DLØSHF station remote control with BeagleBone, Python and ZMQ

Per Dudek
Martin Süfke

2018-09-16 EUCARA am Stockert

www.dl0shf.de
per@per-dudek.de
eucara.2018.masuefke@recursor.net

DLØSHF at a glance

DLØSHF at a glance

1 GHz dish Ø 9m
1296 MHz Rx/Tx
1420 MHz Rx
all transistor PA

DLØSHF at a glance

2 GHz dish Ø 6m
DSN 2.1–2.2 GHz Rx
2.3 GHz Amateur radio
all transistor PA

DLØSHF at a glance

8.4 GHz dish Ø 7.2m
Deep Space Network
Rx only

DLØSHF at a glance

10 GHz dish Ø 7.2m
10 GHz Amateur Radio
Moon beacon
50 W transistor PA
600 W TWTA

DLØSHF at a glance

24 GHz dish Ø 3.7m
24 GHz Amateur Radio
30 W transistor PA

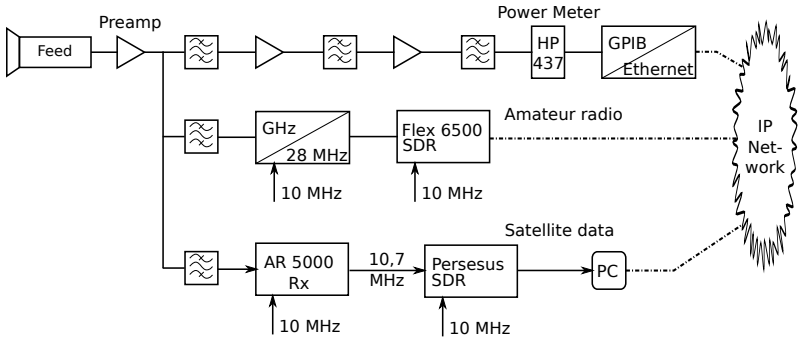
DLØSHF at a glance

32 GHz dish Ø 2m
32.1 GHz Deep Space Network
Rx only

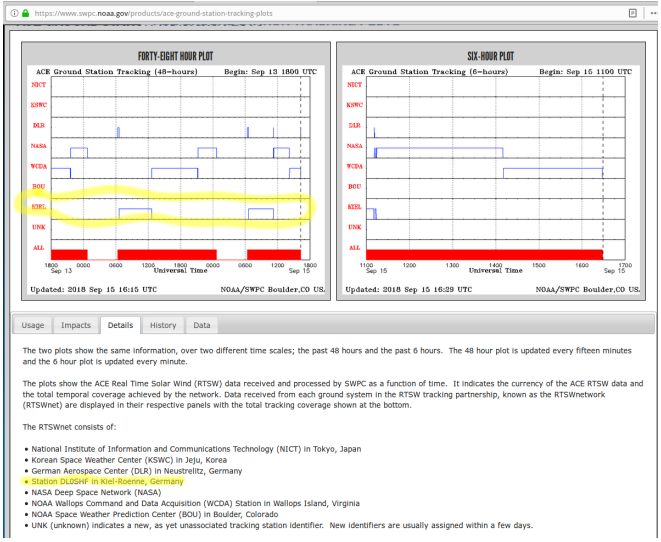
DLØSHF at a glance

60m tower
lower tier: commercial
upper tier:
– 10/24/47 GHz dish Ø 2m
– 1296/2320 MHz dish Ø 2m
– 50/70/432 MHz yagi

