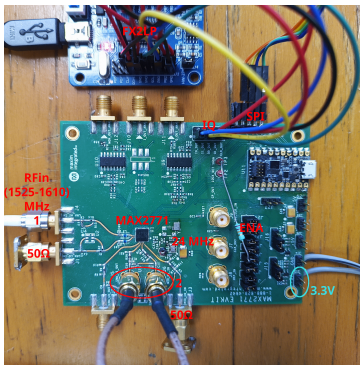
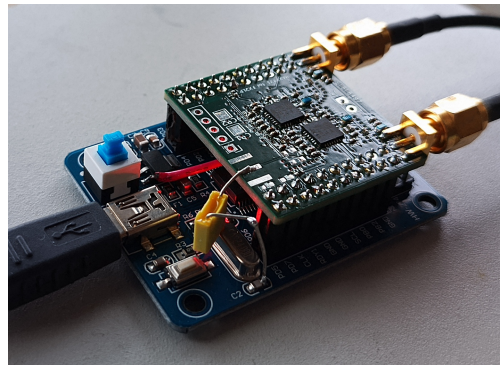
Efficient USB communication under GNU/Linux for a wideband (MAX2771-based) L-band (GNSS) SDR receiver

J.-M Friedt
FEMTO-ST Time & Frequency, Besançon, France

From



to



October 12, 2024

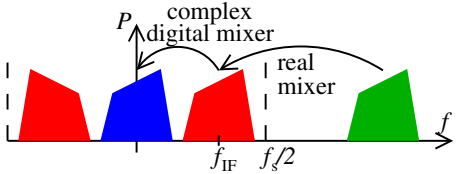
Why do we need high bandwidth data transfer?

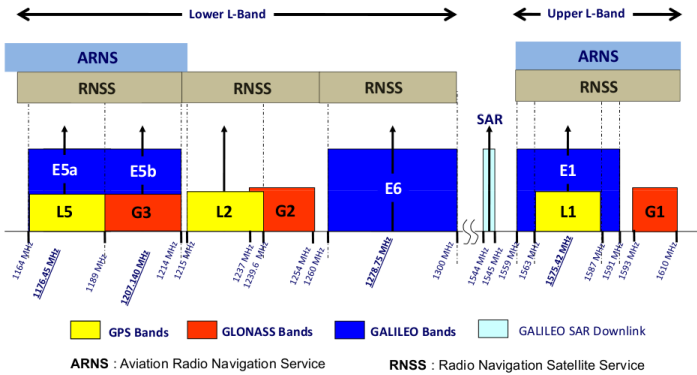
More and more GNSS constellations with increasingly complex (and accurate) coding schemes

- legacy GPS L1 C/A: 2.046 MHz bandwidth (1.023 MSps complex)
- Galileo E1: 4 MHz bandwidth (2 MSps complex)
- GPS L2C: 2.046 MHz bandwidth
- GPS L5: 24 MHz bandwidth (8 MSps complex sufficient, aeronautical band)
- Galileo E5a: 20.46 MHz bandwidth (5 MSps complex sufficient)

Very weak signals (below thermal noise)

$\Rightarrow$ might benefit from an intermediate frequency $\Rightarrow$ more bandwidth needed





GNSS L-band (1–2 GHz) frequency allocations [a]

[a] gssc.esa.int/navipedia/index.php/GNSS_signal

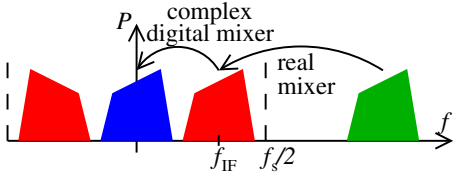
Why do we need high bandwidth data transfer?

