Rosenheim
Technical University
of Applied Sciences

Prof. Dr. Florian Künzner
Technical University of Applied Sciences Rosenheim, Computer Science

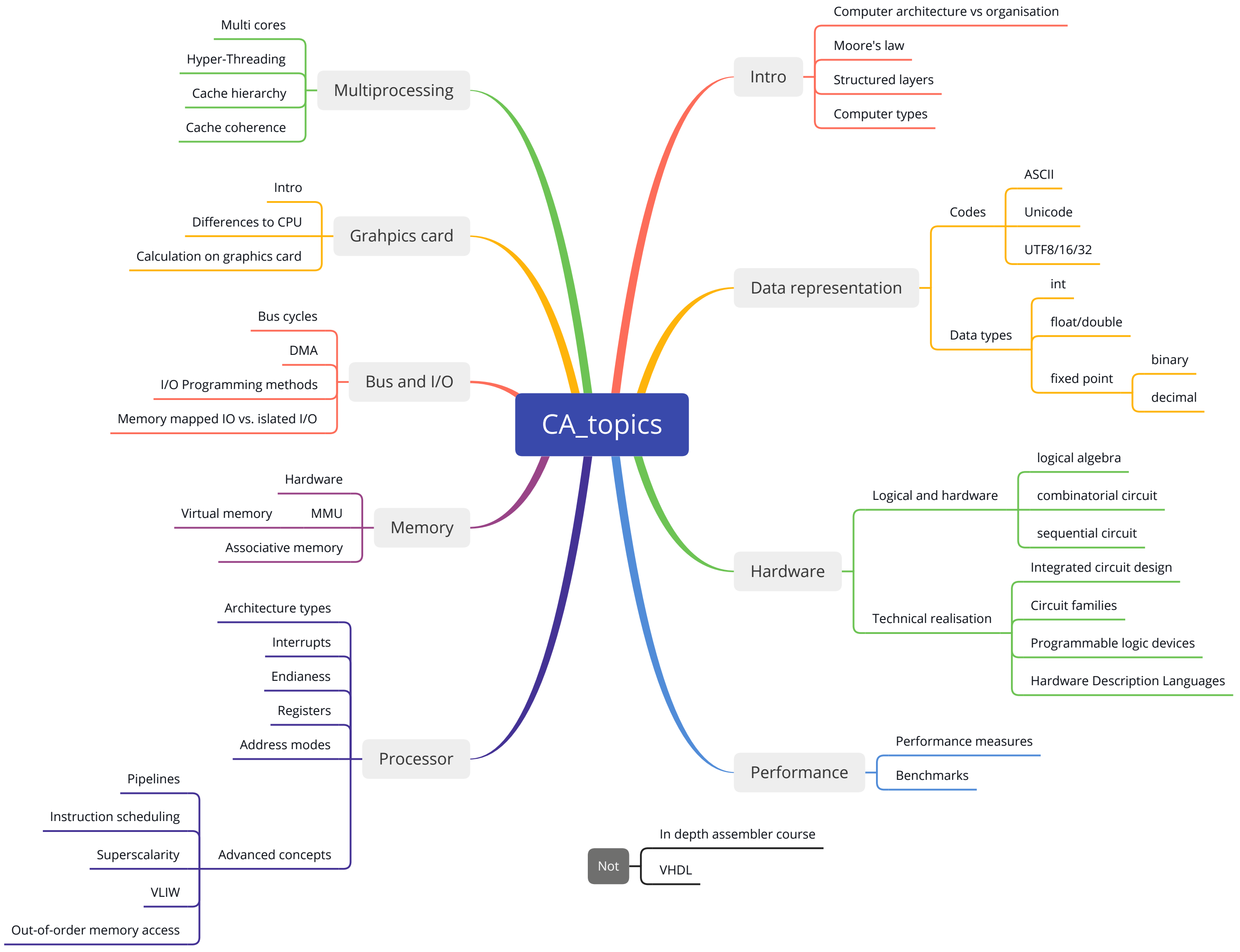
CA 13 – Bus and I/O 2

The lecture is based on the work and the documents of Prof. Dr. Theodor Tempelmeier

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Goal

CA::Bus and I/O

- I/O programming methods
- Programmed I/O
- Interrupts and interrupt driven I/O
- DMA
- FSI programming example
- FDD (DMA) programming example

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

I/O programming methods

Methods:

- Programmed I/O
  - Busy wait
  - Polling
- Interrupt driven I/O
- Direct memory access (DMA)

Difference:

- Who initiates the data transfer
- Who performs the actual data transfer
- How is the completion of the data transfer detected
- How much data is transferred with one instruction

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Programmed I/O (or single transfer)

Idea:

The **processor waits** until a device is ready to transfer data and performs the I/O operation.

