Technische Hochschule Rosenheim
Technical University of Applied Sciences

Start: 8:01

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

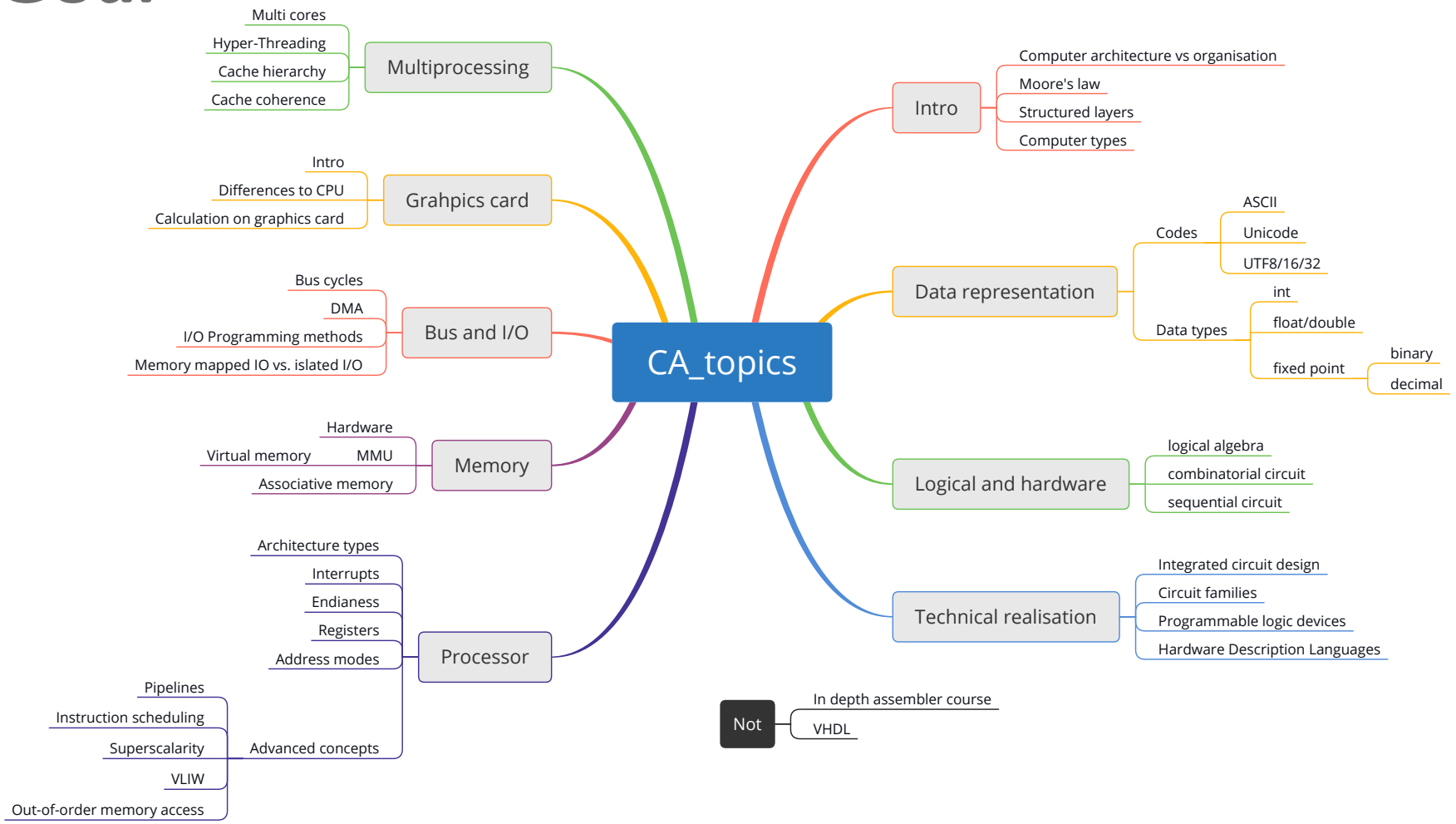
CA 12 – Bus and I/O 2

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Goal



CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Technische Hochschule 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

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

The CPU 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

Technische Hochschule **Rosenheim**
Technical University of Applied Sciences

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

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

We will see a programming example in the FSI chapter.

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

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

We will see a programming example in the FSI chapter.

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

The CPU 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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

The CPU 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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

The CPU 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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Programmed I/O (or single transfer)

Idea:

The processor waits until an I/O operation has finished.

Procedure:

- The processor **initiate** the **I/O** data **transfer**
- The **processor waits** (busy waiting or polling) until the data has been transferred
- The processor **proceeds** with another instruction

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

But:

The CPU 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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupts

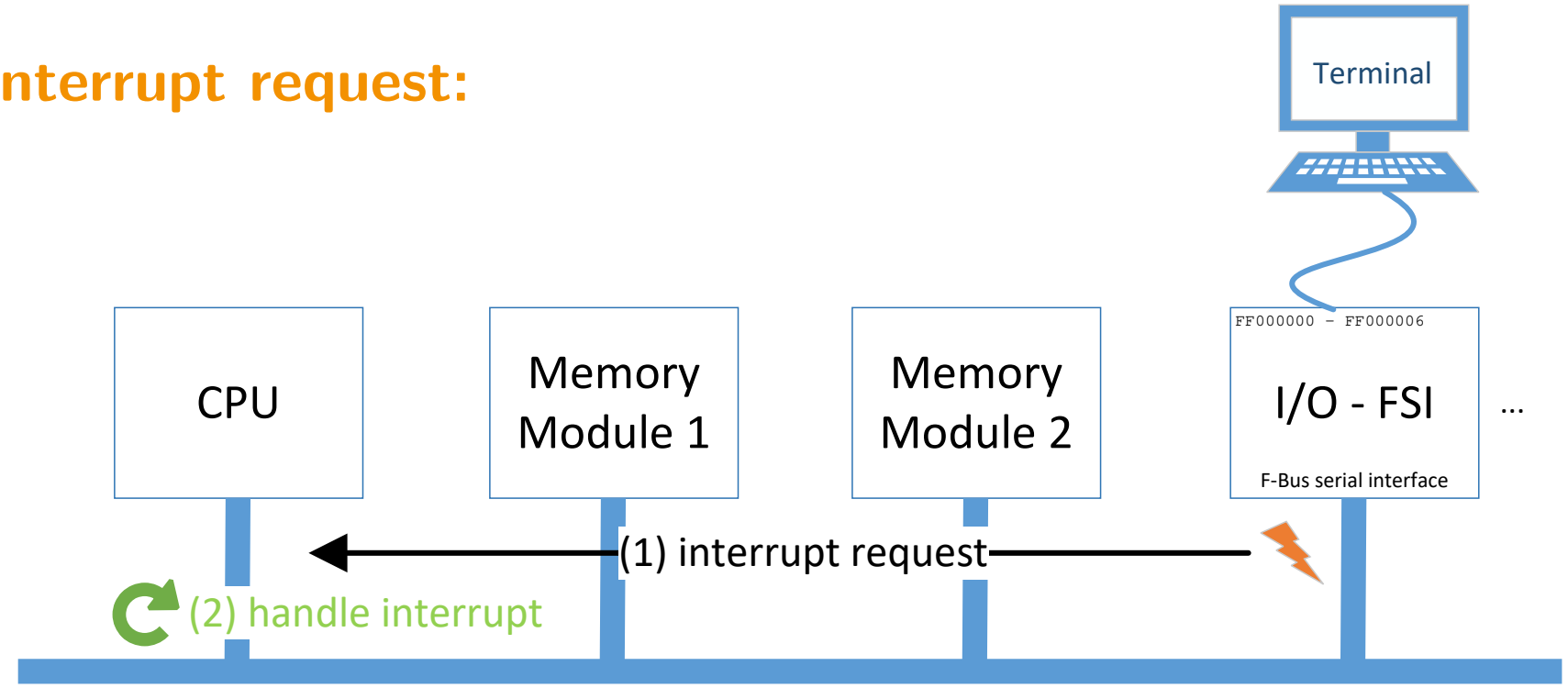
How can a device on the bus **draw attention to itself** (e.g. data available, ready, ...)?

