Embedded Rust Workshop

tweede golf

web / security / embedded

Agenda

This part is divided up into:

- Introduction
- Exploring the Rust embedded ecosystem
- Exercise 1
- Portable drivers in Rust
- Exercise 2

Exploring the Rust embedded ecosystem

- Rust, C++, C

- core, alloc, std
- cargo, crates, rustc

- probe-rs

Cortex-m crates

- Cortex-m
  - Peripheral Access Crate
  - Similar to CMSIS register definitions
- Cortex-m-rt
  - Startup runtime
  - Interrupt setup

Device PACs

Every device has different peripherals. One PAC for every device.

- Generated from SVD
- Imported as dependency

Device PACs

C

```c
#include "samd21e17l.h"

// Raw
bool is_8_cycles = ((WDT->CONFIG.reg & WDT_CONFIG_PER_Msk) << WDT_CONFIG_PER_Pos) == WDT_CONFIG_PER_8_val;
WDT->CONFIG.reg = (WDT->CONFIG.reg & ~WDT_CONFIG_PER_Msk) | WDT_CONFIG_PER_16;

// Bitfield
bool is_8_cycles = WDT->CONFIG.bit.PER == WDT_CONFIG_PER_8_val;
WDT->CONFIG.bit.PER = WDT_CONFIG_PER_16;
```

Rust

```rust
// Take ownership of the peripherals
let dp = atsamd21e::Peripherals::take().unwrap();

let is_8_cycles = dp.WDT.CONFIG.read().per().is_8();
dp.WDT.CONFIG.modify(|_, w| w.per()._8());
```

Overview


PAC
SAMD21E
PAC
nRF9160
PAC
nRF52840
PAC
STM32H743
PAC
STM32H753
PAC
STM32L476
PAC
STM32L496

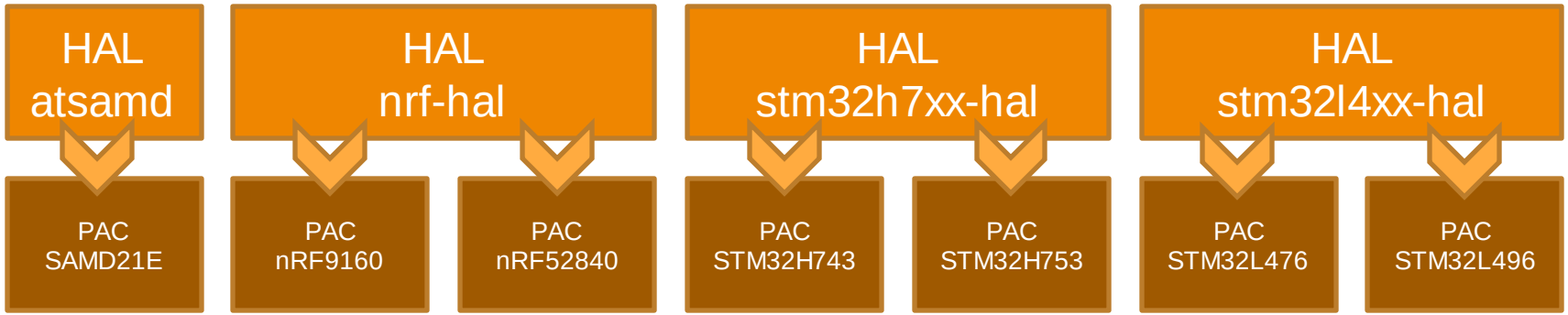
Device HALs

- Many open source HALs
- Most basic operations supported
- Built on top of PACs

Device HALs

```rust
#[entry]
fn main() -> ! {
    // Take the device's peripherals
    let dp = Peripherals::take().unwrap();

    // Create the timer and give it access to the peripheral
    let mut timer = Timer::periodic(dp.TIMER0);
    timer.enable_interrupt();
    timer.start(1000000u32); // Timer runs at 1 Mhz, so it will interrupt every second
    drop(timer);

    // Unmask the timer interrupt in the NVIC, this can be unsafe in some situations,
    // so we have to put it in an unsafe block
    unsafe { NVIC::unmask(Interrupt::TIMER0); }

    loop {}
}

#[interrupt]
fn TIMER0() {
    // Get a reference to the peripheral.
    // This is unsafe because only one instance may exist at a time or we'll trigger UB.
    // In this case it's fine because we dropped the timer in main.
    // Normally we wouldn't do this.
    // We'd have to use a mutex to share the timer peripheral between contexts.
    let timer = unsafe { &*TIMER0::ptr() };
    // Stop the interrupt
    timer.events_compare[0].write(|w| w);
}
```

