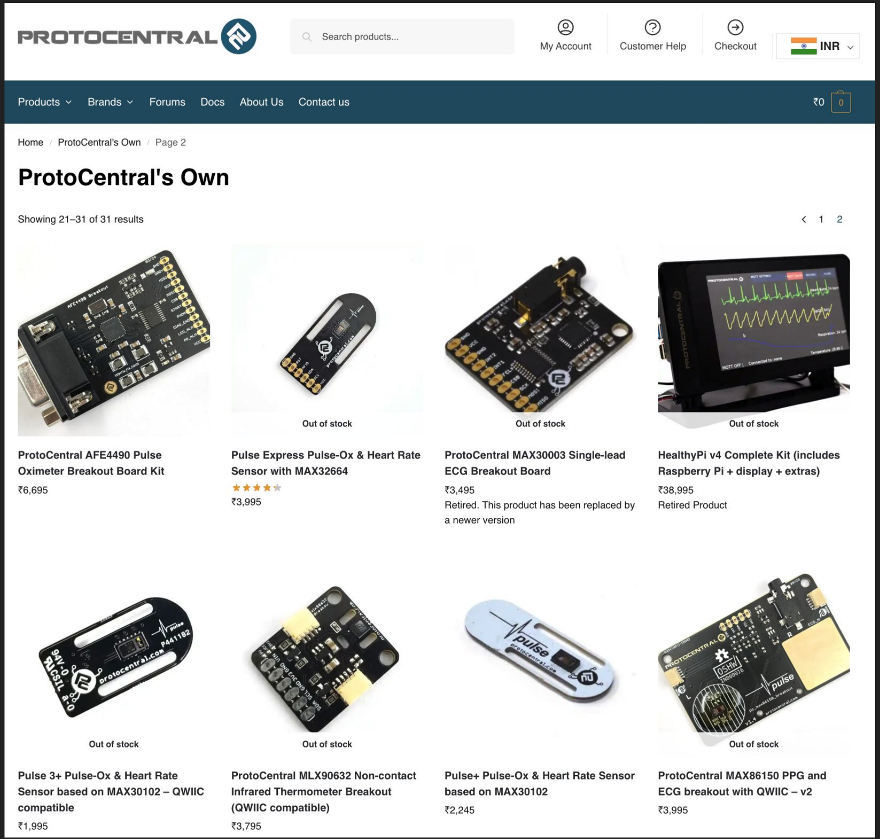
Zephyr and Open Source Health Devices

Ashwin Whitchurch
Protocentral Electronics

ashwin@protocentral.com

About me

- “A hand in every Pie” at ProtoCentral Electronics, otherwise known as CEO
- Primarily an electronics designer for medical electronic products
- Sometimes also a firmware engineer, and other times product designer
- Still fascinated by how what we code controls electrons on a wire
- Zephyr fan since 1.14.0



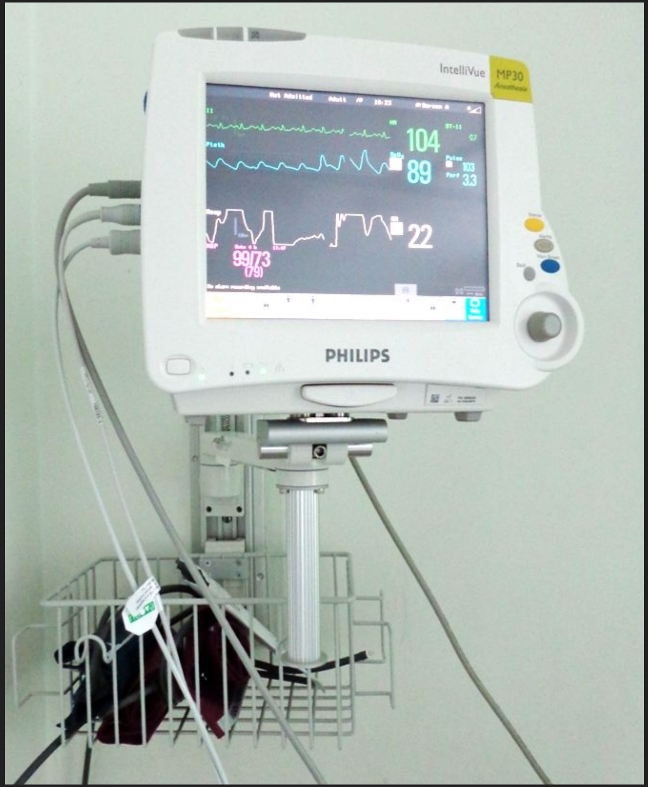
What and why: Open Source Source health devices