CAMPUS Rosenheim
Computer Science

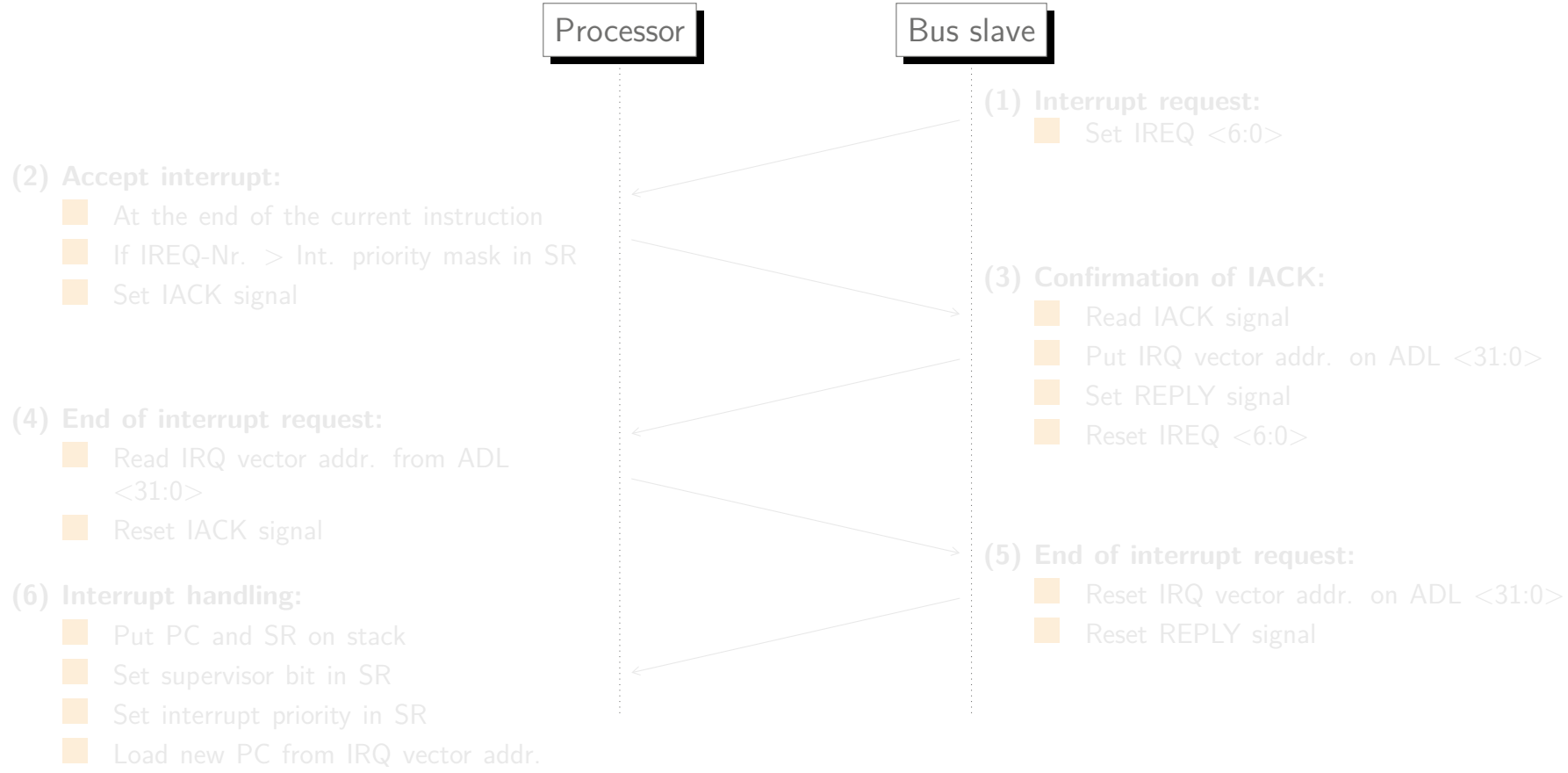
Technische Hochschule Rosenheim
Technical University of Applied Sciences

F-Bus: Interrupt cycle

Interrupt request:

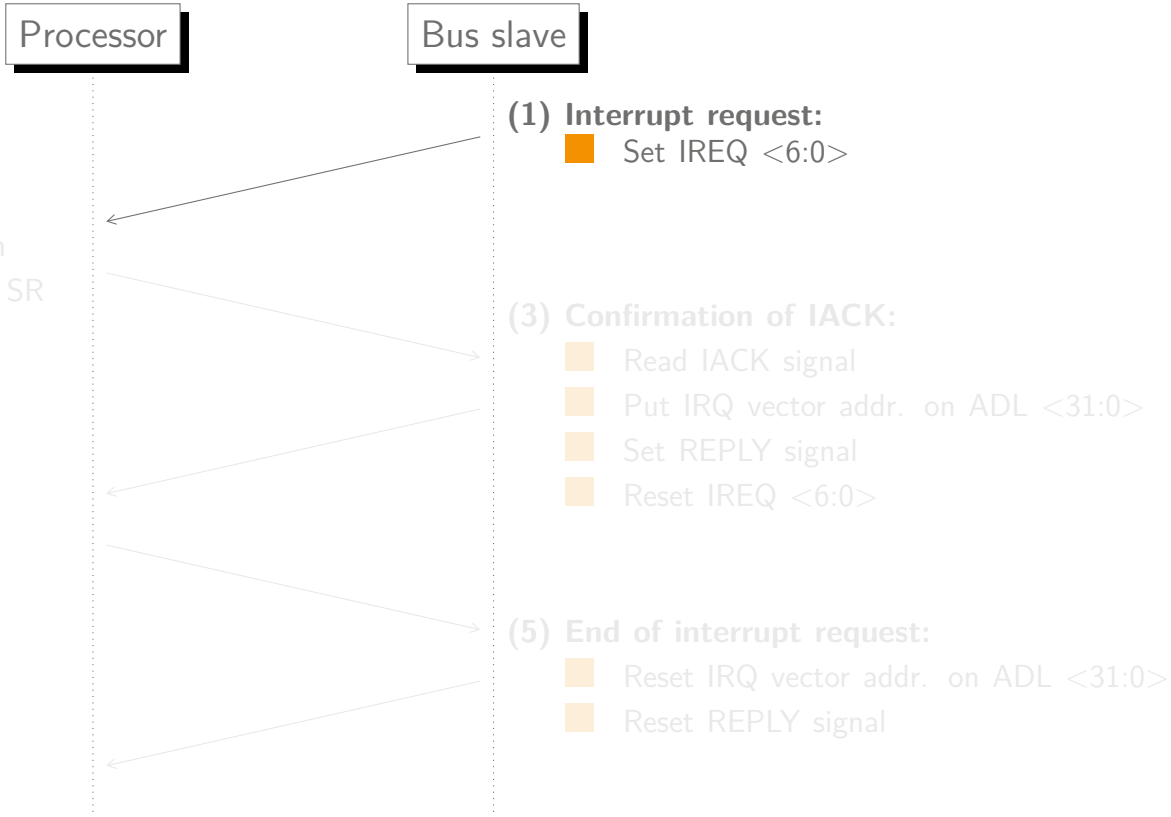


Bus cycle protocol: interrupt



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

Bus cycle protocol: interrupt

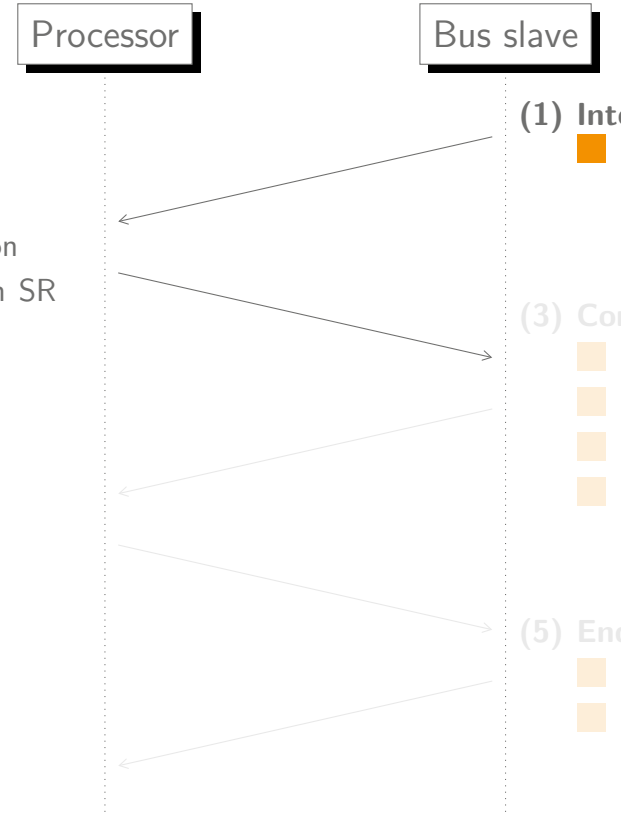


There are 7 interrupt request lines <6:0>, corresponding to 7 priority levels. Each interrupt has its own interrupt vector.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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>

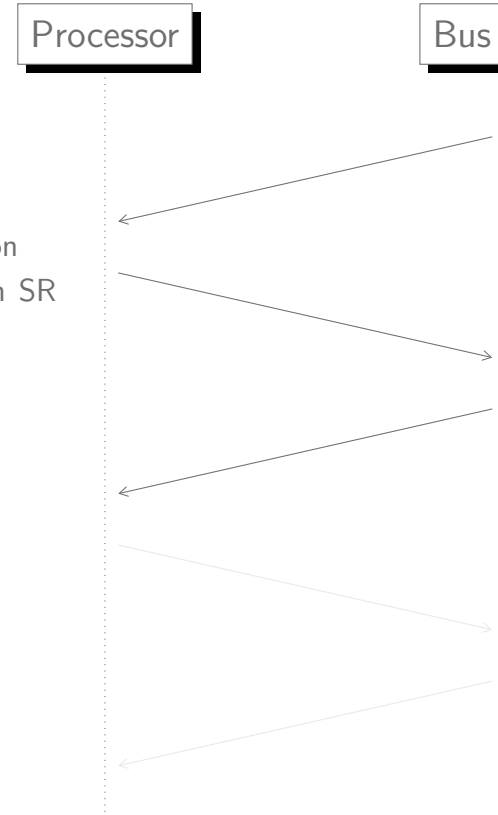
(4) End of interrupt request:
- Read IRQ vector addr. from ADL <31:0>
- Reset IACK signal

(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.

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

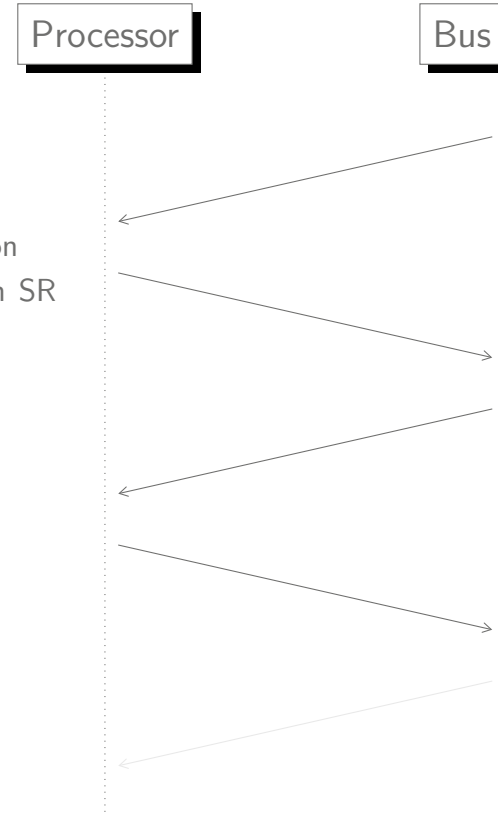
(6) Interrupt handling:
- Put PC and SR on stack
- Set supervisor bit in SR
- Set interrupt priority in SR
- Load new PC from IRQ vector addr.

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

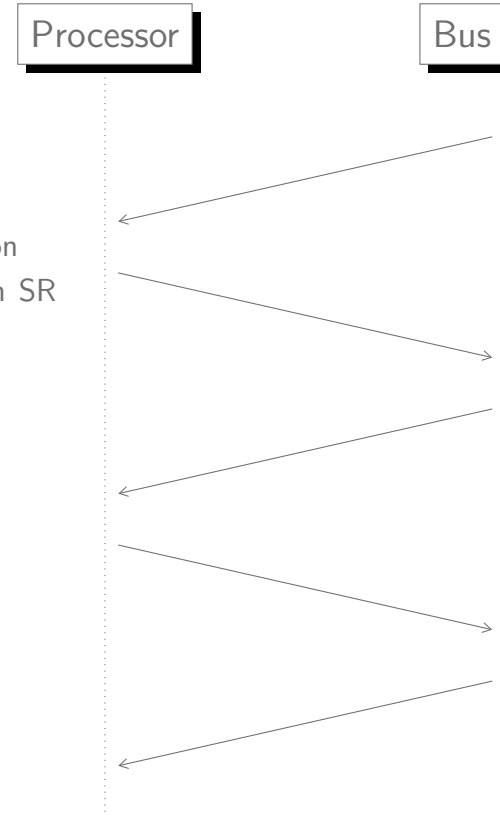
(6) Interrupt handling:
- Put PC and SR on stack
- Set supervisor bit in SR
- Set interrupt priority in SR
- Load new PC from IRQ vector addr.

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

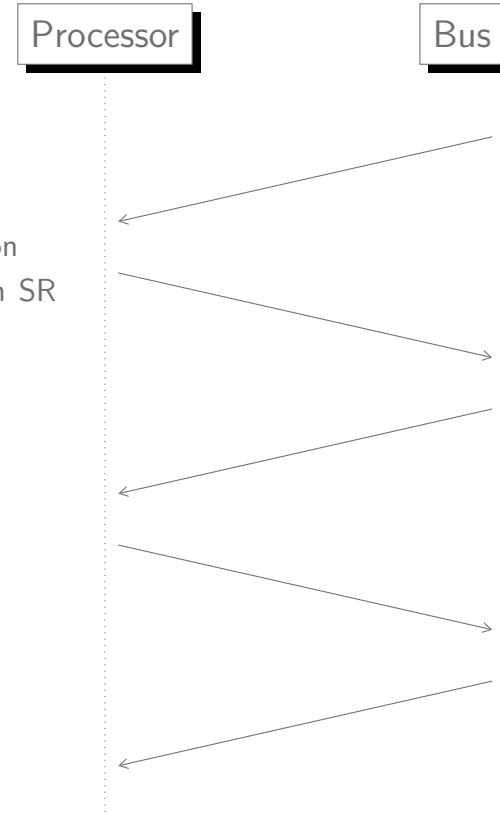
(6) Interrupt handling:
- Put PC and SR on stack
- Set supervisor bit in SR
- Set interrupt priority in SR
- Load new PC from IRQ vector addr.

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

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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.

Interrupt driven I/O programming

Idea:

The processor should not wait until an I/O operation has finished.

Procedure:

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

But:

One obvious drawback is that **one interrupt for each word** is necessary.

We will see a programming example in the FSI chapter.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Interrupt driven I/O programming

Idea:

The processor should not wait until an I/O operation has finished.

Procedure:

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

But:

One obvious drawback is that **one interrupt for each word** is necessary.

