Fast DDS (XRCE) Installation on PineTime64 smart watch

Assignment 2

Heterogeneous Computing (COE 504)

Submitted by

MD Siddiqur Rahman Tanveer (PhD Student, CoE, 202417180)

F M Jahiduzzaman (Masters Student, CoE, 202417000)

Department of Computer Engineering

King Fahd University of Petroleum and Minerals (KFUPM)

**Fast DDS (XRCE)**

Fast DDS XRCE (also known **as eProsima Fast DDS-XRCE**) is an implementation of the DDS-XRCE (DDS for Extremely Resource Constrained Environments) protocol, part of the Data Distribution Service (DDS) family of standards maintained by the Object Management Group (OMG) [1]. The aim of the DDS-XRCE protocol is to provide access to the DDS Global-Data-Space from resource-constrained devices. This is achieved thanks to a client-server architecture, where low resource devices, called *XRCE Clients*, are connected to a server, called *XRCE Agent*, which acts on behalf of its clients in the DDS Global-Data-Space.

The eprosima Micro XRCE-DDS library implements a client-server protocol that enables resource-constrained devices (clients) to take part in DDS communications. The eProsima Micro XRCE-DDS Agent (server) acts as a bridge to make this communication possible. It acts on behalf of the Micro XRCE-DDS Clients by enabling them to take part to the DDS Global Data Space as DDS publishers and/or subscribers. It also allows for Remote Procedure Calls, as defined by the [DDS-RPC standard](https://www.omg.org/spec/DDS-RPC/About-DDS-RPC/), which implement a request/reply communication pattern.

eProsima Micro XRCE-DDS provides both a plug and play eProsima Micro XRCE-DDS Agent and an API layer which allows the user to implement the eProsimaMicro XRCE-DDS Clients.

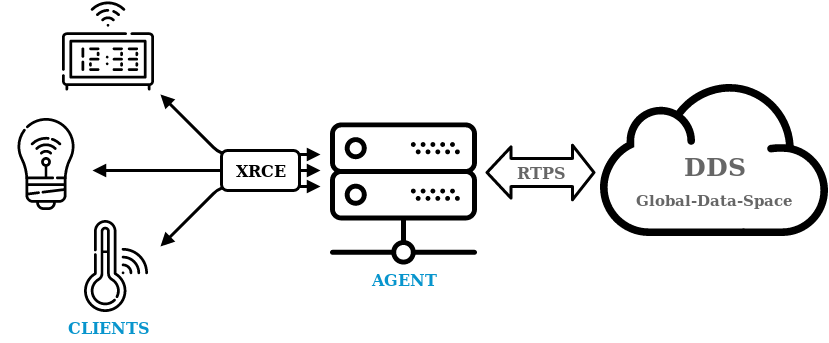


Fig. 1. Micro XRCE-DDS Architecture [1].

It is designed to allow devices with very limited resources, such as IoT sensors, microcontrollers, or embedded systems, to communicate in a DDS network. **Fast DDS-XRCE** serves as a middleware that enables data exchange between highly resource-constrained devices (clients) and full DDS networks (servers or agents), thus making DDS more accessible to a wider range of devices.

**Key Features:**

1. **Client-Server Architecture**: XRCE is built on a client-server model. The client is a lightweight entity on a resource-constrained device, and the server (or agent) acts as an intermediary between the client and the DDS network.
2. **Interoperability with DDS**: XRCE clients can publish or subscribe to data in a DDS network through an XRCE agent, bridging the gap between DDS-enabled systems and resource-constrained devices.
3. **Small Footprint**: It is optimized for environments with limited computational power, memory, and network bandwidth.
4. **Real-Time Communication**: Like DDS, Fast DDS XRCE supports real-time communication with a focus on low latency and high reliability.
5. **Wide Range of Use Cases**: It can be used in various fields such as industrial automation, robotics, automotive systems, and IoT applications, where small devices need to participate in complex communication networks.
6. **Customization**: Fast DDS-XRCE provides different Quality of Service (QoS) settings and can be tailored to specific use cases in constrained environments.

In essence, Fast DDS XRCE is a lightweight extension of the DDS protocol, enabling constrained devices to become part of larger, real-time data communication systems, maintaining compatibility with the broader DDS ecosystem.

Now, we would like to use XRCE DDS on day to day life IoT device such as PineTime smart watch.

**PineTime**

The PineTime [2] is a free and open source smartwatch capable of running custom-built open operating systems. Some of the notable features include a heart rate monitor, a week-long battery, and a capacitive touch IPS display that is legible in direct sunlight. It is a fully community driven side-project which anyone can contribute to, allowing you to keep control of your device.

