KMK FW - Encoder module

Add twist control to your keyboard! Volume, zoom, anything you want.

I2C encoder type has been tested with the Adafruit I2C QT Rotary Encoder with NeoPixel.

Note: If you have a **split** keyboard and encoders on **both sides** should work, it's currently necessary to use the encoder-scanner explained at the bottom of scanners docs.

Enabling the extension

The constructor(`EncoderHandler` class) takes a list of encoders, each one defined as either:

- a list of `pad_a` pin, `pad_b` pin, `button_pin` and optionally a flag set to `True` if you want encoder direction to be reversed;
- a `busio.I2C`, address and optionally a flag set to `True` if you want it to be reversed.

The `encoder_map` is modeled after the keymap and works the same way. It should have as many layers (key pressed on "turned left", key pressed on "turned right", key pressed on "knob pressed") as your keymap, and use `KC.NO` keys for layers that you don't require any action. The encoder supports a velocity mode if you desire to make something for video or sound editing.

How to use

Here is all you need to use this module in your `main.py` / `code.py` file.

1. Load the module: import encoder handler and add it to keyboard modules.

```
from kmk.modules.encoder import EncoderHandler
encoder_handler = EncoderHandler()
keyboard.modules = [layers, holdtap, encoder_handler]
```

1. Define the pins for each encoder: `pin_a`, `pin_b` for rotations, `pin_button` for the switch in the encoder. Set switch to `None` if the encoder's button is handled differently (as a part of matrix for example) or not at all. If you want to invert the direction of the encoder, set the 4th (optional) parameter `is_inverted` to `True`. 5th parameter is encoder divisor (optional), it can be either `2` or `4`. If your encoder button pull direction is not the default of `digitalio.Pull.UP`, you may specify the 6th (optional) parameter `button_pull` as `digitalio.Pull.DOWN`.

```
# Regular GPIO Encoder
encoder_handler.pins = (
    # regular direction encoder and a button
    (board.GP17, board.GP15, board.GP14,), # encoder #1
    # reversed direction encoder with no button handling and divisor of 2
    (board.GP3, board.GP5, None, True, 2,), # encoder #2
    )
```

Or in case you have an I2C encoder on a special PCB (e.g. Adafruit I2C QT Rotary Encoder), define I2C encoder as following.

```
# I2C Encoder

# Setup i2c
SDA = board.GP0
SCL = board.GP1
i2c = busio.I2C(SCL, SDA)

encoder_handler.pins = ((i2c, 0x36, False), (encoder 2 definition), etc. )
```

1. Define the mapping of keys to be called for each layer.

```
# You can optionally predefine combo keys as for your layout
Zoom_in = KC.LCTRL(KC.EQUAL)
Zoom_out = KC.LCTRL(KC.MINUS)

encoder_handler.map = [ (( KC.VOLD, KC.VOLU, KC.MUTE), (encoder 2
definition), etc. ), # Layer 1
                        ((Zoom_out, Zoom_in, KC.NO), (encoder 2 definition),
etc. ), # Layer 2
                        ((KC.A, KC.Z, KC.N1), (encoder 2 definition), etc. ),
# Layer 3
                        ((KC.NO, KC.NO, KC.NO), (encoder 2 definition), etc.
), # Layer 4
                      ]
```

Encoder divisor

Depending on your encoder resolution, it may send 4 or 2 pulses (state changes) on every detent. This number is controlled by the `divisor` property. By default the divisor is set to 4, but if your encoder only activates on every second detent (skips pulses), set the divisor to 2. If the encoder activates twice on every detent, set the value to 4.

You can change the default globally for all encoders **before** initializing the encoder pins (`main.py` file):

```
encoder_handler.divisor = 2
encoder_handler.pins = ( (board.GP14, board.GP15, None), (board.GP26,
board.GP27, None), )
```

Or if you have different types of encoders, set divisor for each encoder individually:

```
encoder_handler.pins = (
    (board.GP14, board.GP15, None, False, 4),
    (board.GP26, board.GP27, None, False, 2),
    (board.GP26, board.GP27, None ), # will be set to global default
)
```

This setting is equivalent to `divisor` in the `rotaryio` module. The divisor of `1` for smooth encoders is not currently supported but you can use the divisor of `2` for them without issues and any noticeable difference.

Handler methods overrides

Encoder methods `on_move_do` and `on_button_do` can be overridden for complex use cases.

Connecting the encoder

Most EC11, EC12 and similar encoders have a common pinout shown below. For EVQWGD001 horizontal roller encoder, the pins are ordered `Pin A`, `Pin B`, `Ground`, and the fourth (furthest from the two switch pins) is not connected. This information is provided just for reference — always refer to the manufacturer datasheet for the correct pinout.

```
            +----------+
 Pin A   ----|          |
             |          |----- Ground
 Ground ----|          |
             |          |----- Switch Pin
 Pin B   ----|          |
            +----------+
```

In this configuration the encoder push button has its own dedicated pin. If the button should instead be a part of the matrix, it needs to be wired to Column and Row like other switches instead of ground.