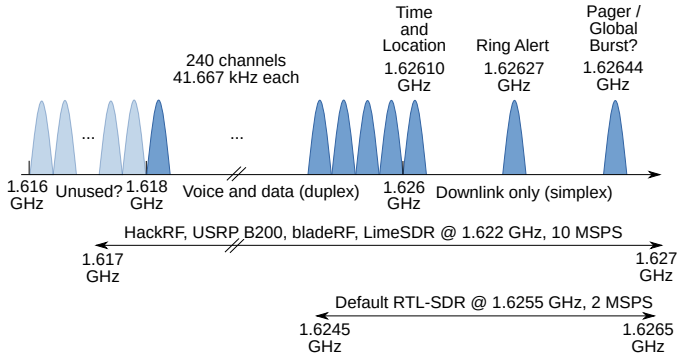
More and more GNSS constellations with increasingly complex (and accurate) coding schemes

- legacy GPS L1 C/A: 2.046 MHz bandwidth (1.023 MSps complex)
- Galileo E1: 4 MHz bandwidth (2 MSps complex)
- GPS L2C: 2.046 MHz bandwidth
- GPS L5: 24 MHz bandwidth (8 MSps complex sufficient, aeronautical band)
- Galileo E5a: 20.46 MHz bandwidth (5 MSps complex sufficient)

Very weak signals (below thermal noise)

$\Rightarrow$ might benefit from an intermediate frequency $\Rightarrow$ more bandwidth needed





Iridium LEO satellite frequency allocations [a]: 1616–1626.5 MHz

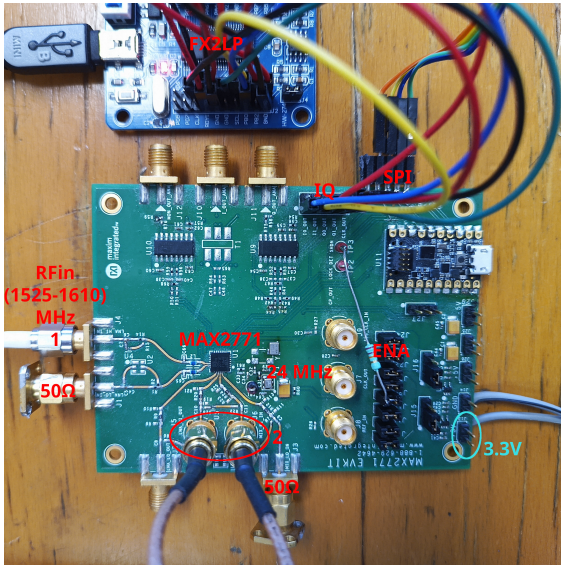
[a] https://github.com/muccc/gr-iridium

MAX2771 and its evaluation board

- MAX2771: general purpose L-band (1–2 GHz) receiver ...
- ... with 2 or 3-bit ADC, up to 48 MS/s parallel output
- dedicated to “upper” (L1, 1575.42 MHz) and “lower” (L2, L5... 1200 MHz) GNSS bands.

fitted on an evaluation by Maxim IC (now ADi)

- control software only running with MS-Windows (fails with Wine)
- after reverse engineering the USB control interface (VirtualBox guest on Linux host and `usbmon` debugging interface [a])... no IQ streaming, only converts USB Vendor Request commands to SPI sentences, but IQ pins are left unconnected

