Пример. ЖКИ. SPI (USCI_B1)

Задача 4.

4.1. С какой частотой выставляются биты данных на SPI?

```
// Set SPI mode: SMCLK for clock, keep reset
    UCB1CTL1 |= UCSWRST | UCSSEL__SMCLK;
```

После сброса DCOCLK = 2,097152 МГц

DCOCLKDIV = DCOCLK/2 ~ 1 МГц

SMCLK = DCOCLKDIV = 1 МГц

UCBRx определяют делитель входной тактовой частоты USCI

$$F_{BitClock} = f_{BRCLK} / UCBRx$$

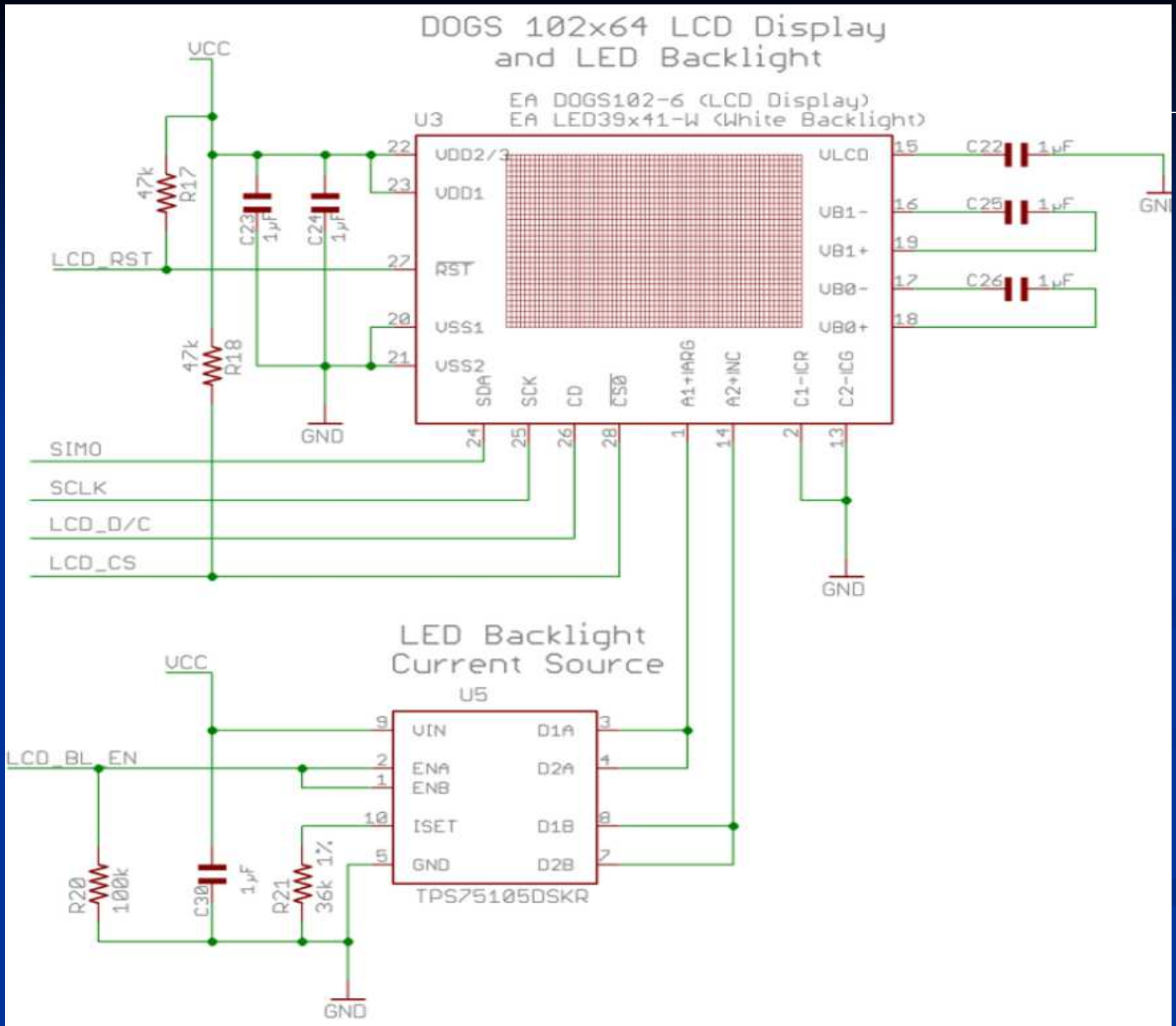
```
UCB1BR0 = 0x30; // Set SPI mode: low byte division
UCB1BR1 = 0;    // Set SPI mode: high byte division
```

