SD Card Party

DMA Transfers, Dateisystemzugriff und SD Initialisierung

Christoffer Anselm, Fabian Hinderer, Clara Scherer

Showcase

DMA Transfers: Datenschubsen für Fortgeschrittene

DmaTransfer - Viele Einstellungen, viele Möglichkeiten

```
pub struct DmaTransfer {
    pub dma: DmaManagerRc,
    pub stream: Stream,
    pub channel: Channel,
    pub priority: PriorityLevel,
    pub direction: Direction,
    pub circular_mode: CircularMode,
    pub double_buffering_mode: DoubleBufferingMode,
    pub flow_controller: FlowContoller,
    pub peripheral_increment_offset_size: PeripheralIncrementOffsetSize,
    pub peripheral: DmaTransferNode,
    pub memory: DmaTransferNode,
    pub transaction_count: u16,
    pub direct_mode: DirectMode,
    pub fifo_threshold: FifoThreshold,
    pub interrupt_transfer_complete: InterruptControl,
    pub interrupt_half_transfer: InterruptControl,
    pub interrupt_transfer_error: InterruptControl,
    pub interrupt_direct_mode_error: InterruptControl,
    pub interrupt_fifo: InterruptControl,
}
```

DmaTransfer::new(. . .) - Einfacherer Generator

```
let mut dma_transfer = dma::DmaTransfer::new(
    dma_2.clone(),
    dma::Stream::S0,
    dma::Channel::C3,
    dma::Direction::MemoryToMemory,
    dma::DmaTransferNode {
        address: source_buffer.as_mut_ptr() as *mut u8,
        burst_mode: dma::BurstMode::SingleTransfer,
        increment_mode: dma::IncrementMode::Increment,
        transaction_width: dma::Width::Word,
    },
    dma::DmaTransferNode {
        address: destination_buffer.as_mut_ptr() as *mut u8,
        burst_mode: dma::BurstMode::SingleTransfer,
        increment_mode: dma::IncrementMode::Increment,
        transaction_width: dma::Width::Word,
    },
    (BUFFER_SIZE / 4) as u16
);
```

Verwendung: Polling

Übliche Verwendung, bis Interrupts etwas komfortabler werden:

```
let finish_time;
let start_time = system_clock::ticks();
dma_transfer.start().expect("Failed to start DMA transfer");

loop {
    if !dma_transfer.is_active() {
        finish_time = system_clock::ticks();
        break;
    }
}

dma_transfer.stop();
println!("s: {:?} - d: {:?}", source_buffer, destination_buffer);
println!("time: {}ms", finish_time - start_time);
```

Verwendung: Warten

Starten, was anderes machen, warten.

```
let start_time = system_clock::ticks();
dma_transfer.start().expect("Failed to start DMA transfer");
dma_transfer.wait();
let finish_time = system_clock::ticks();
dma_transfer.stop();
```

I'm not here for performance - I just want my data.

```
dma_transfer.execute().expect("Failed DMA transfer failed");
```

Wichtige Einstellungen

```
pub enum IncrementMode {
    Fixed = 0,
    Increment = 1,
}

pub enum CircularMode {
    Disable = 0,
    Enable = 1,
}

pub enum Direction {
    PeripheralToMemory = 0b00,
    MemoryToPeripheral = 0b01,
    MemoryToMemory = 0b10,
}
```

Controller, Stream und Channel wählen

Table 24. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	SPDIFRX_DT	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	SPDIFRX_CS	SPI3_TX
Channel 1	I2C1_RX	I2C3_RX	TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2C4 _RX	TIM4_CH2	-	I2C4 _TX	TIM4_UP	TIM4_CH3
Channel 3	-	TIM2_UP TIM2_CH3	I2C3_RX	-	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX	UART7_TX	TIM3_CH4 TIM3_UP	UART7_RX	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

Table 25. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	SAI2_B
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	SPI6_TX	SPI6_RX	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX	SPI5_TX	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDMMC1	-	USART1_RX	SDMMC1	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX	SPI5_TX	TIM8_CH4 TIM8_TRIG TIM8_COM

Vorteile

- Schnell
- Vielseitig
- Asynchron (Hält die CPU frei)

Verwendung

- Peripheral <-> Memory
  - Controller, Stream & Channel wählen - Tabelle beachten!
  - Einstellungen hängen von der Peripherie ab
- Memory <-> Memory
  - Controller, Stream & Channel wählen - freie Wahl
  - Zugriff auf alle via FMC angebundenen Speicher
  - Bis zu 255,99kb (65535 * 4b) pro Transfer möglich
  - Kein Circular oder Direct mode!

