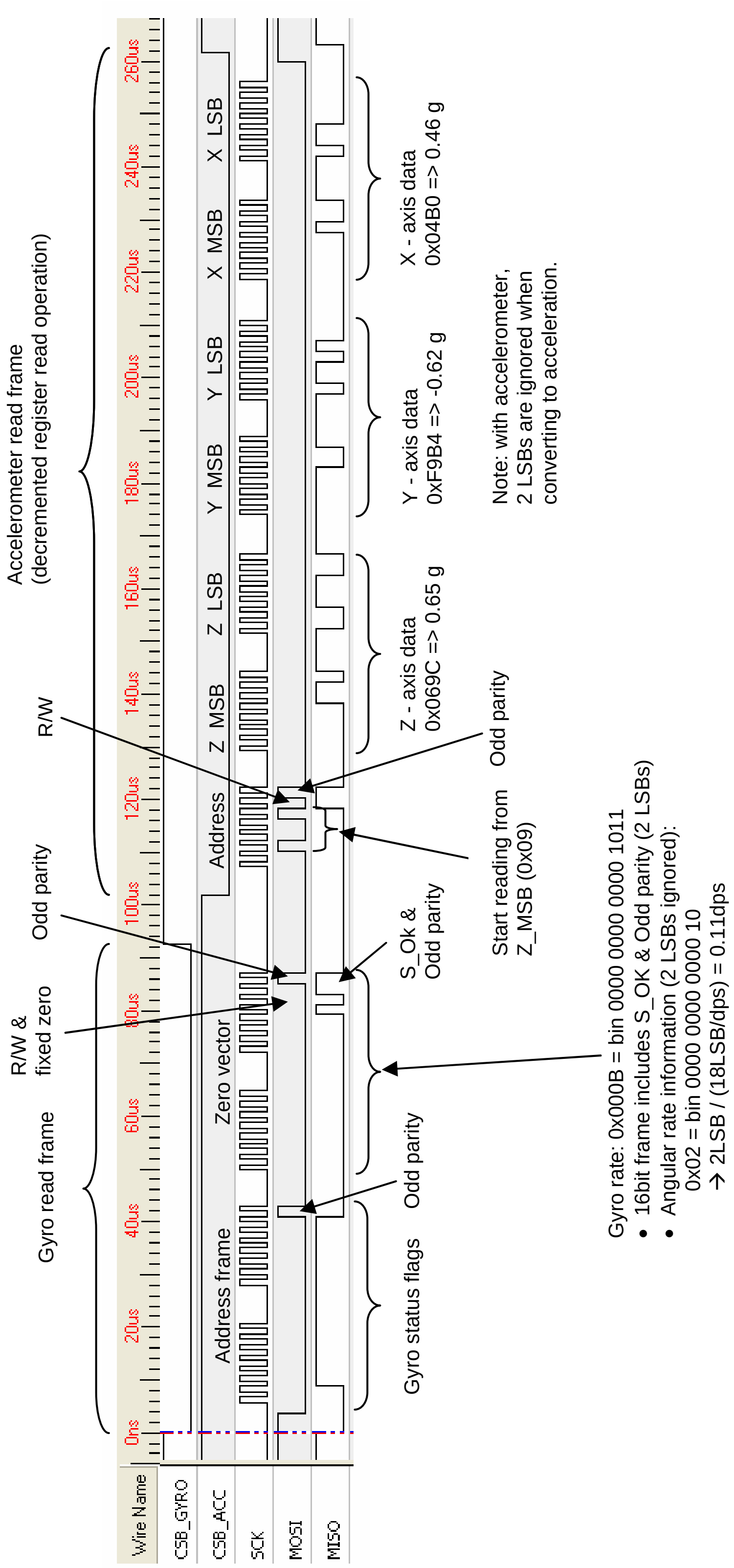

Figure 5. SPI communication with Gyro and Accelerometer (unit conversions apply for SCC1300-D04).