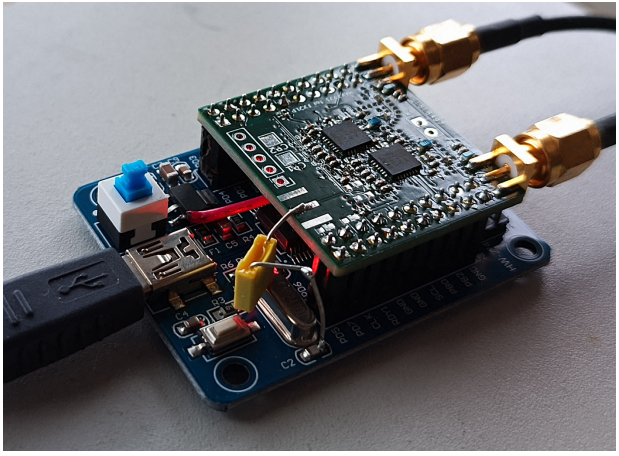


[a] `https://www.baeldung.com/linux/usb-sniffing`

PocketSDR [1]: dual MAX2771 GNSS SDR receiver

- Two or four MAX2771 RF frontends
- either two bands decoded simultaneously, or same band monitored with two antennas (CRPA [a])
- USB communication interface: Cypress EZ-USB FX2LP [b]
- IQ stream either decoded using Python scripts or fed to `gnss-sdr` (FIFO for real time)

Reproduced here with a custom board



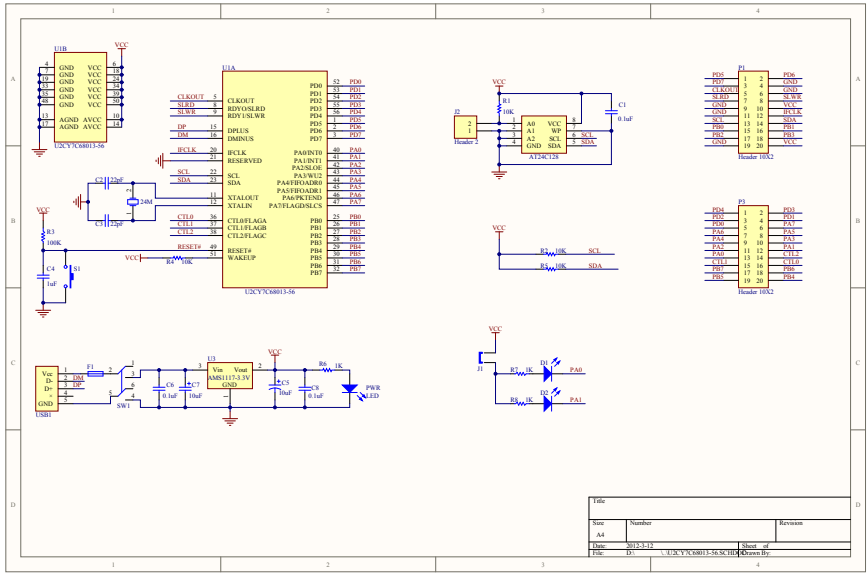
[a] Controlled Radiation Pattern Antenna for jamming/spoofing detection and mitigation

[b] compare with LUFA CDC (Communications Device Class): 100 kB/s

[1] `https://github.com/tomojitakasu/PocketSDR`

Cypress FX2LP EZ-USB

- parallel input or output to USB Bulk stream
- on-board buffers with clock driving parallel data
- EEPROM storing configuration running on the 8051 embedded microcontroller
- 8051 can run custom software, e.g. software emulation of SPI communication

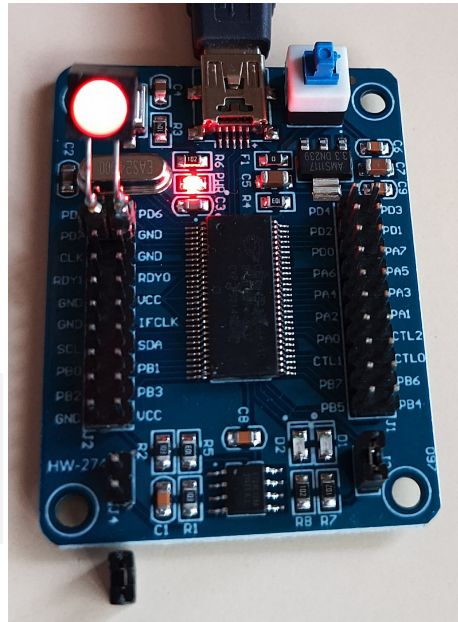


Getting familiar with the 8051 on the Cypress FX2LP EZ-USB

- 8-bit 8051 core is too small for gcc ⇒ use `sdcc`
- custom library for managing FX2LP peripherals: FX2LIB at `https://github.com/djmuhlestein/fx2lib`
- programming using `fxload` (`https://github.com/mbed-ce/fxload`) or `cycfx2prog`
- concept of *Vendor Requests* for sharing commands through USB
- use of Python wrapper of `libusb`