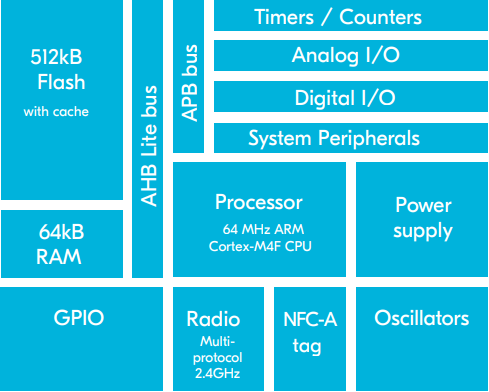


Fig. 2. PineTime64 [2] watch and SoC architecture of nRF52832 [3].

The smart watch is based **nRF52832** chip. That’s SoC [3] architecture is shown in Fig.2. The nRF52832 takes Bluetooth Low Energy SoCs to the next level with its support for Bluetooth 5. It has an ARM Cortex M4 CPU at its heart, running at 64 MHz, it is capable of handling demanding application and communication tasks in a short timeframe. This frees up the CPU handle even more tasks, or to return to sleep mode, thus conserving precious battery energy. The nRF52832, and all SoCs in the nRF52 Series family are flash-based SoCs and are ideal for Device Firmware Updates (DFU). DFU brings total flexibility and control to the firmware running in your product. Enabling new security updates, bugfixing and feature additions in the field.

**Features and Specification**

|  |  |
| --- | --- |
| Display | Square 1.3-inch 240×240 IPS capacitive touch display |
| SoC | Low-power Nordic Semiconductor nRF52832 |
| 64 MHz + Floating Point |
| Software | Any open-source operating systems built on top of   numerous RTOSes |
| Body | Dimensions: 37.5mm x 40mm x 11mm |
| Weight: 38 grams |
| Made with Zinc Alloy and Plastic |
| Dustproof and water-resistant up to 1m (Rated at IP67) |
| Health Tracking | Step Counting (with Accelerometer) |
| Heart Rate Detection |
| Notification features | Notification access |
| Wrist vibration |
| Quick glance via lift-to-wake. |
| Connectivity | Bluetooth 5 and Bluetooth Low Energy |
| Compatible with almost any device |
| Over-the-air update |
| Storage | 4 MB of User Storage |
| 0.5 MB of OS Storage |
| Battery | All-week 180 mAh battery |
| 2-pin USB charging dock |

**IPS touch display**

The PineTime includes a color IPS capacitive touch display

**Dustproof and Water Resistant**

The PineTime comes with a IP67 rating, meaning that it can handle up to one meter of water and is fully dust proof

**Bluetooth 5**

With Bluetooth 5 and Bluetooth Low Energy, the PineTime can pair to other devices

**Heartrate and Step Counting**

With a built-in accelerometer and heart-rate sensor, the PineTime can help you keep track of your health

Since this smart watch has been designed as a very resource constrained manner for lightweight power consumption but to perform suitable tasks on a single chip architecture. That’s why it’s our key interest to implement XRCE DDS on this deice.

**Environment Setup**

Since the smartwatch contains **nRF52832** chip, we need a programmer to upload the firmware of this chip. To do this task we have chosen ESP32 chip as a programmer with SWD interface from Arduino IDE.

**Required Hardware**

* ESP32 WROOM S2 chip
* Bread Board
* PineTime64 smartwatch
* USB Cable
* Jumper wires
* Cables
* LED

**Required Software**

* Arduino IDE <https://www.arduino.cc/>
* ESP32 core Library <https://github.com/espressif/arduino-esp32>
* This version of the WifiManager

<https://github.com/tzapu/WiFiManager/tree/feature_asyncwebserver>

* AsyncTCP <https://github.com/me-no-dev/AsyncTCP>
* ESPAsyncWebServer <https://github.com/me-no-dev/ESPAsyncWebServer>
* ArduinoBufferedStreams <https://github.com/paulo-raca/ArduinoBufferedStreams>

**Arduino IDE Setup and preparation**:

We have downloaded the Arduino IDE and complete the installation

* Copy or install the three downloaded libraries (AsyncTCP, ESPAsyncWebServer, WifiManager) into the Arduino > libraries directory
* Arduino > libraries: Rename:
* AsyncTCP-master to AsyncTCP
* ESPAsyncWebServer-master to ESPAsyncWebServer
* WiFiManager-master to WiFiManager

**Installing libraries for ESP32 chip:**

Now to add the ESP32 Core to Arduino

* File > Additional Boards Manager URLS >

<https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>

**ESP 32 SWD Programmer Interface preparation**

We have followed this github (<https://github.com/atc1441/ESP32_nRF52_SWD>) tutorial for this preparation. We downloaded the repository and open the "ESP32\_SWD\_WIFI.ino" file with Arduino and select the "DOIT ESP32 DEVKIT V1" (Tools > Board: > DOIT ESP32 DEVKIT V1)

While compiling the sketch in IDE there were several issues found in the program that we had to fix. We had to define the pins for the ESP32 module perform as a programmer.

#define LED 2

#define GLITCHER 5

#define NRF\_POWER 22

#define OSCI\_PIN 34

#define swd\_clock\_pin 21

#define swd\_data\_pin 19

The repository didn’t include the WiFiManager.h header file rather it used wifimanager.h and wifimanager.cpp files which made the ambiguity so that it was causing compilation error. We downloaded the WiFiManager.h and WiFiManager.cpp corresponding files from the git (<https://github.com/tzapu/WiFiManager/>) repository and added them to the project.