$$F_{BitClock} = 1\ МГц / 0x30 = 21845{,}3\ Гц = 21{,}8\ КГц$$

Пример. ЖКИ. SPI (USCI_B1)

4.2. Почему не заработал LCD?

ЖКИ


DOGS 102x64 LCD Display
and LED Backlight
EA DOGS102-6 (LCD Display)
EA LED39x41-W (White Backlight)
U3
VCC
47k
R17
C23
1µF
C24
1µF
LCD_RST
47k
R18
GND
VDD2/3
VDD1
RST
VSS1
VSS2
VLCD
VB1-
VB1+
VB0-
VB0+
C22
C25
C26
SDA
SCK
CD
CS0
A1+IARG
A2+INC
C1-ICR
C2-ICG
SIMO
SCLK
LCD_D/C
LCD_CS
LED Backlight
Current Source
U5
VCC
LCD_BL_EN
VIN
ENA
ENB
ISET
GND
D1A
D2A
D1B
D2B
TPS75105DSKR
R20
100k
C30
1µF
R21
36k 1%
GND

ЖКИ

по схеме (по модулю)	по MSP430F5529
LCD_RST (RST)	P5.7 / TB0.1
SIMO (SDA)	P4.1 / PM_UCB1SIMO / PM_UCB1SDA
SCLK (SCK)	P4.3 / PM_UCB1CLK / PM_UCA1STE
LCD_D/C (CD)	P5.6 / TB0.0
LCD_CS (CS0)	P7.4 / TB0.2
LCD_BL_EN (ENA, ENB)	P7.6 / TB0.4

ЖКИ

- LCD_RST (RST) сброс (=0)
- SIMO (SDA) — SIMO данные
- SCLK (SCK) — синхросигнал
- LCD_D/C (CD) — команда (=0) / данные (=1)
- LCD_CS (CS0) — выбор устройства (=0)
- LCD_BL_EN (ENA, ENB) — включение подсветки

Пример. ЖКИ. SPI (USCI_B1)

```
#include <msp430.h>
void DOGS102_SPI(unsigned char byte1);
int main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    // P5.6 & P5.7 (DOGS102 pin CD & RST) set as output
    P5DIR |= (BIT7 | BIT6);
    // P7.4 & P7.6 (DOGS102 pin CS & ENA) set as output
    P7DIR |= (BIT4 | BIT6);
// P4.1 & P4.3 (DOGS102 pin CDA/SIMO & SCK) set as output
    P4DIR |= (BIT1 | BIT3);
// P7.4 & P7.6 (DOGS102 pin CS & ENA) no select, on bkLED
    P7OUT |= (BIT4 | BIT6);
    // device mode: P4.1 & P4.3 is UCB1SIMO & UCB1CLK
    P4SEL |= (BIT1 | BIT3);
    // P5.7 (DOGS102 pin RST) set "0" is reset
    P5OUT &= ~BIT7;
    __delay_cycles(25000);
    // P5.7 (DOGS102 pin RST) set "1" is no reset
    P5OUT |= BIT7;
    __delay_cycles(125000);
```