Why develop Open Source Health Devices

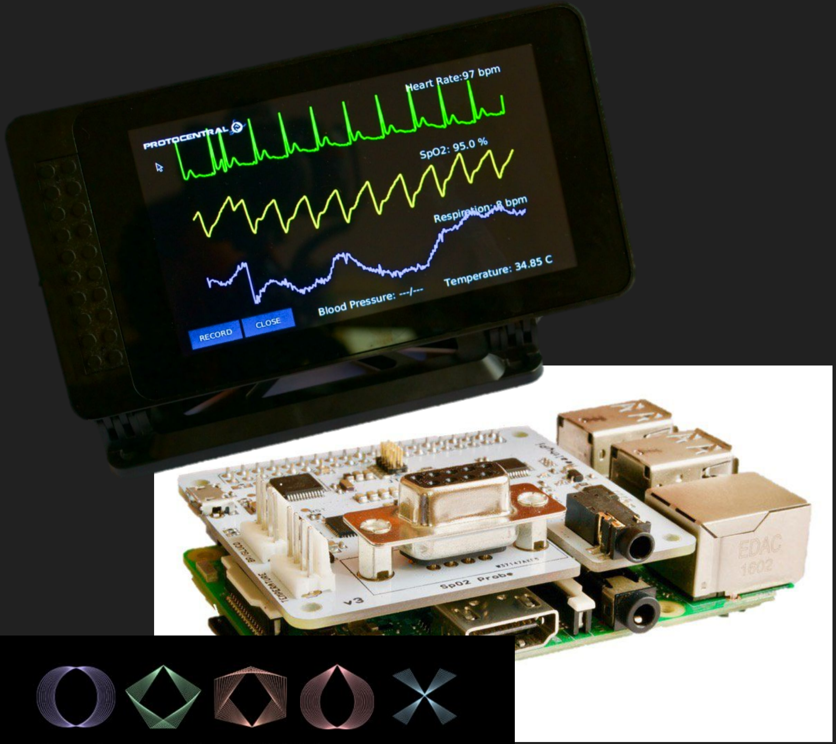
- Improved access to designs without IP restrictions. Localise manufacturing with open designs
- Build secure systems (since all the vulnerabilities can be identified) - anyone can Audit a system
- Collaboration results in faster improvements
- Baseline design decisions allow end-users & manufacturers to build for local regulations
- Accelerates prototyping and academic research
- Make a device work on your terms rather than the other way around

What is a patient/vital signs monitor?

- Monitor vital signs such as ECG, PPG/SpO2, respiration, temperature, other stuff depending on indications
- Reminds you of when you were sick and at the hospital



In the beginning…



HackadayPrize2017

- Open source vital signs/patient monitor with these parameters:
  - ECG (Single lead)
  - Respiration (Impedance Pneumography)
  - SpO2 (Blood Oxygen)
  - Temperature
- Designed as a HAT for Raspberry Pi that uses the RPi as the host computer with display
- Based on the SAMD21 running Arduino
- Tool for medical research and development and a reference design for low cost open medical devices for the resource constrained

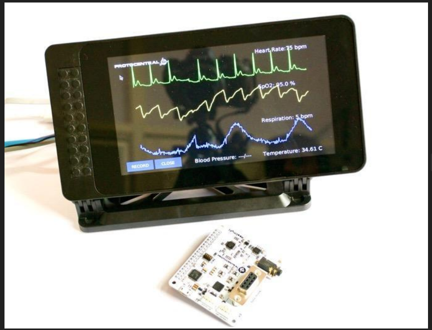
They don’t have to be scary or a “black box”

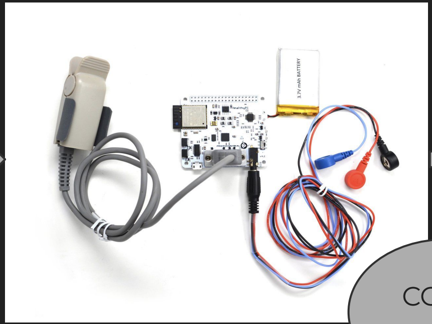
Are open source health devices practical?

National Science Foundation (NSF) POSE grant to OSHWA to foster an Open Healthware Ecosystem.

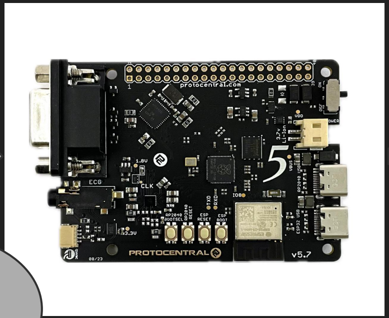
Source: Open Source Harware Association
https://oshwa.org/announcements/session-1-open-healthware-strategy-and-vision-takeaways-and-reflections/

HealthyPi : Evolution





COVID 19



Rev 3 (2017)

- HAT for Raspberry Pi
- USB CDC data output
- **First Open source patient monitor**
- Based on SAMD21

Rev 4 (2019)

- Standalone capability
- Added WiFi/Bluetooth support
- Based on ESP32
- Mobile app for Android