Procedure:

- The **processor waits** (busy waiting or polling) until the data can be transferred
- The processor **performs** the **I/O** data **transfer**
- After the transfer, the processor **proceeds** with another instruction

This is the simplest method to transfer I/O data between the processor and a device.

But:

The processor has to do all the work and it is therefore **slow**!

We will see a programming example in the FSI chapter.

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

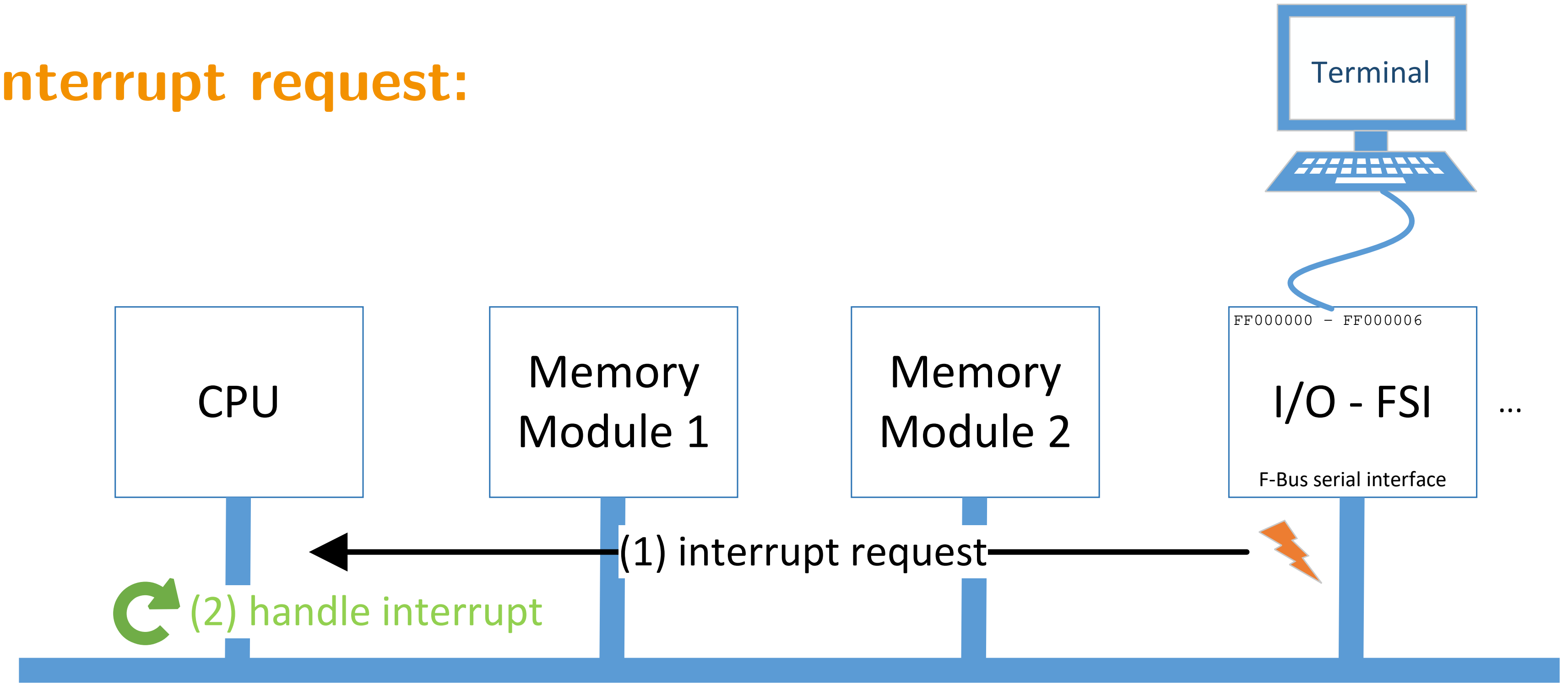
Interrupts

How can a device on the bus **draw attention to itself** (e.g. data available, ready, ...)?