Пример. ЖКИ. SPI (USCI_B1)

```
    UCB1CTL1 |= UCSWRST;//Set SPI mode: reset logic is on
// Set SPI mode: master, MSB first, data latch on rising
    UCB1CTL0 |= UCSYNC | UCMST | UCMSB | UCCKPH;
    // Set SPI mode: SMCLK for clock, keep reset
    UCB1CTL1 |= UCSWRST | UCSSEL__SMCLK;
    UCB1BR0 = 0x30;   // Set SPI mode: low byte division
    UCB1BR1 = 0;      // Set SPI mode: high byte division
    UCB1CTL1 &= ~UCSWRST;//Reset SPI==Start SPI interface

    P5OUT &= ~BIT6;   // DOGS102 format: command
    DOGS102_SPI(0x2F);// SPI transmit.Command: Power on
    DOGS102_SPI(0xAF);// SPI transmit.Command: Display on
    // SPI transmit. Command: Set LSB column address
    DOGS102_SPI(0);
    // SPI transmit. Command: Set MSB column address
    DOGS102_SPI(0x14);
    // SPI transmit. Command: Set page address
    DOGS102_SPI(0xB4);
    P5OUT |= BIT6;        // DOGS102 format: data
    DOGS102_SPI(0xFF);   // SPI transmit. 8 pixels are set
```

Пример. ЖКИ. SPI (USCI_B1)

```
    // Enter LPM0, enable interrupts
    __bis_SR_register(LPM0_bits + GIE);
    __no_operation();   // For debugger
    return 0;
}

void DOGS102_SPI(unsigned char byte1)
{
    // Wait TXIFG == TXBUF is ready for new data
    while (!(UCB1IFG & UCTXIFG));
// P7.4 (DOGS102 pin CS) set "0" is start SPI operation
    P7OUT &= ~BIT4;
    UCB1TXBUF = byte1;          // Start SPI transmit
    // Wait until USCI_B1 SPI interface is no longer busy
    while (UCB1STAT & UCBUSY);
    // P7.4 (DOGS102 pin CS)set "1" is stop SPI operation
    P7OUT |= BIT4;
}
```

Пример. ЖКИ. SPI (USCI_B1)

4.2. Почему не заработал LCD?

Возможно, что-то с синхронизацией приема-передачи? Контроллер DOGS102 допускает CLK передачи до 33 МГц... У нас всего 21,8 КГц

Возможно, пишем за границы экрана?

```
    DOGS102_SPI(0);   // SPI: Set LSB column address
    DOGS102_SPI(0x14);// SPI: Set MSB column address
    DOGS102_SPI(0xB4);// SPI: Set page address
```

Столбец: 0x40 = 64. Допустимо:
0 – 101 (ориентация 6 o'clock)
30 – 131 (ориентация 12 o'clock)
Адрес страницы: 4. Допустимы 0-7.

