```c
 */
#include <zephyr/kernel.h>
#include <nrfx_example.h>
#include <nrfx_spim.h>
#include <zephyr/drivers/gpio.h>
#include <MAX30003.h>
#include <nrfx_log.h>
#include <zephyr/sys/time_units.h>



#define NRFX_LOG_MODULE                 EXAMPLE
#define NRFX_EXAMPLE_CONFIG_LOG_ENABLED 1
#define NRFX_EXAMPLE_CONFIG_LOG_LEVEL   3

// SPIM pin initializaiton
#define SPIM_INST_IDX 1
#define MOSI_PIN 3
#define MISO_PIN 31
#define SCK_PIN 4
#define MSG_TO_SEND "Nordic Semiconductor"
#define SS_PIN 27
nrfx_spim_t spim_inst = NRFX_SPIM_INSTANCE(SPIM_INST_IDX);
static const struct gpio_dt_spec intB_gpio_pin =
GPIO_DT_SPEC_GET_OR(DT_NODELABEL(gpiocust1), gpios, {0});
static const struct gpio_dt_spec intB_gpio_pin_time =
GPIO_DT_SPEC_GET_OR(DT_NODELABEL(gpiocust2), gpios, {0});




// Maxim30003 bit constants
int EINT_STATUS =  1 << 23;
int RTOR_STATUS =  1 << 10;
int DCL_OFF = 1<<20;
int DCL_ON = 1<<11;
int RTOR_REG_OFFSET = 10;
float RTOR_LSB_RES = 0.0078125f;
int FIFO_OVF =  0x7;
int FIFO_VALID_SAMPLE =  0x0;
int FIFO_FAST_SAMPLE =  0x1;
int ETAG_BITS = 0x7;
int FIFO_EMPTY = 0x6;

//Maxim registers
union GeneralConfiguration_u CNFG_GEN_r;

// ECG related buffers and variables
volatile bool ecgIntFlag = false;
static uint8_t m_tx_buffer[4];
static uint8_t m_rx_buffer[4];
static uint8_t txB[1];
static uint8_t rxB[1];
uint32_t RtoR, idx, stat;

uint8_t ETAG[32];
uint32_t ecgFIFO;
int32_t ecgSample[17];
float BPM=0.0f;



// #define nodeId DT_NODELABEL(test)
// timing vars
int64_t ecg_start, ecg_end, ecg_start_m, ecg_end_m;



void writeRegister(enum Registers_e reg, const uint32_t data)
{
   // NRFX_LOG_INFO("Inside write");
    m_tx_buffer[0] = reg << 1;
    m_tx_buffer[1] = ((0x00FF0000 & data) >> 16);
    m_tx_buffer[2] = ((0x0000FF00 & data) >> 8);
    m_tx_buffer[3] = ( 0x000000FF & data);
    //nrfx_spim_xfer_desc_t spim_xfer_desc =
NRFX_SPIM_XFER_TX(m_tx_buffer, sizeof(m_tx_buffer));
     nrfx_spim_xfer_desc_t spim_xfer_desc =
NRFX_SPIM_XFER_TRX(m_tx_buffer, sizeof(m_tx_buffer), m_rx_buffer,
sizeof(m_rx_buffer));
```

```
                                                                        //
nrfx_spim_xfer_desc_t spim_xfer_desc = NRFX_SPIM_XFER_TX(m_tx_buffer,
sizeof(m_tx_buffer),
                                                                        //
m_rx_buffer, sizeof(m_rx_buffer));

   // printf("\nwriting reg %x => D:%x -> tx:%x",reg,data, m_tx_buffer[0]
);
    // for(int i =0; i<4; i++)
    // {
    //     txB[0] = m_tx_buffer[i];
        nrfx_spim_xfer(&spim_inst, &spim_xfer_desc, 0);
    // printk("tx:%x ",m_tx_buffer[i]);
    // }
   // printf("\n---\n");



}

uint32_t readRegister(enum Registers_e reg)
{


    uint32_t data = 0;
    m_tx_buffer[0] = ((reg << 1) | 1);
    m_tx_buffer[1] = 0;
    m_tx_buffer[2] = 0;
    m_tx_buffer[3] = 0;

    m_rx_buffer[0] = 7;
    m_rx_buffer[1] = 7;
    m_rx_buffer[2] = 7;
    m_rx_buffer[3] = 7;
    txB[0] = m_tx_buffer[0];
    nrfx_spim_xfer_desc_t spim_xfer_desc = NRFX_SPIM_XFER_TRX(m_tx_buffer,
sizeof(m_tx_buffer), m_rx_buffer, sizeof(m_rx_buffer));
    nrfx_spim_xfer(&spim_inst, &spim_xfer_desc, 0);
    data |= m_rx_buffer[1] ;
    data = data << 8;
    data |= m_rx_buffer[2] ;
    data = data << 8;
     data |= m_rx_buffer[3] ;
     //printk(" -> %x\n----\n",data);
    return data;
}


int32_t readdECG(uint32_t dataIn)
{
   // NRFX_LOG_INFO("Inside Read");
    int32_t data = 0;

    // if(reg == CNFG_GEN)
    // printk("rcvd: %x ",m_rx_buffer[1]);
    data |= (dataIn & 0x00FF0000) << 24;
    // if(reg == CNFG_GEN)
    // printk("\ndata: %x 16shifted rcvd: %x\n",data, (m_rx_buffer[1] <<