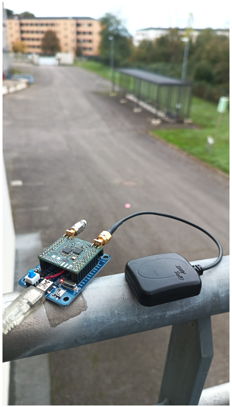
```
import usb
import binascii
VID = 0x04B4
PID = 0x8613
dev = usb.core.find(idVendor = VID, idProduct = PID)
ret=dev.ctrl_transfer(0xC0,0xa9, 0, 0, 32)
    # read EEPROM content
print(binascii.hexlify(ret))
```

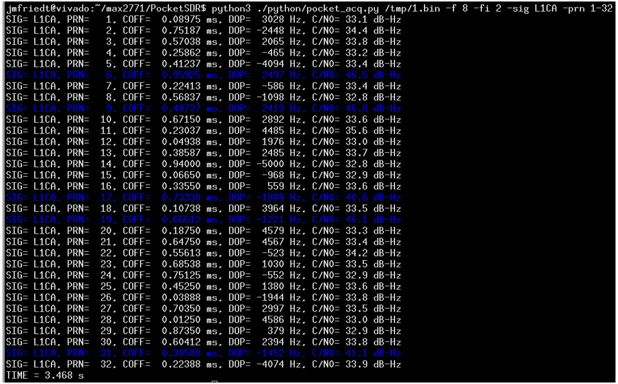
- software implementation of SPI communication protocol