Overview


HAL
atsamd
PAC
SAMD21E
HAL
nrf-hal
PAC
nRF9160
PAC
nRF52840
HAL
stm32h7xx-hal
PAC
STM32H743
PAC
STM32H753
HAL
stm32l4xx-hal
PAC
STM32L476
PAC
STM32L496

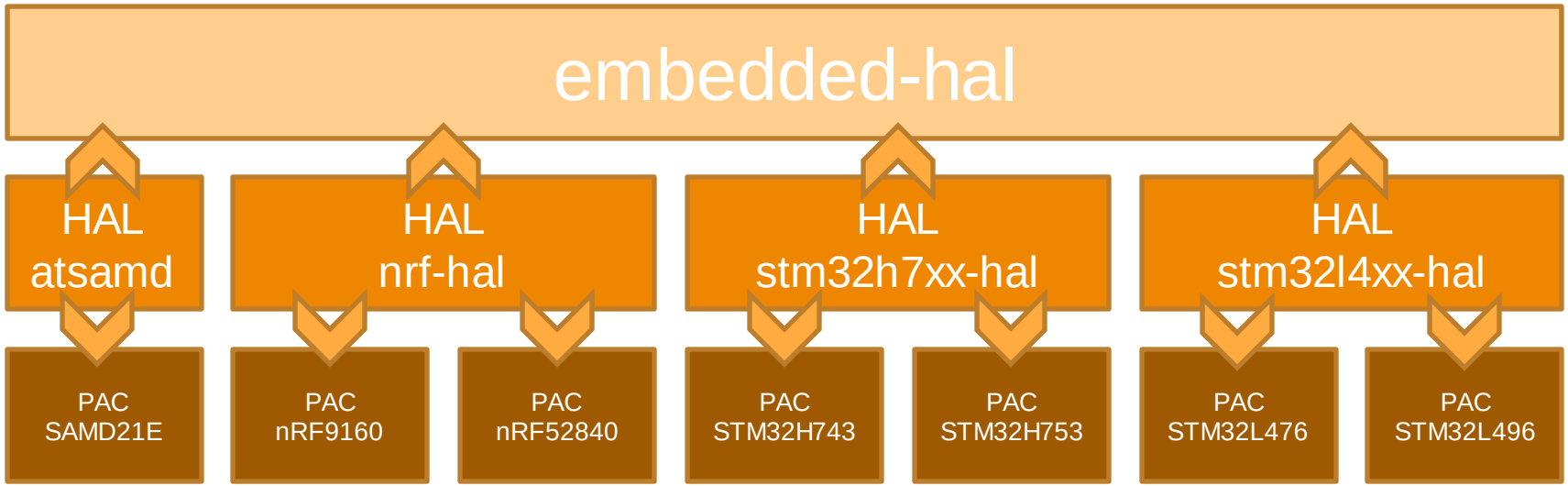
Embedded-hal

The glue of the entire ecosystem

- Contains abstractions for many common operations
- SPI example trait:

```rust
pub trait Transfer<W> {

    type Error;

    fn transfer<'w>(
        &mut self,
        words: &'w mut [W]
    ) -> Result<&'w [W], Self::Error>;
}
```

Overview


embedded-hal
HAL
atsamd
HAL
nrf-hal
HAL
stm32h7xx-hal
HAL
stm32l4xx-hal
PAC
SAMD21E
PAC
nRF9160
PAC
nRF52840
PAC
STM32H743
PAC
STM32H753
PAC
STM32L476
PAC
STM32L496

Typestate

State encoded in the type of the variable

```rust
use nrf52840_hal::gpio::{Pin, p0::P0_04, Input, PullDown, Output, PushPull};

/// Take an nrf pin.
/// It must be:
/// - Port 0 pin 4 (Compile time known)
/// - Configured as input
/// - Pulldown enabled
fn do_something_1(pin: P0_04<Input<PullDown>>) {}

/// Take an nrf pin.
/// It must be:
/// - Any port and pin (Runtime known)
/// - Configured as output
/// - Configured as push-pull
fn do_something_2(pin: Pin<Output<PushPull>>) {}
```

Runtimes

- Bare metal + interrupts
- RTIC (Real-Time Interrupt-driven Concurrency)
- RTOS
- Async

Questions so far?