CAMPUS Rosenheim
Computer Science

F-Bus: Interrupt cycle

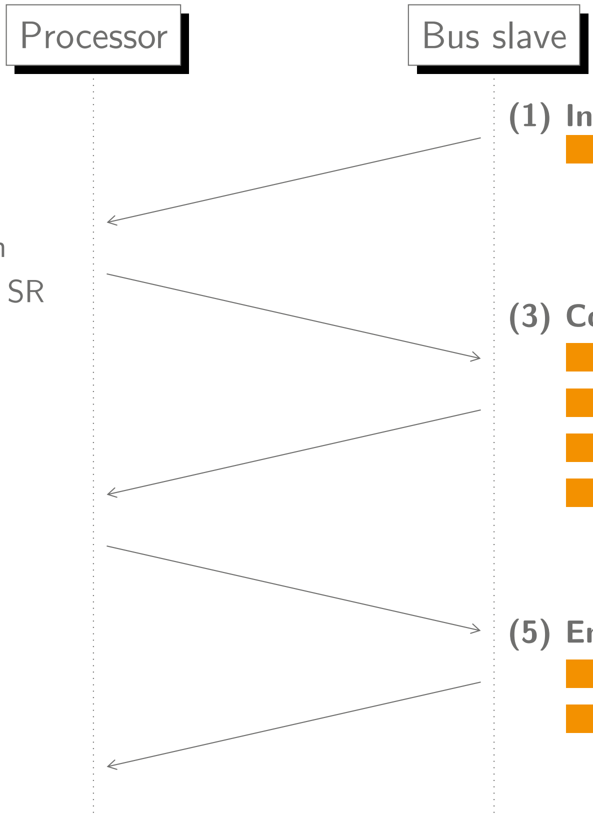
Interrupt request:



CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Bus cycle protocol: interrupt



(1) Interrupt request:
- Set IREQ <6:0>

(2) Accept interrupt:
- At the end of the current instruction
- If IREQ-Nr. > Int. priority mask in SR
- Set IACK signal

(3) Confirmation of IACK:
- Read IACK signal
- Put IRQ vector addr. on ADL <31:0>
- Set REPLY signal
- Reset IREQ <6:0>

(4) End of interrupt request:
- Read IRQ vector addr. from ADL <31:0>
- Reset IACK signal

(5) End of interrupt request:
- Reset IRQ vector addr. on ADL <31:0>
- Reset REPLY signal

(6) Interrupt handling:
- Put PC and SR on stack
- Set supervisor bit in SR
- Set interrupt priority in SR
- Load new PC from IRQ vector addr.

There are 7 interrupt request lines <6:0>, corresponding to 7 priority levels. Each interrupt has its own interrupt vector.

CAMPUS Rosenheim
Computer Science

Interrupt driven I/O programming

Idea:

The **processor should not wait** until a device is ready for transferring data.

Procedure:

- The processor **configures** the device for **interrupt driven I/O**
- The processor **immediately proceeds** with another instruction
- After the **I/O** data is **available** in the device or the device is ready to receive new data, an **interrupt** is generated
- The processor **performs** the **I/O** data **transfer**

But:

One obvious drawback is that **one interrupt for each word** is necessary and the **processor performs** the **I/O data transfer**.

We will see a programming example in the FSI chapter.

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FSI

The F-Bus serial interface

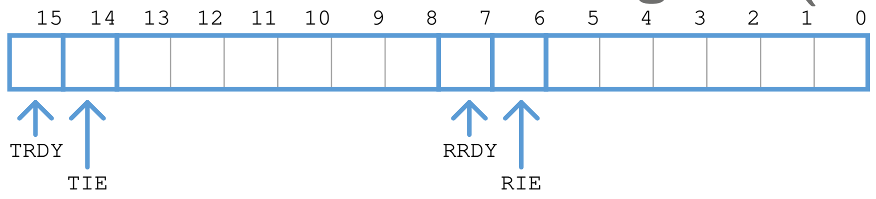
Disclaimer:

- For simplification without multiplexing of the N interfaces
- For simplification without any error handling
- For simplification without modem signals
- For simplification without "echo"

CAMPUS Rosenheim
Computer Science

FSI registers (1)

FSI Control and status register (CSR)



TRDY – Transmitter ready (CPU to device)
Set by the HW when a character has been sent completely.
Is cleared by the hardware when writing TBUF register.