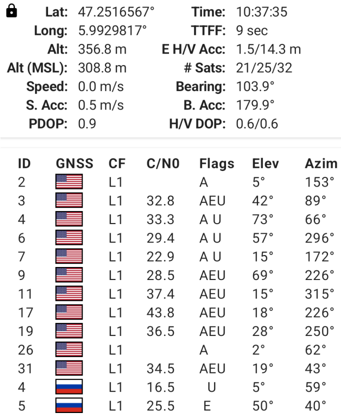
Objective

GNSS receiver with the ability to identify which satellite is broadcasting and solve position





```
jmfriedt@vivado:~/max2771/PocketSDR$ python3 ./python/pocket_acq.py /tmp/1.bin -f 8 -fi 2 -sig L1CA -prn 1-32
SIG= L1CA, PRN=   1, COFF=  0.08975 ms, DOP=  3028 Hz, C/N0= 33.1 dB-Hz
SIG= L1CA, PRN=   2, COFF=  0.75187 ms, DOP= -2448 Hz, C/N0= 34.4 dB-Hz
SIG= L1CA, PRN=   3, COFF=  0.57038 ms, DOP=  2065 Hz, C/N0= 33.8 dB-Hz
SIG= L1CA, PRN=   4, COFF=  0.25862 ms, DOP=  -465 Hz, C/N0= 33.2 dB-Hz
SIG= L1CA, PRN=   5, COFF=  0.41237 ms, DOP= -4094 Hz, C/N0= 33.4 dB-Hz
SIG= L1CA, PRN=   6, COFF=  0.95925 ms, DOP=  2497 Hz, C/N0= 46.5 dB-Hz
SIG= L1CA, PRN=   7, COFF=  0.22413 ms, DOP=  -586 Hz, C/N0= 33.4 dB-Hz
SIG= L1CA, PRN=   8, COFF=  0.56837 ms, DOP= -1098 Hz, C/N0= 32.8 dB-Hz
SIG= L1CA, PRN=   9, COFF=  0.48737 ms, DOP=  2419 Hz, C/N0= 46.8 dB-Hz
SIG= L1CA, PRN=  10, COFF=  0.67150 ms, DOP=  2892 Hz, C/N0= 33.6 dB-Hz
SIG= L1CA, PRN=  11, COFF=  0.23037 ms, DOP=  4485 Hz, C/N0= 35.6 dB-Hz
SIG= L1CA, PRN=  12, COFF=  0.04938 ms, DOP=  1976 Hz, C/N0= 33.0 dB-Hz
SIG= L1CA, PRN=  13, COFF=  0.38587 ms, DOP=  2485 Hz, C/N0= 33.7 dB-Hz
SIG= L1CA, PRN=  14, COFF=  0.94000 ms, DOP= -5000 Hz, C/N0= 32.8 dB-Hz
SIG= L1CA, PRN=  15, COFF=  0.06650 ms, DOP=  -968 Hz, C/N0= 32.9 dB-Hz
SIG= L1CA, PRN=  16, COFF=  0.33550 ms, DOP=   559 Hz, C/N0= 33.6 dB-Hz
SIG= L1CA, PRN=  17, COFF=  0.73338 ms, DOP= -1888 Hz, C/N0= 40.6 dB-Hz
SIG= L1CA, PRN=  18, COFF=  0.10738 ms, DOP=  3964 Hz, C/N0= 33.5 dB-Hz
SIG= L1CA, PRN=  19, COFF=  0.66612 ms, DOP= -1221 Hz, C/N0= 46.1 dB-Hz
SIG= L1CA, PRN=  20, COFF=  0.18750 ms, DOP=  4579 Hz, C/N0= 33.3 dB-Hz
SIG= L1CA, PRN=  21, COFF=  0.64750 ms, DOP=  4567 Hz, C/N0= 33.4 dB-Hz
SIG= L1CA, PRN=  22, COFF=  0.55613 ms, DOP=  -523 Hz, C/N0= 34.2 dB-Hz
SIG= L1CA, PRN=  23, COFF=  0.68538 ms, DOP=  1030 Hz, C/N0= 33.5 dB-Hz
SIG= L1CA, PRN=  24, COFF=  0.75125 ms, DOP=  -552 Hz, C/N0= 32.9 dB-Hz
SIG= L1CA, PRN=  25, COFF=  0.45250 ms, DOP=  1380 Hz, C/N0= 33.6 dB-Hz
SIG= L1CA, PRN=  26, COFF=  0.03888 ms, DOP= -1944 Hz, C/N0= 33.8 dB-Hz
SIG= L1CA, PRN=  27, COFF=  0.70350 ms, DOP=  2997 Hz, C/N0= 33.5 dB-Hz
SIG= L1CA, PRN=  28, COFF=  0.01250 ms, DOP=  4586 Hz, C/N0= 33.0 dB-Hz
SIG= L1CA, PRN=  29, COFF=  0.87350 ms, DOP=   379 Hz, C/N0= 32.9 dB-Hz
SIG= L1CA, PRN=  30, COFF=  0.60412 ms, DOP=  2394 Hz, C/N0= 33.8 dB-Hz
SIG= L1CA, PRN=  31, COFF=  0.30588 ms, DOP= -1452 Hz, C/N0= 41.1 dB-Hz
SIG= L1CA, PRN=  32, COFF=  0.22388 ms, DOP= -4074 Hz, C/N0= 33.9 dB-Hz
TIME = 3.468 s
```