Exercise 1

Meet the LIS3DH

- Clone the repository: https://github.com/tweedegolf/workshop-embedded-twoparter
- Open your favorite text editor in `‘assignments/part_1’`
- Open the book and read the intructions

- We are going to read the ID register of the chip

Solution

- How did you solve it?
- What were the problems you encountered?
- What did you like or dislike?

Portable drivers

- We want drivers for devices
- We want them to be portable

- Abstraction in C?
- Abstraction in Rust?
- Low level & high level drivers?
- A low level driver
- Possibilities in Rust

Abstraction in C

Fill-in functions

- Efficient
- Inconvenient

```c
#include <stddef.h>

// Todo: Change to your spi type
#define SPI_TYPE void

static SPI_TYPE* spi = NULL;
void init(SPI_TYPE* spi_instance);

static void enable_cs(SPI_TYPE* spi) {
    // Todo: Enable the cs pin
}

static void disable_cs(SPI_TYPE* spi) {
    // Todo: Disable the cs pin
}

static char transfer(SPI_TYPE* spi, char value) {
    // Todo: Transfer 'value' over the bus and return the response
}

char example() {
    enable_cs(spi);
    transfer(spi, 0xDE);
    char result = transfer(spi, 0xAD);
    disable_cs(spi);

    return result;
}
```

C source #1

Save/Load Add new... Vim C

```c
#include <stddef.h>

// Todo: Change to your spi type
#define SPI_TYPE void

static SPI_TYPE* spi = NULL;
void init(SPI_TYPE* spi_instance);

static void enable_cs(SPI_TYPE* spi) {
    // Todo: Enable the cs pin
    (*(volatile char*)0x2000000) = 1;
}

static void disable_cs(SPI_TYPE* spi) {
    // Todo: Disable the cs pin
    (*(volatile char*)0x2000000) = 0;
}

static char transfer(SPI_TYPE* spi, char value) {
    // Todo: Transfer 'value' over the bus and return the response
    (*(volatile char*)0x2000001) = value;
    return (*(volatile char*)0x2000001);
}

char example() {
    enable_cs(spi);
    transfer(spi, 0xDE);
    char result = transfer(spi, 0xAD);
    disable_cs(spi);

    return result;
}
```

tweede golf

x86-64 clang 10.0.0 (Editor #1, Compiler #1) C

x86-64 clang 10.0.0 -O3

Output... Filter... Libraries Add new... Add tool...

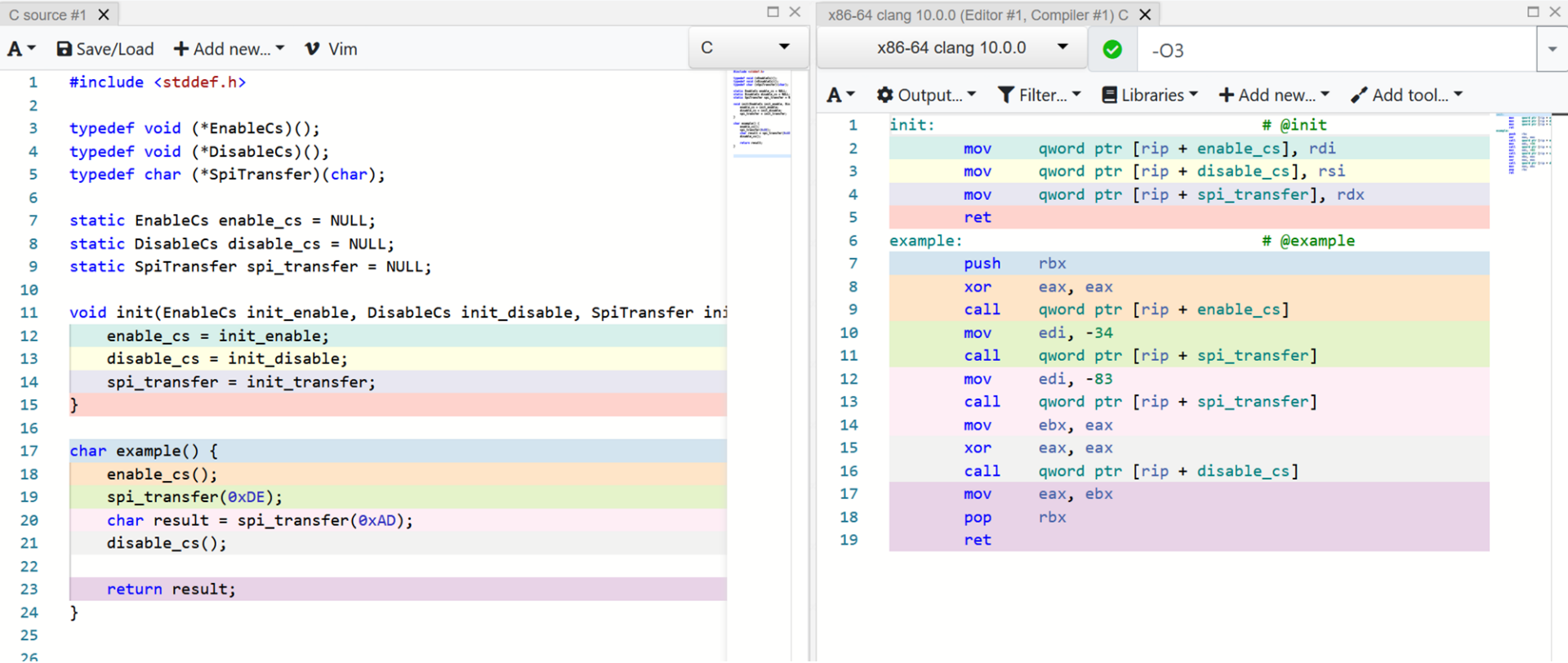
```
example:                                # @example
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], -34
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554433], -83
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
        ret
```