**Firmware upload to the ESP32 chip**

Our chip is connected to the COM3 Port in the computer

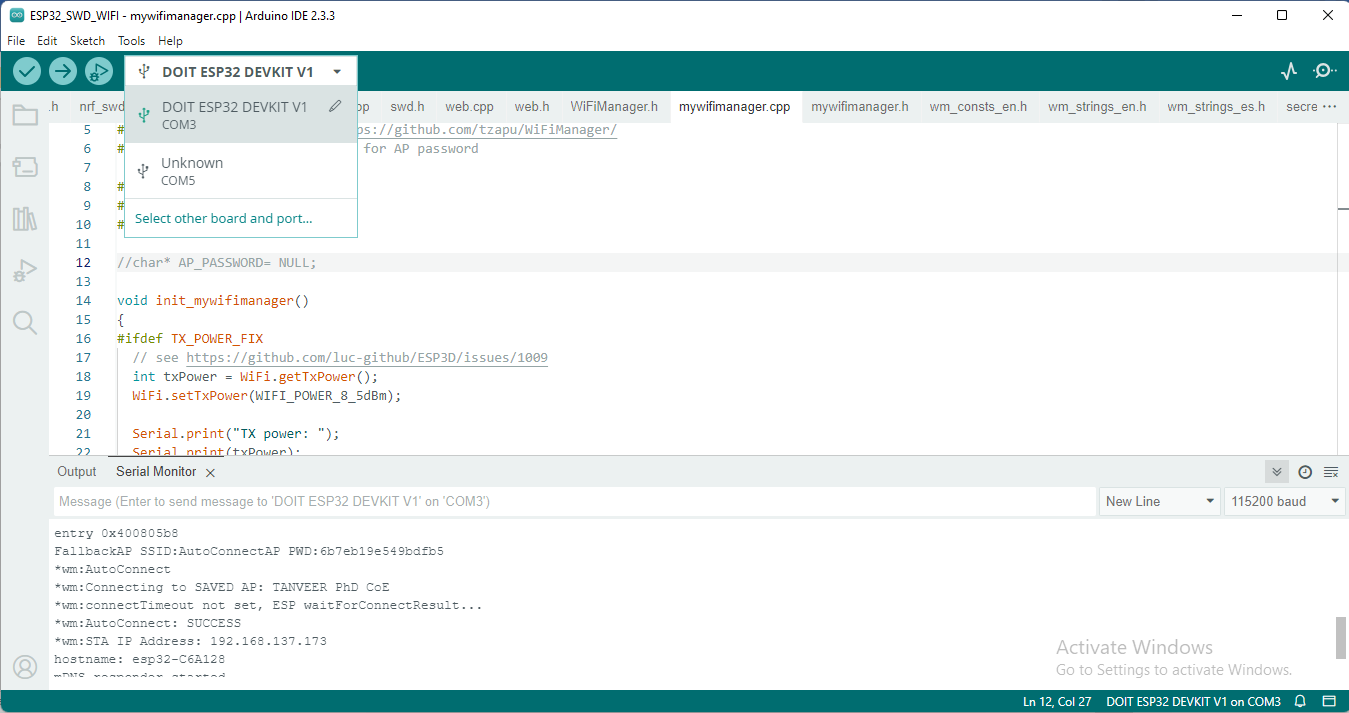


Fig. 3. Arduino IDE connection with ESP32.

After successful upload the ESP32 is start working as SWD programmer over the air (OTA).

That’s why the chip has been connected to our computer hotspot via WiFi obtaining the IP Address **192.168.137.173**