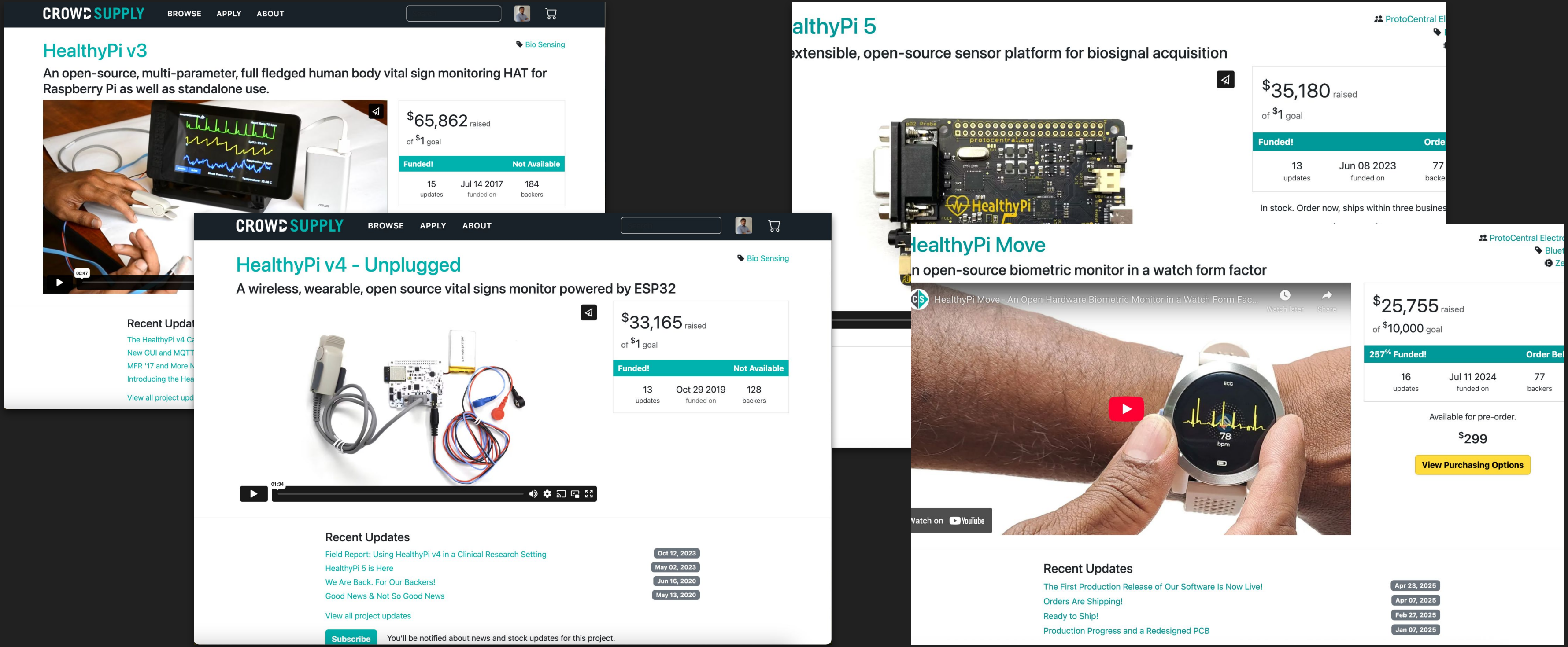
Rev 5 (2023)

- Complete redesign
- Based on RP2040+ESP32C3
- Datalogging capabilities

Healthy Pi Move (2024)

- Smartwatch form factor
- Low Power
- Based on nRF5340 & Zephyr

Scaling HealthyPi - Crowdfunding

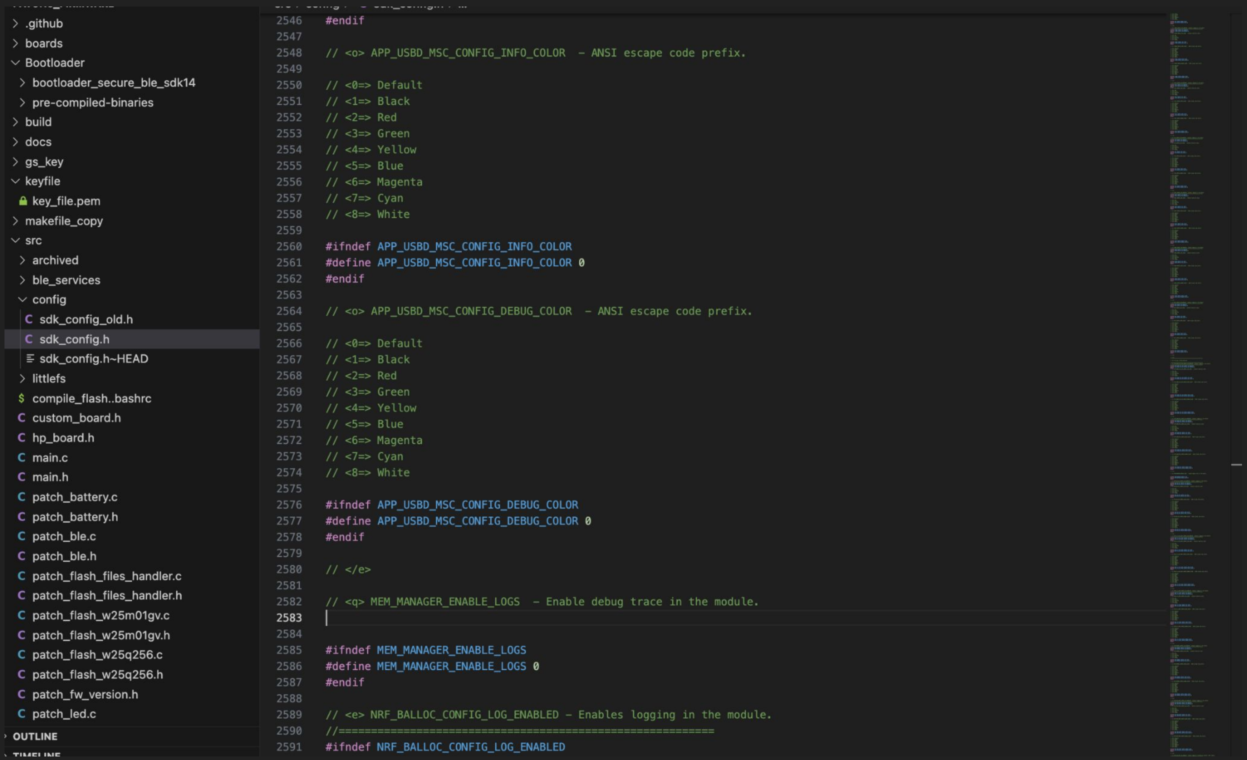


Where does Zephyr fit in?