Abstraction in C

Function pointers

- Inefficient
- Convenient

```c
#include <stddef.h>

typedef void (*EnableCs)();
typedef void (*DisableCs)();
typedef char (*SpiTransfer)(char);

static EnableCs enable_cs = NULL;
static DisableCs disable_cs = NULL;
static SpiTransfer spi_transfer = NULL;

void init(EnableCs init_enable, DisableCs init_disable, SpiTransfer init_transfer) {
    enable_cs = init_enable;
    disable_cs = init_disable;
    spi_transfer = init_transfer;
}

char example() {
    enable_cs();
    spi_transfer(0xDE);
    char result = spi_transfer(0xAD);
    disable_cs();

    return result;
}
```

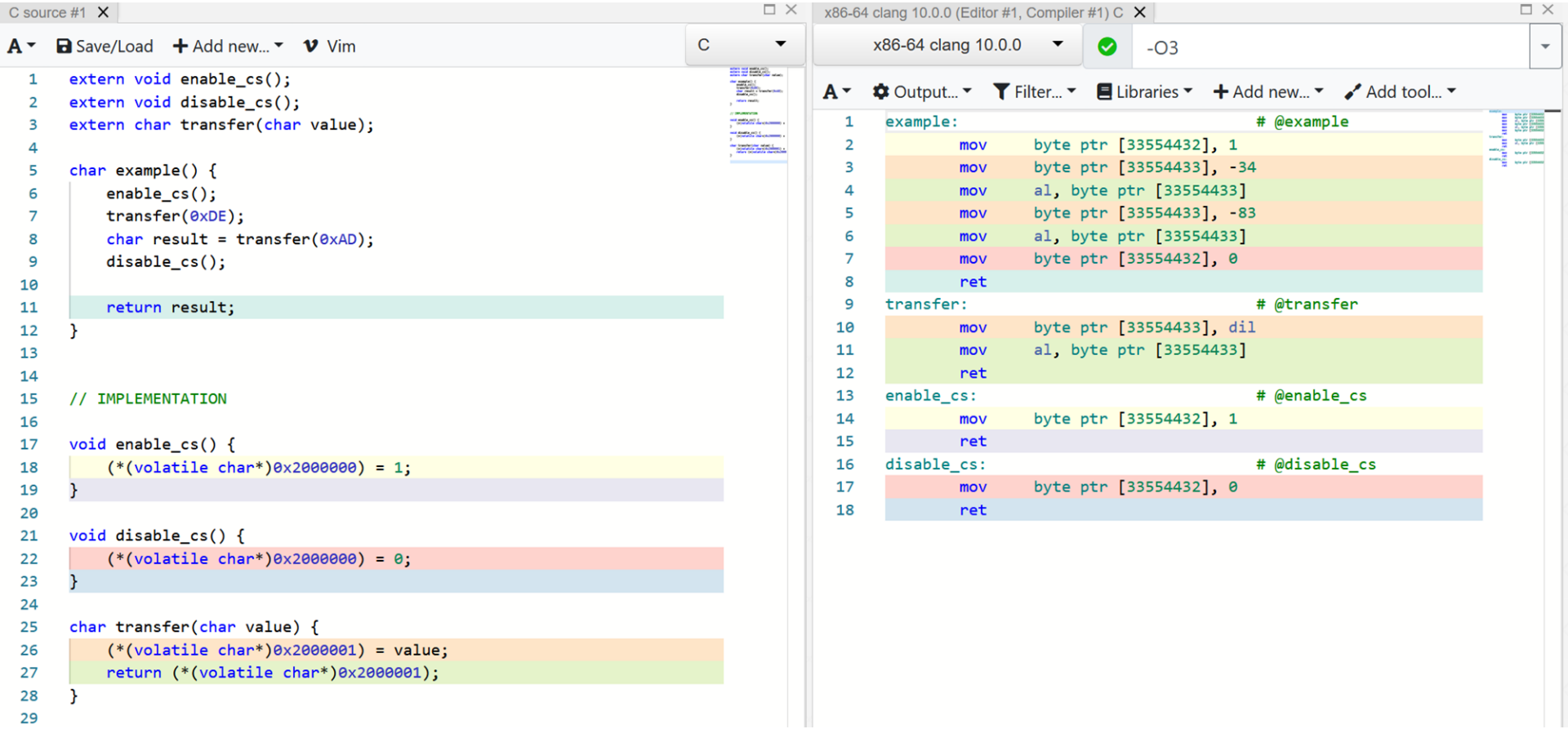

C source #1
Save/Load
Add new...
Vim
C
#include <stddef.h>
typedef void (*EnableCs)();
typedef void (*DisableCs)();
typedef char (*SpiTransfer)(char);
static EnableCs enable_cs = NULL;
static DisableCs disable_cs = NULL;
static SpiTransfer spi_transfer = NULL;
void init(EnableCs init_enable, DisableCs init_disable, SpiTransfer ini
enable_cs = init_enable;
disable_cs = init_disable;
spi_transfer = init_transfer;
}
char example() {
enable_cs();
spi_transfer(0xDE);
char result = spi_transfer(0xAD);
disable_cs();
return result;
}
x86-64 clang 10.0.0 (Editor #1, Compiler #1) C
x86-64 clang 10.0.0
-O3
Output...
Filter...
Libraries
Add new...
Add tool...
init: # @init
mov qword ptr [rip + enable_cs], rdi
mov qword ptr [rip + disable_cs], rsi
mov qword ptr [rip + spi_transfer], rdx
ret
example: # @example
push rbx
xor eax, eax
call qword ptr [rip + enable_cs]