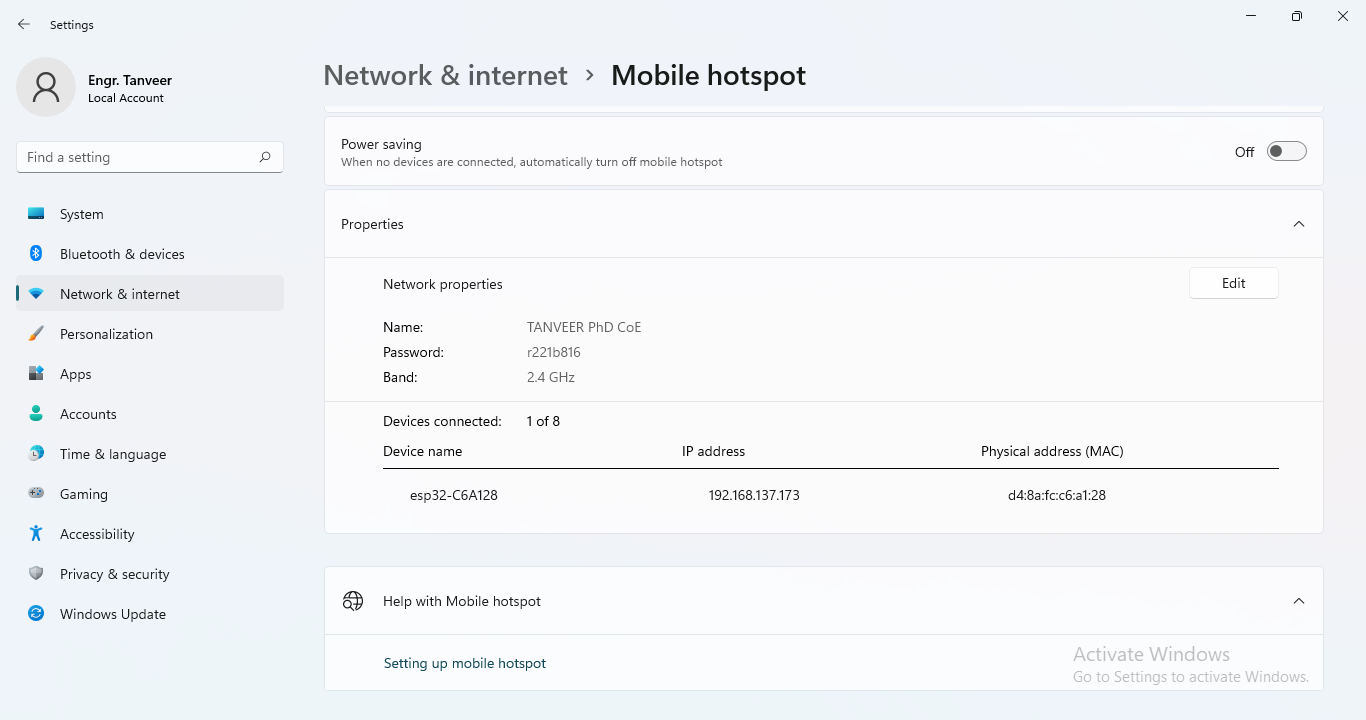


Fig. 4. ESP32 connected to the host PC through mobile hotspot WiFi.

The Arduino Serial monitor is reading at 115200 baud rate with the ESP32 chip.

wm:AutoConnect

\*wm:Connecting to SAVED AP: TANVEER PhD CoE

\*wm:connectTimeout not set, ESP waitForConnectResult...

\*wm:AutoConnect: SUCCESS

\*wm:STA IP Address: 192.168.137.173

hostname: esp32-C6A128

mDNS responder started

DP Write reg: 0x00 : 0x0000001e

DP Write reg: 0x04 : 0x50000000

SWD Id: 0x00000000

**Connecting the pinetime64 to ESP32 chip**

The SWD pinout for the PineTime watch is shown below

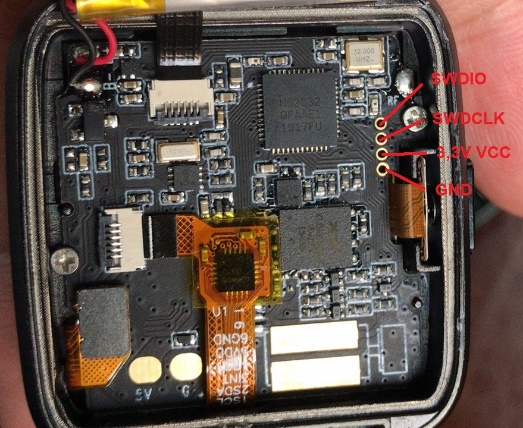


Fig. 5. SWD pinout of the PineTime64 smartwatch [4]

To flash an nRF52 connect the following:

* nRF52 **SWDCLK** to ESP32 **GPIO 21**
* nRF52 **SWDIO** to ESP32 **GPIO 19**
* nRF52 **GND** to ESP32 **GND**
* Then power the nRF52 as needed

