```rust
//! A driver for generic GPIO driven CD74HC4067
//!
//! This driver was built using [`embedded-hal`] traits.
//!
//! [`embedded-hal`]: https://docs.rs/embedded-hal/~0.2
//!
//! TODO # Examples

#![deny(unsafe_code)]
#![deny(missing_docs)]
#![cfg_attr(not(test), no_std)]

use core::marker::PhantomData;

/// Errors of this crate
#[derive(Debug, Copy, Clone, PartialEq, Eq)]
pub enum Error<P: OutputPin, E: OutputPin> {
    /// Error setting a pin
    SelectPinError(P::Error),
    /// Error enabling/disabling
    EnablePinError(E::Error),
}

struct EnabledState;
struct DisabledState;

use embedded_hal as hal;

use hal::digital::v2::OutputPin;

/// A structure representing the 4 input pins and pin_enable pin
pub struct CD74HC4067<P, E, State> {
    pin_0: P,
    pin_1: P,
    pin_2: P,
    pin_3: P,
    pin_enable: E,
    state: PhantomData<State>,
}

impl<P, E> CD74HC4067<P, E, DisabledState>
where
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    E: OutputPin,
{
    /// Create a new CD74HC4067 structure by passing in 5 GPIOs implementing the
    /// `OutputPin` trait for `a`, `b`, `c`, `d`
    /// Mux is initially disabled, and all select pins are set low, selecting channel 0.
    pub fn new(
        mut pin_0: P,
        mut pin_1: P,
        mut pin_2: P,
        mut pin_3: P,
        mut pin_enable: E,
    ) -> Result<Self, Error<P, E>> {
        // Disable the mux
        pin_enable.set_high().map_err(Error::EnablePinError)?;
        // Set to output 0
        pin_0.set_low().map_err(Error::SelectPinError)?;
        pin_1.set_low().map_err(Error::SelectPinError)?;
        pin_2.set_low().map_err(Error::SelectPinError)?;
        pin_3.set_low().map_err(Error::SelectPinError)?;

        Ok(Self {
            pin_0,
            pin_1,
            pin_2,
            pin_3,
            pin_enable,
            state: PhantomData::<DisabledState>,
        })
    }

    /// Release the 5 GPIOs previously occupied
    pub fn release(self) -> (P, P, P, P, E) {
        (
            self.pin_0,
            self.pin_1,
            self.pin_2,
            self.pin_3,
            self.pin_enable,
        )
    }

    /// Enable the mux display by pulling `pin_enable` low
    /// If Error::EnablePinError occurs, the struct is dropped.
    pub fn enable(mut self) -> Result<CD74HC4067<P, E, EnabledState>, Error<P, E>> {
        self.pin_enable.set_low().map_err(Error::EnablePinError)?;
        Ok(CD74HC4067 {
            pin_0: self.pin_0,
            pin_1: self.pin_1,
            pin_2: self.pin_2,
            pin_3: self.pin_3,
            pin_enable: self.pin_enable,
            state: PhantomData::<EnabledState>,
        })
    }

    /// Enable output `n`. `n` must be between 0 and 15 inclusive.
    /// If a SelectPinError occurs, the select is left in a possibly unwanted state, but it is disabled here.
    pub fn set_output_active(&mut self, n: u8) -> Result<(), Error<P, E>> {
        assert!(n < 16);
        let is_bit_set = |b: u8| -> bool { n & (1 << b) != 0 };

        if is_bit_set(0) {
            self.pin_0.set_high().map_err(Error::SelectPinError)?;
        } else {
            self.pin_0.set_low().map_err(Error::SelectPinError)?;
        }
        if is_bit_set(1) {
            self.pin_1.set_high().map_err(Error::SelectPinError)?;
        } else {
```

```rust
            self.pin_1.set_low().map_err(Error::SelectPinError)?;
        }
        if is_bit_set(2) {
            self.pin_2.set_high().map_err(Error::SelectPinError)?;
        } else {
            self.pin_2.set_low().map_err(Error::SelectPinError)?;
        }
        if is_bit_set(3) {
            self.pin_3.set_high().map_err(Error::SelectPinError)?;
        } else {
            self.pin_3.set_low().map_err(Error::SelectPinError)?;
        }
        Ok(())
    }
}

impl<P, E> CD74HC4067<P, E, EnabledState>
where
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    P: OutputPin,
    E: OutputPin,
{
    /// Disable the mux display by pulling `pin_enable` high
    pub fn disable(mut self) -> Result<CD74HC4067<P, E, DisabledState>, Error<P, E>> {
        self.pin_enable.set_high().map_err(Error::EnablePinError)?;
        Ok(CD74HC4067 {
            pin_0: self.pin_0,
            pin_1: self.pin_1,
            pin_2: self.pin_2,
            pin_3: self.pin_3,
            pin_enable: self.pin_enable,
            state: PhantomData::<DisabledState>,
        })
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    use embedded_hal_mock::pin::{
        Mock as PinMock, State as PinState, Transaction as PinTransaction,
    };

    #[test]
    fn make_mux() {
        let pin_0 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_1 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_2 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_3 = PinMock::new(&[PinTransaction::set(PinState::Low)]);

        let expectations = [PinTransaction::set(PinState::High)];

