Exercise sheet 13 – I/O

Goals:

- Programmed I/O
- Interrupt driven I/O
- DMA

Exercise 13.1: Synchronisation commands

(a) Describe a machine instruction for synchronisation and its interaction with the application, OS (operating system), and bus level. *Hint: You may consider the TAS command and semaphores, learned in the Operating Systems (Betriebssysteme) lecture as well as in this lecture. Use some pseudo-code to describe your ideas.*

Proposal for solution: Remember the *TAS* (test and set) command from lecture the Operating Systems? We learned that the OS is using it to provide semaphores for synchronization.

Application	OS	Bus	
P(s);	P_OP(s) {		
	do {	read cycle	atomic
	TAS_LOCK;	write cycle	
	} while(!successful)		
	}		
// critical section	. . .		

V(s);	V_OP(s) {
	LOCK = 0; // release
	}		

Semaphore (P()/V() operation)	TAS instruction with (LOCK variable)	Atomic read/write bus cycle (LOCK bus signal)	

Exercise 13.2: Programmed I/O (single transfer) with busy wait (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the busy wait approach. Compare the lecture for that.

(a) Update the `CA_exercises` repository with `git pull`.

(b) In
`CA_exercises/sheet_13_io/io_pc_prog_io_busy_wait/io_pc_prog_io_busy_wait.c`
you will find a skeleton file.

(c) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```
#include <stdlib.h>
#include <stdbool.h>
#include <inttypes.h>

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
    uint16_t RBUF; //receive buffer register
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

#define BUFFER_SIZE (16)
uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

int main(void) {
    for(int i = 0; i < BUFFER_SIZE; ++i) {
        while((bool)(FSI.CSR & RRDY) == false) { //wait until FSI has provided next ch
            //busy wait (do nothing)
        }
        buffer[i] = (uint8_t)FSI.RBUF;
    }

    return EXIT_SUCCESS;
}
```

Exercise 13.3: Programmed I/O (single transfer) with polling (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the polling approach. Compare the lecture for that.

(a) In
`CA_exercises/sheet_13_io/io_pc_prog_io_polling/io_pc_prog_io_polling.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```
#include <stdlib.h>
#include <inttypes.h>
#include <stdbool.h> //bool

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
```

```c
    uint16_t RBUF; //receive buffer register
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

#define BUFFER_SIZE (16)
uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

int main(void) {
    for(int i = 0; i < BUFFER_SIZE; ++i) {

        while(true) {
            if((bool)(FSI.CSR & RRDY) == true) { //if the FSI has provided next charac
                buffer[i] = (uint8_t)FSI.RBUF;   //copy the received character
                break;                           //proceed with next character
            } else {
                //do something else...
            }
        }
    }

    return EXIT_SUCCESS;
}
```

Exercise 13.4: Interrupt driven I/O (single transfer) (pseudo C code)

Consider a system with the *F-Bus serial interface* (FSI). You want to receive data (characters) from the FSI with the interrupt control approach. Compare the lecture for that.

(a) In
`CA_exercises/sheet_13_io/io_pc_interrupt_io/io_pc_interrupt_io.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to read 16 bytes (characters) from the *F-Bus serial interface* (FSI) into the memory buffer.

Proposal for solution:

```c
#include <stdlib.h>
#include <inttypes.h>

typedef volatile struct { //FSI interface
    uint16_t CSR;  //control and status register
    uint16_t TBUF; //transmit buffer register
    uint16_t RBUF; //receive buffer register
    uint16_t CFR;  //configuration register
} FsiStruct;

//FSI: FsiStruct is mapped to the memory position 0xFF000000
#define FSI  (*((FsiStruct*)(0xFF000000)))

#define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
```

```c
#define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
#define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
#define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
//more defines...

typedef void (*ISR_t)(void); //Function pointer for an ISR
//INTVECTOR: ISR_t is mapped to the memory position 0x000000C8
#define INTVECTOR (*((ISR_t*)(0x000000C8)))

#define BUFFER_SIZE (16)
volatile uint8_t counter = 0;
volatile uint8_t buffer[BUFFER_SIZE] = {0}; //initialise all elements with 0

void ISR_serial_read(); //prototype

int main(void) {
    //Start transfer
    INTVECTOR = &ISR_serial_read; //set ISR for receive character
    FSI.CSR |= RIE; //enable RIE flag

    //do something else while transfer is in progress...

    return EXIT_SUCCESS;
}

//an interrupt is triggered if
//the hardware has provided a new character
void ISR_serial_read() {
    buffer[counter] = (uint8_t)FSI.RBUF;
    counter++;
    if (counter == BUFFER_SIZE){
        FSI.CSR &= ~RIE; //delete the RIE flag to stop the transfer
                         //~ is the bitwise inversion.

        //buffer full -> inform outer world (somehow)
    }
}
```

Exercise 13.5: DMA programming (pseudo C code)

Consider a system with the *F-Bus DMA disk* (FDD). You want to write data (some words) from the memory with the DMA approach to the disk. Compare the lecture for that.

(a) In
`CA_exercises/sheet_13_io/io_pc_dma/io_pc_dma.c`
you will find a skeleton file.

(b) Complete the skeleton with pseudo C to write 16 words (4 bytes per word) from the memory to the *F-Bus DMA disk* (FDD).

- Source (memory) starting address: 0x400000
- Target (disk) starting address: 0x4711

Hint: Source, destination, how much, GO!