mov edi, -34
call qword ptr [rip + spi_transfer]
mov edi, -83
call qword ptr [rip + spi_transfer]
mov ebx, eax
xor eax, eax
call qword ptr [rip + disable_cs]
mov eax, ebx
pop rbx
ret

Abstraction in C

Link-time binding

- Efficient
- Somewhat convenient
- Error-prone

```
1  extern void enable_cs();
2  extern void disable_cs();
3  extern char transfer(char value);
4
5  char example() {
6      enable_cs();
7      transfer(0xDE);
8      char result = transfer(0xAD);
9      disable_cs();
10
11     return result;
12 }
13
```


C source #1
Save/Load
Add new...
Vim
C
x86-64 clang 10.0.0 (Editor #1, Compiler #1) C
x86-64 clang 10.0.0
-O3
Output...
Filter...
Libraries
Add new...
Add tool...
extern void enable_cs();
extern void disable_cs();
extern char transfer(char value);
char example() {
enable_cs();
transfer(0xDE);
char result = transfer(0xAD);
disable_cs();
return result;
}
// IMPLEMENTATION
void enable_cs() {
(*(volatile char*)0x2000000) = 1;
}
void disable_cs() {
(*(volatile char*)0x2000000) = 0;
}
char transfer(char value) {
(*(volatile char*)0x2000001) = value;
return (*(volatile char*)0x2000001);
}
example: # @example
mov byte ptr [33554432], 1
mov byte ptr [33554433], -34
mov al, byte ptr [33554433]
mov byte ptr [33554433], -83
mov al, byte ptr [33554433]
mov byte ptr [33554432], 0
ret
transfer: # @transfer
mov byte ptr [33554433], dil
mov al, byte ptr [33554433]
ret
enable_cs: # @enable_cs
mov byte ptr [33554432], 1
ret
disable_cs: # @disable_cs
mov byte ptr [33554432], 0
ret

