OLED Front Panel Driver

- I²C slave operating at 100kHz or 400kHz
- Reports input events in an event FIFO
- Rotary encoder quadrature decode
- Button press debounce
- Piezo sounder tone generation
- 4 GPIO lines capable of output or input, including change reporting

Basic Operation

The control chip on an OLED-mini Front Panel Module is a self-contained microcontroller accessible as an I²C slave. This chip handles operation of the rotary encoder and push-button, piezo sounder, and GPIO header on the board.

The chip responds to wheel rotation and press events, and can autonomously drive the piezo sounder to make a beep sound when these occur, independently of the host MCU.

An 8-level FIFO of events is maintained, which reports input events on the rotary encoder or changes in state of the GPIO lines. Whenever this FIFO is non-empty, the INT output line is pulled low. When the FIFO is emptied, the INT line is returned to hi-Z state.

I²C Operation

The control chip is accessed via a standard I²C bus, which can operate at 100kHz or 400kHz. The 1MHz "fast mode" is not supported. The slave address is fixed at `0x3D` (`0x7A` write, `0x7B` read). The chip makes use of SCL clock stretching to insert delays on the bus when it requires extra time to respond, so needs a bus master that can accept this. If any bus isolation or multiplexing is used, that too will need to support SCL clock stretching by slaves.

The chip operates as a simple register-based slave, containing a set of individually-numbered registers. Each register may be read or written to (though some registers are notionally read-only, and attempts to write will be ignored).

Register Writes

Registers may be written to in a single WRITE transaction, consisting of a START condition, slave-addressing byte, register address, value, and STOP condition.

START	**Slave (W)**	ACK	**Addr**	ACK	**Value**	ACK	STOP
	Master sends the slave writing address; `0x7A`		Master sends a register number - see the section below		Master sends the new value for the register		

Register Reads

Registers may be read from in a single WRITE-then-READ transaction, consisting of a START, slave-addressing byte, register address, repeated START, another slave-addressing byte, reading the value, STOP.

START	Slave (W)	ACK	Addr	ACK	RE-START	Slave (R)	ACK	Value	ACK	STOP
	Master sends the slave writing address; `0x7A`		Master sends a register number - see the section below			Master sends the slave reading address; `0x7B`		Slave sends the current value of the register		

Detailed Register List

EVENT

Address: `0x01`

Direction: R

Default value: `0x00`

Implements an 8-level deep FIFO of input events. Each event is formed of a single byte. The top three bits of the event classify it into a category, the remaining bits are specific to that category.

`0b000xxxxx`	**No event** `0b00000000` / `0x00`
`0b001xxxxx`	**Wheel rotation**: lower bits give the wheel direction `0b00100001` / `0x21` - wheel downwards (anticlockwise) `0b00100010` / `0x22` - wheel upwards (clockwise)
`0b010xxxxx`	**Button**: lowest bit gives the button press/release state `0b01000000` / `0x40` - wheel button release `0b01000001` / `0x41` - wheel button press
`0b011xxxxx`	**GPIO state change**: lower 4 bits give the GPIO line state at the time the event occurred `0b01100000` / `0x60` to `0b00101111` / `0x6F` - GPIO state captured

When an input event occurs, the INT line will be pulled low. Reading from this register will clear the event from the queue, causing the next to appear. If there are no more events waiting, the register will read as zero and the INT line will be returned to high impedance state.

RELEASEMASK

Address: `0x02`

Direction: R/W

Default value: `0x00`

Controls whether buttons will send event reports on release as well as press (bits set to `1`), or on press only (bits set to `0`).

`0b.......x`	Ignored
`0b......x.`	Wheel button

KEYBEEP_DURATION

Address: `0x10`

Direction: R/W

Default value: `0x0A` / `10`

Gives the duration in centiseconds for an autonomous keybeep.

KEYBEEP_MASK

Address: `0x11`

Direction: R/W

Default value: `0x00`

Controls which button events cause an autonomous keybeep.

`0b.......x`	Wheel rotation
`0b......x.`	Wheel button

BEEP_DURATION

Address: `0x12`

Direction: R/W

Default value: `0x00`

Writing a non-zero value into this register will immediately cause a beep of the given duration, in centiseconds. Reading the value will give a current countdown of the remaining time for the ongoing beep.

BEEP_TONE

Address: `0x13`

Direction: R/W

Default value: `0xC8` / `200`

Sets the frequency of the beep tone, in units of 10 Hz. The default value of `200` sets a tone of 2 kHz, which is ideal for the piezo sounder being used. Note that altering this value will also change the tone used by the autonomous keybeep.

GPIO_DIR

Address: `0x30`

Direction: R/W

Default value: `0x00`

Sets the drive direction on each of the GPIO lines. Bits set low (`0`) are inputs; set high (`1`) are outputs. The lower four bits are used; the upper four are ignored.

GPIO_IO

Address: `0x31`

Direction: R/W

Default value: `0x00`

Writes to this register set the output state for any GPIO lines currently set as outputs. The state of pins set as inputs is ignored. The lower four bits are used; the upper four are ignored.

Reads from this register return the current state of all four GPIO lines, regardless of drive state.

GPIO_PULLUP

Address: `0x32`

Direction: R/W

Default value: `0x00`

Enables pullup resistors on any of the GPIO lines with bits set high (`1`). This pullup is about 50kohm; sufficient to hold a high state around a button or similar input. The lower four bits are used; the upper four are ignored.

GPIO_EVENTMASK

Address: `0x33`

Direction: R/W

Default value: `0x00`

Sets a mask used for level change event detection on the GPIO lines. A change of state on any lines which are set as inputs and have the mask bit high (`1`) will be reported as a GPIO event into the event FIFO. This event will capture the current value of all four lines at the time the level changed.

Possible Future Expansions

The operation of the control chip is defined by re-flashable firmware which can be programmed via the main IO pin header on the OLED panel board itself. The following possible ideas may be implemented in a future update:

- Timer to detect long press of buttons, allowing offload of that from the host MCU too
- Use of (a fixed) 3 GPIO lines for SPI transfer, allowing offload of frontpanel LEDs via 74'595 chains or similar