16));
    data |= (dataIn & 0x0000FF00) << 16;
    // if(reg == CNFG_GEN)
    // printk("\ndata: %x 8shifted rcvd: %x %x\n",data, (m_rx_buffer[1] <<
8), m_rx_buffer[2]);

    data |= dataIn    & 0xC0;
    // if(reg == CNFG_GEN)
    // printk("\ndata: %x  rcvd: %x\n",data, m_rx_buffer[3]);

    return data;
}

    union EnableInterrupts_u EN_INT_r;
    union ECGConfiguration_u CNFG_ECG_r;
    union RtoR1Configuration_u CNFG_RTOR_r;
    union ManageInterrupts_u MNG_INT_r;
     union ManageDynamicModes_u MNG_DYN_r;
    union MuxConfiguration_u CNFG_MUX_r;
void regCheck(){

    // printk(" Freq check %x",);
     NRFX_SPIM_FREQUENCY_STATIC_CHECK(1, 125000);
    writeRegister( SW_RST , 0);
    EN_INT_r.bits.en_eint = 1;              // Enable EINT interrupt
    EN_INT_r.bits.en_dcloffint = 0; // ak
    EN_INT_r.bits.en_loint = 0 ;//ak
    EN_INT_r.bits.en_pllint = 0 ;//ak
    EN_INT_r.bits.en_fstint = 0;
    EN_INT_r.bits.en_rrint = 0;             // Enable R-to-R interrupt
    EN_INT_r.bits.intb_type = 3;            // Open-drain NMOS with
internal pullup
    //CNFG_GEN_r.bits.imag = 2;
int i=0;
    writeRegister( EN_INT , EN_INT_r.all);
 //writeRegister( CNFG_GEN , CNFG_GEN_r.all);

//    while ( true ) {
//     printk("\n####reading## \n");
//     readRegister( EN_INT);
//     NRFX_EXAMPLE_LOG_PROCESS();
//     k_msleep(250);

//    }



    //while(true){
    readRegister(EN_INT);
    //}

    //readRegister(INFO);
}
void ecg_config() {
    writeRegister( SW_RST , 0);

    // General config register setting

    CNFG_GEN_r.bits.en_ecg = 1;      // Enable ECG channel
    CNFG_GEN_r.bits.rbiasn = 1   ;   // Enable resistive bias on negative
input
    CNFG_GEN_r.bits.rbiasp = 1;      // Enable resistive bias on positive
input
    CNFG_GEN_r.bits.en_rbias = 1;    // Enable resistive bias
```

```
     CNFG_GEN_r.bits.imag = 2;       // Current magnitude = 10nA
    // CNFG_GEN_r.bits.fmstr = 0;
    CNFG_GEN_r.bits.en_dcloff = 0;  // Enable DC lead-off detection
    //printk("CNFG_GEN_r.all %x\n",CNFG_GEN_r.all);
    writeRegister( CNFG_GEN , CNFG_GEN_r.all);


    // ECG Config register setting

    CNFG_ECG_r.bits.dlpf = 1;       // Digital LPF cutoff = 40Hz
    CNFG_ECG_r.bits.dhpf = 1;       // Digital HPF cutoff = 0.5Hz
    CNFG_ECG_r.bits.gain = 0;       // ECG gain = 160V/V, 3
    CNFG_ECG_r.bits.rate = 0;   //chng ss from 2(128sps) to 0(512sps)
// Sample rate = 128 sps
    writeRegister( CNFG_ECG , CNFG_ECG_r.all);
    //printk("CNFG_ECG_r.all %u\n",CNFG_ECG_r.all);


    CNFG_RTOR_r.bits.wndw = 0b0011;         // WNDW       = 96ms
    CNFG_RTOR_r.bits.en_rtor = 1; // enabling rtor detection
    CNFG_RTOR_r.bits.rgain = 0b1111;        // Auto-scale gain
    CNFG_RTOR_r.bits.pavg = 0b11;           // 16-average
    CNFG_RTOR_r.bits.ptsf = 0b0011;         // PTSF = 4/16
    CNFG_RTOR_r.bits.en_rtor = 1;           // Enable R-to-R detection
1->0
    writeRegister( CNFG_RTOR1 , CNFG_RTOR_r.all);


    //Manage interrupts register setting

    MNG_INT_r.bits.efit = 4;//16;//;          // Assert EINT w/ 4 unread
samples
    MNG_INT_r.bits.clr_rrint = 0b01;// Clear R-to-R on RTOR reg. read back
     MNG_INT_r.bits.clr_samp= 1;
    writeRegister( MNGR_INT , MNG_INT_r.all);

    // printk("MNG_INT_r.all %u\n",MNG_INT_r.all);


    //Enable interrupts register setting
    EN_INT_r.bits.en_eint = 1;                // Enable EINT interrupt
    EN_INT_r.bits.en_dcloffint = 0; // ak
    EN_INT_r.bits.en_loint = 0 ;//ak
    EN_INT_r.bits.en_pllint = 1 ;//ak
EN_INT_r.bits.en_samp = 0;
    EN_INT_r.bits.en_rrint = 1;               // Enable R-to-R interrupt