We will see a programming example in the FSI chapter.

Interrupt driven I/O programming

Idea:

The processor should not wait until an I/O operation has finished.

Procedure:

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

But:

One obvious drawback is that **one interrupt for each word** is necessary.

We will see a programming example in the FSI chapter.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FSI

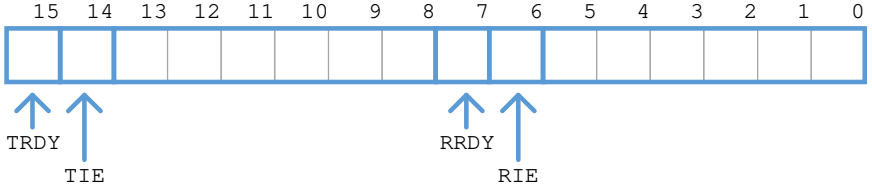
The F-Bus serial interface

Disclaimer:

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

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.

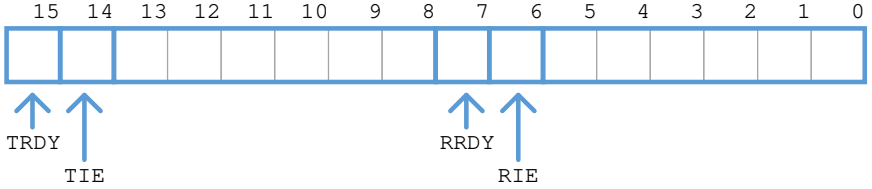
Deleted from the HW when reading RBUF register.

RIE – Receiver interrupt enable

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

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

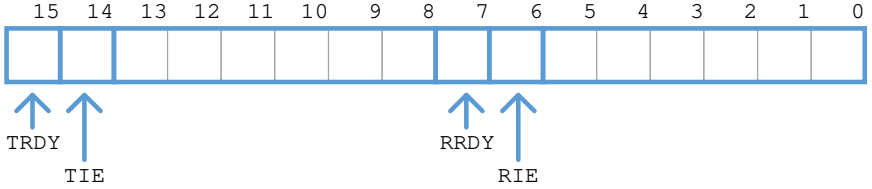
If RIE is activated, an interrupt is generated if RRDY is set to 1.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

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.

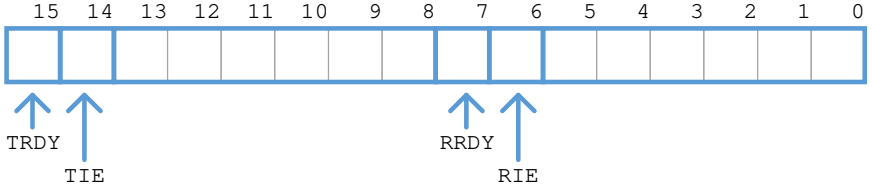
Deleted from the HW when reading RBUF register.

RIE – Receiver interrupt enable

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

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

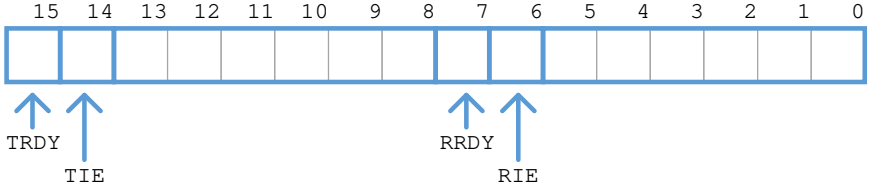
If RIE is activated, an interrupt is generated if RRDY is set to 1.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

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

Technische Hochschule 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.

CAMPUS Rosenheim
Computer Science

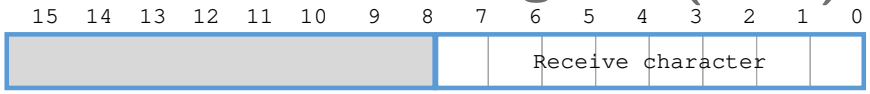
Technische Hochschule Rosenheim
Technical University of Applied Sciences

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)



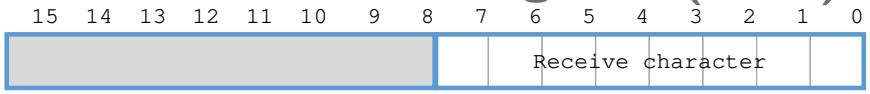
The receive character is sent from the serial interface to the CPU.

FSI registers (2)

FSI Transmit buffer register (TBUF)

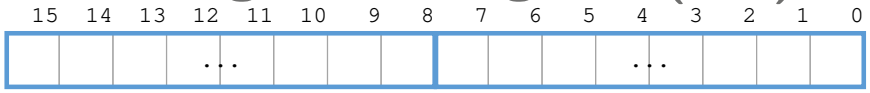
The transmit character is sent from the CPU to the serial interface.

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, ...

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FSI programming example – common

FSI programming example – common definitions

```
1  #include <stdlib.h>
2  #include <stdbool.h>
3  #include <inttypes.h>
4
5  typedef volatile struct { //FSI interface
6      uint16_t CSR;  //control and status register
7      uint16_t TBUF; //transmit buffer register
8      uint16_t RBUF; //receive buffer register
9      uint16_t CFR;  //configuration register
10 } FsiStruct;
11
12 //FSI: FsiStruct is mapped to the memory position 0xFF000000
13 #define FSI  (*((FsiStruct*)(0xFF000000)))
14
15 #define TRDY (0B1000000000000000) //Mask for TRDY (or: 0x8000)
16 #define TIE  (0B0100000000000000) //Mask for TIE  (or: 0x4000)
17 #define RRDY (0B0000000010000000) //Mask for RRDY (or: 0x0080)
18 #define RIE  (0B0000000001000000) //Mask for RIE  (or: 0x0040)
19 //more defines...
```

→ FSI.CSR = 1

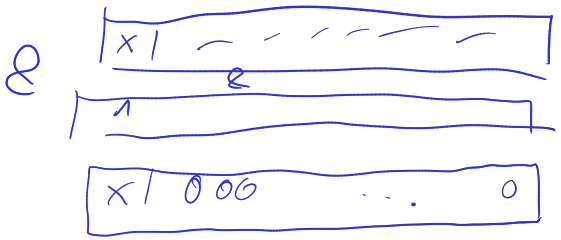
CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FSI programming example (1)

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

```
21 int main(void) {
22     char char_to_transmit = 'A';
23
24     while ((bool)(FSI.CSR & TRDY) == false) //wait until FSI is ready to
25     {                                       //transmit data (CPU to device)
26         //busy wait (do nothing)
27     }
28     FSI.TBUF = char_to_transmit;
29
30     return 0;
31 }
```



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FSI programming example (2)

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

```
21 int main(void) {
22     char char_to_transmit = 'A';
23
24     while(true) { //polling
25         if((bool)(FSI.CSR & TRDY) == true){
26             FSI.TBUF = char_to_transmit;
27             //repeat with next character or break
28         } else {
29             //do something else...
30         }
31     }
32
33     return EXIT_SUCCESS;
34 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FSI programming example (3)

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

```
21 typedef void (*ISR_t)(void); //Function pointer for an ISR
22 //INTVECTOR: ISR_t is mapped to the memory position 0x000000C8
23 #define INTVECTOR (*((ISR_t*)(0x000000C8)))
24
25 char volatile received_char = ' ';
26 void ISR() { //interrupt service routine for received characters from FSI
27     received_char = FSI.RBUF;
28 }
29
30 int main(void) {
31     INTVECTOR = &ISR; //ISR address is set to INTVECTOR (address 0x000000C8)
32     FSI.CSR = FSI.CSR | RIE; //enable the receiver interrupt (RIE)
33
34     //The input is now done "automatically" via the ISR when the
35     //next character has been received completely.
36
37     //do something else...
38
39     return EXIT_SUCCESS;
40 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

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, ...

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, ...