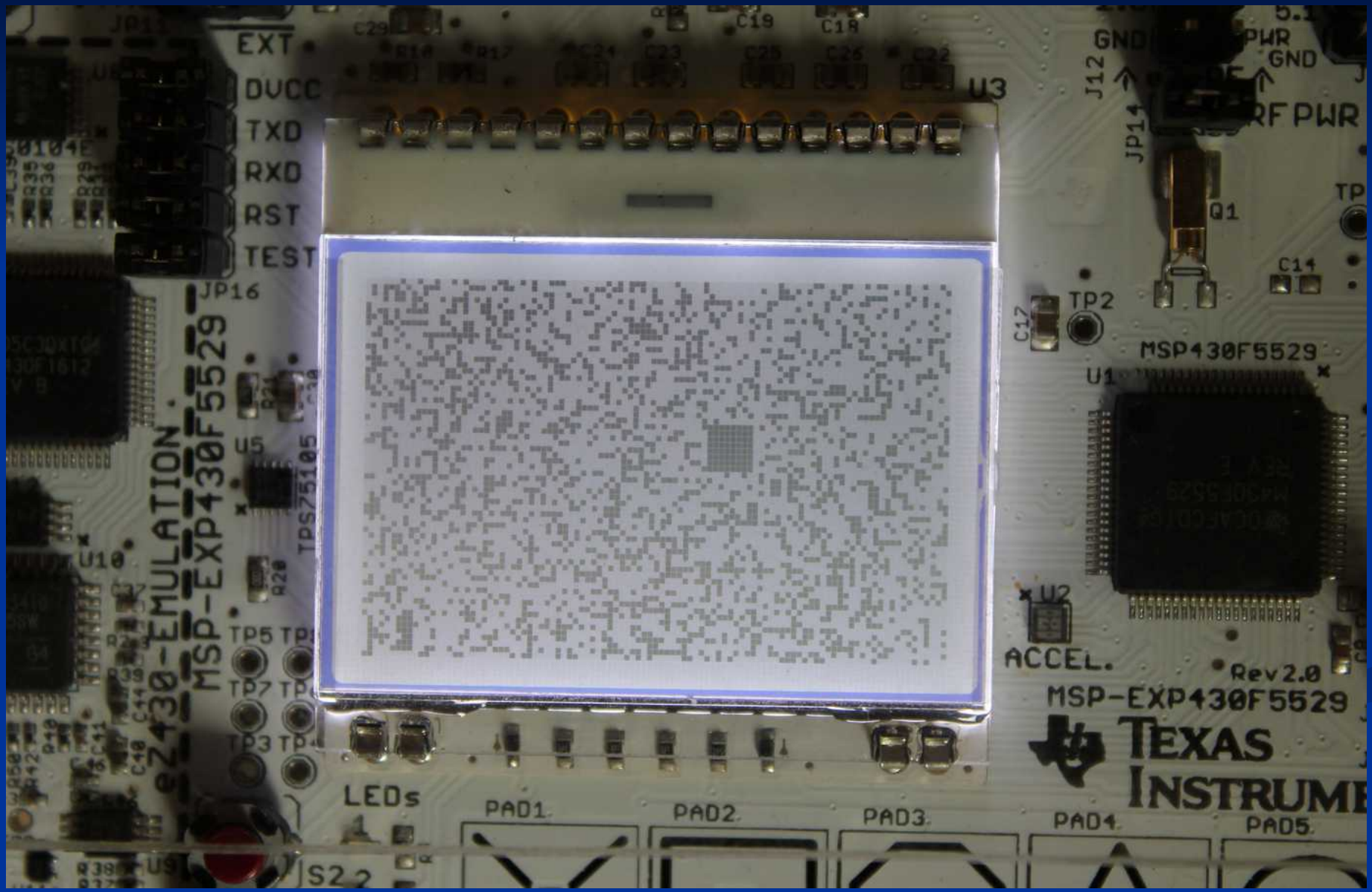
Пример. ЖКИ. SPI (USCI_B1)

4.2. Почему не заработал LCD?

```
int main(void) {
unsigned char i;
...
for(i=0;i<8;i++)          // For 8 columns
{
// SPI transmit. Command: Set LSB column addr
DOGS102_SPI(i);
P5OUT |= BIT6;            // DOGS102 format: data
// SPI transmit. 8 pixels are
setDOGS102_SPI(0xFF);
P5OUT &= ~BIT6;   // DOGS102 format: command
}
```

Пример. ЖКИ. SPI (USCI_B1)

Задача 5. Почему не заработал LCD?