Before Zephyr

- nRF 5 SDK
- Atmel SDK
- STM32 SDK
- FreeRTOS
- Raspberry Pi SDK
- Arduino (prototyping)

"The coder walks a narrow trail, each driver handmade, each feature alone in the forest."



From nRF5 SDK to nRF Connect SDK

Aspect	nRF5 SDK	nRF Connect SDK (NCS)
Architecture	Monolithic (architecture only)	Modular (output maybe a monolithic binary)
BLE Stack	Proprietary SoftDevice	Open-source Zephyr Bluetooth stack
OS Support	Bare-metal / Simple schedulers	Zephyr RTOS with full thread support
Extensibility	Limited	Highly extensible
Ecosystem	- -	Upstream Zephyr + community-wide

Zen and the Art of Firmware Evolution

Migration Stage	Control 🧠	Simplicity 🧺	Portability 🚀	BLE Stack
nRF5 SDK	Full	Low	Low	SoftDevice
nRF Connect SDK	Medium	High	Medium	Zephyr (forked)
Zephyr (main)	Full	Medium	High	Zephyr BLE

Unified by Zephyr: One Codebase, Many Platforms

HealthyPi 5 RP2040+ ESP32	HealthyPi Move nRF5340	HealthyPi 6 (In development) STM32H7 ARM CM7/CM4 ESP32	HealthyPi EEG (Future)	Future versions

Platform Specific Overlays (Available data sources, screen size, storage modes, memory layouts

Algorithms & Signal Processing (Zephyr Libraries)

HealthyPi Zephyr Drivers

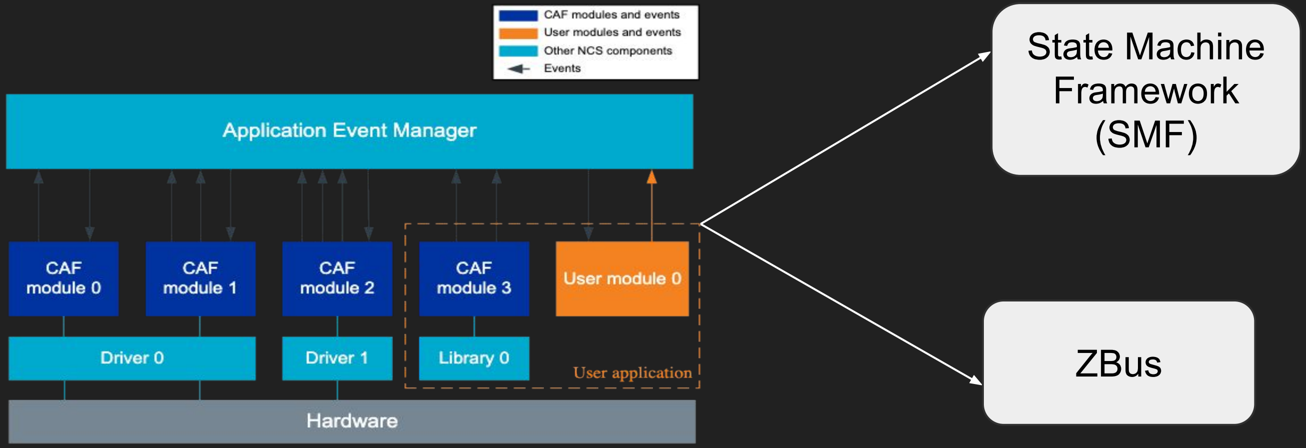
HealthyPi Zephyr Codebase (Skeleton services, tasks, work modules)

Development experience with HealthyPi Move

- **Wrist-wearable** device with 390x390 Full color OLED display
- **nRF5340** microcontroller with BLE
- Onboard 128 MB NOR Flash
- Parameters measured
  - ECG and Heart rate
  - Fingertip PPG and SpO2
  - Heart-rate Variability
  - Blood Pressure Trending (from PPG)
  - EDA/GSR
  - Skin Temperature
- Completely open sourced