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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

DMA

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

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**

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

Technische Hochschule 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:

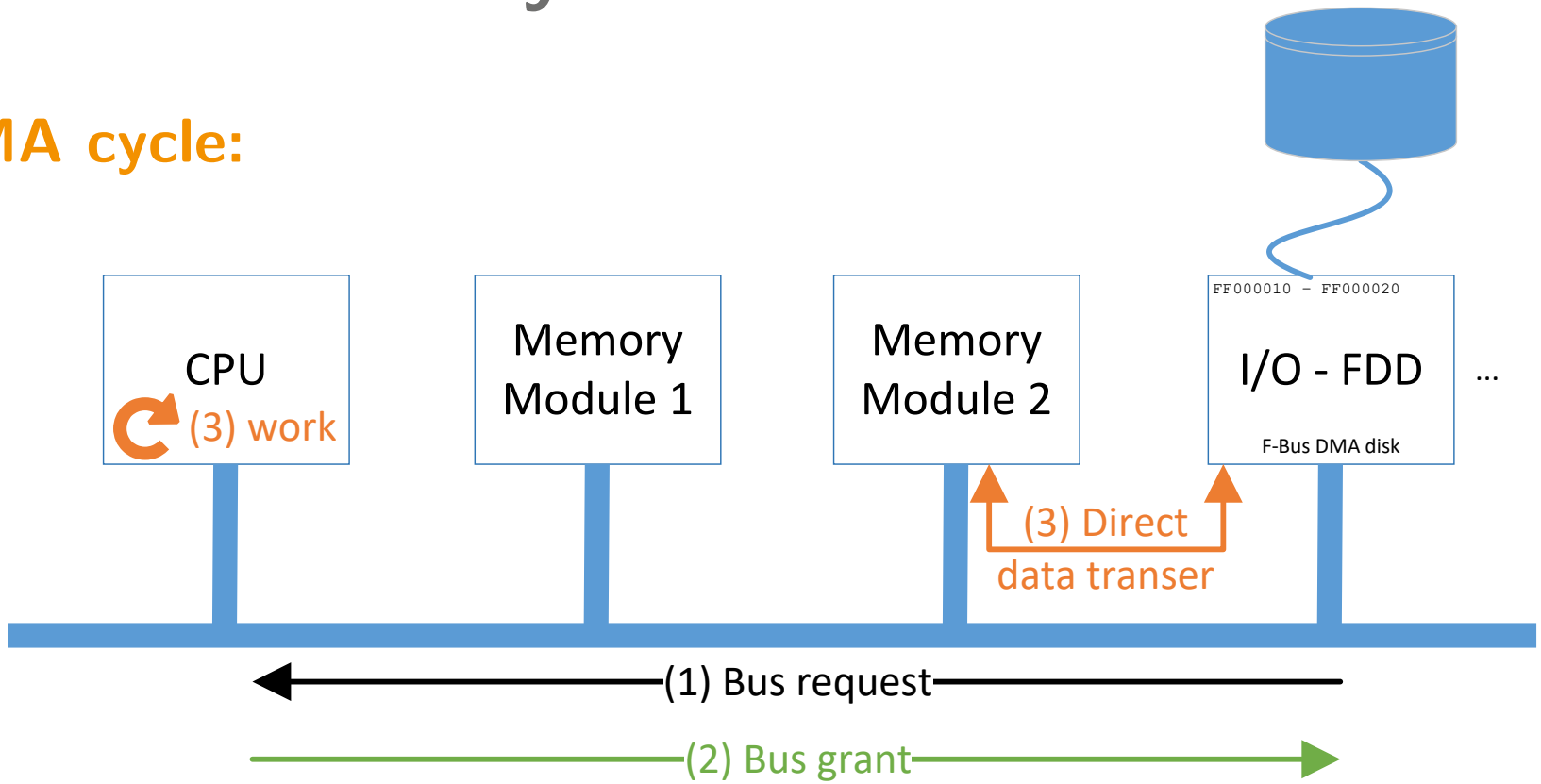
- If someone other than the CPU is bus master
- Then the DMA interface **steals the CPU *possible* bus cycles**

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

F-Bus: DMA cycle

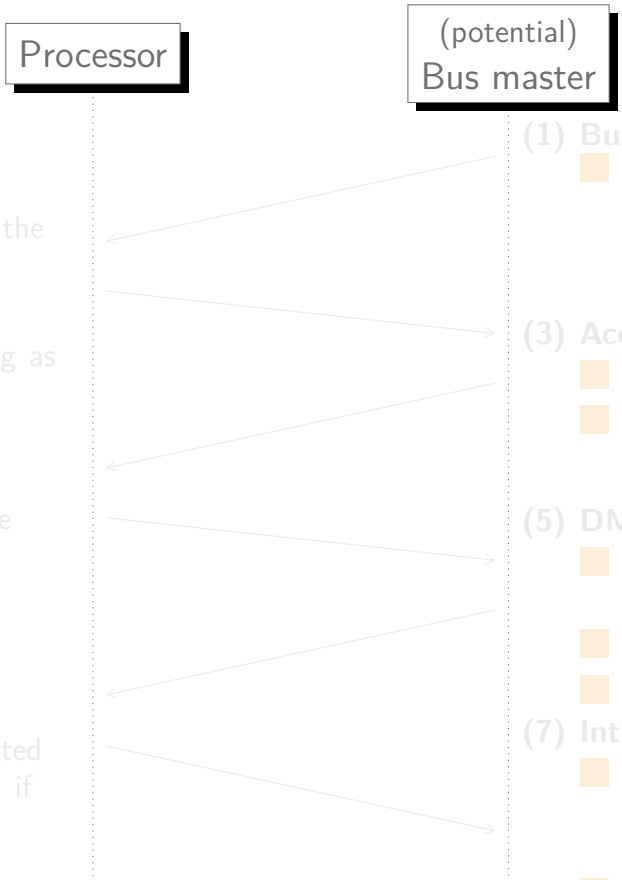
DMA cycle:



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

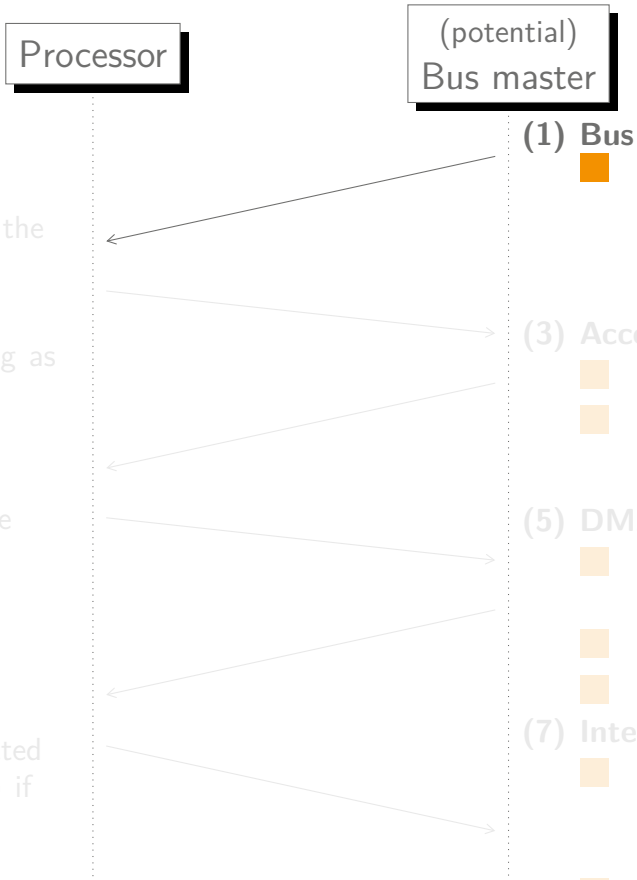
Bus cycle protocol: DMA



CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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