According to this we have connected the watch as per figure

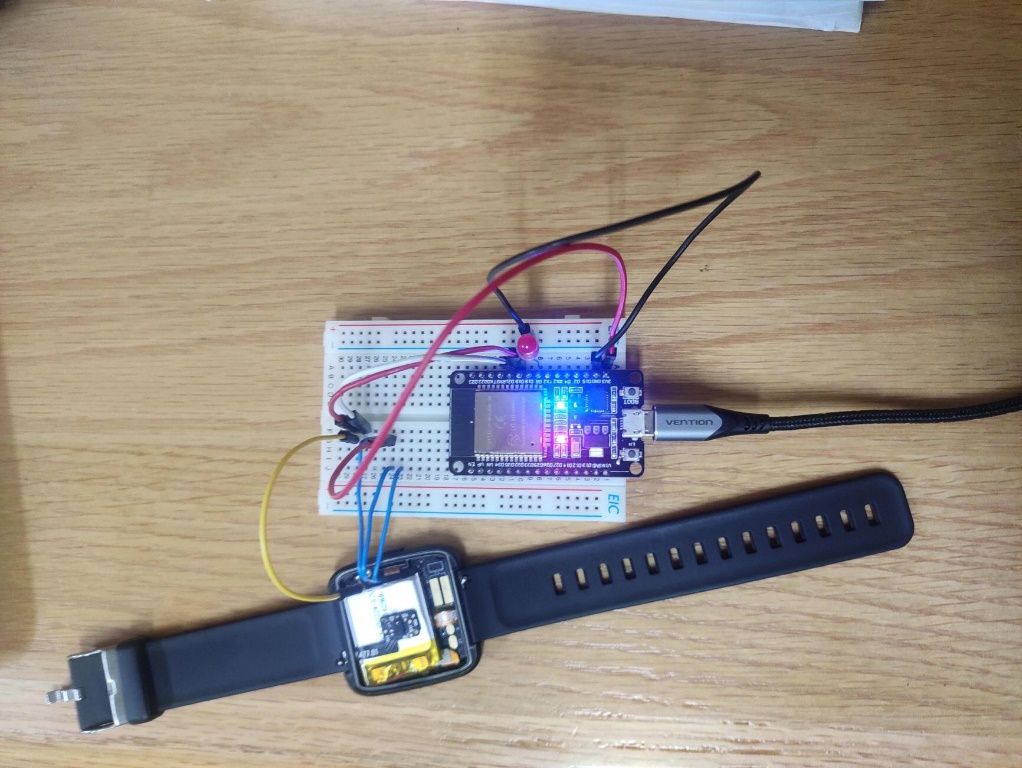


Fig. 6. PineTime64 smart watch SWD pin connection to ESP32 chip.

Firmware update to PineTime smart watch

* Once fully connected enter: "[http://swd.local](http://swd.local/)" in our internet browser and it showed a first page from the ESP32
* login with admin:admin: "<http://swd.local/edit>"
* We Choose File and browse for the "data/index.htm" file and clicked Upload
* Went to: "[http://swd.local](http://swd.local/)" again, the ESP32 SWD Flasher page was displayed
* We connected the nRF52 via SWD. Click the button "Init SWD" and wait for the response in the info page or look in the Arduino UART terminal if something doesn't work. The nRF chip should be detected and it will display a notification about whether or not the nRF52 is locked

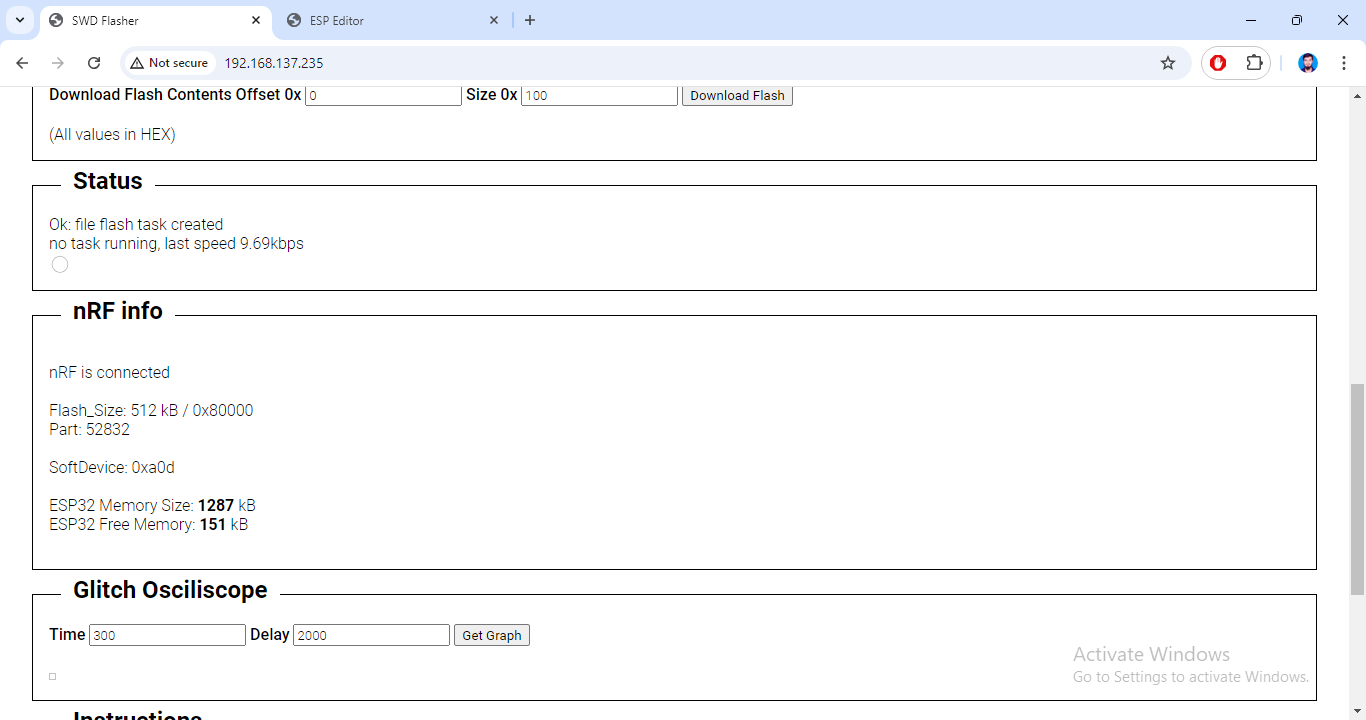


Fig. 7. PineTime64 smart watch chip nRF connected to the ESP32.