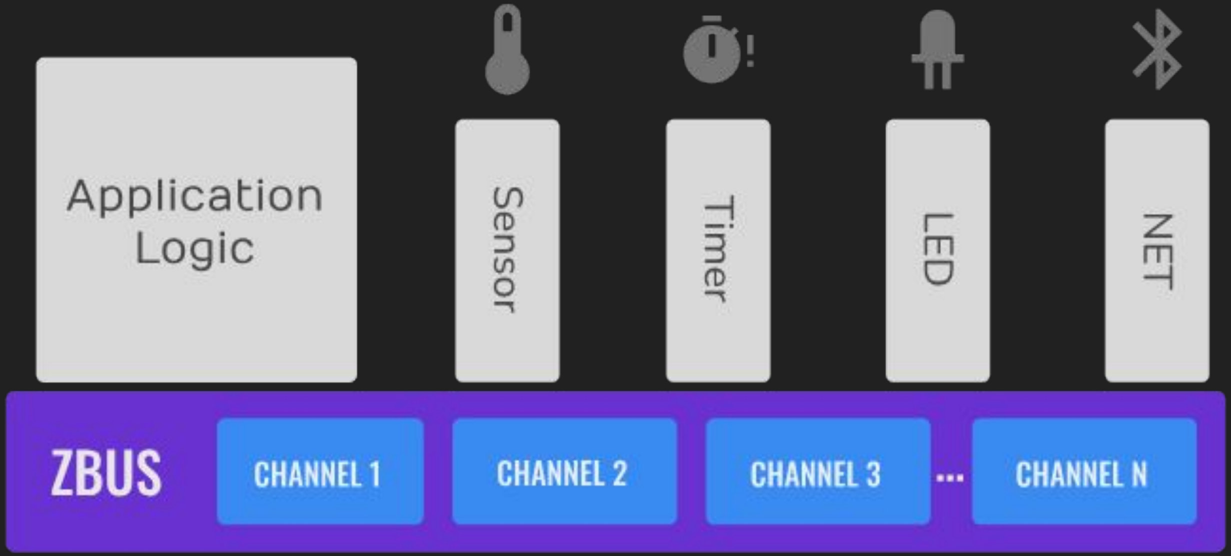
Migrating from nRF Connect to mainstream Zephyr

- Replace Nordic-centered implementations to platform-neutral methods
  - Common Application framework (CAF)
  - Vendor-specific Bluetooth functions
  - nRF bootloaders to MCUBoot



Source: Nordic Semiconductor
https://docs.nordicsemi.com/bundle/ncs-latest/page/nrf/libraries/caf/caf_overview.html

Event management with ZBus



Source: Zephyr Project Docs
https://docs.zephyrproject.org/latest/services/zbus/index.html

Event management with ZBus

```c
ZBUS_CHAN_DEFINE(ecg_timer_chan, /* Name */
                 struct hpi_ecg_timer_t,
                 NULL, /* Validator */
                 NULL, /* User Data */
                 ZBUS_OBSERVERS(disp_ecg_timer_lis),
                 ZBUS_MSG_INIT(0) /* Initial value {0} */
);

ZBUS_CHAN_DEFINE(ecg_hr_chan, /* Name */
                 uint16_t,
                 NULL, /* Validator */
                 NULL, /* User Data */
                 ZBUS_OBSERVERS(disp_ecg_hr_lis),
                 ZBUS_MSG_INIT(0) /* Initial value {0} */
);
```

Channel definitions

```c
static void disp_ecg_timer_listener(const struct zbus_channel *chan)
{
    const struct hpi_ecg_timer_t *ecg_timer = zbus_chan_const_msg(chan);
    m_disp_ecg_timer = ecg_timer->timer_val;
}
ZBUS_LISTENER_DEFINE(disp_ecg_timer_lis, disp_ecg_timer_listener);

static void disp_ecg_hr_listener(const struct zbus_channel *chan)
{
    const uint16_t *ecg_hr = zbus_chan_const_msg(chan);
    m_disp_ecg_hr = *ecg_hr;
    LOG_DBG("ZB ECG HR: %d", *ecg_hr);
}
ZBUS_LISTENER_DEFINE(disp_ecg_hr_lis, disp_ecg_hr_listener);
```

Listeners

```c
uint16_t ecg_hr = ecg_bioz_sensor_sample.hr;
zbus_chan_pub(&ecg_hr_chan, &ecg_hr, K_SECONDS(1));
```

Publish updates

State Management with SMF

State Management with SMF

```c
static const struct smf_state display_states[] = {
    [HPI_DISPLAY_STATE_INIT] = SMF_CREATE_STATE(st_display_init_entry, NULL, NULL, NULL, NULL),
    [HPI_DISPLAY_STATE_SPLASH] = SMF_CREATE_STATE(st_display_splash_entry, st_display_splash_ru
    [HPI_DISPLAY_STATE_BOOT] = SMF_CREATE_STATE(st_display_boot_entry, st_display_boot_run, st_

    [HPI_DISPLAY_STATE_SCR_PROGRESS] = SMF_CREATE_STATE(st_display_progress_entry, st_display_p
    [HPI_DISPLAY_STATE_ACTIVE] = SMF_CREATE_STATE(st_display_active_entry, st_display_active_ru
    [HPI_DISPLAY_STATE_SLEEP] = SMF_CREATE_STATE(st_display_sleep_entry, st_display_sleep_run,
    [HPI_DISPLAY_STATE_ON] = SMF_CREATE_STATE(st_display_on_entry, NULL, NULL, NULL, NULL),
};
```