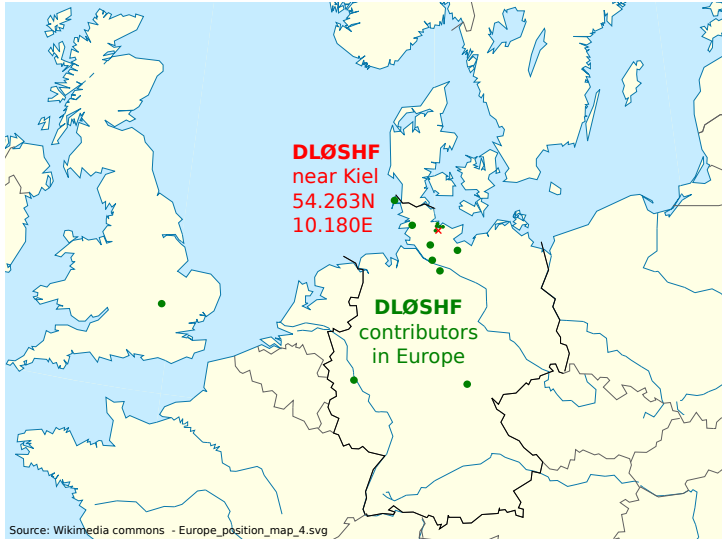
DLØSHF default Rx path



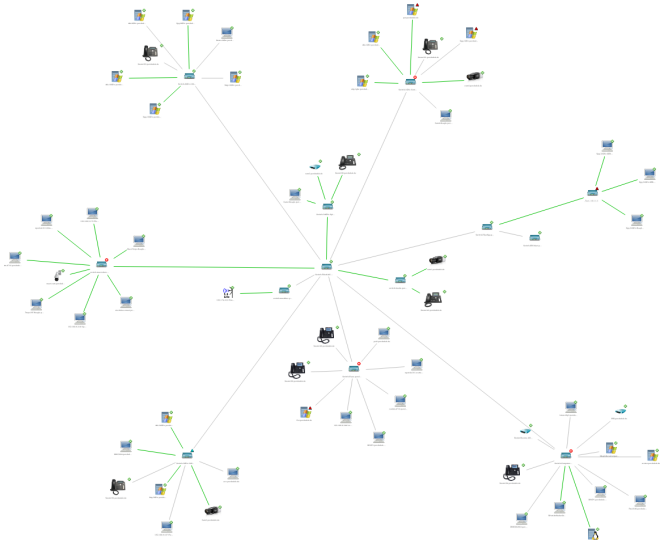
ACE satellite downlink



DLØSHF users



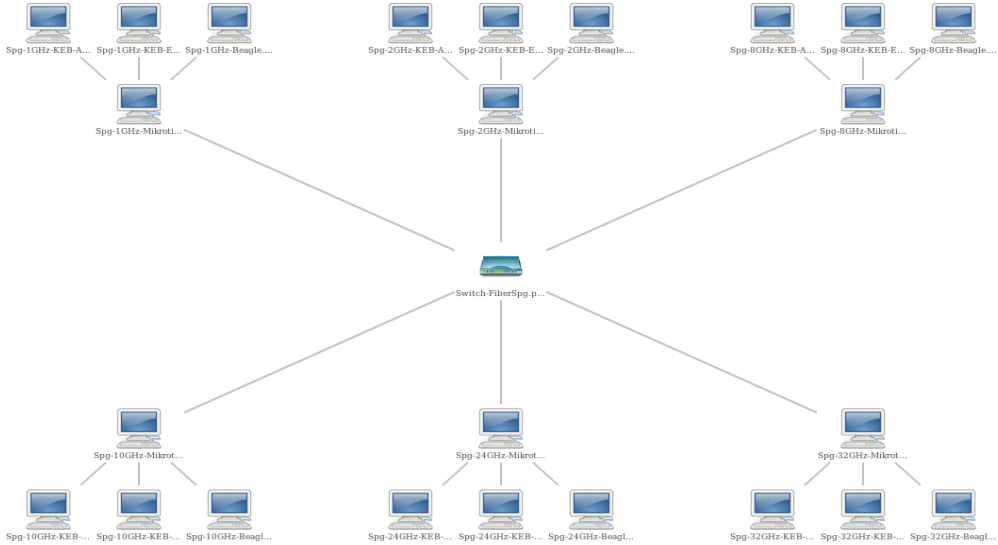
DLØSHF Ethernet (data/user)



DLØSHF Ethernet – 130 nodes active

Active node Free IP on the network 192.168.9.0/24 at 2018-09-14.