Пример. Акселерометр. SPI (USCI A0)

4.3. Какая тактовая частота поступает на CLK вход акселерометра (какая частота передачи бит)?

```
// Set SPI mode: SMCLK for clock, keep reset
    UCA0CTL1 |= UCSWRST | UCSSEL__SMCLK;
```

После сброса DCOCLK = 2,097152 МГц

DCOCLKDIV = DCOCLK/2 ~ 1 МГц

SMCLK = DCOCLKDIV = 1 МГц

UCBRx определяют делитель входной тактовой частоты USCI

$$F_{BitClock} = f_{BRCLK} / UCBRx$$

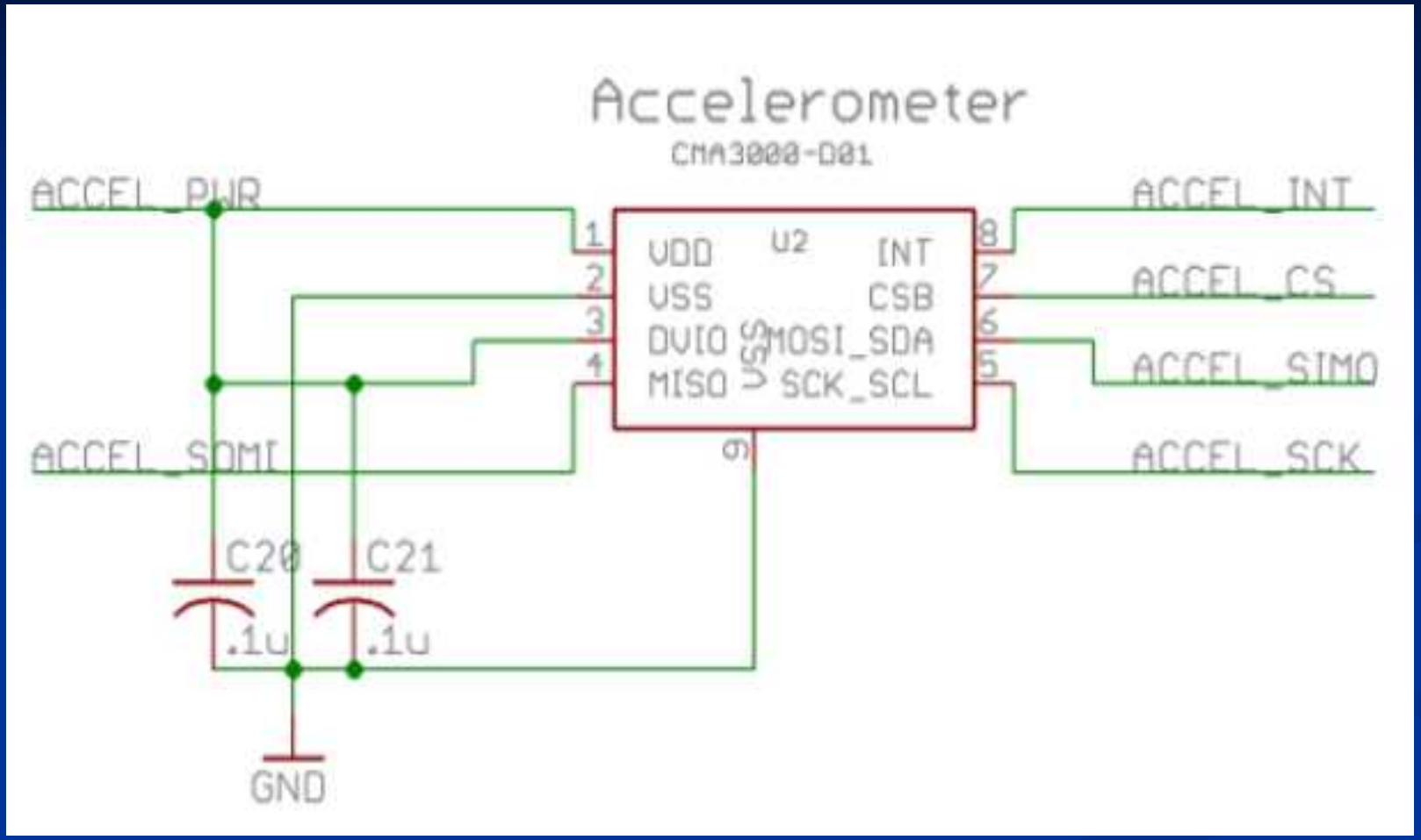
```
UCA0BR0 = 0x30; // Set SPI mode: low byte division
UCA0BR1 = 0;    // Set SPI mode: high byte division
```

$$F_{BitClock} = 1\ МГц / 0x30 = 21845,3\ Гц = 21,8\ КГц$$

Пример. SPI. Акселерометр

4.4. Почему плата не реагирует на перемещение?