Fazit

- Implementation rein auf Basis der Dokumentation -> :-(
- Referenzimplementation to the rescue!
- Rust, clippy und rls sehr hilfreich

Dateisystemzugriff: Datei im FAT32 Root-Directory auslesen

Datenträgerinhalt

- MBR
  - ...
  - Partitionstabelle
  - ...
- ...
- erste Partition
  - FAT32 Dateisystem
    - Metadaten
    - FAT -> u32 Liste; Einträge stehen für Cluster in den
    - Nutzdaten
- ...

Modellierung

- BlockDevice (Trait)
  - abstrahiert Datenzugriff
  - read_blocks(..), number_of_blocks(..), block_size(..)
- MbrDeviceDriver
  - Zugriff: BlockDevice
  - liefert: erste Partition
- Partition
  - ist: BlockDevice
  - hat: Dateisystemtyp
- Fat32DeviceDriver
  - Zugriff: BlockDevice
  - liefert: Datei (Vec)

Fat32DeviceDriver Dateizugriff

1. Metadaten liefern: erster Cluster vom Root-Directory
2. Einträge dort enthalten: is_file, name_extension, first_cluster
3. Damit: FAT Clusterkette durchlaufen und
4. entsprechende Cluster in den Nutzdaten konkatenieren (Vec)

Verwendung Codebeispiel

```
let mbr_device_driver = MbrDeviceDriver::new(&block_device);
let partition = mbr_device_driver.get_first_partition();
if partition.get_partition_type() != 0x0B {
    panic!("not FAT32");
}
let fat32_device_driver = Fat32DeviceDriver::new(partition);
let file_vec = fat32_device_driver.read_file_to_vec("tst.txt");
if file_vec.is_some() {
    let file = String::from_utf8(file_vec.unwrap()).unwrap();
    println!("{:?}", file);
} else {
    println!("file not found");
}
```

Showcase

SD Initialisierung

SdHandle

```rust
/// SD handle
// represents SD_HandleTypeDef
pub struct SdHandle {
    registers: &'static mut Sdmmc,
    lock_type: LockType,
    rx_dma_transfer: dma::DmaTransfer,
    tx_dma_transfer: dma::DmaTransfer,
    context: Context,
    state: State,
    error_code: low_level::SdmmcErrorCode,
    sd_card: CardInfo,
}
```

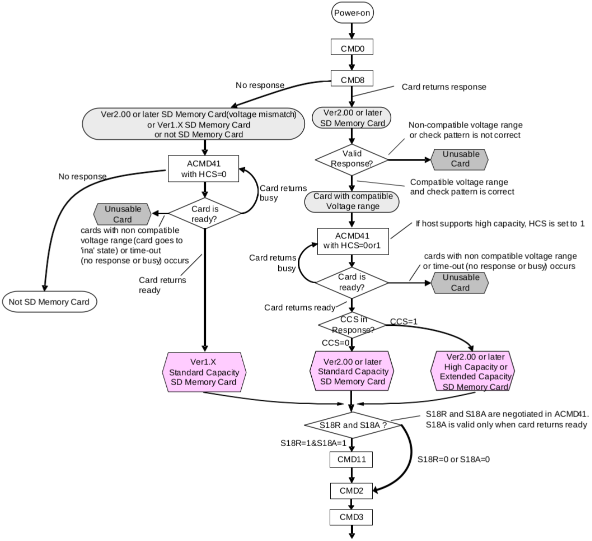
CardInfo

```
// represents HAL_SD_CardInfoTypeDef
#[derive(Debug, PartialEq, Eq)]
struct CardInfo {
    card_type: CardType,
    version: CardVersion,
    class: u16, // einfach Resp2 >> 20
    relative_card_address: u16,
    number_of_blocks: usize,
    block_size: usize,
    logical_number_of_blocks: usize,
    logical_block_size: usize,
    cid: [u32; 4], // Card indentification number data
    csd: [u32; 4], // Card specific data
}
```

Hardware initialisieren

- GPIO Pins per Alternate Function setzen
  - SDMMC Clock
  - SDMMC Command
  - SDMMC Data
- Clock initialisieren und anschalten
- Power On
- Initialisierung der SD Karte

SD Karte initialisieren



Benutzung

- DmaManager initialisieren → dma::DmaManager::init_dma2(dma_2, rcc);
- neues SdHandle struct erzeugen → sd::SdHandle::new(sdmmc, &dma_2, &mut sdram_addr);
- SdHandle initialisieren → sd_handle.init(&mut gpio, rcc);

```
// DMA2 init
let dma_2 = dma::DmaManager::init_dma2(dma_2, rcc);

// SD stuff
let mut sd_handle = sd::SdHandle::new(sdmmc, &dma_2, &mut sdram_addr);
sd_handle.init(&mut gpio, rcc);
```