DLØSHF Ethernet (antennas)



Antenna control (one antenna)

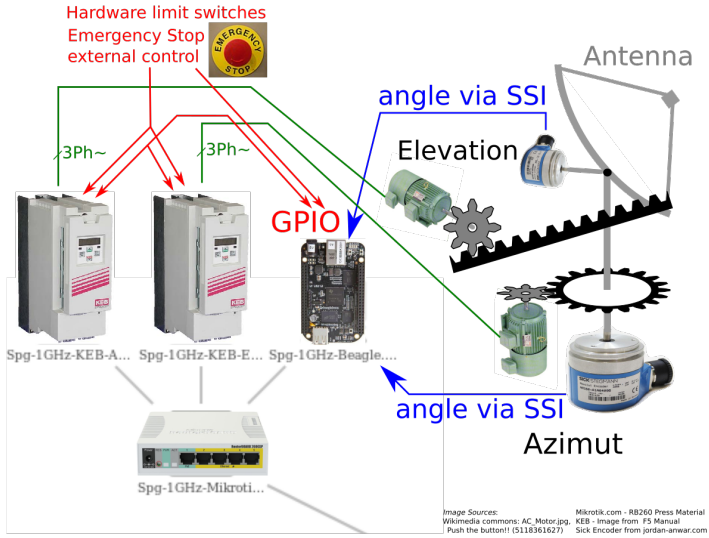
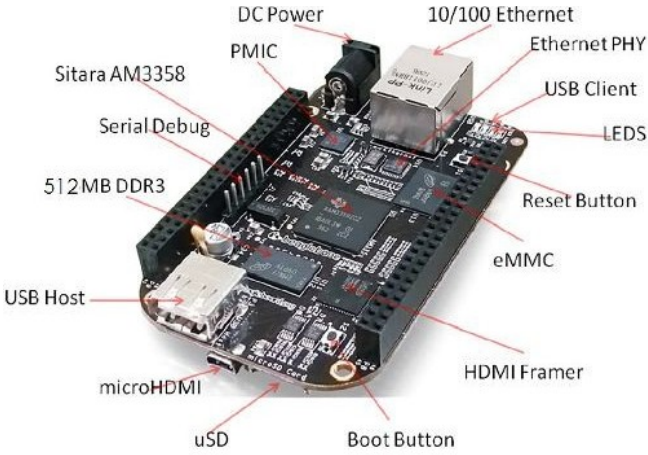


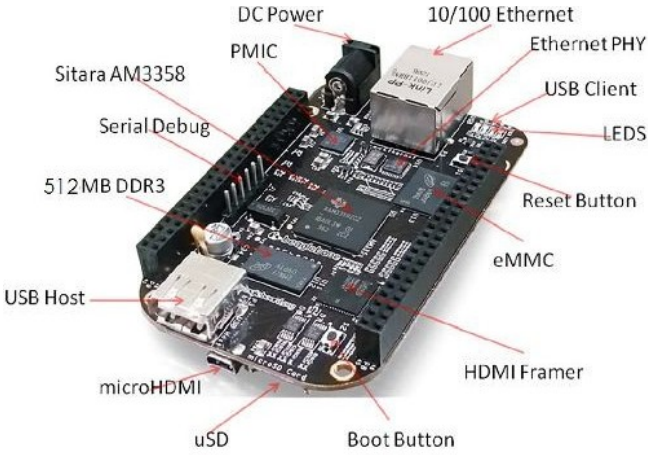
Image Sources: Wikimedia commons: AC_Motor.jpg, Push the button!! (5118361627) Mikrotik.com - RB260 Press Material KEB - Image from F5 Manual Sick Encoder from jordan-anwar.com