Now downloaded the PineTime64 latest firmware hex file from github repository <https://github.com/InfiniTimeOrg/InfiniTime/releases> the **merged-internal.hex**



Fig. 8. PineTime64 smartwatch firmware hex file

Now we opened the .hex file in HxD to save it as infinitime.bin file.

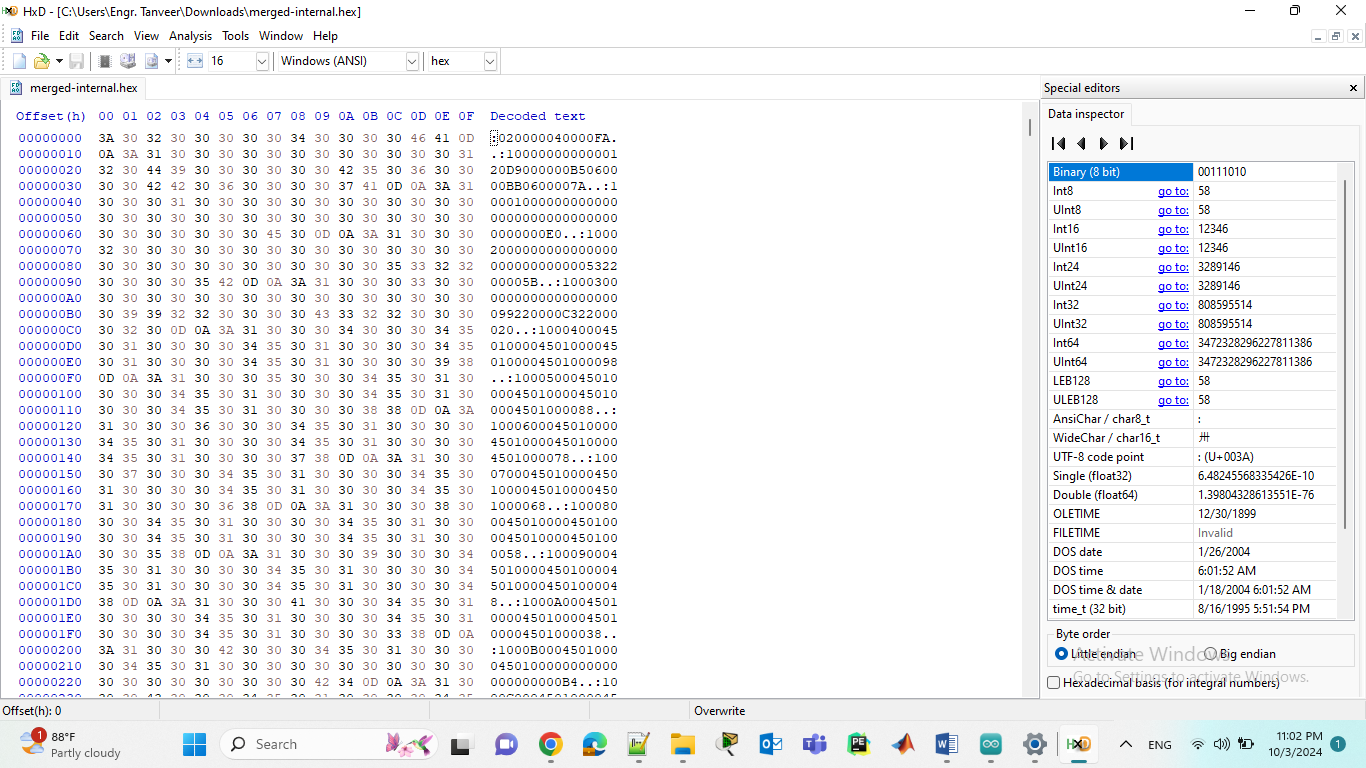


Fig. 8. Opening .hex file in HxD to save it as .bin file

Now we upload the infinitime.bin file in the esp32 provided web interface. And then clicked the flash file by showing the .bin file in the web interface it then started flashing the Chip.