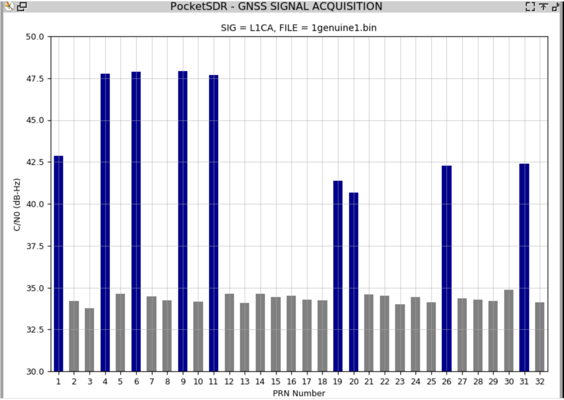


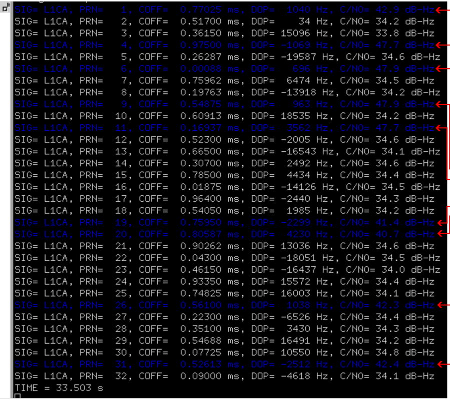
Lat:	47.2516567°	Time:	10:37:35
Long:	5.9929817°	TTFF:	9 sec
Alt:	356.8 m	E H/V Acc:	1.5/14.3 m
Alt (MSL):	308.8 m	# Sats:	21/25/32
Speed:	0.0 m/s	Bearing:	103.9°
S. Acc:	0.5 m/s	B. Acc:	179.9°
PDOP:	0.9	H/V DOP:	0.6/0.6

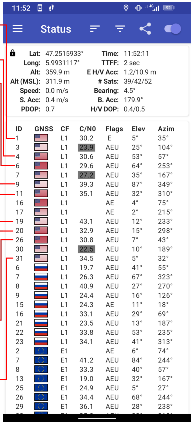
ID	GNSS	CF	C/N0	Flags	Elev	Azim
2		L1		A	5°	153°
3		L1	32.8	AEU	42°	89°
4		L1	33.3	A U	73°	66°
6		L1	29.4	A U	57°	296°
7		L1	22.9	A U	15°	172°
9		L1	28.5	AEU	69°	226°
11		L1	37.4	AEU	15°	315°
17		L1	43.8	AEU	18°	226°
19		L1	36.5	AEU	28°	250°
26		L1		A	2°	62°
31		L1	34.5	AEU	19°	43°
4		L1	16.5	U	5°	59°
5		L1	25.5	E	50°	40°

Outline

1. become familiar with the FX2LP and its 8051 (blinking LED) + toolchain and library
2. become familiar with USB communication (endpoints, PID:VID, HID vs Bulk, Vendor Requests)
3. later: add MAX2771 L-band receiver (PCB design + programming)
4. later: characterize MAX2771 receiver (signal synthesizers, Iridium signals)
5. later: receive IQ stream and analyze GNSS data (PocketSDR Python software, `gnss-sdr`)