BeagleBoneBlack



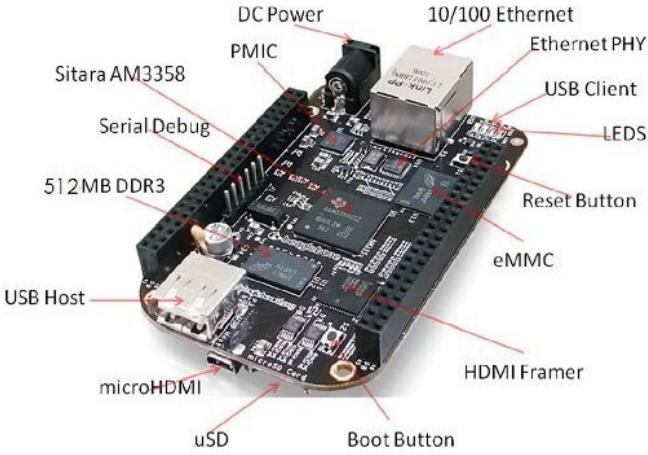
- CPU: 32 bit ARM v7

BeagleBoneBlack



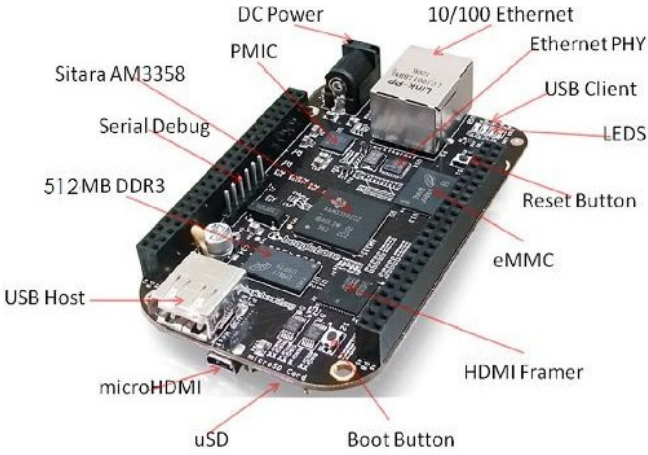
- CPU: 32 bit ARM v7
- 512 MB DDR3

BeagleBoneBlack



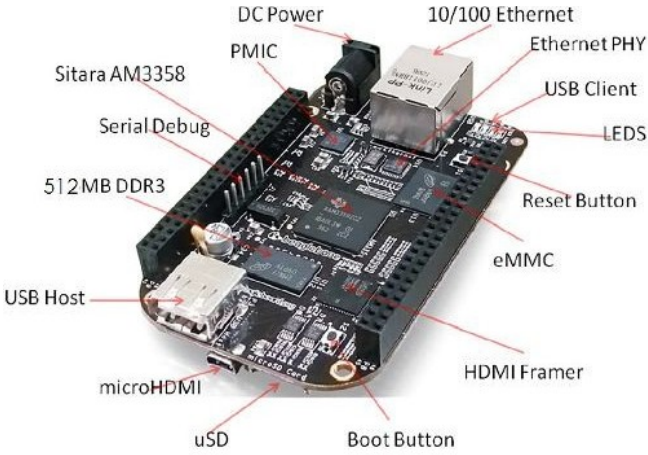
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard

BeagleBoneBlack



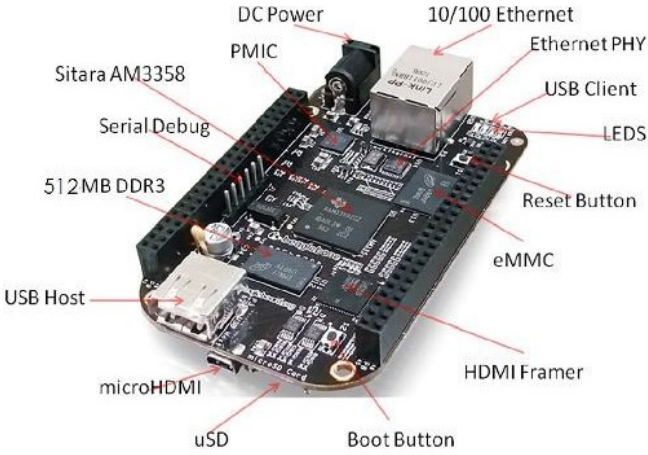
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO

BeagleBoneBlack



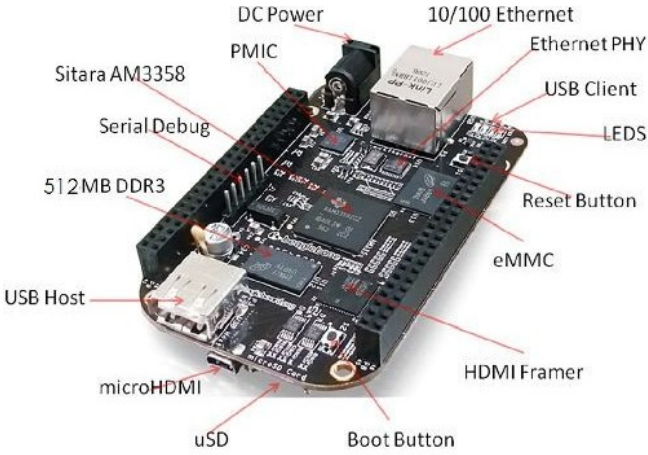
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232

BeagleBoneBlack



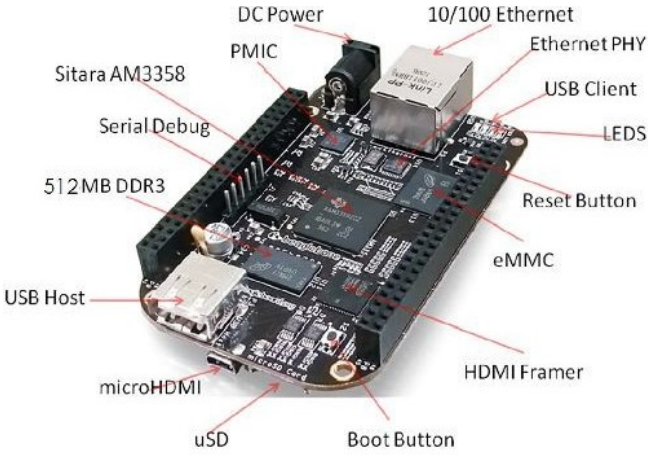
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232
- 2 I^2C

BeagleBoneBlack



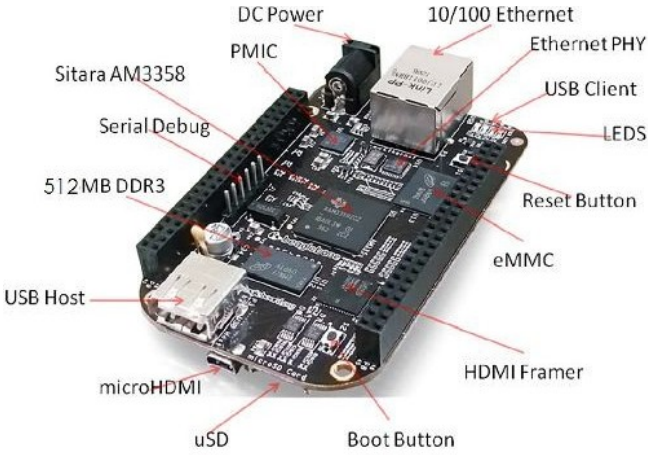
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232
- 2 I^2C
- 2 SPI

BeagleBoneBlack



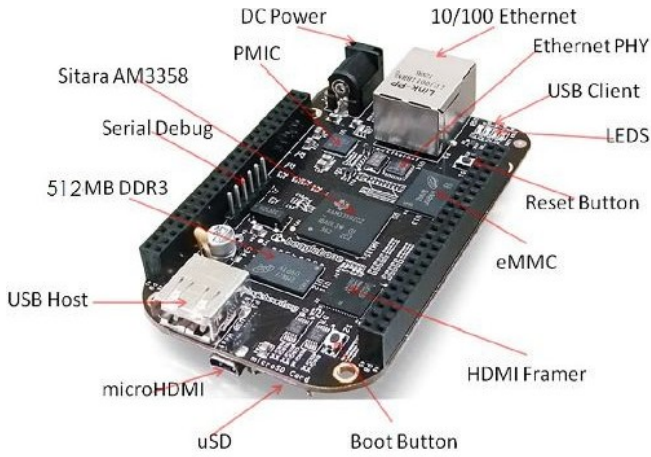
- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232
- 2 I^2C
- 2 SPI
- 2 CAN