    EN_INT_r.bits.intb_type = 3;              // Open-drain NMOS with
internal pullup
    writeRegister( EN_INT , EN_INT_r.all);


    //Dyanmic modes config

    MNG_DYN_r.bits.fast = 0;                  // Fast recovery mode disabled
    writeRegister( MNGR_DYN , MNG_DYN_r.all);
    // MUX Config

    CNFG_MUX_r.bits.openn = 0;            // Connect ECGN to AFE channel
ss-> (0->1)
    CNFG_MUX_r.bits.openp = 0;            // Connect ECGP to AFE channel
ss-> (0->1)
    //ss
//     CNFG_MUX_r.bits.caln_sel = 1;
   writeRegister( CNFG_EMUX , CNFG_MUX_r.all);
// //ss
//    union CalConfiguration_u CAL_CONFG_r;
//    CAL_CONFG_r.bits.en_vcal = 1;
//    CAL_CONFG_r.bits.vmode = 0;
//    CAL_CONFG_r.bits.fcal = 1;
//    writeRegister( CNFG_CAL , CAL_CONFG_r.all);

    return;
}
static struct gpio_callback maxim_intB;

void maxim_interrupt(){
    // ecg_start = k_uptime_get();
    //printk("\nintterupted\n");
    ecgIntFlag = true;
```

```
}
void configureGPIO()
{
    int ret;
    ret = gpio_pin_configure_dt(&intB_gpio_pin, GPIO_INPUT );
    ret = gpio_pin_configure_dt(&intB_gpio_pin_time, GPIO_OUTPUT );
    bool status = false;
    ret = gpio_pin_interrupt_configure_dt(&intB_gpio_pin,
GPIO_INT_EDGE_TO_ACTIVE );
    gpio_init_callback(&maxim_intB, maxim_interrupt,
BIT(intB_gpio_pin.pin));
    gpio_add_callback(intB_gpio_pin.port, &maxim_intB);
}

int main(void)
{
    nrfx_err_t status1;
    (void) status1;
    printk("\n main funciton \n");

    NRFX_EXAMPLE_LOG_INIT();


    nrfx_spim_config_t spim_config = NRFX_SPIM_DEFAULT_CONFIG(SCK_PIN,
                                                              MOSI_PIN,
                                                              MISO_PIN,
                                                              SS_PIN);
                spim_config.frequency = 125000;
                spim_config.orc = 0x00;
    status1 = nrfx_spim_init(&spim_inst, &spim_config, NULL, NULL);
    NRFX_ASSERT(status1 == NRFX_SUCCESS);

    NRFX_ASSERT(status1 == NRFX_SUCCESS);

    // ecg_config();

    configureGPIO();
    //regCheck();
    ecg_config();
    //printk("MNG_INT_r.all %u\n",MNG_INT_r.all);



    bool state = false;
    int ecg_count = 0;


    NRFX_EXAMPLE_LOG_PROCESS();
    ecg_start = 0;

   int toggle = 0;
    uint32_t arr ;
        int pllCheck =0;

    // while (  true  ){

    uint32_t en_int = readRegister(EN_INT);
    uint32_t rtor = readRegister(CNFG_RTOR1);
    uint32_t mngr_int = readRegister(MNGR_INT);
    uint32_t cnfg_gen = readRegister(CNFG_GEN);
    uint32_t cnfg_ecg = readRegister(CNFG_ECG);

    printk("\n%x EN_INT %x \n",en_int, EN_INT_r.all);
    printk("\n%x CNFG_RTOR1 %x\n",rtor, CNFG_RTOR_r.all);
    printk("\n%x MNGR_INT %x\n",mngr_int, MNG_INT_r.all);
    printk("\n%x CNFG_GEN %x\n",cnfg_gen,  CNFG_GEN_r.all);
    printk("\n%x CNFG_ECG %x\n",cnfg_ecg, CNFG_ECG_r.all);

  if(en_int == EN_INT_r.all && rtor == CNFG_RTOR_r.all && mngr_int ==
MNG_INT_r.all && cnfg_gen == CNFG_GEN_r.all && cnfg_ecg ==  CNFG_ECG_r.all
){
        printk("All registers done");
        //break;
    }

    while(true){
        arr = readRegister(STATUS );
        if((arr & 256)){
        printk("\n PLL Interrupt - pre SYNCH \n");
            // break;
        }

        else {
               printk(" \n zero\n");
            break;

        }
    }

         NRFX_EXAMPLE_LOG_PROCESS();
 //    }
    writeRegister( SYNCH , 0);


// break;

    while (1)
    {
    //   printk("\n looping \n");
        NRFX_EXAMPLE_LOG_PROCESS();

        if(intB_gpio_pin.port)
        {

            // printk("\necg %d\n",ecgIntFlag);

