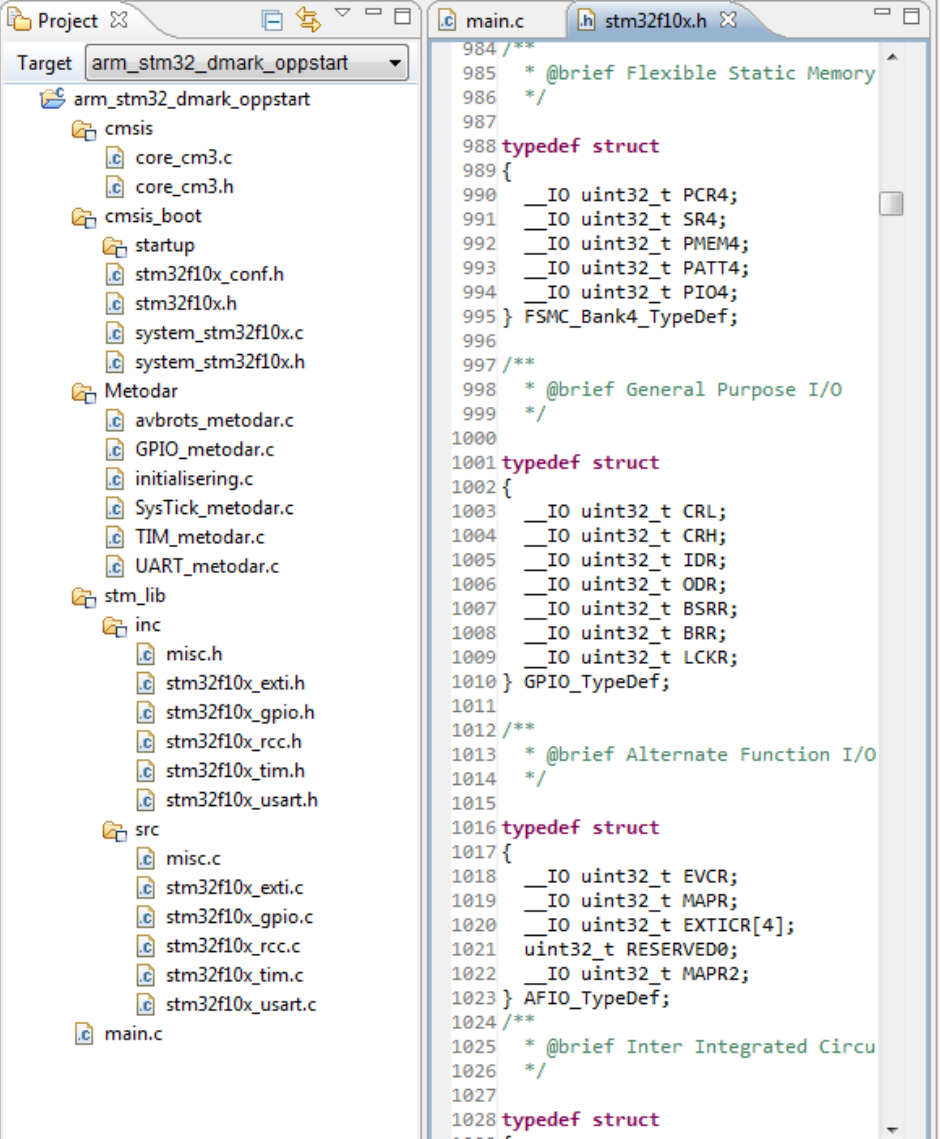

GPIO

Overview

Each Port have 16 pins, each I/O port bit is freely programmable to modes as follows: Input floating, Input pull-up, Input pull-down, Analog Input, Output open-drain, Output push-pull, Alternate function push-pull and Alternate function open-drain. All GPIOs are APB2 peripheral. In output mode, it is possible to modify only one or several bits in a single atomic APB2 write access. This component can be used to set pins mode and speed, to input and output, to set pins as Event output or EXTI Line, to remap the pin, to selects the Ethernet media interface, tp lock GPIO Pins configuration.