TIE – Transmit interrupt enable
If TIE is activated, an interrupt is generated if TRDY is set to 1.

RRDY – Receiver ready (device to CPU)
Set by the HW when a new character has been received completely.
Deleted from the HW when reading RBUF register.

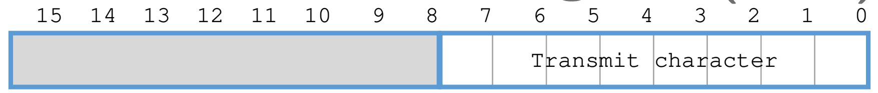
RIE – Receiver interrupt enable
If RIE is activated, an interrupt is generated if RRDY is set to 1.

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

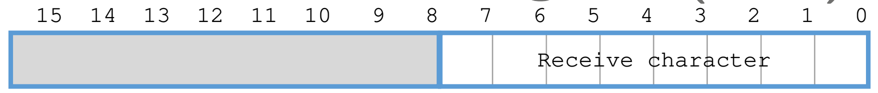
FSI registers (2)

FSI Transmit buffer register (TBUF)



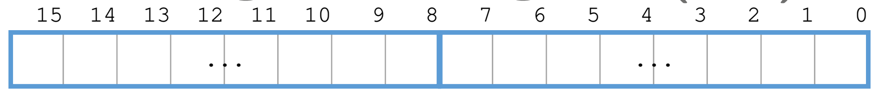
The transmit character is sent from the CPU to the serial interface.

FSI Receive buffer register (RBUF)



The receive character is sent from the serial interface to the CPU.

FSI Configuration register (CFR)



Configuration of the serial interface: stop bits, data bits, parity, baud rate, ...

FSI programming example – common

FSI programming example – common definitions

```c
#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...
```

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FSI programming example (1)

Output of a character with busy waiting: Programmed I/O with busy wait

```
int main(void) {
    char char_to_transmit = 'A';

    while ((bool)(FSI.CSR & TRDY) == false) //wait until FSI is ready to
    {                                       //transmit data (CPU to device)
        //busy wait (do nothing)
    }
    FSI.TBUF = char_to_transmit;

    return 0;
}
```

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FSI programming example (2)

Output of a character with polling: Programmed I/O with polling

```c
int main(void) {
    char char_to_transmit = 'A';

    while(true) { //polling
        if((bool)(FSI.CSR & TRDY) == true){
            FSI.TBUF = char_to_transmit;
            //repeat with next character or break
        } else {
            //do something else...
        }
    }

    return EXIT_SUCCESS;
}
```

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

FSI programming example (3)

Input of a character with interrupt driven I/O: Interrupt driven I/O

```
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)))

volatile char received_char = ' ';
void ISR() { //interrupt service routine for received characters from FSI
    received_char = FSI.RBUF;
}

int main(void) {
    INTVECTOR = &ISR; //ISR address is set to INTVECTOR (address 0x000000C8)
    FSI.CSR = FSI.CSR | RIE; //enable the receiver interrupt (RIE)

    //The input is now done "automatically" via the ISR when the
    //next character has been received completely.

    //do something else...

    return EXIT_SUCCESS;
}
```

Bus arbitration and prioritisation

The arbiter (Schiedsrichter)

Interrupt request

- If two ore more bus devices generates interrupts
- The arbiter prioritises and decides which interrupt request is handled first

Bus master request

- If two ore more bus devices wants to become bus master
- The arbiter prioritises and decides which bus device can become bus master first

Memory request, ...

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

DMA

Does the processor always have to be involved in data transfer on the bus?

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

DMA - direct memory access

Idea:
In addition to the CPU, other *smart* bus devices can also be allowed to **become temporarily bus master**.