Акселерометр



Акселерометр

по схеме (по модулю)	по MSP430F5529
ACCEL_PWR (VDD, DVIO)	P3.6 / TB0.6
ACCEL_SOMI (MISO)	P3.4 / UCA0RXD / UCA0SOMI
ACCEL_INT (INT)	P2.5 / TA2.2
ACCEL_CS (CSB)	P3.5 / TB0.5
ACCEL_SIMO (MOSI_SDA)	P3.3 / UCA0TXD / UCA0SIMO
ACCEL_SCK (SCK_SCL)	P2.7 / UCB0STC / UCA0CLK

Пример. SPI. Акселерометр

```
#include <msp430.h>

char cma3000_SPI(unsigned char byte1, unsigned char
byte2);

int main(void) {
WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
P1DIR |= BIT0;              // P1.0 (LED1) set as output
P8DIR |= (BIT1 | BIT2); // P8.1 & P8.2(LED2 & LED3)output
P1OUT &= ~BIT0;             // LED1 off
P8OUT &= ~(BIT1 | BIT2);// LED2 & LED3 off
P2DIR &= ~BIT5;             // P2.5 (cma3000 pin INT) input
P2OUT |= BIT5;// P2.5 (cma3000 pin INT) pull-up resistor
P2REN |= BIT5;// P2.5 (cma3000 pin INT) enable resistor
P2IE |= BIT5; // P2.5 (cma3000 pin INT) interrupt enable
// P2.5 (cma3000 pin INT) edge for interrupt: low-to-high
P2IES &= ~BIT5;
P2IFG &= ~BIT5;// P2.5 (cma3000 pin INT) clear int flag
```

Пример. SPI. Акселерометр

```
P3DIR |= BIT5;     // P3.5 (cma3000 pin CSB) set as output
// P3.5 (cma3000 pin CSB) set "1" is disable cma3000
P3OUT |= BIT5;
P2DIR |= BIT7;     // P2.7 (cma3000 pin SCK) set as output
P2SEL |= BIT7;     // device mode: P2.7 is UCA0CLK
// P3.3 & P3.6 (cma3000 pin MOSI, PWR) set as output
P3DIR |= (BIT3 | BIT6);
P3DIR &= ~BIT4;    // P3.4 (cma3000 pin MISO) set as input
// device mode: P3.3 - UCA0SIMO, P3.4 - UCA0SOMI
P3SEL |= (BIT3 | BIT4);
// P3.6 (cma3000 pin PWR) set "1" is power cma3000
P3OUT |= BIT6;

UCA0CTL1 |= UCSWRST; // Set SPI mode: reset logic is on
// Set SPI mode: master, MSB first, data latch on rising
UCA0CTL0 |= UCSYNC | UCMST | UCMSB | UCCKPH;
// Set SPI mode: SMCLK for clock, keep reset
UCA0CTL1 |= UCSWRST | UCSSEL__SMCLK;
```

Пример. SPI. Акселерометр

```
UCA0BR0 = 0x30;    // Set SPI mode: low byte division
UCA0BR1 = 0;       // Set SPI mode: high byte division
UCA0MCTL = 0;      // Set SPI mode: no modulation
UCA0CTL1 &= ~UCSWRST;// Reset SPI == Start SPI interface

// Start SPI transmit. cma3000 format:
// 1 = REVID register, 0 - read, 0 - predefined
// Second byte for read operation can be any
cma3000_SPI(0x4, 0);
__delay_cycles(1250); // ????

// SPI: 10 = CTRL register, 1 - write, 0 - predefined
// cma3000 CTRL register set: 2g, disable I2C, 400 Hz
cma3000_SPI(0xA, BIT7 | BIT4 | BIT2);
__delay_cycles(25000); // ????

// Enter LPM0, enable interrupts
__bis_SR_register(LPM0_bits + GIE);
__no_operation();              // For debugger
return 0; }
```