API Reference

GPIO_DeInit	Deinitializes the GPIOx peripheral registers to their default reset values.
GPIO_AFIODeInit	Deinitializes the Alternate Functions (remap, event control and EXTI configuration) registers to their default reset values.
GPIO_Init	Initializes the GPIOx peripheral according to the specified parameters in the GPIO_InitStruct.
GPIO_StructInit	Fills each GPIO_InitStruct member with its default value.
GPIO_ReadInputDataBit	Reads the specified input port pin.
GPIO_ReadInputData	Reads the specified GPIO input data port.
GPIO_ReadOutputDataBit	Reads the specified output data port bit.
GPIO_ReadOutputData	Reads the specified GPIO output data port.
GPIO_SetBits	Sets the selected data port bits.
GPIO_ResetBits	Clears the selected data port bits.
GPIO_WriteBit	Sets or clears the selected data port bit.
GPIO_Write	Writes data to the specified GPIO data port.

```
//
// Fil: SPI_metodar.c
// m.t.210814
//---------------------------------------

//---------------------------------------
// Inklusjonar og definisjonar
//---------------------------------------

#include "stm32f30x.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_spi.h"
#include "stm32f30x_rcc.h"

//---------------------------------------
// Globale variablar
//---------------------------------------

#include "extern_dekl_globale_variablar.h"

//---------------------------------------
// Funksjonsprototypar
//---------------------------------------

void SPI_oppstart(void);
int8_t SPI2_SendByte_Sokkel1(int8_t data);
void Exp_click_sokkel1_oppstart(void);
void Exp_click_sokkel1_sett_retningAB(int8_t moenster_A,int8_t moenster_B);
void Exp_click_sokkel1_skriv_til_AB(int8_t moenster_A,int8_t moenster_B);
//---------------------------------------
// Funksjonsdeklarasjonar
//---------------------------------------

void SPI_oppstart(void)  {
```

```
//Foerst oppsett av sjølve SPI-modulen
//----------------------------------------
  //Deklarasjon av initialiseringsstrukturen.
    SPI_InitTypeDef   SPI_InitStructure;

  //Slepp først til klokka paa SPI2.
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
```

1. Deklarasjon av struct

2. Finn og sett inn rett klokkefunksjon (Sjå blokkskjema i brukarmanual.)

3. Finn struct-en for modulen og rett val av bitmønster. Her må ein av og til inn i ref.manualen for å bli kjent med modulen.

```
  //Oppsett av SPI2
      SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;          /*!< Specifies
the SPI unidirectional or bidirectional data mode.
                                        This parameter can be a value of @ref SPI_data_direction */
      SPI_InitStructure.SPI_Mode = SPI_Mode_Master;               /*!< Specifies the SPI mode
(Master/Slave).
                                        This parameter can be a value of @ref SPI_mode */
      SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;           /*!< Specifies the SPI data size.
                                        This parameter can be a value of @ref SPI_data_size */
      SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;                /*!< Specifies the serial clock
steady state.
                                        This parameter can be a value of @ref SPI_Clock_Polarity */
      SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;                /*!< Specifies the clock active
edge for the bit capture.
                                        This parameter can be a value of @ref SPI_Clock_Phase */
      SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;                 /*!< Specifies whether the NSS
signal is managed by
                                        hardware (NSS pin) or by software using the SSI bit.
                                        This parameter can be a value of @ref
SPI_Slave_Select_management */
      SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;   /*!< Specifies the Baud
Rate prescaler value which will be
                                        used to configure the transmit and receive SCK clock.
                                        f_sclk = (f_sysklk/2)/preskalering */
      SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;            /*!< Specifies whether data
transfers start from MSB or LSB bit.
```

```
                                        This parameter can be a value of @ref
SPI_MSB_LSB_transmission */

//    SPI_InitStructure.SPI_CRCPolynomial;      /*!< Specifies the polynomial used for the CRC
calculation. */
```

4. Last ned oppsettet.

```
  //Initialiser, dvs. last ned konfigurasjonen i modulen
     SPI_Init(SPI2, &SPI_InitStructure);


//Så oppsett av GPIO-pinnane PB13 (SCLK), 14 (MISO) og 15 (MOSI) som blir brukt av SPI2-modulen
//-----------------------------------------
//Deklarasjon av initialiseringsstrukturen.
     GPIO_InitTypeDef GPIO_InitStructure_SPI2;
```

5. Så oppsett av dei GPIO-pinnane som er involverte og oppsett av klokke til GPIO-modulane.

```
  //Slepp til klokka paa GPIO-portB.
     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

  //Konfigurer PB13 - 15.
     GPIO_InitStructure_SPI2.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
     GPIO_InitStructure_SPI2.GPIO_Mode = GPIO_Mode_AF;
     GPIO_InitStructure_SPI2.GPIO_Speed = GPIO_Speed_Level_3;
     GPIO_InitStructure_SPI2.GPIO_PuPd  = GPIO_PuPd_NOPULL;

  //Initialiser, dvs. last ned konfigurasjonen i modulen
     GPIO_Init(GPIOB, &GPIO_InitStructure_SPI2);
```