            if(ecgIntFlag)
            {
                // Samples obtained in 1000 ms

                // if(ecg_end-ecg_start >= 1000){
                //     //printk("{time for %d in ms =
%lld}\n",ecg_count,ecg_end-ecg_start);
                //     ecg_count = 0;
                //     ecg_start = k_uptime_get();
                // }
                uint8_t sampleCount = 0;
                ecgIntFlag = false;


                uint32_t cond_bits = readRegister( STATUS );      // Read
the STATUS register
                //  if((cond_bits & 256)){
                // printk("\nPLL INTERRUPT\n");

                //  writeRegister( SYNCH , 0);
                // }
                // NRFX_ASSERT(!cond_bits & DCL_OFF);
                // NRFX_ASSERT(!cond_bits & DCL_ON);

                   if((cond_bits & 256)){
          printk("\nPLL Interrupt\n");
              }

                if( ( cond_bits & RTOR_STATUS ) == RTOR_STATUS ){
                    RtoR = readRegister( RTOR ) >>  RTOR_REG_OFFSET;
                    BPM = 1.0f / ( RtoR * RTOR_LSB_RES / 60.0f );
                }
                // Check if EINT interrupt asserted
                if ( ( cond_bits & EINT_STATUS ) == EINT_STATUS ) {

                    do {
                        ecgFIFO = readRegister( ECG_FIFO );       // Read
FIFO

                        ecgSample[sampleCount] = ( ecgFIFO & 0x00FFFFFF )
>> 6;

                        if( ecgSample[sampleCount]  & 0x0002FFFF )
ecgSample[sampleCount] = ecgSample[sampleCount] | 0xFFFC0000;

                        ETAG[sampleCount] = ( ecgFIFO >> 3 ) & ETAG_BITS;
// Isolate ETAG
                      // printk("\necgFIFO %x - %x - %x %d\n", ecgFIFO, ((
ecgFIFO >> 3 )), ( ecgFIFO >> 3 ) & ETAG_BITS, ETAG[sampleCount]);
                     printk("\nuint32 fifo %x 18bits ECG - %x ETAG -
%u\n",ecgFIFO, ecgSample[sampleCount], ETAG[sampleCount] );
                        // printk("\n%x -> %x|0x%u %u\n", ecgFIFO,
ecgSample[sampleCount],ETAG[sampleCount] , sampleCount);
```

```
                        // printk("\n%d", ecgSample[sampleCount]  );
                        sampleCount++;
// Increment sample counter

                        ecg_count++;
                    //  printk("\nFifo overflow check  %u\n",
sampleCount);
                    //   NRFX_EXAMPLE_LOG_PROCESS();
                        if(sampleCount > 32){
                            printk("\nFifo overflow %u\n",  sampleCount);
                            //  NRFX_EXAMPLE_LOG_PROCESS();
                            break;
                        }


                    } while ( ETAG[sampleCount-1] == FIFO_VALID_SAMPLE ||
                            ETAG[sampleCount-1] == FIFO_FAST_SAMPLE);
                    //   printk("\n size : %d\n", sampleCount);
                    // printk("\n ecg read %u \n",sampleCount);
                    if( ETAG[sampleCount - 1] == FIFO_OVF ){
                         printk("\n writing FIFO RST \n");
                        writeRegister( FIFO_RST , 0); // Reset FIFO

                    }

                    // for( idx = 0; idx < sampleCount; idx++ ) {
                    //     //   printk("%d|0x%u\n",
ecgSample[idx],ETAG[idx]);

                    //     }

                    if( ETAG[sampleCount - 1] == FIFO_EMPTY ){
                         printk("\n  FIFO empty/invalid \n");
                    }




                }
```

```
            }
            //  printk("\n ENDIF\n");
            //  NRFX_EXAMPLE_LOG_PROCESS();
        }
        // else printk("\n port error\nn");

    }
    printk("\n while broken \n");
}
```