Abstraction in Rust

Traits + generics

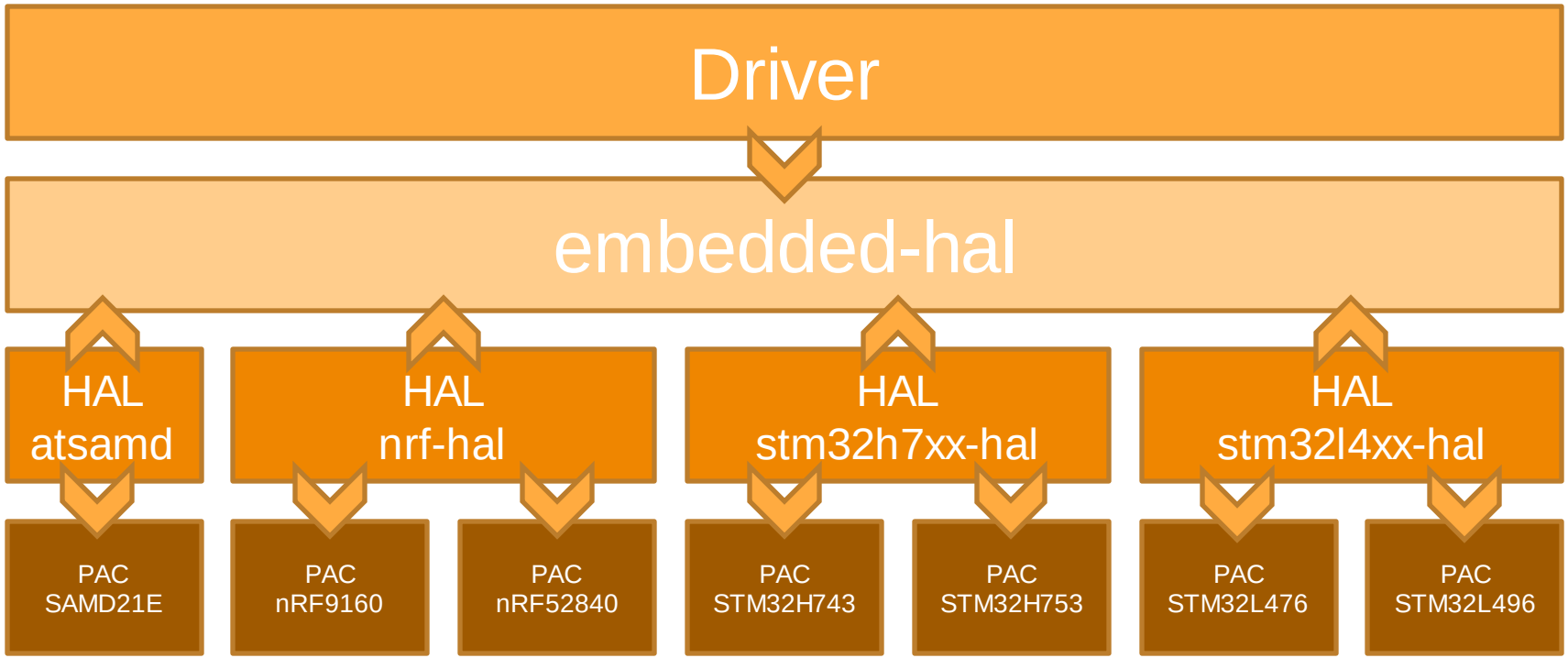
- Reuse traits from embedded-hal
- Efficient
- Convenient

```rust
use embedded_hal::blocking::spi;
use embedded_hal::digital::v2::OutputPin;

pub struct Device<SPI, CS>
where
    SPI: spi::Transfer<u8>,
    CS: OutputPin,
{
    bus: SPI,
    chipselect: CS,
}

impl<SPI, CS> Device<SPI, CS>
where
    SPI: spi::Transfer<u8>,
    CS: OutputPin,
{
    pub fn new(bus: SPI, chipselect: CS) -> Self {
        Self { bus, chipselect }
    }

    pub fn example(&mut self) -> u8 {
        self.chipselect.set_low().ok();
        self.bus.transfer(&mut [0xDE]).ok();
        let result = self.bus.transfer(&mut [0xAD]).ok().unwrap()[0];
        self.chipselect.set_high().ok();

        result
    }
}
```

```rust
struct Spi;
impl spi::Transfer<u8> for Spi {
    type Error = core::convert::Infallible;

    fn transfer<'w>(&mut self, words: &'w mut [u8]) -> Result<&'w [u8], Self::Error> {
        unsafe {
            core::ptr::write_volatile(0x2000001 as *mut u8, words[0]);
            words[0] = core::ptr::read_volatile(0x2000001 as *mut u8);
        }
        Ok(words)
    }
}

struct Pin;
impl OutputPin for Pin {
    type Error = core::convert::Infallible;

    fn set_low(&mut self) -> Result<(), Self::Error> {
        unsafe { core::ptr::write_volatile(0x2000000 as *mut u8, 0) };
        Ok(())
    }

    fn set_high(&mut self) -> Result<(), Self::Error> {
        unsafe { core::ptr::write_volatile(0x2000000 as *mut u8, 1) };
        Ok(())
    }
}

pub fn make_assembly_show_up() {
    let mut device = Device::new(Spi, Pin);
    device.example();
}
```

```
$ cargo asm workshop_code::make_assembly_show_up
workshop_code::make_assembly_show_up:
 mov     byte, ptr, [33554432], 0
 mov     byte, ptr, [33554433], -34
 mov     al, byte, ptr, [33554433]
 mov     byte, ptr, [33554433], -83
 mov     al, byte, ptr, [33554433]
 mov     byte, ptr, [33554432], 1
 ret
```