6. Så spesifikasjon av den alternative funksjonen og ruting av kvar pinne til rett modul.

```
  //Knytt SPI2-pinnane til AF */
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource12, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource13, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource14, GPIO_AF_5);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource15, GPIO_AF_5);

//Så oppsett av GPIO-pinnen PB12 (CS) som blir brukt av SPI2-modulen
//-----------------------------------------
```

```
    //Deklarasjon av initialiseringsstrukturen.
    GPIO_InitTypeDef   GPIO_InitStructure;
    //Konfigurer som ein vanleg GPIO_pinne.
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_Level_3;
      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
      GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;

     //Initialiser, dvs. last ned konfigurasjonen i modulen
      GPIO_Init(GPIOB, &GPIO_InitStructure);

      //Sett CS = 1 (aktiv låg)
    GPIOB->ODR = GPIOB->ODR | GPIO_Pin_12;

// Aktiver SPI2                  7. Oppstart av modul
      SPI_Cmd(SPI2, ENABLE);


}

int8_t SPI2_SendByte_Sokkel1(int8_t data)  {

        GPIOB->ODR = GPIOB->ODR & (~(GPIO_Pin_12));
//        SPI2->DR = data;
           SPI_SendData8(SPI2, 0x7A);
           while(SPI2->SR & SPI_SR_FTLVL);
           while(SPI2->SR & SPI_SR_BSY);
         GPIOB->ODR = GPIOB->ODR | GPIO_Pin_12;
return  SPI_ReceiveData8(SPI2); //SPI2->DR;
}

void Exp_click_sokkel1_oppstart(void) {
// Sett retningsregistra i expansjonsmodulen i sokkel 1
      Exp_click_sokkel1_sett_retningAB(0x00, 0x00); // Alle er utgangar
```

```
}

void Exp_click_sokkel1_sett_retningAB(int8_t moenster_A,int8_t moenster_B) {
    GPIOB->ODR = GPIOB->ODR & (~(GPIO_Pin_12)); // CS låg
//        SPI2->DR = data;
//    SPI_I2S_SendData16(SPI2, 0x4000);
      SPI_SendData8(SPI2, 0x40);
      SPI_SendData8(SPI2, 0x00);
      SPI_SendData8(SPI2, moenster_A);
      SPI_SendData8(SPI2, moenster_B);
      while(SPI2->SR & SPI_SR_FTLVL);
      while(SPI2->SR & SPI_SR_BSY);

    GPIOB->ODR = GPIOB->ODR | GPIO_Pin_12;
}

void Exp_click_sokkel1_skriv_til_AB(int8_t moenster_A,int8_t moenster_B) {
    GPIOB->ODR = GPIOB->ODR & (~(GPIO_Pin_12)); // CS låg
//        SPI2->DR = data;
//    SPI_I2S_SendData16(SPI2, 0x4000);
      SPI_SendData8(SPI2, 0x40);
      SPI_SendData8(SPI2, 0x12); // GPIOA-adressa er 0x12
      SPI_SendData8(SPI2, moenster_A);
      SPI_SendData8(SPI2, moenster_B);
      while(SPI2->SR & SPI_SR_FTLVL);
      while(SPI2->SR & SPI_SR_BSY);

    GPIOB->ODR = GPIOB->ODR | GPIO_Pin_12;
}
/* This funtion is used to transmit and receive data
 * with SPI1
 *               data --> data to be transmitted
 *               returns received value
 */
//uint8_t SPI1_send(uint8_t data){
```

```
//
//    SPI1->DR = data; // write data to be transmitted to the SPI data register
//    while( !(SPI1->SR & SPI_I2S_FLAG_TXE) ); // wait until transmit complete
//    while( !(SPI1->SR & SPI_I2S_FLAG_RXNE) ); // wait until receive complete
//    while( SPI1->SR & SPI_I2S_FLAG_BSY ); // wait until SPI is not busy anymore
//    return SPI1->DR; // return received data from SPI data register
//}



//void InitMCU(){
//  // SPI config
//  SPI3_Init_Advanced(_SPI_FPCLK_DIV4, _SPI_MASTER  | _SPI_8_BIT |
//                     _SPI_CLK_IDLE_LOW | _SPI_FIRST_CLK_EDGE_TRANSITION |
