Encoders

Basic (EC11 compatible) encoders are supported by adding this to your `rules.mk`:

```
ENCODER_ENABLE = yes
```

and this to your `config.h`:

```
#define ENCODERS_PAD_A { B12 }
#define ENCODERS_PAD_B { B13 }
```

Each PAD_A/B variable defines an array so multiple encoders can be defined, e.g.:

```
#define ENCODERS_PAD_A { encoder1a, encoder2a }
#define ENCODERS_PAD_B { encoder1b, encoder2b }
```

If your encoder's clockwise directions are incorrect, you can swap the A & B pad definitions. They can also be flipped with a define:

```
#define ENCODER_DIRECTION_FLIP
```

Additionally, the resolution, which defines how many pulses the encoder registers between each detent, can be defined with:

```
#define ENCODER_RESOLUTION 4
```

It can also be defined per-encoder, by instead defining:

```
#define ENCODER_RESOLUTIONS { 4, 2 }
```

For 4× encoders you also can assign default position if encoder skips pulses when it changes direction. For example, if your encoder send high level on both pins by default, define this:

```
#define ENCODER_DEFAULT_POS 0x3
```

Split Keyboards

If you are using different pinouts for the encoders on each half of a split keyboard, you can define the pinout (and optionally, resolutions) for the right half like this:

```
#define ENCODERS_PAD_A_RIGHT { encoder1a, encoder2a }
#define ENCODERS_PAD_B_RIGHT { encoder1b, encoder2b }
#define ENCODER_RESOLUTIONS_RIGHT { 2, 4 }
```

If the `_RIGHT` definitions aren't specified in your `config.h`, then the non-`_RIGHT` versions will be applied to both sides of the split.

Additionally, if one side does not have an encoder, you can specify `{}` for the pins/resolution – for example, a split keyboard with only a right-side encoder:

```
#define ENCODERS_PAD_A { }
#define ENCODERS_PAD_B { }
#define ENCODER_RESOLUTIONS { }
#define ENCODERS_PAD_A_RIGHT { B12 }
#define ENCODERS_PAD_B_RIGHT { B13 }
#define ENCODER_RESOLUTIONS_RIGHT { 4 }
```

Keep in mind that whenver you change the encoder resolution, you will need to reflash the half that has the encoder affected by the change.

Encoder map

Encoder mapping may be added to your `keymap.c`, which replicates the normal keyswitch layer handling functionality, but with encoders. Add this to your keymap's `rules.mk`:

```
ENCODER_MAP_ENABLE = yes
```

Your `keymap.c` will then need an encoder mapping defined (for four layers and two encoders):

```
#if defined(ENCODER_MAP_ENABLE)
const uint16_t PROGMEM encoder_map[][NUM_ENCODERS][NUM_DIRECTIONS] = {
    [0] = { ENCODER_CCW_CW(KC_MS_WH_UP, KC_MS_WH_DOWN),
```

```
ENCODER_CCW_CW(KC_VOLD, KC_VOLU)  },
    [1] = { ENCODER_CCW_CW(RGB_HUD, RGB_HUI),
ENCODER_CCW_CW(RGB_SAD, RGB_SAI)  },
    [2] = { ENCODER_CCW_CW(RGB_VAD, RGB_VAI),
ENCODER_CCW_CW(RGB_SPD, RGB_SPI)  },
    [3] = { ENCODER_CCW_CW(RGB_RMOD, RGB_MOD),
ENCODER_CCW_CW(KC_RIGHT, KC_LEFT) },
};
#endif
```

This should only be enabled at the keymap level.

Using encoder mapping pumps events through the normal QMK keycode processing pipeline, resulting in a *keydown/keyup* combination pushed through `process_record_xxxxx()`. To configure the amount of time between the encoder “keyup” and “keydown”, you can add the following to your `config.h`:

```
#define ENCODER_MAP_KEY_DELAY 10
```

By default, the encoder map delay matches the value of `TAP_CODE_DELAY`.

Callbacks

Default Behaviour: all encoders installed will function as volume up (`KC_VOLU`) on clockwise rotation and volume down (`KC_VOLD`) on counter-clockwise rotation. If you do not wish to override this, no further configuration is necessary.

If you would like the alter the default behaviour, and are not using `ENCODER_MAP_ENABLE = yes`, the callback functions can be inserted into your `<keyboard>.c`:

```
bool encoder_update_kb(uint8_t index, bool clockwise) {
    if (!encoder_update_user(index, clockwise)) {
      return false; /* Don't process further events if user function exists
and returns false */
    }
    if (index == 0) { /* First encoder */
        if (clockwise) {
            tap_code(KC_PGDN);
        } else {
            tap_code(KC_PGUP);
        }
    } else if (index == 1) { /* Second encoder */
        if (clockwise) {
```

```
            rgb_matrix_increase_hue();
        } else {
            rgb_matrix_decrease_hue();
        }
    }
    return true;
}
```

or `keymap.c`:

```
bool encoder_update_user(uint8_t index, bool clockwise) {
    if (index == 0) { /* First encoder */
        if (clockwise) {
            tap_code(KC_PGDN);
        } else {
            tap_code(KC_PGUP);
        }
    } else if (index == 1) { /* Second encoder */
        if (clockwise) {
            rgb_matrix_increase_hue();
        } else {
            rgb_matrix_decrease_hue();
        }
    }
    return false;
}
```

If you return `true` in the keymap level `_user` function, it will allow the keyboard/core level encoder code to run on top of your own. Returning `false` will override the keyboard level function, if setup correctly. This is generally the safest option to avoid confusion.

Hardware

The A an B lines of the encoders should be wired directly to the MCU, and the C/common lines should be wired to ground.

Multiple Encoders

Multiple encoders may share pins so long as each encoder has a distinct pair of pins when the following conditions are met:

- using detent encoders

- pads must be high at the detent stability point which is called ‘default position’ in QMK
- no more than two encoders sharing a pin can be turned at the same time

For example you can support two encoders using only 3 pins like this

```
#define ENCODERS_PAD_A { B1, B1 }
#define ENCODERS_PAD_B { B2, B3 }
```

You could even support three encoders using only three pins (one per encoder) however in this configuration, rotating two encoders which share pins simultaneously will often generate incorrect output. For example:

```
#define ENCODERS_PAD_A { B1, B1, B2 }
#define ENCODERS_PAD_B { B2, B3, B3 }
```

Here rotating Encoder 0 `B1 B2` and Encoder 1 `B1 B3` could be interpreted as rotating Encoder 2 `B2 B3` or `B3 B2` depending on the timing. This may still be a useful configuration depending on your use case

---