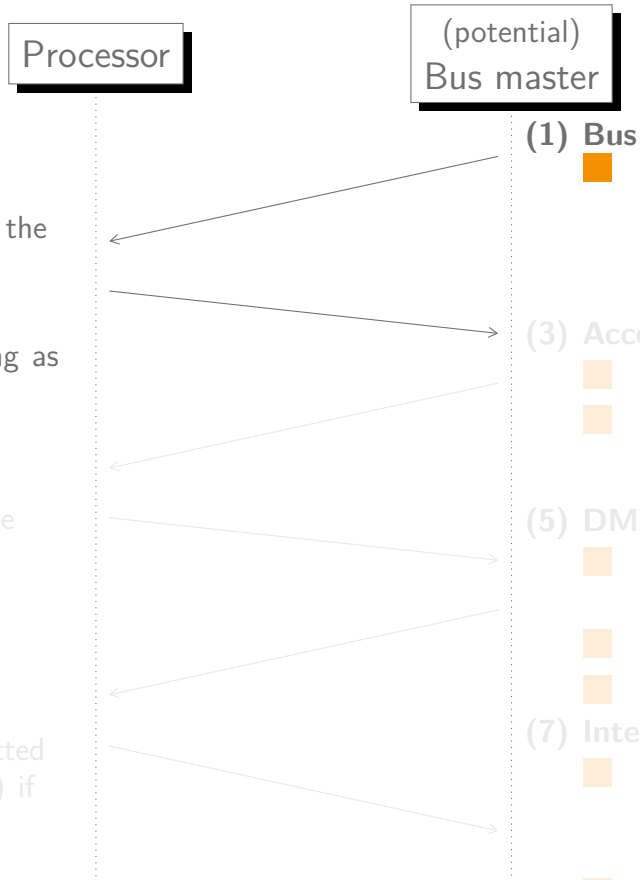
(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles.

(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.

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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

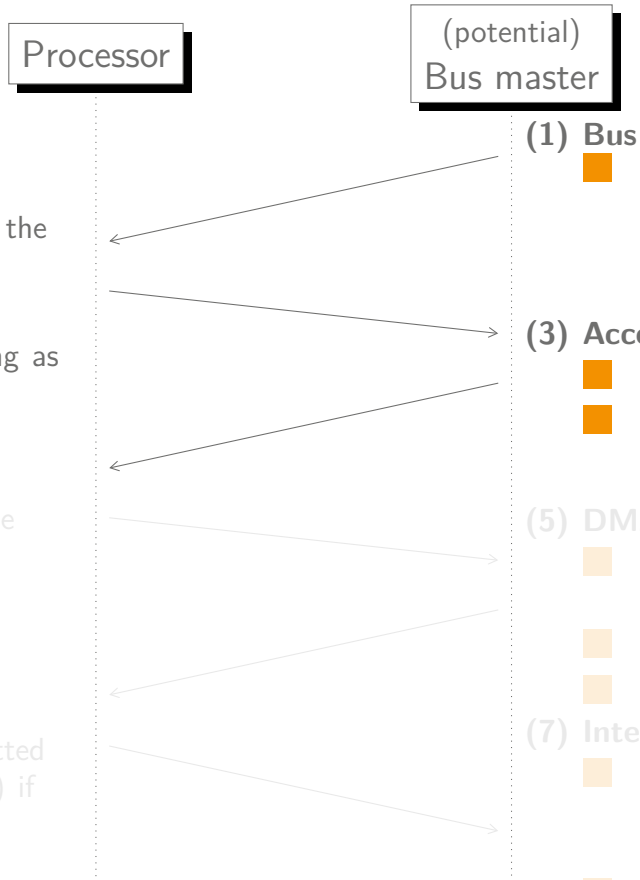
(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles.

(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.

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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>

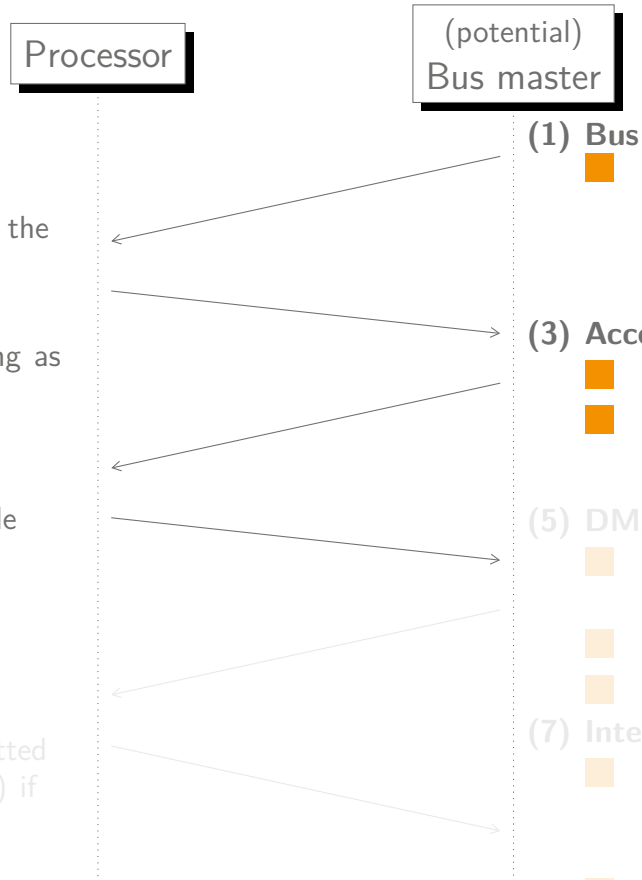
(6) Continue with normal bus cycles:
- Reset BGNT <7:0>
- Creation of new bus cycles is permitted again, also new bus request (BREQ) if required.

(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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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

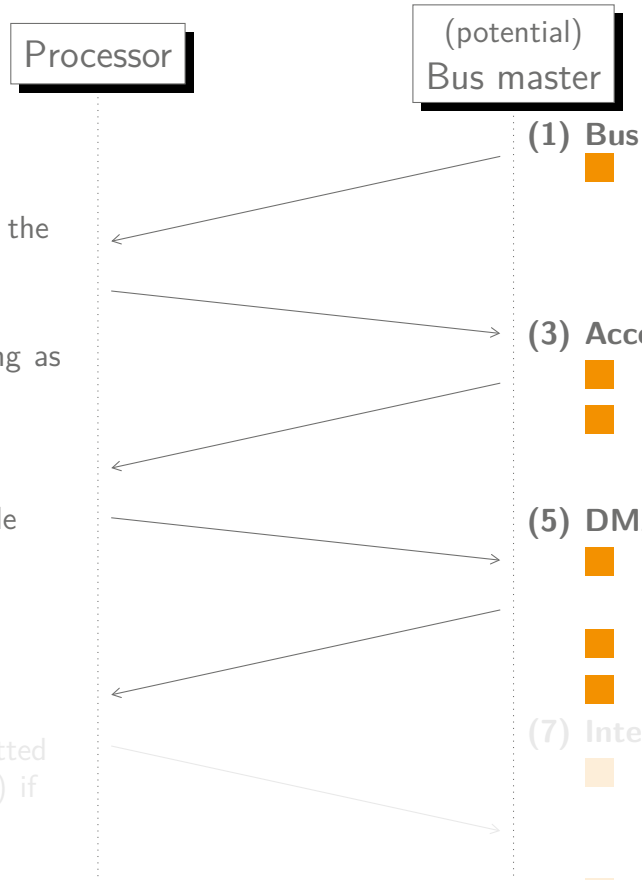
(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles.

(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.