//                     _SPI_MSB_FIRST | _SPI_SS_DISABLE | _SPI_SSM_ENABLE | _SPI_SSI_1,
//                     &_GPIO_MODULE_SPI3_PC10_11_12);
//}
//-------------------------------------------------------------------------------

 /* This file is part of the libopencm3 project.
 *
 * Copyright (C) 2009 Uwe Hermann <uwe@hermann-uwe.de>
 * Copyright (C) 2011 Stephen Caudle <scaudle@doceme.com>
 * Modified by Fernando Cortes <fermando.corcam@gmail.com>
 * modified by Guillermo Rivera <memogrg@gmail.com>
 *
 * This library is free software: you can redistribute it and/or modify
 * it under the terms of the GNU Lesser General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
 * GNU Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this library.  If not, see <http://www.gnu.org/licenses/>.
 */

//#include <libopencm3/stm32/rcc.h>
//#include <libopencm3/stm32/usart.h>
//#include <libopencm3/stm32/spi.h>
//#include <libopencm3/stm32/gpio.h>


//static void spi_setup(void)
//{
//    rcc_periph_clock_enable(RCC_SPI1);
//    /* For spi signal pins */
//    rcc_periph_clock_enable(RCC_GPIOA);
//    /* For spi mode select on the l3gd20 */
//    rcc_periph_clock_enable(RCC_GPIOE);
//
//    /* Setup GPIOE3 pin for spi mode l3gd20 select. */
//    gpio_mode_setup(GPIOE, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO3);
//    /* Start with spi communication disabled */
//    gpio_set(GPIOE, GPIO3);
//
//    /* Setup GPIO pins for AF5 for SPI1 signals. */
//    gpio_mode_setup(GPIOA, GPIO_MODE_AF, GPIO_PUPD_NONE,
//                GPIO5 | GPIO6 | GPIO7);
//    gpio_set_af(GPIOA, GPIO_AF5, GPIO5 | GPIO6 | GPIO7);
//
//    //spi initialization;
//    spi_set_master_mode(SPI1);
//    spi_set_baudrate_prescaler(SPI1, SPI_CR1_BR_FPCLK_DIV_64);
//    spi_set_clock_polarity_0(SPI1);
//    spi_set_clock_phase_0(SPI1);
```

```
//    spi_set_full_duplex_mode(SPI1);
//    spi_set_unidirectional_mode(SPI1); /* bidirectional but in 3-wire */
//    spi_set_data_size(SPI1, SPI_CR2_DS_8BIT);
//    spi_enable_software_slave_management(SPI1);
//    spi_send_msb_first(SPI1);
//    spi_set_nss_high(SPI1);
//    //spi_enable_ss_output(SPI1);
//    spi_fifo_reception_threshold_8bit(SPI1);
//    SPI_I2SCFGR(SPI1) &= ~SPI_I2SCFGR_I2SMOD;
//    spi_enable(SPI1);
//}
//
//static void usart_setup(void)
//{
//    /* Enable clocks for GPIO port A (for GPIO_USART2_TX) and USART2. */
//    rcc_periph_clock_enable(RCC_USART2);
//    rcc_periph_clock_enable(RCC_GPIOA);
//
//    /* Setup GPIO pin GPIO_USART2_TX/GPIO9 on GPIO port A for transmit. */
//    gpio_mode_setup(GPIOA, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO2 | GPIO3);
//    gpio_set_af(GPIOA, GPIO_AF7, GPIO2| GPIO3);
//
//    /* Setup UART parameters. */
//    usart_set_baudrate(USART2, 115200);
//    usart_set_databits(USART2, 8);
//    usart_set_stopbits(USART2, USART_STOPBITS_1);
//    usart_set_mode(USART2, USART_MODE_TX_RX);
//    usart_set_parity(USART2, USART_PARITY_NONE);
//    usart_set_flow_control(USART2, USART_FLOWCONTROL_NONE);
//
//    /* Finally enable the USART. */
//    usart_enable(USART2);
//}
//
//static void gpio_setup(void)
```

```
//{
//    rcc_periph_clock_enable(RCC_GPIOE);
//    gpio_mode_setup(GPIOE, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE,
//          GPIO8 | GPIO9 | GPIO10 | GPIO11 | GPIO12 | GPIO13 |
//          GPIO14 | GPIO15);
//}
//
//static void my_usart_print_int(uint32_t usart, int32_t value)
//{