Пример. SPI. Акселерометр

```
char cma3000_SPI(unsigned char byte1, unsigned char
byte2) {
char indata;
// P3.5 (cma3000 pin CSB) set "0" is start SPI operation
P3OUT &= ~BIT5;
indata = UCA0RXBUF; // ??????
// Wait TXIFG == TXBUF is ready for new data
while (!(UCA0IFG & UCTXIFG)) ;
UCA0TXBUF = byte1; // Start SPI transmit. Send first byte
// Wait RXIFG == RXBUF have new data
while (!(UCA0IFG & UCRXIFG));
indata = UCA0RXBUF; // ??????
// Wait TXIFG == TXBUF is ready for new data
while (!(UCA0IFG & UCTXIFG));
UCA0TXBUF = byte2; // Start SPI transmit. Send second byte
// Wait RXIFG == RXBUF have new data
while (!(UCA0IFG & UCRXIFG));
```

Пример. SPI. Акселерометр

```
// Read SPI data from accel. in 2 byte in read command
// ????? in write command
indata =UCA0RXBUF;
// Wait until USCI_A0 SPI interface is no longer busy
while (UCA0STAT & UCBUSY) ;
// P3.5 (cma3000 pin CSB) set "1" is stop SPI operation
P3OUT |= BIT5;
return indata;
}

// Port 2 interrupt service routine == cma3000 interrupt
#pragma vector=PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
 char dx, dy, dz;
 if (P2IN & BIT5)  {
 P1OUT &= ~BIT0;            // LED1 off
 P8OUT &= ~(BIT1 | BIT2);   // LED2 & LED3 off
```

Пример. SPI. Акселерометр

```
// Start SPI transmit. cma3000 format:
// 110 = DOUTX register, 0 - read, 0 - predefined
// Second byte for read operation can be any
dx = cma3000_SPI(0x18, 0);
__delay_cycles(1250); // ????
// SPI: 111 = DOUTY register, 0 - read, 0 - predefined
dy = cma3000_SPI(0x1C, 0);
__delay_cycles(1250); // ????
// SPI: 1000 = DOUTZ register, 0 - read, 0 - predefined
dz = cma3000_SPI(0x20, 0);
__delay_cycles(1250); // ????
// LED1 on if acceleration on X more than threshold
if (dx > 32) P1OUT |= BIT0;
// LED2 on if acceleration on Y more than threshold
if (dy > 32) P8OUT |= BIT1;
// LED3 on if acceleration on Z more than threshold
if (dz > 32) P8OUT |= BIT2;
P2IFG &= ~BIT5;          // reset interrupt flag
    }
}
```

Пример. SPI. Акселерометр

Видео

05. Accelerometer_3

Что хотели получить

Пример. SPI. Акселерометр

Видео

05. Accelerometer_0

Что на самом деле вышло

Пример. SPI. Акселерометр

```
// P2IFG &= ~BIT5;                  // reset interrupt flag
    }
}
```

Видео

05. Accelerometer_2

Акселерометр работает, данные поступают, но...

Пример. Акселерометр. SPI (USCI_A0)

4.4.