Overview


Driver
embedded-hal
HAL
atsamd
HAL
nrf-hal
HAL
stm32h7xx-hal
HAL
stm32l4xx-hal
PAC
SAMD21E
PAC
nRF9160
PAC
nRF52840
PAC
STM32H743
PAC
STM32H753
PAC
STM32L476
PAC
STM32L496

Low level & high level drivers

Low level driver

- Hardware abstractions + reading/writing registers
- Register/buffer/command definitions

High level driver

- Implement common routines
- Requirement assertions
- Common interface export

Low level driver in C

- Enum & #define for definitions
- Functions for read/write

C source #1

```c
// -------------------- REGISTER DEFINITIONS --------------------
enum Reg {
    REG_ID = 0x01,
    REG_IER = 0x02,
    REG_IDR = 0x03,
    REG_ISR = 0x04,
    REG_PIN = 0x05,
    REG_PORT = 0x06,
};

#define ID_MANUFACTURER_MASK 0xF0
#define ID_MANUFACTURER_POS 4

// -------------------- LOW LEVEL DEVICE INTERFACE --------------------
static char read_register(enum Reg address) {
    enable_cs();
    transfer(address);
    char result = transfer(0x00);
    disable_cs();

    return result;
}

static void write_register(enum Reg address, char value) {
    enable_cs();
    transfer(address);
    transfer(value);
    disable_cs();
}

// -------------------- EXAMPLE CODE --------------------
void example() {
    char manufacturer =
        (read_register(REG_ID) & ID_MANUFACTURER_MASK) >> ID_MANUFACTURER_POS;

    if (manufacturer != 0) {
        char reg = read_register(REG_PORT);
        reg |= 0x80;
        write_register(REG_PORT, reg);
    }
}
```

x86-64 clang 10.0.0 (Editor #1, Compiler #1) C

x86-64 clang 10.0.0 -O3

```
example:
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 1
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554433], 0
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
        cmp     al, 16
        jb      .LBB0_2
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 6
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554433], 0
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
        or      al, -128
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 6
        mov     cl, byte ptr [33554433]
        mov     byte ptr [33554433], al
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
.LBB0_2:
        ret
```

tweede golf

Low level driver in Rust

- Enum for definitions
- Device struct representing an instance of a chip
- Functions for read/write

Rust source #1

```rust
// -------------------- REGISTER DEFINITIONS --------------------
#[repr(u8)]
pub enum Register {
    Id = 0x01,
    Ier = 0x02,
    Idr = 0x03,
    Isr = 0x04,
    Pin = 0x05,
    Port = 0x06,
}

// -------------------- LOW LEVEL DEVICE INTERFACE --------------------
pub struct Device<I: Interface> {
    interface: I,
}

impl<I: Interface> Device<I> {
    pub fn new(interface: I) -> Self {
        Self { interface }
    }

    pub fn read_register(&mut self, register: Register) -> u8 {
        self.interface.enable_cs();
        self.interface.transfer(register as u8);
        let result = self.interface.transfer(0x00);
        self.interface.disable_cs();

        result
    }

    pub fn write_register(&mut self, register: Register, value: u8) {
        self.interface.enable_cs();
        self.interface.transfer(register as u8);
        self.interface.transfer(value);
        self.interface.disable_cs();
    }
}

// -------------------- EXAMPLE CODE --------------------
pub fn example() {
    let mut device = Device::new(Spi);

    let manufacturer = (device.read_register(Register::Id) & 0xF0) >> 4;

    if manufacturer != 0 {
        let mut reg = device.read_register(Register::Port);
        reg |= 0x80;
        device.write_register(Register::Port, reg);
    }
}
```