//    int8_t i;
//    int8_t nr_digits = 0;
//    char buffer[25];
//
//    if (value < 0) {
//          usart_send_blocking(usart, '-');
//          value = value * -1;
//    }
//
//    if (value == 0) {
//          usart_send_blocking(usart, '0');
//    }
//
//    while (value > 0) {
//          buffer[nr_digits++] = "0123456789"[value % 10];
//          value /= 10;
//    }
//
//    for (i = nr_digits-1; i >= 0; i--) {
//          usart_send_blocking(usart, buffer[i]);
//    }
//
//    usart_send_blocking(usart, '\r');
//    usart_send_blocking(usart, '\n');
//}
//
```

```
//static void clock_setup(void)
//{
//   rcc_clock_setup_hsi(&hsi_8mhz[CLOCK_64MHZ]);
//}
//
//#define GYR_RNW                (1 << 7) /* Write when zero */
//#define GYR_MNS                (1 << 6) /* Multiple reads when 1 */
//#define GYR_WHO_AM_I           0x0F
//#define GYR_OUT_TEMP           0x26
//#define GYR_STATUS_REG         0x27
//#define GYR_CTRL_REG1          0x20
//#define GYR_CTRL_REG1_PD (1 << 3)
//#define GYR_CTRL_REG1_XEN(1 << 1)
//#define GYR_CTRL_REG1_YEN(1 << 0)
//#define GYR_CTRL_REG1_ZEN(1 << 2)
//#define GYR_CTRL_REG1_BW_SHIFT 4
//#define GYR_CTRL_REG4          0x23
//#define GYR_CTRL_REG4_FS_SHIFT 4
//
//#define GYR_OUT_X_L      0x28
//#define GYR_OUT_X_H      0x29
//
//int main(void)
//{
//   uint8_t temp;
//   int16_t gyr_x;
//   clock_setup();
//   gpio_setup();
//   usart_setup();
//   spi_setup();
//
//   gpio_clear(GPIOE, GPIO3);
//   spi_send8(SPI1, GYR_CTRL_REG1);
//   spi_read8(SPI1);
//   spi_send8(SPI1, GYR_CTRL_REG1_PD | GYR_CTRL_REG1_XEN |
```

```
//              GYR_CTRL_REG1_YEN | GYR_CTRL_REG1_ZEN |
//              (3 << GYR_CTRL_REG1_BW_SHIFT));
//    spi_read8(SPI1);
//    gpio_set(GPIOE, GPIO3);
//
//    gpio_clear(GPIOE, GPIO3);
//    spi_send8(SPI1, GYR_CTRL_REG4);
//    spi_read8(SPI1);
//    spi_send8(SPI1, (1 << GYR_CTRL_REG4_FS_SHIFT));
//    spi_read8(SPI1);
//    gpio_set(GPIOE, GPIO3);
//
//    while (1) {
//
//         gpio_clear(GPIOE, GPIO3);
//         spi_send8(SPI1, GYR_WHO_AM_I | GYR_RNW);
//         spi_read8(SPI1);
//         spi_send8(SPI1, 0);
//         temp=spi_read8(SPI1);
//         my_usart_print_int(USART2, (temp));
//         gpio_set(GPIOE, GPIO3);
//
//         gpio_clear(GPIOE, GPIO3);
//         spi_send8(SPI1, GYR_STATUS_REG | GYR_RNW);
//         spi_read8(SPI1);
//         spi_send8(SPI1, 0);
//         temp=spi_read8(SPI1);
//         my_usart_print_int(USART2, (temp));
//         gpio_set(GPIOE, GPIO3);
//
//         gpio_clear(GPIOE, GPIO3);
//         spi_send8(SPI1, GYR_OUT_TEMP | GYR_RNW);
//         spi_read8(SPI1);
//         spi_send8(SPI1, 0);
//         temp=spi_read8(SPI1);
```

```
//         my_usart_print_int(USART2, (temp));
//         gpio_set(GPIOE, GPIO3);
//
//         gpio_clear(GPIOE, GPIO3);
//         spi_send8(SPI1, GYR_OUT_X_L | GYR_RNW);
//         spi_read8(SPI1);
//         spi_send8(SPI1, 0);
//         gyr_x=spi_read8(SPI1);
//         gpio_set(GPIOE, GPIO3);
//
//         gpio_clear(GPIOE, GPIO3);
//         spi_send8(SPI1, GYR_OUT_X_H | GYR_RNW);
//         spi_read8(SPI1);
//         spi_send8(SPI1, 0);
//         gyr_x|=spi_read8(SPI1) << 8;
//         my_usart_print_int(USART2, (gyr_x));
//         gpio_set(GPIOE, GPIO3);
//
//         int i;
//         for (i = 0; i < 80000; i++)    /* Wait a bit. */
//              __asm__("nop");
//    }
//
//    return 0;
//}
```