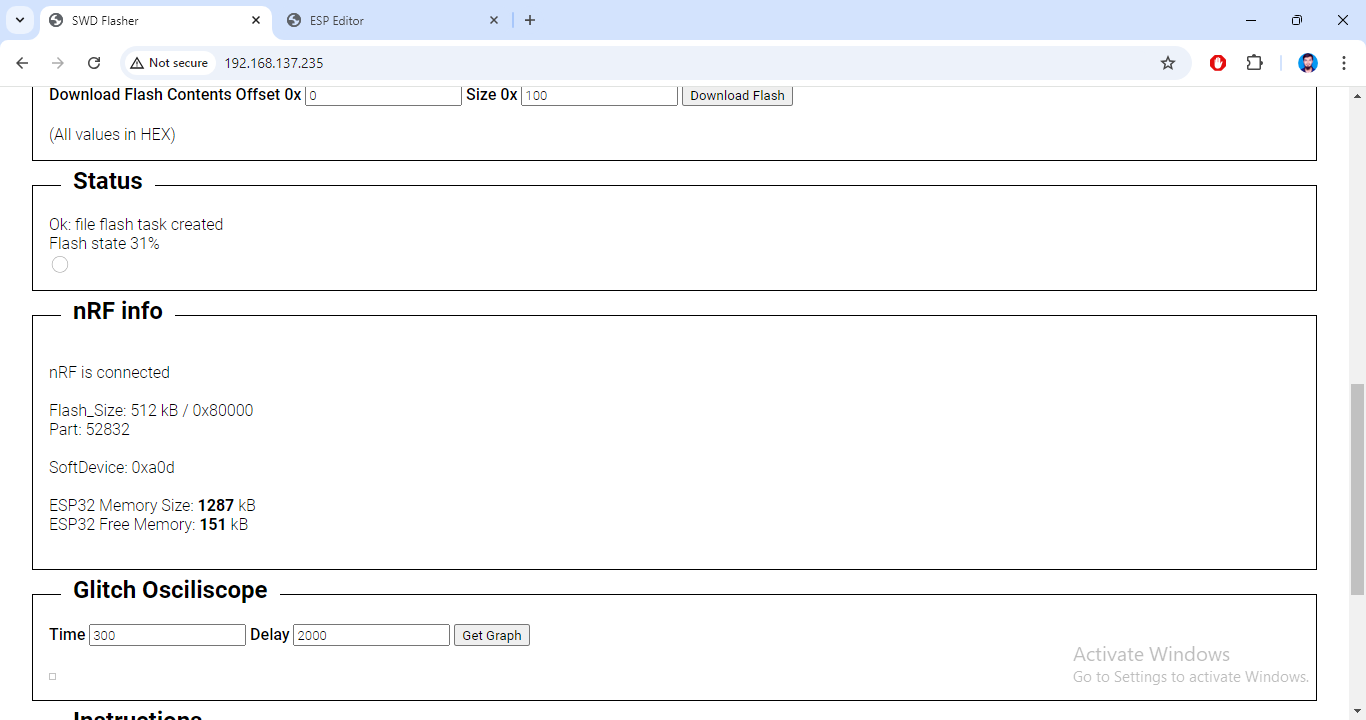


Fig. 9. Flash task is running 31%

During this time we can check the arduino IDE terminal that Register Read and Write operation is performing.