Condition:

- The **bus arbiter (CPU)** has to be asked whether it is possible that someone other can become bus master
- **Only if the bus arbiter allows** it, then a DMA data transfer can happen

Cycle stealing:

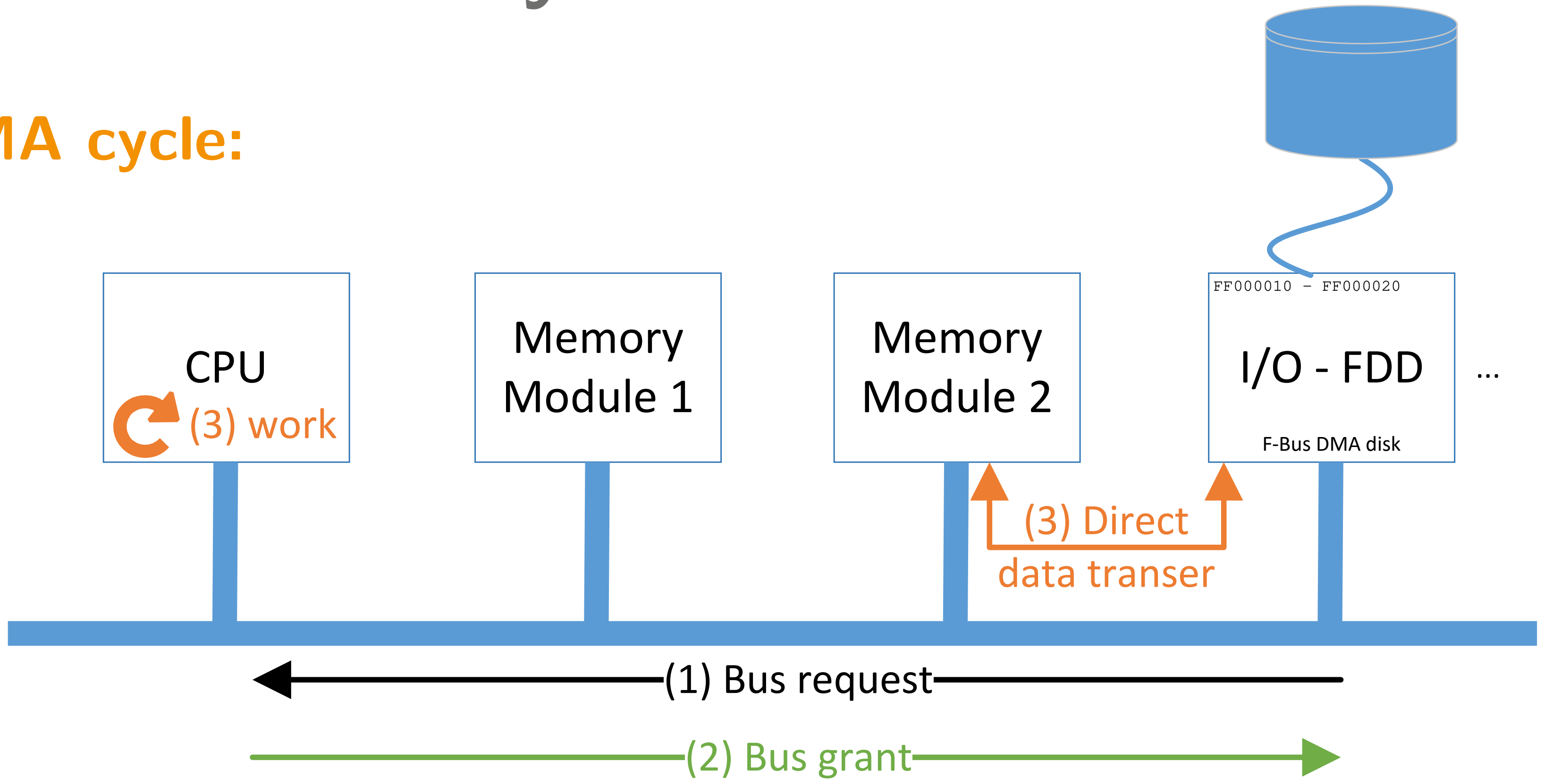
- If someone other than the CPU is bus master
- Then the DMA interface **steals the CPU *possible* bus cycles**