rustc 1.44.0 (Editor #1, Compiler #1) Rust

rustc 1.44.0 -C opt-level=3

```
example::example:
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 1
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554433], 0
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
        cmp     al, 16
        jb      .LBB0_2
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 6
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554433], 0
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
        or      al, -128
        mov     byte ptr [33554432], 1
        mov     byte ptr [33554433], 6
        mov     cl, byte ptr [33554433]
        mov     byte ptr [33554433], al
        mov     al, byte ptr [33554433]
        mov     byte ptr [33554432], 0
.LBB0_2:
        ret

<example::Spi as example::Interface>::enable_cs:
        mov     byte ptr [33554432], 1
        ret

<example::Spi as example::Interface>::disable_cs:
        mov     byte ptr [33554432], 0
        ret

<example::Spi as example::Interface>::transfer:
        mov     byte ptr [33554433], sil
        mov     al, byte ptr [33554433]
        ret
```

tweede golf

Possibilities in Rust

Much more is possible:

```rust
// -------------------- EXAMPLE CODE --------------------
pub fn example() {
    let mut device = Device::new(Spi);

    let manufacturer = device.id().read().manufacturer();

    if manufacturer != 0 {
        device.port().modify(|_, w| w.enable_7(true));
    }
}
```

```c
// -------------------- EXAMPLE CODE --------------------
void example() {
    char manufacturer =
        (read_register(REG_ID) & ID_MANUFACTURER_MASK) >> ID_MANUFACTURER_POS;

    if (manufacturer != 0) {
        char reg = read_register(REG_PORT);
        reg |= 0x80;
        write_register(REG_PORT, reg);
    }
}
```

Possibilities in Rust

- Add typestate to high level driver

Struct dw1000::hl::DW1000 [-][src]

[+] Show declaration

[-] Entry point to the DW1000 driver API

Implementations

[-] `impl<SPI, CS> DW1000<SPI, CS, Uninitialized>` [src]

```
where
    SPI: Transfer<u8> + Write<u8>,
    CS: OutputPin,
```

[-] `pub fn new(spi: SPI, chip_select: CS) -> Self` [src]

Create a new instance of `DW1000`

Requires the SPI peripheral and the chip select pin that are connected to the DW1000.

[-] `pub fn init(mut self: Self) -> Result<DW1000<SPI, CS, Ready>, Error<SPI, CS>>` [src]

Initialize the DW1000

The DW1000's default configuration is somewhat inconsistent, and the user manual (section 2.5.5) has a long list of default configuration values that should be changed to guarantee everything works correctly. This method does just that.

Please note that this method assumes that you kept the default configuration. It is generally recommended not to change configuration before calling this method.

[+] `impl<SPI, CS> DW1000<SPI, CS, Ready>` [src]

```
where
    SPI: Transfer<u8> + Write<u8>,
    CS: OutputPin,
```

[+] `impl<SPI, CS> DW1000<SPI, CS, Sending>` [src]

```
where
    SPI: Transfer<u8> + Write<u8>,
    CS: OutputPin,
```

[+] `impl<SPI, CS> DW1000<SPI, CS, Receiving>` [src]

```
where
    SPI: Transfer<u8> + Write<u8>,
    CS: OutputPin,
```

[+] `impl<SPI, CS, State> DW1000<SPI, CS, State>` [src]

```
where
    SPI: Transfer<u8> + Write<u8>,
    CS: OutputPin,
```

Possibilities in Rust

Much more is possible:

- Embedded-hal
- Radio
- Embedded-nal
- Usb-device
- Embedded-graphics
- Accelerometer
- Embedded-storage

Questions so far?

Exercise 2

A real driver for the LIS3DH

- Open the book and read the intructions

Solution

- How did you solve it?
- What were the problems you encountered?
- What did you like or dislike?

tweede golf

web / security / embedded

Castellastraat 26, 6512 EX Nijmegen
info@tweedegolf.com
024 3010 484