        let pin_enable = PinMock::new(&expectations);

        let mux = CD74HC4067::new(pin_0, pin_1, pin_2, pin_3, pin_enable).unwrap();
        let (mut pin_0, mut pin_1, mut pin_2, mut pin_3, mut pin_enable) = mux.release();
        pin_enable.done();
        pin_0.done();
        pin_1.done();
        pin_2.done();
        pin_3.done();
    }

    #[test]
    fn enable() {
        let pin_0 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_1 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_2 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_3 = PinMock::new(&[PinTransaction::set(PinState::Low)]);

        let expectations = [
            PinTransaction::set(PinState::High),
            PinTransaction::set(PinState::Low),
            PinTransaction::set(PinState::High),
        ];

        let pin_enable = PinMock::new(&expectations);

        let mux = CD74HC4067::new(pin_0, pin_1, pin_2, pin_3, pin_enable).unwrap();
        let enabled_mux = mux.enable().unwrap();
        let mux = enabled_mux.disable().unwrap();

        let (mut pin_0, mut pin_1, mut pin_2, mut pin_3, mut pin_enable) = mux.release();
        pin_enable.done();
        pin_0.done();
        pin_1.done();
        pin_2.done();
        pin_3.done();
    }

    #[test]
    fn set_output_to_9() {
        let pin_0 = PinMock::new(&[PinTransaction::set(PinState::High)]);
        let pin_1 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_2 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_3 = PinMock::new(&[PinTransaction::set(PinState::High)]);

        let pin_enable = PinMock::new(&[]);

        let mut mux = CD74HC4067 {
            pin_0,
            pin_1,
            pin_2,
            pin_3,
            pin_enable,
            state: PhantomData::<DisabledState>,
        };

        mux.set_output_active(9).unwrap();

        let (mut pin_0, mut pin_1, mut pin_2, mut pin_3, mut pin_enable) = mux.release();

        pin_0.done();
        pin_1.done();
        pin_2.done();
        pin_3.done();
        pin_enable.done();
    }

    #[test]
    fn set_output_to_6() {
        let pin_0 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_1 = PinMock::new(&[PinTransaction::set(PinState::High)]);
        let pin_2 = PinMock::new(&[PinTransaction::set(PinState::High)]);
        let pin_3 = PinMock::new(&[PinTransaction::set(PinState::Low)]);

        let pin_enable = PinMock::new(&[]);

        let mut mux = CD74HC4067 {
            pin_0,
            pin_1,
            pin_2,
            pin_3,
            pin_enable,
            state: PhantomData::<DisabledState>,
        };

        mux.set_output_active(6).unwrap();

        let (mut pin_0, mut pin_1, mut pin_2, mut pin_3, mut pin_enable) = mux.release();

        pin_0.done();
        pin_1.done();
        pin_2.done();
        pin_3.done();
        pin_enable.done();
    }

    #[test]
    fn set_output_to_10() {
        let pin_0 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_1 = PinMock::new(&[PinTransaction::set(PinState::High)]);
        let pin_2 = PinMock::new(&[PinTransaction::set(PinState::Low)]);
        let pin_3 = PinMock::new(&[PinTransaction::set(PinState::High)]);

        let pin_enable = PinMock::new(&[]);

        let mut mux = CD74HC4067 {
            pin_0,
            pin_1,
            pin_2,
            pin_3,
            pin_enable,
            state: PhantomData::<DisabledState>,
        };

        mux.set_output_active(10).unwrap();

        let (mut pin_0, mut pin_1, mut pin_2, mut pin_3, mut pin_enable) = mux.release();

        pin_0.done();
        pin_1.done();
        pin_2.done();
        pin_3.done();
        pin_enable.done();
    }

    struct DumbPin;
    impl OutputPin for DumbPin {
        type Error = ();

        fn set_low(&mut self) -> Result<(), Self::Error> {
            Ok(())
        }

        fn set_high(&mut self) -> Result<(), Self::Error> {
            Ok(())
        }
    }

    #[test]
    fn dumb_pin_does_nothing() {
        let mut d = DumbPin;
        assert!(d.set_high().is_ok());
        assert!(d.set_low().is_ok());
    }

    #[test]
    #[should_panic]
    fn set_output_panic_16() {
        let mut mux = CD74HC4067 {
            pin_0: DumbPin,
            pin_1: DumbPin,
            pin_2: DumbPin,
            pin_3: DumbPin,
            pin_enable: DumbPin,
            state: PhantomData::<DisabledState>,
        };

        let _unreachable_result = mux.set_output_active(16);
    }

    #[test]
    #[should_panic]
    fn set_output_panic_20() {
        let mut mux = CD74HC4067 {
            pin_0: DumbPin,
            pin_1: DumbPin,
            pin_2: DumbPin,
            pin_3: DumbPin,
            pin_enable: DumbPin,
            state: PhantomData::<DisabledState>,
        };

        let _unreachable_result = mux.set_output_active(20);
    }
}
```