State Machine Definition

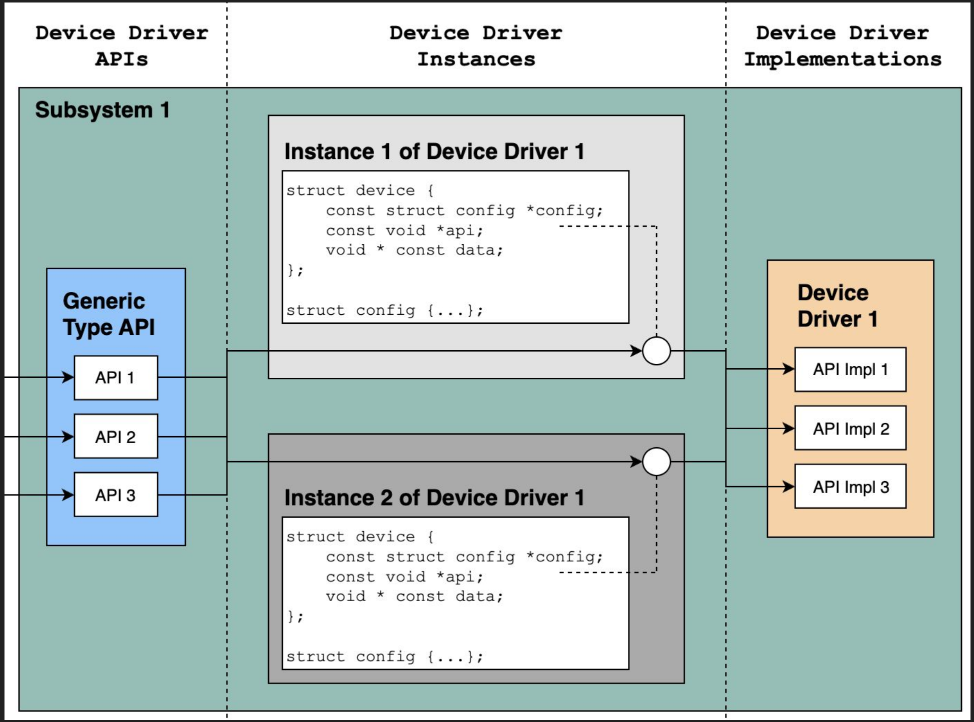
```c
smf_set_initial(SMF_CTX(&s_disp_obj), &display_states[HPI_DISPLAY_STATE_INIT]);

for (;;)
{
    ret = smf_run_state(SMF_CTX(&s_disp_obj));
    if (ret != 0)
    {
        LOG_ERR("SMF Run error: %d", ret);
        break;
    }
    k_msleep(lv_task_handler());
}
```

Running the SM

```c
static void st_display_sleep_run(void *o)
{
    int inactivity_time = lv_disp_get_inactive_time(NULL);
    // LOG_DBG("Inactivity Time: %d", inactivity_time);
    if (inactivity_time < DISP_SLEEP_TIME_MS)
    {
        // hpi_display_sleep_on();
        smf_set_state(SMF_CTX(&s_disp_obj), &display_states[HPI_DISPLAY_STATE_ACTIVE])
    }
}

static void st_display_sleep_exit(void *o)
{
    LOG_DBG("Display SM Sleep Exit");
    hpi_display_sleep_off();
}

static void st_display_on_entry(void *o)
{
    LOG_DBG("Display SM On Entry");
}
```

Define and transitions between states

Device Drivers - driver model



Source: Zephyr Project Docs
https://docs.zephyrproject.org/latest/kernel/drivers/index.html

Sensor Driver API

FETCH & GET

How It Works:

- `sensor_sample_fetch(dev)`
 → Initiates a new sensor reading (fetches data into a buffer).

- `sensor_channel_get(dev, SENSOR_CHAN_*, &value)`
 → Retrieves the value of a specific sensor channel from the last fetch.

Conceptual Flow:

- `fetch()` → talks to hardware
- `get_*()` → reads from internal cache/buffer

READ & DECODE

How It Works:

- `read()` → Reads raw bytes from the device (e.g., over I2C or SPI)

- `decode()` → Converts raw data into physical values

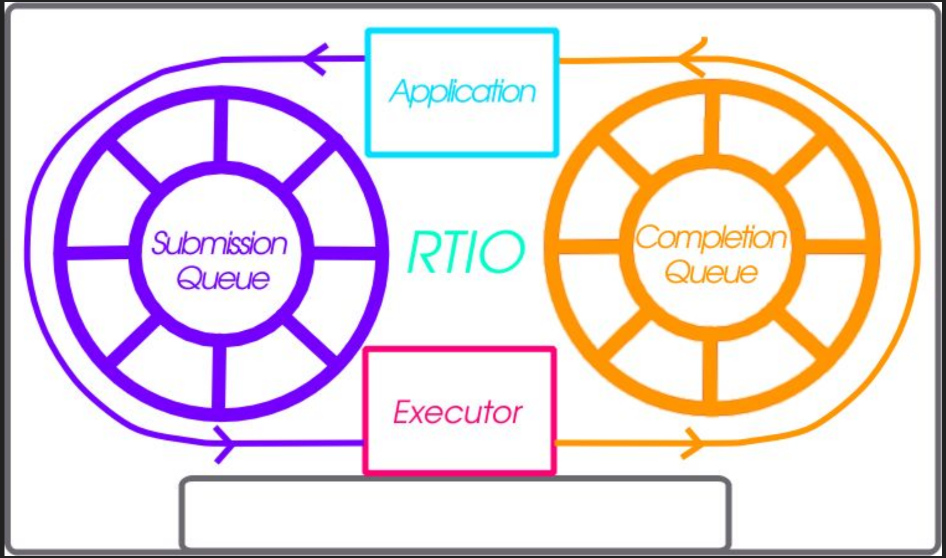
Conceptual Flow:

- `read()` → hardware data acquisition (often raw)
- `decode()` → interpretation/parsing logic (driver or app-level)

Writing Sensor Drivers

Used read and decode for most

- High Speed Sensors do not perform well with fetch and get
- Access to raw sensor data - some types are not encodable into the sensor_value type
- Supports sensors with built-in FIFOs which give multiple samples in one burst and require custom decoding
- Submission and completion can be sync or async (callback-based)



Uses RTIO for asynchronous reads

Implementing Read and Decode in our drivers

```c
int max32664d_submit(const struct device *dev, struct rtio_iodev_sqe *iodev_sqe)
{
    uint32_t min_buf_len = sizeof(struct max32664d_encoded_data);
    int rc;
    uint8_t *buf;
    uint32_t buf_len;

    struct max32664d_encoded_data *edata;
    struct max32664_enc_calib_data *calib_data;
    struct max32664d_data *data = dev->data;

    /* Get the buffer for the frame, it may be allocated dynamically by the rtio context */
    rc = rtio_sqe_rx_buf(iodev_sqe, min_buf_len, min_buf_len, &buf, &buf_len);
    if (rc != 0)
    {
        LOG_ERR("Failed to get a read buffer of size %u bytes", min_buf_len);
        rtio_iodev_sqe_err(iodev_sqe, rc);
        return rc;
    }

    if ((data->op_mode == MAX32664D_OP_MODE_BPT_EST) || (data->op_mode == MAX32664D_OP_MODE_R
    {
        edata = (struct max32664d_encoded_data *)buf;
        edata->header.timestamp = k_ticks_to_ns_floor64(k_uptime_ticks());
        rc = max32664_async_sample_fetch(dev, edata->ir_samples, edata->red_samples, &edata->
                                         &edata->hr, &edata->bpt_status, &edata->bpt_progress
    }
```

Driver - Handle an RTIO request

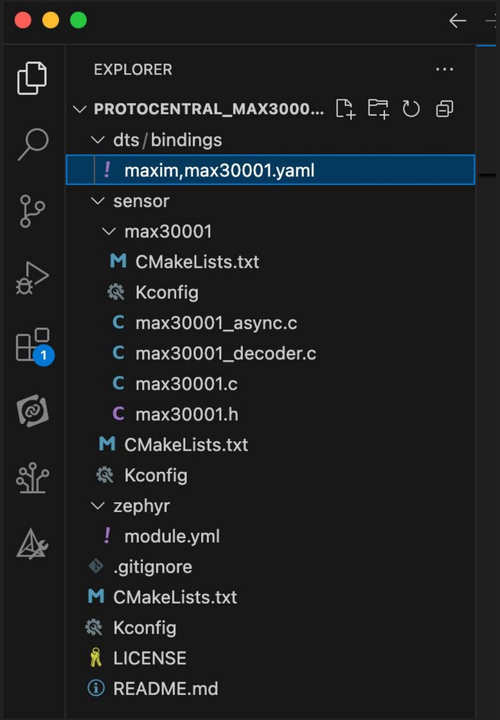
```c
void work_fi_sample_handler(struct k_work *work)
{
    uint8_t data_buf[384];

    int ret = 0;
    ret = sensor_read(&max32664d_iodev, &max32664d_read_rtio_poll_ctx, data_buf, sizeof(data_
    if (ret < 0)
    {
        LOG_ERR("Error reading sensor data");
        return;
    }
    sensor_ppg_finger_decode(data_buf, sizeof(data_buf), sens_decode_ppg_fi_op_mode);
}
K_WORK_DEFINE(work_fi_sample, work_fi_sample_handler);

static void ppg_fi_sampling_handler(struct k_timer *timer_id)
{
    k_work_submit(&work_fi_sample);
}
```

App - Submit an RTIO request and wait for data (blocking or callback based

```c
static void sensor_ppg_finger_decode(uint8_t *buf, uint32_t buf_len, uint8_t m_ppg_op_mode)
{
    const struct max32664d_encoded_data *edata = (const struct max32664d_encoded_data *)buf;
    struct hpi_ppg_fi_data_t ppg_sensor_sample;

    uint16_t _n_samples = edata->num_samples;
    //printk("FNS: %d ", edata->num_samples);
    if (_n_samples > 16)
    {
        _n_samples = 16;
    }

    if (_n_samples > 0)
    {
        ppg_sensor_sample.ppg_num_samples = _n_samples;

        for (int i = 0; i < _n_samples; i++)
        {
            ppg_sensor_sample.raw_red[i] = edata->red_samples[i];
```

App - Directly decode the data

Portability - Case in point: Drivers



Driver module (repo)

```yaml
manifest:
    self:
        west-commands: scripts/west-commands.yml

    remotes:
        - name: zephyrproject-rtos
        url-base: https://github.com/zephyrproject-rtos

    projects:
        - name: zephyr
            remote: zephyrproject-rtos
            revision: v3.5-branch
        - name: MAX30001
            path: modules/protocentral_max30001
            revision: main
            url: https://github.com/Protocentral/protocentr
```