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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

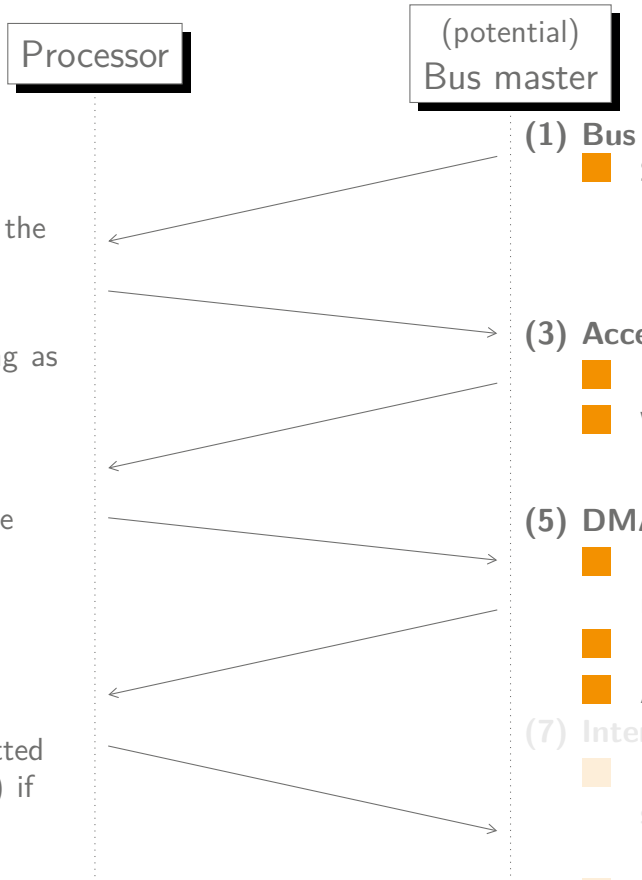
(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles.

(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.

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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

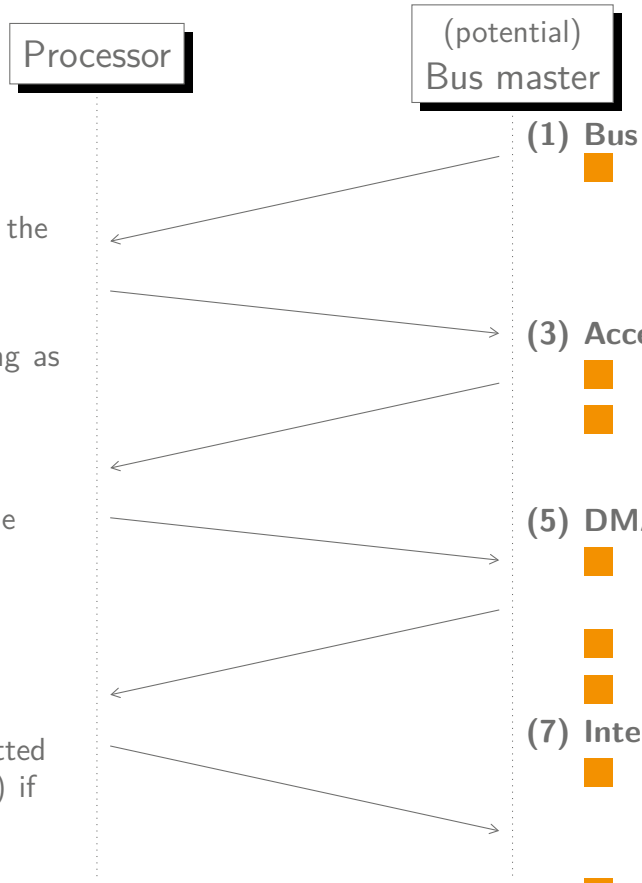
(4) End bus grant:
- Wait until the end of the DMA cycle before generating new bus cycles.

(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.

Bus cycle protocol: DMA



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

(2) Bus grant:
- Wait until shortly before 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.

DMA programming

Idea:

The **data transfer** runs **without** further actions by the **CPU**.

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 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.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

DMA programming

Idea:

The **data transfer** runs **without** further actions by the **CPU**.

Procedure:

- The processor **initiate** the DMA device interface with `{Source, Destination, How much, IE | R/W | GO!}` ← Prinzip
- The processor **immediately proceeds** with another instruction
- The **DMA** devices **does all the work**
- After the 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.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

DMA programming

Idea:

The **data transfer** runs **without** further actions by the **CPU**.

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 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.

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

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.

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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FDD

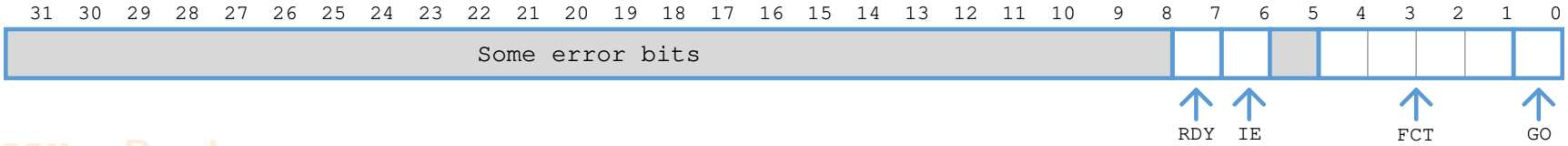
The F-Bus DMA disk

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

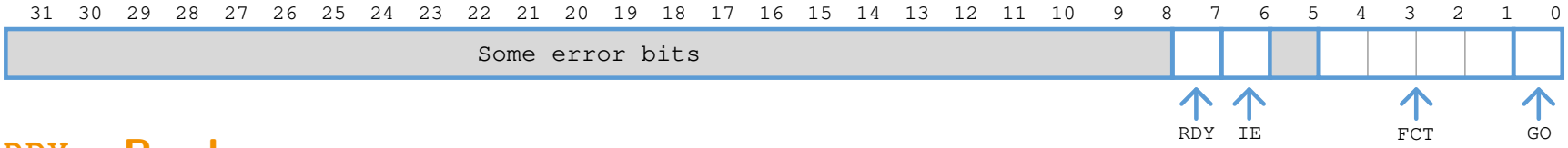
If set to 1, then the DMA operation starts.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

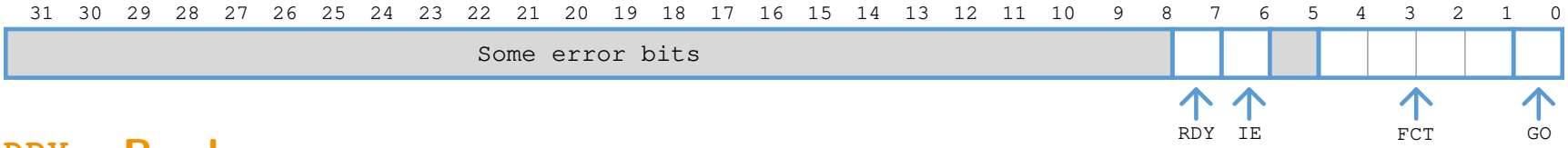
If set to 1, then the DMA operation starts.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

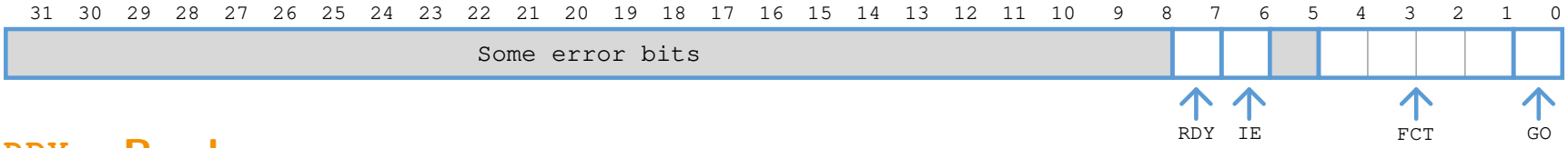
If set to 1, then the DMA operation starts.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

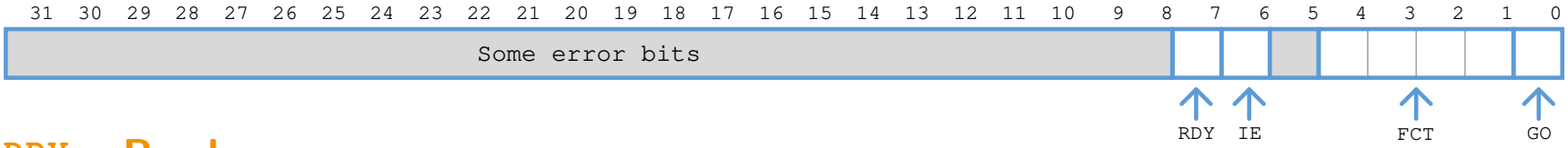
If set to 1, then the DMA operation starts.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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.

