Rust stm32f4xx_hal Crate - UART

Introduction

Out of curiosity, I decided to try out the stm32f4xx_hal crate. As an embedded target, I used my STM32F401RE Discovery Board. I don't really use HALs and rather do baremetal like a l33t haxxx0rrr but whatever, can't always be cool. My goal was to send data via the UART interface to a FTDI-to-USB connector and display the data sent by the STM32F4 via minicom.

Setup

The examples provided on the crate's github page did not work for me, for what reason ever. Chances are high I messed up. So I got to coding. My implementation is very similar to the example: serial.rs Let's go through the project setup Use cargo to create a new project with the name of your choice. For me that name is haltest. `cargo new haltest`

```
.
├── .cargo
│   └── config
├── Cargo.lock
├── Cargo.toml
├── connect.sh
├── memory.x
└── src
    └── main.rs
```

Next, we add a `config` file under `.cargo/config`

```
mkdir .cargo
# add this to config
[target.thumbv7em-none-eabihf]
runner = 'gdb-multiarch -q'
rustflags = [
  "-C", "link-arg=-Tlink.x",
]
[build]
target = "thumbv7em-none-eabihf"
[env]
DEFMT_LOG = "info"
```

Please note the gdb-multiarch -q if gdb has a different name on your system, this will not work. To connect to the MCU, I use openocd. I packed the connection into `connect.sh`

```
#!/bin/bash

# start openocd and connect to target
openocd -f interface/stlink-v2-1.cfg -f target/stm32f4x.cfg
```

... and of course, we need the memory layout in `memory.x`

```
MEMORY
{
    /* NOTE K = KiBi = 1024 bytes */
    FLASH : ORIGIN = 0x08000000, LENGTH = 256K
```

```
    RAM : ORIGIN = 0x20000000, LENGTH = 64K
}   /* original file featured some comments below ... */
```

Last thing is the Cargo.toml

```
[package]
name = "haltest"
version = "0.1.0"
edition = "2021"
[dependencies]
embedded-hal = "0.2"
nb = "1"
cortex-m = "0.7"
cortex-m-rt = "0.7"
panic-halt = "0.2"
[dependencies.stm32f4xx-hal]
version = "0.11.1"
features = ["rt", "stm32f401"]
```

Source Code

So, here is my source code with lots of comments. As mentioned, very similar to the example in the repo.

```
#![no_main]

#![no_std]
use panic_halt as _;
use cortex_m_rt::entry;
use stm32f4xx_hal as hal;
use cortex_m::Peripherals as _core_periph;
use crate::hal::{pac, prelude::*};
use stm32f4xx_hal::serial::{config, Serial};
use stm32f4xx_hal::delay::Delay;
use core::fmt::Write;

#[entry]
fn main() -> ! {
    // peripheral access crate periphs
    let dp = pac::Peripherals::take().unwrap();
    // core peripherals
    let mut cp = _core_periph::take().unwrap();
    // get some peripheral handles
    let gpioa = dp.GPIOA.split();
    let rcc = dp.RCC.constrain();
    // set the clock which we want to use including the frequency
    let clocks = rcc.cfgr.use_hse(8.mhz()).freeze();
    // this configures the system timer as a delay provider
    let mut delay = Delay::new(cp.SYST, &clocks);
    // set alternate function mode for GPIOA pin 9
    let mut tx_pin = gpioa.pa9.into_alternate();
    // configure the USART. use the default, with a baudrate of 9600
    let usart_config = config::Config::default().baudrate(9600.bps());
    // now get a serial tx we can write to, implement the config on USART1
    let mut tx = Serial::tx(dp.USART1, tx_pin, usart_config, &clocks).unwrap();
    loop {
        tx.write_str("hello\r\n").unwrap();
        delay.delay_ms(1_000_u16);
    }
```

```
}
```

Testing

To test the code, I compiled it with cargo build first and then ran it using cargo run. Thanks to the .cargo/config, everything is configured for cross-compiling and no additional flags must be passed to cargo. Here are the steps:

```
# terminal 1 - openOCD
./connect.sh

# terminal 2 - GDB
cargo run
... in GDB:
target remote :3333
load
continue

# terminal 3 - minicom

# 9600 8N1 Flow Control off (HW and SW)
sudo minicom -s
```

In the minicom window, you should be able to see data coming in.

Conclusion

The stm32f4xx_hal crate is awesome in my opinion, amazing work by the authors. I'm gonna stick with this for future projects and will keep the posts coming. See Ya!

<= back