ITAMS
Applied Microcontroller Systems

Indholdsfortegnelse

LAB 1

1.1 Lektion 08-02-2018

1.1.1 Simulator and JTAG debugging

1. To be able to use the Atmel Studio 7 simulator.
2. To be able to set up and use the Atmel - ICE for debugging purposes

Exercise, Part 1

- Create a project and write a C program (intended for debugging).

Exercise, Part 2

- Simulate the program using the Atmel Studio Simulator ("debugger"). The simulator is integrated in the Atmel Studio IDE.
- When using the simulator, the program execution can be simulated in Atmel Studio (not in real time).
- **Important** to disable the compiler optimization.
  - Select "Project" ⇒ "Properties"(Alt + F7).
  - Select "Toolchain", and mark "Optimization".
  - Select "Optimization Level"to "None".
- Remember after debugging, normally the "Optimization Level"should be set to generate effective code.
- Select "Debug" ⇒ "Start Debugging and Break"(ALT + F5).
- Note that the program is "just"simulated, it will take more time than executing the program on the microcontroller.

- Registers can be watched in the window "Processor".
- Register contents that has been changed by an instruction, will be marked with a red color.
- "Stop Watch"show the exact time elapsed since program execution started.
- Monitoring specific variables and registers can be done using the "Watch Window".
  - Select "Debug" ⇒ "Windows" ⇒ "Watch" ⇒ "Watch1".
- Monitoring I/O registers of the microcontroller using "I/O View".
  - Select "Debug" ⇒ "Windows" ⇒ "I/O View".

Exercise, Part 3

- Use the Atmel-ICE for debugging the program while it is executing in target.
- Possible since Mega2560 has an on chip hardware JTAG interface.
- Advantage is that the program is executed in real time.
- Disadvance is that 4 pins of PORTF are allocated for the JTAG interface.
- **Important** to enable JTAG interface by setting a fuse. Can only be done using ISP programming.
- Connect the ISP 6 pin connector of the ICE to the Mega2560 ISP connector (plastic tab → Mega2560 chip).
  - Select "Device Programming".
  - Select Atmel-ICE as tool and interface as ISP.
- The Arduino board comes with a boot loader installed. When using the ICE, we will probably erase the boot loader.
- Start by reading the flash content of your board to a .hex file.
- Go to tab "Fuses"and check "JTAGEN". Click Program.
- Set up Atmel Studio to use the Atmel-ICE for debugging instead of the simulator.

  - Select "Project Properties"(Alt + F7).
  - Select "Tool" ⇒ "Atmel-ICE"as debugger ⇒ "JTAG"as interface.

- Connect the mini squid cable of the Atmel-ICE.

Signal	ICE port pin	Mega2560 pin
TCK	1	PORTF, 4
TMS	5	PORTF, 5
TDO	3	PORTF, 6
TDI	9	PORTF, 7

- **Restore** the Arduino with the original bootloader, program the device with the .hex file, saved earlier.

- To free the JTAG pins of PORT F, you will have to clear the JTAG fuse "JTAGEN".

LAB 3

2.1 Lektion 15-02-2018

2.1.1 GraphicTFT display driver

Driver for "ITDB02 320 x 240 TFT display module, Version 2"mounted at "ITDB02 Arduino Mega2560 Shield".

Display controller = ILI 9341.