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

F-Bus: DMA cycle

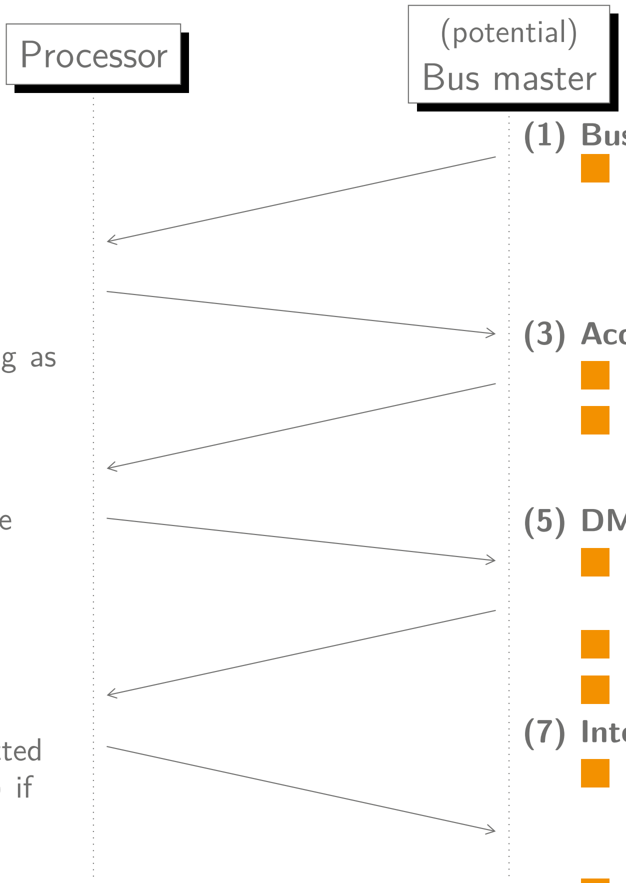
DMA cycle:



CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Bus cycle protocol: DMA



(1) Bus request:
- Set BREQ <7:0>

(2) Bus grant:
- Wait until the end of the currently running bus cycle
- Set BGNT <7:0>
- Do not create new bus cycles as long as the DMA cycle is running

(3) Accept bus grant:
- Read BGNT <7:0>
- Wait until FRAME signal is reset

(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles

(5) DMA transfer:
- Execute normal bus cycles: read, write, or burst
- If burst: repeat cycle b_{max} times
- At the end: Reset BREQ <7:0>

(6) Continue with normal bus cycles:
- Reset BGNT <7:0>
- Creation of new bus cycles is permitted again, also new bus request (BREQ) if required

(7) Interrupt handling:
- Possibly waiting time, before the same bus device creates a new bus request (BREQ)
- If everything is transferred: generate an interrupt

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

DMA programming

Idea:
The **I/O data transfer** is performed directly from the **devices** (without the **processor**).

Procedure:

- The processor **initiate** the DMA device interface with `{Source, Destination, How much, IE | R/W | GO!}`
- The processor **immediately proceeds** with another instruction
- The **DMA** devices **does all the work**
- After the whole DMA data transfer has **finished**, an **interrupt** is generated

Pro:
Only one interrupt after the whole DMA data (block) transfer has been finished.

We will see a programming example in the FDD chapter.

DMA: Command chaining

Multiple DMA data transfers required?

Command chaining:

- `{Source, Destination, How much, IE | R/W | GO!}`$_1$
- `{Source, Destination, How much, IE | R/W | GO!}`$_2$
- `{Source, Destination, How much, IE | R/W | GO!}`$_3$
- ...

The DMA interface automatically executes the entire sequence of the individual DMA data (block) transfers.