Add to Project's west.yml

```
CONFIG_SENSOR=y
CONFIG_MAX30001=y
```

Enable Driver in prj.conf

```
&spi3 {
        compatible = "nordic,nrf-spim";
        status = "okay";

        pinctrl-0 = <&spi3_default>;
        pinctrl-1 = <&spi3_sleep>;
        pinctrl-names = "default", "sleep";
        cs-gpios = <&gpio1 10 GPIO_ACTIVE_LOW>;

    max30001: max30001@0 {
                compatible = "maxim,max30001";
                status = "okay";
                reg = <0x0>;
                spi-max-frequency = <DT_FREQ_M(1)>;
                intb-gpios = <&gpio1 9 GPIO_ACTIVE_LOW>;
                rtor-enabled;
                ecg-enabled;
                bioz-enabled;
                ecg-gain = <3>;
                ecg-invert;
        };
};
```

Add device to devicetree (DTS) or DTS overlay

Source: https://github.com/Protocentral/protocentral_max30001_zephyr_driver

Portability - Case in Point: Libraries

```
EXPLORER
∨ PC_MAX32664_UPDATER
  > msbl
  ∨ zephyr
    ! module.yml
  .gitignore
  CMakeLists.txt
  Kconfig
  LICENSE
  max32664_updater.c
  max32664_updater.h
  max32664d_bl.c
  README.md
```

Library as module (repo)

```yaml
- name: max32664_updater_zephyr
  remote: protocentral
  repo-path: pc_max32664_updater_zephyr
  revision: main
  path: app/lib/max32664_updater
```

Add to Project’s west.yml

```
CONFIG_MAX32664_UPDATER=y
CONFIG_MAX32664_UPDATER_LOG_LEVEL_DBG=y
```

Enable Library in prj.conf

```c
if ((ver_get.val1 < hpi_max32664c_req_ver.major) 0    Generate Copilot summary    x32664
{
    LOG_INF("MAX32664C App update required");
    hw_add_boot_msg("\tUpdate required", false, false, false, 0);
    k_sem_give(&sem_boot_update_req);
    max32664_updater_start(max32664c_dev, MAX32664_UPDATER_DEV_TYPE_MAX32664C);
}
```

Call library directly from your code

HealthyPi firmware architecture

HealthyPi 5

RP2040+ ESP32

HealthyPi Move

nRF5340

HealthyPi 6 (In development)

STM32H7 ARM CM7/CM4 ESP32

HealthyPi EEG (Future)

Future versions

Platform Specific Overlays (Available data sources, screen size, storage modes, memory layouts

Algorithms & Signal Processing (Zephyr Libraries)

HealthyPi Zephyr Drivers

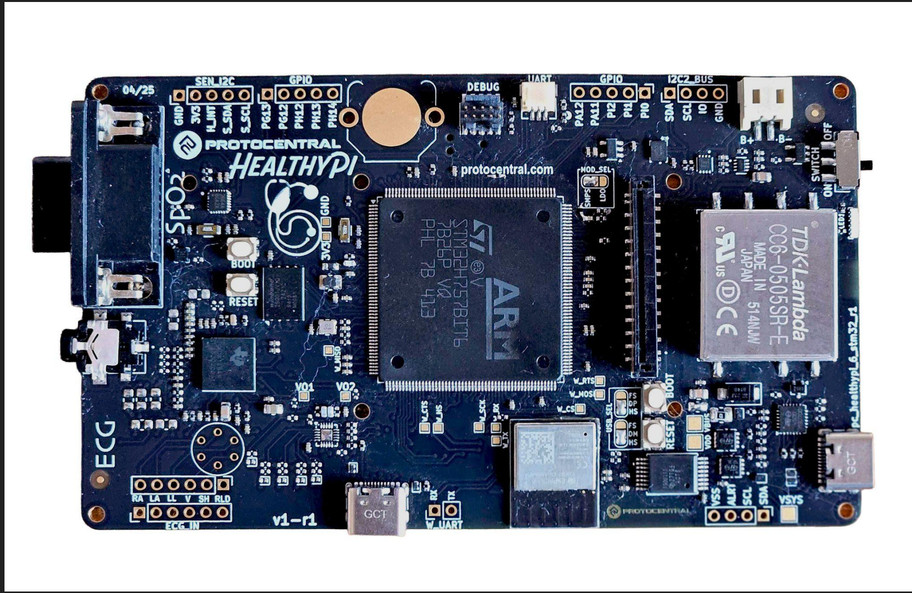
HealthyPi Zephyr Codebase (Skeleton services, tasks, work modules)

Other great features that I use

- Code relocation (whether for XIP in flash or display framebuffers in external SDRAM)
- LittleFS filesystem on external flash
- MCUBoot and the SMP Bluetooth service for device firmware updates (DFU)
- Security - support for Crypto and “root of trust”
- LLEXT (Loadable modules) - still exploring
-

Product in development: HealthyPi 6

- Main MCU: STM32H757 (ARM Cortex M7+M4)
- Wireless: ESP32C6 (connected through SPI)
- 32 MBytes External SDRAM
- 128 MBytes External NOR Flash
- 5-inch 1280x720 LCD display with MIPI-DSI interface
- Sensors
    - ADS1294R 5-lead ECG + Respiration
    - MAX32664A fingertip SpO2
    - Digital temperature sensor
    - Other extensible sensors based on user application



Conclusions

- Arduino removed the fear of starting. Zephyr removes the fear of scaling.
- Multi-platforms, one codebase, unified by Zephyr
- You decide which features you want to implement

Thank You

Questions?

ashwin@protocentral.com