Full example (with 1 encoder)

```
import board

from kmk.kmk_keyboard import KMKKeyboard
from kmk.consts import UnicodeMode
from kmk.keys import KC
from kmk.scanners import DiodeOrientation
from kmk.modules.layers import Layers
from kmk.modules.encoder import EncoderHandler


keyboard = KMKKeyboard()
layers = Layers()
encoder_handler = EncoderHandler()
keyboard.modules = [layers, encoder_handler]


keyboard.col_pins = (
    board.GP0, board.GP1, board.GP2, board.GP3, board.GP4, board.GP5,
    board.GP6, board.GP7, board.GP8, board.GP9, board.GP10, board.GP11,
    board.GP12, board.GP13,
)
keyboard.row_pins = (board.GP28, board.GP27, board.GP22, board.GP26,
board.GP21)
keyboard.diode_orientation = DiodeOrientation.COLUMNS

# I2C example
#import busio
#SDA = board.GP0
#SCL = board.GP1
#i2c = busio.I2C(SCL, SDA)
#encoder_handler.i2c = ((i2c, 0x36, False),)

# encoder_handler.divisor = 2 # for encoders with more precision
encoder_handler.pins = ((board.GP17, board.GP15, board.GP14, False),)

keyboard.tap_time = 250
keyboard.debug_enabled = False


# Filler keys
_______ = KC.TRNS
```

```
xxxxxxx = KC.NO
tbdtbd = KC.A


# Layers
LYR_STD, LYR_EXT, LYR_NUM, LYR_GAME = 0, 1, 2, 3

TO_STD = KC.DF(LYR_STD)
MT_EXT = KC.MO(LYR_EXT)
TO_NUM = KC.MO(LYR_NUM)
TO_GAME = KC.DF(LYR_GAME)


# Keymap

keyboard.keymap = [
    # Standard (ISO) Layer
    [
        KC.ESC , KC.N1  , KC.N2  , KC.N3  , KC.N4  , KC.N5  , KC.N6  , KC.N7
, KC.N8  , KC.N9  , KC.N0  , KC.MINS, KC.EQL , KC.BSPC,
        KC.TAB , KC.Q   , KC.W   , KC.E   , KC.R   , KC.T   , KC.Y   , KC.U
, KC.I   , KC.O   , KC.P   , KC.LBRC, KC.RBRC, KC.DEL ,
        xxxxxxx, KC.A   , KC.S   , KC.D   , KC.F   , KC.G   , KC.H   , KC.J
, KC.K   , KC.L   , KC.SCLN, KC.QUOT, KC.NUHS, xxxxxxx,
        KC.LSFT, KC.NUBS, KC.Z   , KC.X   , KC.C   , KC.V   , KC.B   , KC.N
, KC.M   , KC.COMM, KC.DOT , KC.SLSH, KC.UP  , KC.ENT ,
        KC.LCTL, KC.LGUI, xxxxxxx, KC.LALT, MT_EXT , xxxxxxx, KC.SPC ,
xxxxxxx, KC.RALT, TO_NUM , KC.RSFT, KC.LEFT, KC.DOWN, KC.RGHT,
    ],
    # Extra Keys Layer
    [
        TO_STD , KC.F1  , KC.F2  , KC.F3  , KC.F4  , KC.F5  , KC.F6  , KC.F7
, KC.F8  , KC.F9  , KC.F10 , KC.F11 , KC.F12 , KC.RESET,
        _______, KC.N1  , KC.N2  , KC.N3  , KC.N4  , KC.N5  , KC.N6  , KC.N7
, KC.N8  , KC.N9  , KC.N0  , KC.MINS, KC.EQL , _______,
        xxxxxxx, _______, _______, _______, _______, _______, _______,
_______, _______, _______, _______, _______, _______, xxxxxxx,
        KC.LSFT, _______, _______, _______, _______, _______, _______,
_______, _______, _______, _______, _______, KC.PGUP, _______,
        KC.LCTL, KC.LGUI, xxxxxxx, KC.LALT, MT_EXT , xxxxxxx, _______,
xxxxxxx, _______, TO_NUM , _______, KC.HOME, KC.PGDN, KC.END ,
    ],
    # NumPad Layer
    [
        TO_STD , xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, KC.P7
, KC.P8  , KC.P9  , KC.PSLS, xxxxxxx, xxxxxxx, KC.BSPC,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, KC.P4
, KC.P5  , KC.P6  , KC.PAST, xxxxxxx, xxxxxxx, KC.DEL ,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, KC.LPRN, KC.P1
, KC.P2  , KC.P3  , KC.PPLS, xxxxxxx, xxxxxxx, xxxxxxx,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, KC.RPRN, KC.P0
, KC.PDOT, _______, KC.PMNS, xxxxxxx, xxxxxxx, KC.PENT,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, MT_EXT , xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, TO_NUM , xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
    ],
    # Gaming Layer
    [
        TO_STD , xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
        xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx, MT_EXT , xxxxxxx, xxxxxxx,
xxxxxxx, xxxxxxx, TO_NUM , xxxxxxx, xxxxxxx, xxxxxxx, xxxxxxx,
    ],
]

# Rotary Encoder (1 encoder / 1 definition per layer)
encoder_handler.map = [ ((KC.UP, KC.DOWN, KC.MUTE),), # Standard
                        ((KC.VOLD, KC.VOLU, KC.MUTE),), # Extra
                        ((KC.A, KC.Z, KC.N1),), # NumPad not yet properly
configured
                        ((KC.A, KC.Z, KC.N1),), # Gaming not yet properly
configured
                        ]

if __name__ == "__main__":
    keyboard.go()
```