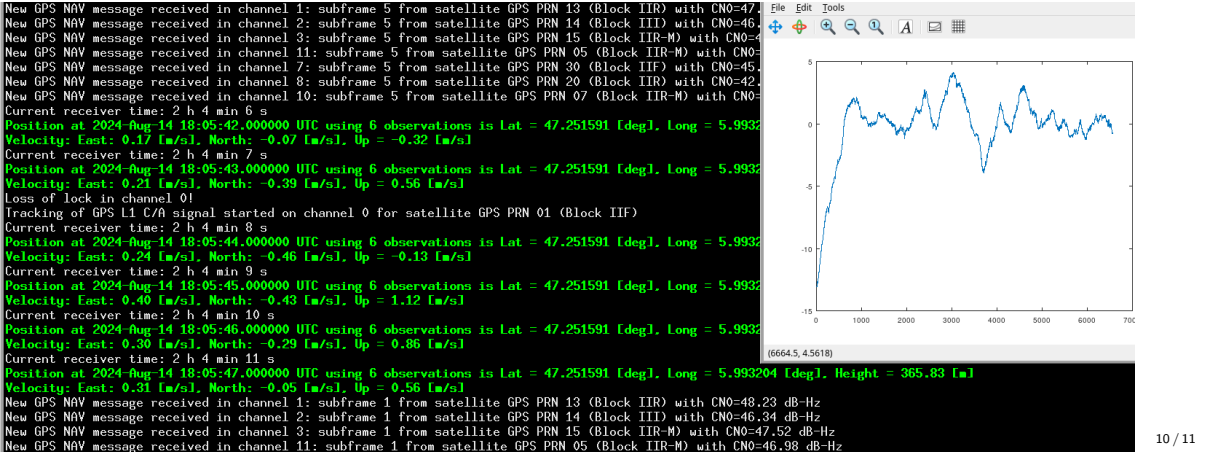






Outline

1. become familiar with the FX2LP and its 8051 (blinking LED) + toolchain and library
2. become familiar with USB communication (endpoints, PID:VID, HID vs Bulk, Vendor Requests)
3. later: add MAX2771 L-band receiver (PCB design + programming)
4. later: characterize MAX2771 receiver (signal synthesizers, Iridium signals)
5. later: receive IQ stream and analyze GNSS data (PocketSDR Python software, `gnss-sdr`)



Resources

- `lsusb` and `lsusb -vvv`
- from `https://www.baeldung.com/linux/usb-sniffing`: `sudo mount -t debugfs device_debug /sys/kernel/debug` and `sudo modprobe usbmon` to gain access to `/sys/kernel/debug/usb/usbmon/Xu` for sniffing USB communication
- `http://jmfriedt.free.fr/hackable_max2771_1.pdf`
- `https://github.com/jmfriedt/max2771_fx2lp`
- `https://github.com/djmuhlestein/fx2lib`
- `https://github.com/tomojitakasu/PocketSDR`

Schedule

1. 2 weeks 14–25 October 2024: USB communication & FX2LP
2. 2 weeks 06–17 January 2025: MAX2771 L-band receiver (PCB + software configuration), IQ collection
3. 2 weeks 03–14 March 2025: Iridium (`gr-iridium`, ZeroMQ and interfacing with GNU Radio) & GNSS signal decoding (PocketSDR, `gnss-sdr` post-processing of file or FIFO)
4. 1 week 05–09 May 2025: signal analysis, GPS v.s Galileo, direction of arrival measurement

End of M1: report + oral presentation ... to be continued: M2.

PROGRAMMATION USB SOUS GNU/LINUX :
APPLICATION DU FX2LP POUR UN RÉCEPTEUR DE RADIO LOGICIELLE DÉDIÉ AUX SIGNAUX DE NAVIGATION PAR SATELLITE (1/2)

Jean-Michel Friedt, enseignant-chercheur à l'université de Franche-Comté à Besançon

Alors que l'USB est souvent abordé comme un bus émulant un port série, tirer pleinement profit de sa bande passante nécessite d'exploiter les interfaces disponibles les plus appropriées, en particulier Human Interface Device (HID) et transferts en volume (Bulk). Nous proposons d'appréhender le bus USB exposé par le noyau Linux en vue d'en tirer le maximum du débit disponible, et appliquer cette connaissance en réalisant un récepteur de radio logicielle dédié à la réception des signaux de navigation par satellite (GNSS) en bande L (1–2 GHz) grâce au MAX2771. Nous démontrons le bon fonctionnement du circuit avec l'acquisition et le traitement de signaux issus de diverses constellations, de GNSS en orbite intermédiaire MEO et Iridium en orbite basse LEO, observés avec une bande passante pouvant aller jusqu'à 44 MHz.