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

FDD

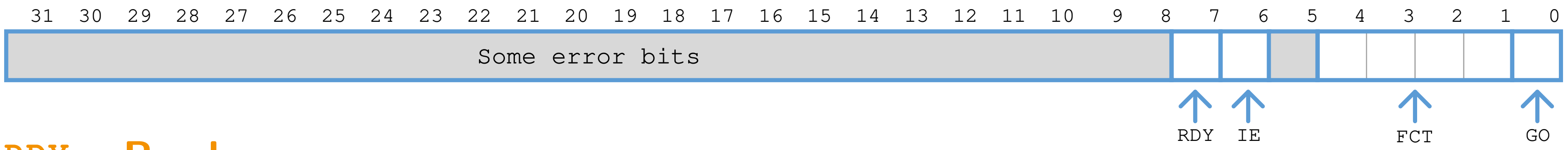
The F-Bus DMA disk

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FDD registers (1)

FDD Control and status register (CSR)



RDY – Ready

Set by the HW when the operation has been finished.

IE – Interrupt enable

If IE is activated, an interrupt is generated if RDY is set to 1.

FCT – Function

Decimal value	Hex value	Bin value	Function
0	0x0	0000	Reset
1	0x1	0001	Write
2	0x2	0010	Read
...			
4	0x4	0100	Seek (positioning)
...			

GO

If set to 1, then the DMA operation starts.

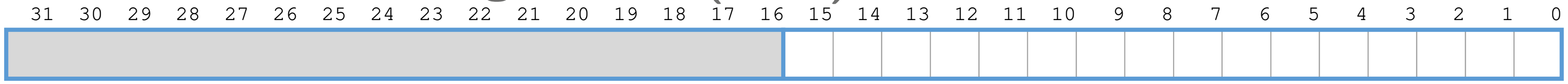
CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FDD registers (2)

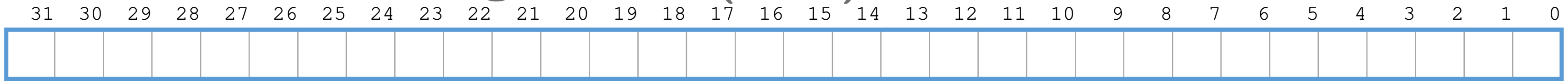
Disk address: The controller uses an **LBA** (logic block address) that is a linear address of blocks, instead of cylinder/head/sector. The LBA has **48 bits** (32 are not enough).

FDD Disk address register HI (`DARH`)



Bits 32 to 47 of the LBA number.

FDD Disk address register LO (`DARL`)



Bits 0 to 31 of the LBA number.

CAMPUS Rosenheim
Computer Science

FDD registers (3)

FDD Bus address register (BAR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Main memory address for transfer.

FDD Byte count register (BCR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Number of bytes to be transmitted.

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FDD programming example

Transmit 0x100 bytes from memory to disk.

Addresses:

- Start address: 0x10000 (memory)
- LBA address: 0x1234 (disk)

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

FDD programming example – common

FDD programming example – common definitions

```
#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...
```

FDD programming example (1)

FDD programming example – DMA

```c
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, IE | R/W | GO}
    FDD.BAR  = 0x10000;         //source memory address
    FDD.DARH = 0x0;             //destination LBA address (bit 32 to 47)
    FDD.DARL = 0x1234;          //destination LBA address (bit 0 to 31)
    FDD.BCR  = 0x100;           //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;
}
```

CAMPUS Rosenheim
Computer Science

Rosenheim
Technical University
of Applied Sciences

FDD programming example (2)

FDD programming example – inside the OS

```
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_fdd_dma_transmitted(); //prototype

//-----------------------------------------------------------------
//Part of the glibc: system call interface (SVC) (very high level)

//Reads an array of count elements, each one with a size of size bytes,
//from the stream and stores them in the block of memory specified by ptr.
size_t fread(void* ptr, size_t size, size_t count, FILE* stream) {
    INTVECTOR = &ISR_fdd_dma_transmitted; //ISR address is set to INTVECTOR
    //Set registers of the FDD just like in the previous example.
    P(transmit_ready);

    return /*total amount of bytes read*/;
}

//-----------------------------------------------------------------
//Inside the OS kernel
void ISR_fdd_dma_transmitted(){
    V(transmit_ready);
}
```

CAMPUS Rosenheim
Computer Science

Rosenheim Technical University of Applied Sciences

Summary and outlook

Summary

- I/O programming methods
- Programmed I/O
- Interrupts and interrupt driven I/O
- DMA
- FSI programming example
- FDD (DMA) programming example

Outlook

- Multiprocessing