$F_{BitClock}$ = 1 МГц / 0x30 = 21845,3 Гц = 21,8 КГц

Или 2730,7 Байта / сек

На одну команду акселерометра надо 2 байта:

= 1365,3 команды / сек

На полное чтение данных (X, Y, Z) надо три команды:

455,1 операций чтения в сек

+ программные задержки и циклы ожидания

< 400 операций чтения в сек

```
// cma3000 CTRL reg set: 2g, disable I2C, 400 Hz
cma3000_SPI(0xA, BIT7 | BIT4 | BIT2);
```

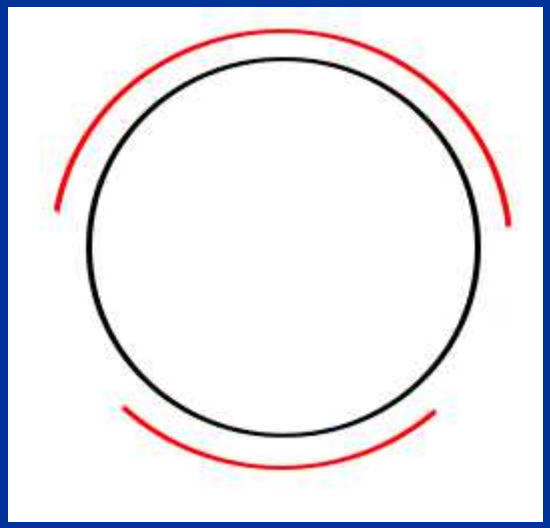
Частота готовности данных (400) > частота обмена

При сбросе флага прерывания уже есть сигнал на линии. Перепад Lo / Hi не обнаруживается, т. е. перепад проходит, когда флаг и так еще установлен

Пример. Акселерометр. SPI (USCI_A0)

4.4. Частота готовности данных (400) > частота обмена. При сбросе флага прерывания уже есть сигнал на линии. Перепад Lo / Hi не обнаруживается

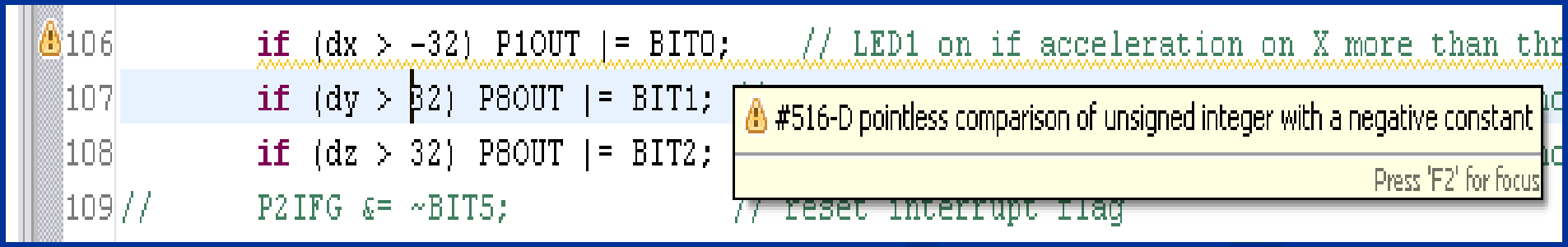
```
// SPI. cma3000 format: 10 = CTRL register, 1 - write, 0
// SPI. cma3000 CTRL set: 2g, disable I2C, 100 Hz
    cma3000_SPI(0xA, BIT7 | BIT4 | BIT1);
    __delay_cycles(25000);
```



Очень похоже, что вместо отрицательных данных передаются очень большие положительные числа

Пример. Акселерометр. SPI (USCI_A0)

```
__interrupt void PORT2_ISR(void)
{ char dx, dy, dz;
...
if (dx > -32) P1OUT |= BIT0;
```




```
signed char dx, dy, dz;
```

Пример. SPI. Акселерометр

Видео

05. Accelerometer_3

Что хотели получить и наконец получили

Пример. SPI. Акселерометр

4.5. Зачем __delay_cycle?

Абсолютно ни к чему. Если их все удалить и вернуть частоту данных 400 Гц, все будет работать, т.к. без задержек

f готовности данных (400) < f обмена (455)

Пример. SPI. Акселерометр

4.6. Зачем при передаче данных в акселерометр выполняется чтение данных?

Освобождаем регистр приема данных. Иначе операция чтения возьмет предыдущий байт из него, не дожидаясь приема на линии