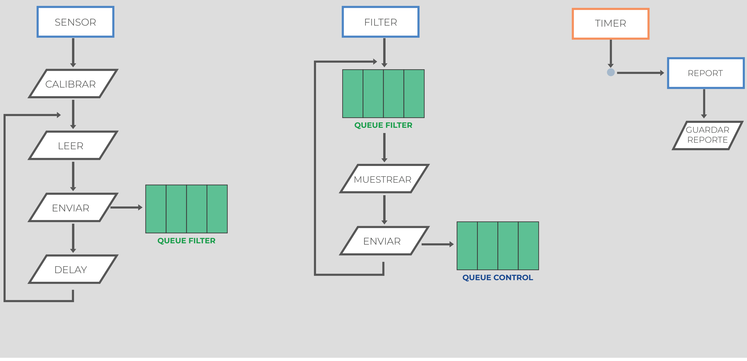

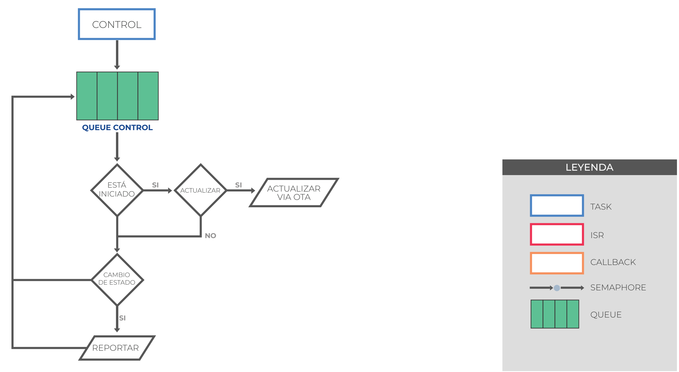

DESCRIPCION:

Dispositivo que permite detectar cuando una ventana pasa de abierta a cerrada o viceversa, con ayuda del sensor hall.

TABLA DE PARTICIONES:

```
# Name,   Type, SubType, Offset,   Size, Flags
# Note: if you have increased the bootloader size, make sure to update the offsets to avoid overlap
nvs,      data, nvs,     ,        0x4000,
otadata,  data, ota,     ,        0x2000,
phy_init, data, phy,     ,        0x1000,
factory,  app,  factory, ,        1M,
ota_0,    app,  ota_0,   ,        1M,
storage,  data, fat,     ,        1M,
```

ESTRUCTURA DEL PROYECTO:

```
window_detect git:master ❯ tree main
main
├── CMakeLists.txt
├── component.mk
├── controller.c
├── controller.h
├── filter.c
├── filter.h
├── Kconfig.projbuild
├── main.c
├── ota.c
├── ota.h
├── sensor.c
├── sensor.h
└── typedef.h

0 directories, 13 files
window_detect git:master ❯
```

Cabe destacar que ota.c es una copia de **native_ota_example.c** modificada ligeramente. Todos los demás archivos presentes y desarrollo se han realizado específicamente para este proyecto con una arquitectura escalable.

MENÚ DE CONFIGURACIÓN:

```
(Top) → OTA Configuration

(https://192.168.0.18:8070/firmware.bin) Firmware Upgrade URL
[ ] Skip server certificate CN fieldcheck
[ ] Skip firmware version check
(4) Number of the GPIO input for diagnostic
(5000) OTA Receive Timeout
```

Esta configuración viene heredada de **native_ota_example.c** aunque esta se ha renombrado a **OTA Configuration.**

```
(Top) → FIRMWARE Configuration

(15) Number of samples for filter sensor
(30) Range window detect
```

La configuración del firmware se ha creado para poder ajustar cuántas muestras desea que aplique el filtro para evitar el ruido y el **Range Window Detect,** para detectar cuando una ventana está abierta o cerrada, teniendo en cuenta que el proceso de calibración inicial ajusta el valor del sensor Hall a cero y sin importar la dirección del imán, la distancia siempre será la misma.

FORMATO DE REPORTE

```
37  void report_state_window(window_t * w, const char * from)
38  {
39          ESP_LOGI(TAG, "%s: n %04d: window is open: %d, t0: %d, t1: %d, tf: %d",
40                           from,
41                           w->n_close,
42                           w->open,
43                           w->t0,
44                           w->t1,
45                           w->tf);
46
47          int err = fprintf(fd, "n: %d, open: %d, t0: %d, t1: %d, tf: %d\n",
48                                                          w->n_close,
49                                                          w->open,
50                                                          w->t0,
51                                                          w->t1,
52                                                          w->tf);
53          if (err < 0)
54          {
55                  ESP_LOGE(TAG, "%s: write dump.log %s", from, strerror(errno));
56          }
57  }
```

Como se presenta en la imagen anterior, es la función de ayuda utilizada para generar un reporte, la cual siempre sigue estos pasos:

1. Notificarlo por la salida estándar
2. Escribirlo en el almacenamiento permanente con el nombre dump.log con un formato fácilmente parseable, para su tratamiento y análisis a futuro.
3. Si en el paso 2 se detecta un error, notificarlo.

CÓDIGO FUENTE:

- **https://github.com/formatcom/window_detect**