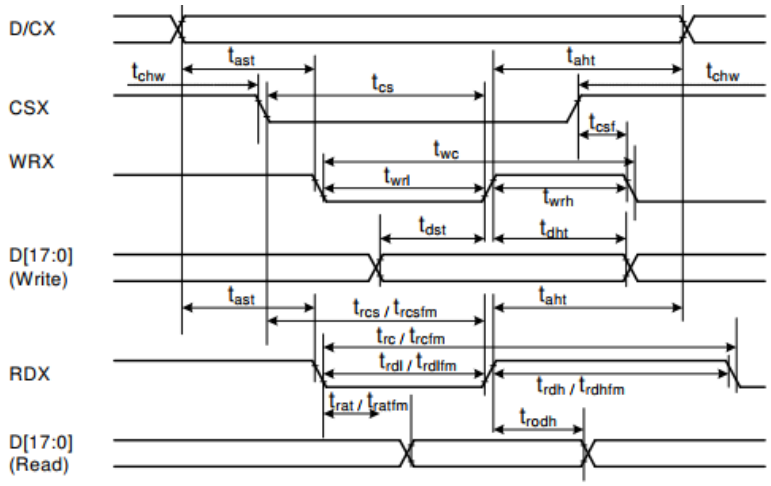
Connections	
DB15-DB8:	PORT A
DB7-DB0:	PORT C
RESETx:	PORT G, bit 0
CSx:	PORT G, bit 1
WRx:	PORT G, bit 2
RS (=D/Cx):	PORT D, bit 7

```
// Data port definitions:
#define DATA_PORT_HIGH PORTA
#define DATA_PORT_LOW  PORTC

// Control port definitions:
#define WR_PORT PORTG
#define WR_BIT 2
#define DC_PORT PORTD
#define DC_BIT  7  // SHIELD RS
#define CS_PORT PORTG
#define CS_BIT  1
#define RST_PORT PORTG
```

```
#define RST_BIT 0
```

Start by implementing the basic, time-critical functions.



Figur 2.1: Timing Characteristics (8080- system).

`PORTD &= ~(1 << n);` will set PIN n low.

`PORTD |= (1 << n);` will set PIN n high.

```
void WriteCommand(unsigned int command)
{
DATA_PORT_LOW = command;
DC_PORT &= ~(1<<DC_BIT);        // DCX LOW = COMMAND MODE
CS_PORT &= ~(1<<CS_BIT);        // CSX LOW
WR_PORT &= ~(1<<WR_BIT);        // WRX LOW
_NOP();                         // DELAY = twrl 15ns
WR_PORT |= (1<<WR_BIT);         // WRX HIGH
_NOP();                         // DELAY = tcf 10ns
}

void WriteData(unsigned int data)
{
DATA_PORT_HIGH = (data >> 8);       // MSB
DATA_PORT_LOW  = data;              // LSB
DC_PORT |= (1<<DC_BIT);             // DCX HIGH = DATA MODE
CS_PORT &= ~(1<<CS_BIT);            // CSX LOW
WR_PORT &= ~(1<<WR_BIT);            // WRX LOW
_NOP();                             // DELAY = twrl 15ns
WR_PORT |= (1<<WR_PORT);            // WRX HIGH
```

```
_NOP();                                    // DELAY = twcf 10ns
}
```

LAB7

3.1 Lektion 01-03-2018

3.1.1 FreeRTOS

Understanding and Hands On the FreeRTOS real time operating system.

Exercise, Part 1

Exercise, Part 2

- The first (extra) task shall wait for switch SW0 to be pressed. When this is the case, the semaphore shall be given. Use the switch port driver to read the switch.
- The second task shall wait for the semaphore (take it). When taken, LED7 shall flash (on for a few milliseconds). Use the LED driver to control LED7.

```
xSemaphoreHandle xSemaphore1 = NULL;

void vSwitchTask( void * pvParameters )
{
xSemaphoreTake( xSemaphore1, portMAX_DELAY );
    while(1)
    {
    /* We now have the semaphore and can wait for SW0 to
    give the semaphore. */
    if (switchOn(0))
        xSemaphoreGive(xSemaphore1);
    }
}
```

```
void vLEDFlashTask3( void * pvParameters )
{
    /* If the semaphore is not available wait to see if it
    becomes free. */
    if( xSemaphoreTake( xSemaphore1, portMAX_DELAY ))
    {
        /* We were able to obtain the semaphore. */
        turnOnLED(7);
        vTaskDelay(500);
        turnOffLED(7);

        /* Release the semaphore. */
        xSemaphoreGive( xSemaphore1 );
    }
}

int main(void)
{
    initLEDport();
    initSwitchPort();

    /* Create the semaphore to guard a shared resource. */
    vSemaphoreCreateBinary(xSemaphore1);

    xTaskCreate( vSwitchTask,    ( signed char * ) "SW0", ...
    configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
    xTaskCreate( vLEDFlashTask3, ( signed char * ) "LED7", ...
    configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
    vTaskStartScheduler();
    while(1)
    {}
}
```

Exercise, Part 3

Write a program having 3 tasks.