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).

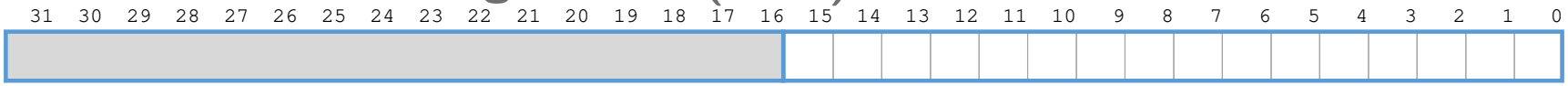
CAMPUS Rosenheim
Computer Science

Technische Hochschule 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)

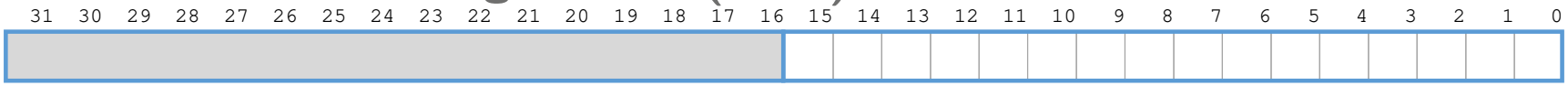


Bits 32 to 47 of the LBA number.

FDD registers (2)

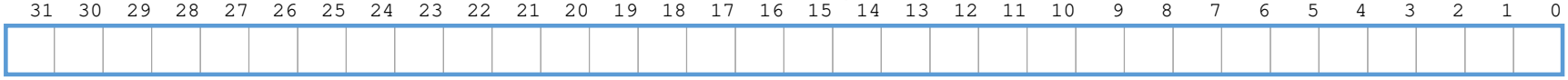
Disk address: The controller uses an **LBA** (logic block address) that is a linear address of blocks, instead of cylinder/head/sector. The LBA has **48 bits** (32 are not enough).

FDD Disk address register HI (DARH)



Bits 32 to 47 of the LBA number.

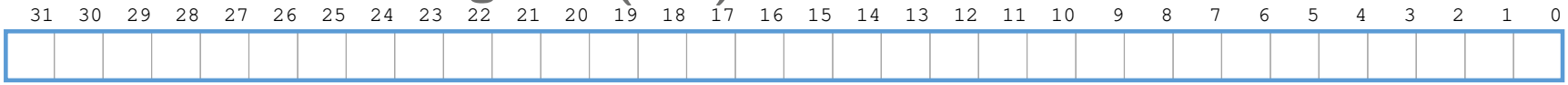
FDD Disk address register LO (DARL)



Bits 0 to 31 of the LBA number.

FDD registers (3)

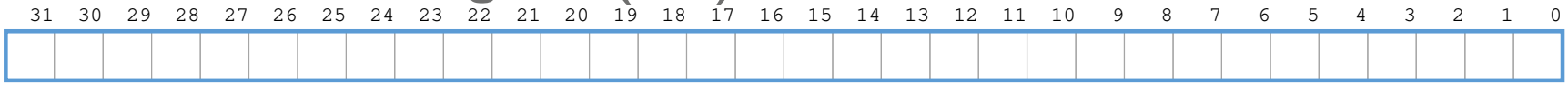
FDD Bus address register (BAR)



Main memory address for transfer.

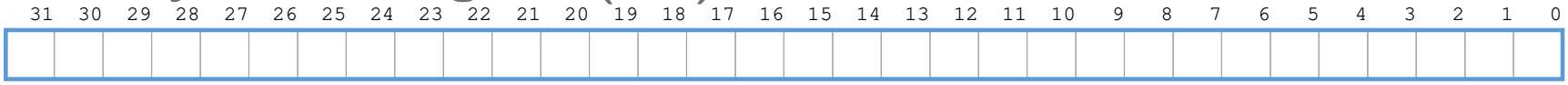
FDD registers (3)

FDD Bus address register (BAR)



Main memory address for transfer.

FDD Byte count register (BCR)



Number of bytes to be transmitted.

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FDD programming example – common

FDD programming example – common definitions

```
1  #include <stdlib.h>
2  #include <stdbool.h>
3  #include <inttypes.h>
4
5  typedef volatile struct { //FDD interface
6      uint32_t CSR;  //control and status register
7      uint32_t DARH; //disk address register HI
8      uint32_t DARL; //disk address register LO
9      uint32_t BAR;  //bus address register
10     uint32_t BCR;  //byte count register
11 } FddStruct;
12
13 //FDD: FddStruct is mapped to the memory position 0xFF000010
14 #define FDD  (*((FddStruct*)(0xFF000010)))
15
16 #define GO    (0x01) //Mask for GO
17 #define IE    (0x40) //Mask for IE
18 #define WRITE (0x02) //Mask for WRITE
19 //more defines...
```

FDD.CSR =

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

FDD programming example (1)

FDD programming example – DMA

```
21 typedef void (*ISR_t)(void); //Function pointer for an ISR
22 //INTVECTOR: ISR_t is mapped to the memory position 0x00000108
23 #define INTVECTOR (*((ISR_t*)(0x00000108)))
24
25 void ISR() { //interrupt service routine for the end of the transfer
26     //notify application that everything is transferred
27 }
28
29 int main(void) {
30     //In principle: {source, destination, how much, IE | R/W | GO}
31     INTVECTOR = &ISR; //ISR address is set to INTVECTOR (address 0x00000108)
32
33     //Configure DMA interface
34     FDD.BAR  = 0x10000; //source memory address
35     FDD.DARH = 0x0;     //destination LBA address (bit 32 to 47)
36     FDD.DARL = 0x1234;  //destination LBA address (bit 0 to 31)
37     FDD.BCR  = 0x100;   //how much: number of bytes
38     FDD.CSR  = 0x43;    //IE | WRITE | GO
39
40     //DMA transmits data now without CPU.
41     //At the end there is an interrupt!
42
43     return EXIT_SUCCESS;
44 }
```

FDD programming example (2)

FDD programming example – inside the OS

```
21 typedef void (*ISR_t)(void); //Function pointer for an ISR
22 //INTVECTOR: ISR_t is mapped to the memory position 0x00000108
23 #define INTVECTOR (*((ISR_t*)(0x00000108)))
24
25 void ISR_fdd_dma_transmitted(); //prototype
26
27 //-------------------------------------------------------------
28 //Part of the glibc: system call interface (SVC) (very high level)
29
30 //Reads an array of count elements, each one with a size of size bytes,
31 //from the stream and stores them in the block of memory specified by ptr.
32 size_t fread(void* ptr, size_t size, size_t count, FILE* stream) {
33     INTVECTOR = &ISR_fdd_dma_transmitted; //ISR address is set to INTVECTOR
34     //Set registers of the FDD just like in the previous example.
35     P(transmit_ready);
36
37     return /*total amount of bytes read*/;
38 }
39
40 //-------------------------------------------------------------
41 //Inside the OS kernel
42 void ISR_fdd_dma_transmitted(){
43     V(transmit_ready);
44 }
```

CAMPUS Rosenheim
Computer Science

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Questions?

All right? ⇒ 

Question? ⇒  and use **chat**

or

speak *after* I ask you to

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

- GPUs
- Multiprocessing

CAMPUS Rosenheim
Computer Science

Technische Hochschule 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

- ~~GPUs~~
- Multiprocessing