Proposal for solution:

```c
#include <stdlib.h>
#include <stdbool.h>
#include <inttypes.h>
```

```

typedef volatile struct { //FDD interface
    uint32_t CSR;  //control and status register
    uint32_t DARH; //disk address register HI
    uint32_t DARL; //disk address register LO
    uint32_t BAR;  //bus address register
    uint32_t BCR;  //byte count register
} FddStruct;

//FDD: FddStruct is mapped to the memory position 0xFF000010
#define FDD  (*((FddStruct*)(0xFF000010)))

#define GO    (0x01) //Mask for GO
#define IE    (0x40) //Mask for IE
#define WRITE (0x02) //Mask for WRITE
//more defines...

typedef void (*ISR_t)(void); //Function pointer for an ISR
//INTVECTOR: ISR_t is mapped to the memory position 0x00000108
#define INTVECTOR (*((ISR_t*)(0x00000108)))

void ISR() { //interrupt service routine for the end of the transfer
    //notify application that everything is transferred
}

int main(void) {
    INTVECTOR = &ISR; //ISR address is set to INTVECTOR (address 0x00000108)

    //Configure DMA interface
    //In principle: {source, destination, how much, GO}
    FDD.BAR  = 0x400000;        //source memory address
    FDD.DARH = 0x0;             //destination LBA address (bit 32 to 47)
    FDD.DARL = 0x4711;          //destination LBA address (bit 0 to 31)
    FDD.BCR  = 0x40;            //how much: number of bytes
    FDD.CSR  = IE | WRITE | GO; //(IE) 0x40 + (WRITE) 0x02 + (GO) 0x01 = 0x43

    //DMA transmits data now without CPU.
    //At the end there is an interrupt!

    return EXIT_SUCCESS;
}
```

Exercise 13.6: DMA programming (coding)

Please use the Arduino MKR WiFi 1010 for this exercise, as the Arduino Mega 2560 does not support DMA. This exercise is based on the simple DMA example of Adafruit. Take the provided Arduino skeleton sketch to copy data using *direct memory access (DMA)* . *Hint: Check out Adafruit_ZeroDMA.h.*

(a) Open the provided
`CA_exercises/sheet_13_io/io_prog_dma_io/io_prog_dma_io.ino`
skeleton file and fill in the TODOs.

Proposal for solution:

```
// This exercise is based on the simple ZeroDMA example by Adafruit
// Because we use DMA, unlike memcpy(), our code could do other
// things simultaneously while the copy operation runs.

```

```
// You may need to checkout the header to fill in the Todos
// https://github.com/adafruit/Adafruit_ZeroDMA/blob/master/Adafruit_ZeroDMA.h

#include <Adafruit_ZeroDMA.h>  // lib Version 1.04
#include "utility/dma.h"

Adafruit_ZeroDMA myDMA;
ZeroDMAstatus    stat; // DMA status codes returned by some functions

// The amount of memory we'll be moving:
#define DATA_LENGTH 1024

//Arrays for source and destination
uint8_t source_memory[DATA_LENGTH];
uint8_t destination_memory[DATA_LENGTH];

volatile bool is_transfer_done = false;

//TODO: write a callback for the end-of-DMA-transfer and set
//      the is_transfer_done value to true
void dma_callback(Adafruit_ZeroDMA *dma) {
  is_transfer_done = true;
}

void setup() {
  uint32_t t;

  //Establish a connection to the serial monitor
  Serial.begin(9600);
  while (!Serial); //wait until a serial port is connected

  Serial.println("DMA test: memory copy");
  Serial.print("Allocating DMA channel...");

  stat = myDMA.allocate(); //TODO: allocate a DMA channel
  myDMA.printStatus(stat);

  Serial.println("Setting up transfer");
  //TODO: set src, dest, count
  myDMA.addDescriptor(source_memory, destination_memory, DATA_LENGTH);

  //TODO: set your callback, use the default type
  myDMA.setCallback(dma_callback);

  // Fill the source buffer with incrementing bytes, dest buf with 0's
  for (uint32_t i = 0; i < DATA_LENGTH; i++) {
    source_memory[i] = i;
  }
  // Todo clear destination memory in one line
  memset(destination_memory, 0, DATA_LENGTH);

  // Show the destination buffer is empty before transfer
  Serial.println("Destination buffer before transfer:");
  dump();

  Serial.println("Starting transfer job");
  //TODO: start the DMA transfer job
  stat = myDMA.startJob();
  myDMA.printStatus(stat);
```

```

  Serial.println("Triggering DMA transfer...");
  t = micros();
  // digitalWrite(LED_BUILTIN, HIGH);
  myDMA.trigger();

  // Your code could do other things here while copy happens!
  int32_t x = 0;
  while (!is_transfer_done) {
    ++x; // Chill until DMA transfer completes
  }

  // digitalWrite(LED_BUILTIN, LOW);
  t = micros() - t; // Elapsed time

  Serial.print("Done! ");
  Serial.print(t);
  Serial.println(" microseconds");

  Serial.print("Did ");
  Serial.print(x);
  Serial.println(" loops while waiting until DMA has completed");

  myDMA.free();

  Serial.println("Destination buffer after transfer:");
  dump();

  copy_data_manually();
}

// Show contents of destination_memory[] array
void dump() {
  for (uint32_t i = 0; i < DATA_LENGTH; i++) {
    Serial.print(destination_memory[i], HEX); Serial.print(' ');
    if ((i & 15) == 15) Serial.println();
  }
}


// Repeat the same operation "manually" without DMA
void copy_data_manually() {
  uint32_t t = micros();

  //TODO: copy the memory "manually" without DMA in one line
  memcpy(destination_memory, source_memory,  DATA_LENGTH);

  t = micros() - t; // Elapsed time

  Serial.print("Same operation without DMA: ");
  Serial.print(t);
  Serial.println(" microseconds");
}


void loop() { }
```

(b) Flash your sketch on the provided *Arduino MKR Wifi 1010* and open a serial monitor. Please make sure, that you set the right baud rate according to your sketch.