Also the program has a global variable called ”count”(type unsigned char), being a common resource for two of the tasks. Therefore this variable has to be protected using a (binary) semaphore!

A queue for 10 bytes is used for data transfer between the tasks.

- The first task shall decrement "count"when switch SW0 is pressed. Then the value of "count"shall be put onto the queue.
- The second task shall increment count when SW1 is pressed. Then the value of "count"shall be put onto the queue.
- The third task must receive items from the queue. Each time a new item is received, the value of the element shall be displayed at the LEDs (use the LED driver). Also the same value shall be sent as a text string to an attached terminal (use the UART driver function SendInteger()).
- Use an interrupt (for example external INT0 edge-triggered) to reset "count"to 0. Also the value 0 shall be put onto the queue. Remember to use special FreeRTOS functions when called form an ISR

```
unsigned char count = 0;
xSemaphoreHandle xSemaphore1 = NULL;
xQueueHandle xQueue1 = NULL;

void initExtInts()
{
    /* INT2:Falling edge. */
    EICRA = 0b00100000;
    /* Enable extern interrupt INT2. */
    EIMSK |= 0b00001000;
}

ISR (INT2_vect)
{
    if( xSemaphoreTake( xSemaphore1, 0 ) == pdTRUE )
    {
        count = 0;
        /* Don't block if the queue is already full. */
        xQueueSendToBackFromISR( xQueue1, &count, 0 );
        xSemaphoreGiveFromISR( xSemaphore1, NULL );
    }
}
```

```
void vDecCountTask( void * pvParameters )
{
    while(1)
    {
        if (switchOn(0))
        {
            xSemaphoreTake( xSemaphore1, portMAX_DELAY );
            count--;
            /* Don't block if the queue is already full. */
            xQueueSendToBack( xQueue1, &count, 0 );
            xSemaphoreGive( xSemaphore1 );
        }
        vTaskDelay(50);
    }
}

void vIncCountTask( void * pvParameters )
{
    while(1)
    {
        if (switchOn(1))
        {
            xSemaphoreTake( xSemaphore1, portMAX_DELAY );
            count++;
            /* Don't block if the queue is already full. */
            xQueueSendToBack( xQueue1, &count, 0 );
            xSemaphoreGive( xSemaphore1 );
        }
        vTaskDelay(50);
    }
}

void vReceiveQueue(void *pvParameters)
{
    unsigned char localCount;
    while(1)
    {
        /* Receive an item on the created queue.
        Block for portMAX_DELAY ticks if a message is not
        immediately available.*/
        if( xQueueReceive( xQueue1, &localCount, portMAX_DELAY ) )
        {
```

```
            writeAllLEDs(localCount);
            SendInteger(localCount);
            SendChar('\r');
            SendChar('\n');
        }
    }
}


int main(void)
{
    initLEDport();
    initSwitchPort();
    initExtInts();
    InitUART(115200, 8, 'N');

    /* Create the semaphore to guard a shared resource. */
    vSemaphoreCreateBinary(xSemaphore1);

    /* Create a queue capable of containing 10 unsigned
    char values.*/
    xQueue1 = xQueueCreate(10, sizeof(unsigned char));

    xTaskCreate( vDecCountTask, ( signed char * ) "DEC", ...
    configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
    xTaskCreate( vIncCountTask, ( signed char * ) "INC", ...
    configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
    xTaskCreate( vReceiveQueue, ( signed char * ) "REC", ...
    configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
    vTaskStartScheduler();
    while(1)
    {}
}
```