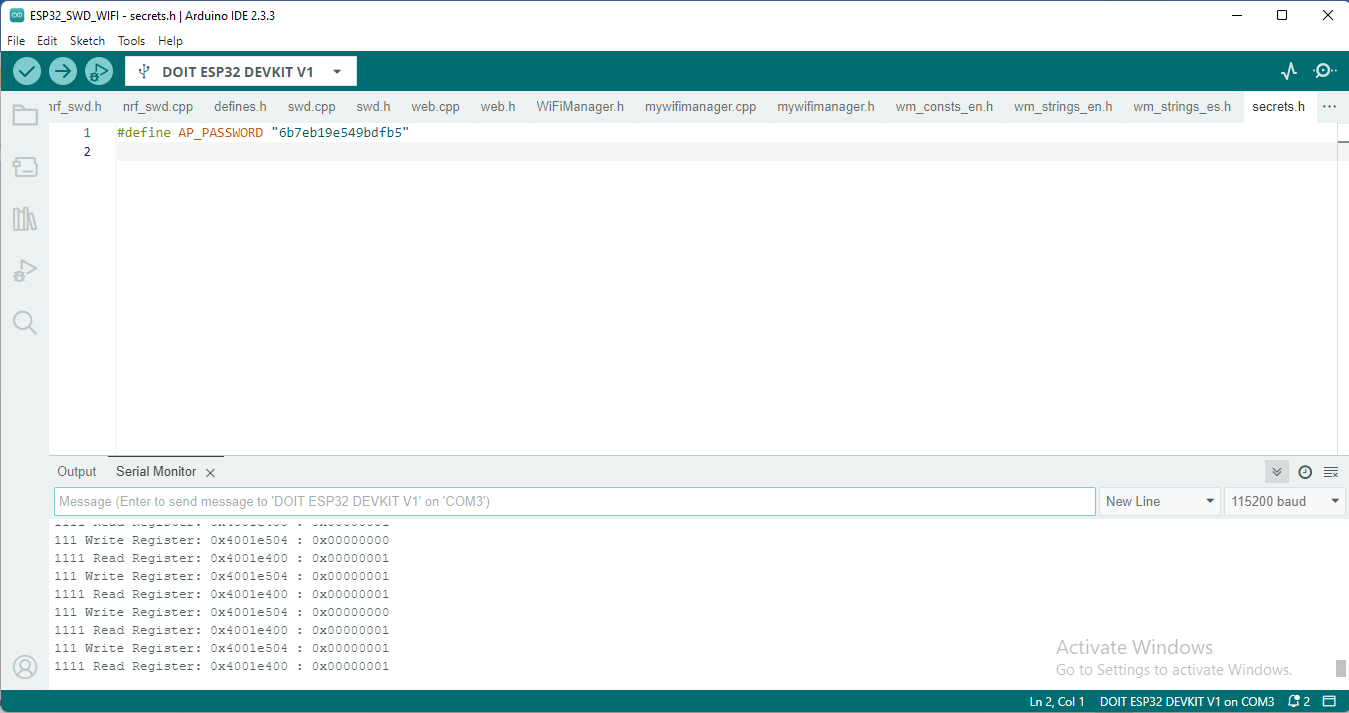


Fig. 10. During Flash operation Register Read and Write is performing

After flashing the chip we clicked on the reset button to start the watch

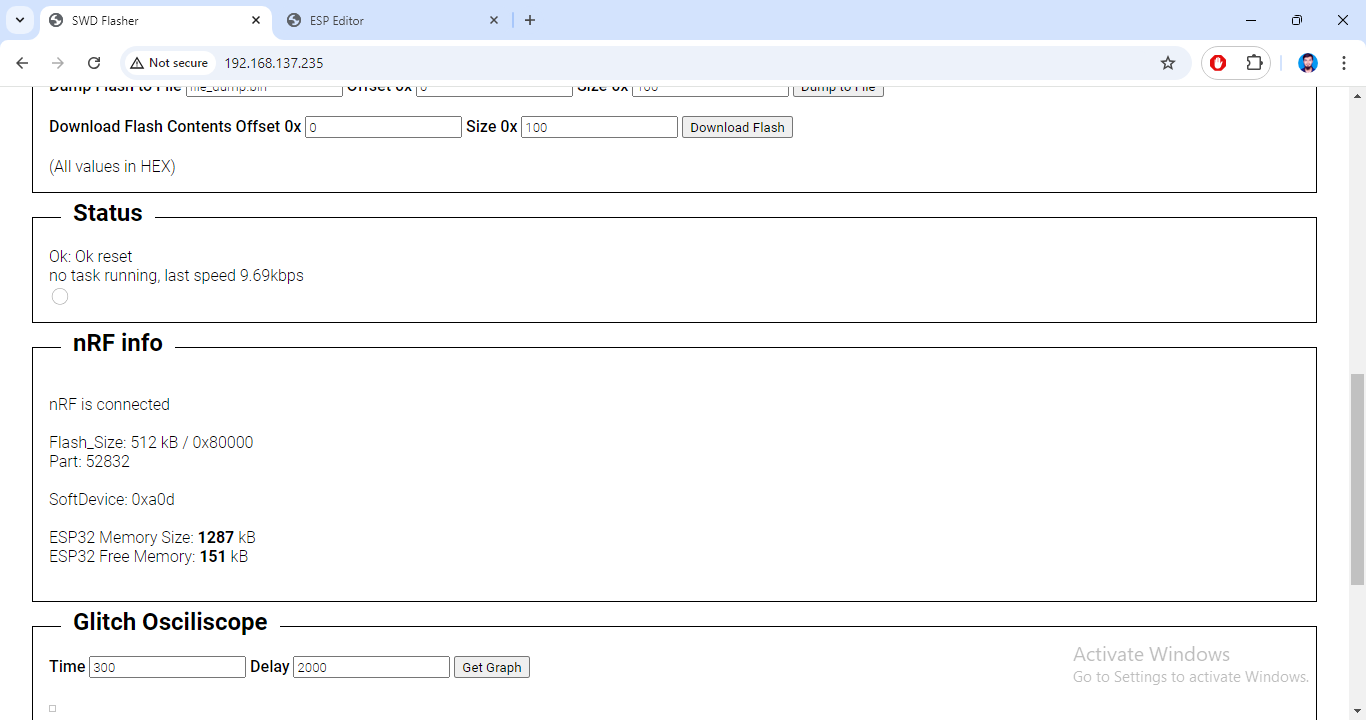


Fig. 11. PineTime smart reset after Flash

**Outcomes:**

We found that the watch didn’t boot. We several time performed the flash operation with other .hex files from the git release. But couldn’t find the glitch, in addition to this we can’t check whether the firmware is correct or not. Though the flash operation was successful. We hope that we will investigate this later.

**References**

[1] Micro XRCE-DDS (<https://micro.ros.org/docs/concepts/middleware/Micro_XRCE-DDS/>)

[2] PineTime, <https://pine64.org/documentation/PineTime/>

[3] nRF52, NordicSemiconductor, <https://docs.nordicsemi.com/>

[4] PineTime SWD pinout, <https://wiki.pine64.org/wiki/PineTime_Devkit_Wiring>