BeagleBoneBlack



- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232
- 2 I^2C
- 2 SPI
- 2 CAN
- 8 analog in

BeagleBoneBlack



- CPU: 32 bit ARM v7
- 512 MB DDR3
- 4 GB eMMC/ any SDcard
- 65 GPIO
- 6 RS232
- 2 I^2C
- 2 SPI
- 2 CAN
- 8 analog in
- 2 PRU

PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller

PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller
- running at 200 MHz

PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller
- running at 200 MHz
- with 8 kB instructions + 8 kB RAM

PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller
- running at 200 MHz
- with 8 kB instructions + 8 kB RAM
- Assembler + C-compiler available

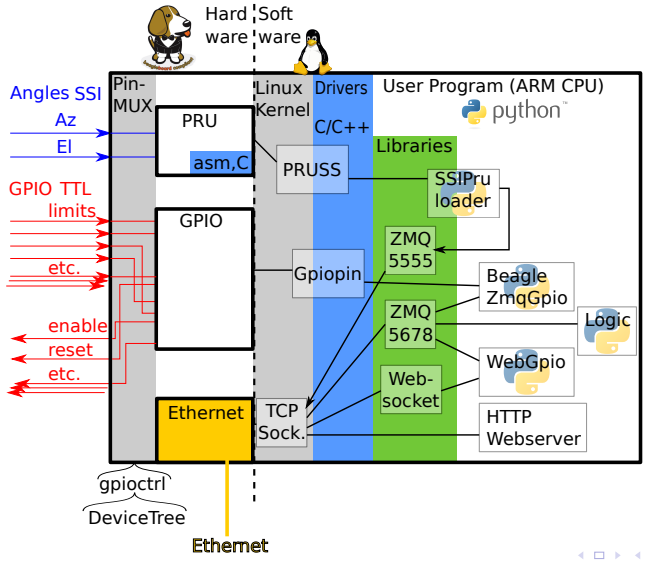
PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller
- running at 200 MHz
- with 8 kB instructions + 8 kB RAM
- Assembler + C-compiler available
- 2 of them in **BeagleBoneBlack**

PRU: Programmable Real-time Unit

- 32 bit RISC microcontroller
- running at 200 MHz
- with 8 kB instructions + 8 kB RAM
- Assembler + C-compiler available
- 2 of them in **BeagleBoneBlack**
- well-connected to the ARM CPU

PRU+GPIO in BeagleBone



- invented 1991 by Guido van Rossum

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library
- humongous eco-system (3rd-party projects, tutorials. . .)

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library
- humongous eco-system (3rd-party projects, tutorials. . .)
- integrates C/C++ code easily

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library
- humongous eco-system (3rd-party projects, tutorials. . .)
- integrates C/C++ code easily
- quite fast

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library
- humongous eco-system (3rd-party projects, tutorials. . .)
- integrates C/C++ code easily
- quite fast
- tolerable overhead

- invented 1991 by Guido van Rossum
- compiled on runtime, interpreted, source-code always available
- current version 3.7.x (DLØSHF: 3.4.5)
- extensive library
- humongous eco-system (3rd-party projects, tutorials. . .)
- integrates C/C++ code easily
- quite fast
- tolerable overhead
- interactive development/debugging possible

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching
- many peers

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching
- many peers
- automatic (re-)connects

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching
- many peers
- automatic (re-)connects

==> *a socket on steroids*

ØMQ

- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching
- many peers
- automatic (re-)connects

==> *a socket on steroids*

- quite fast

ØMQ

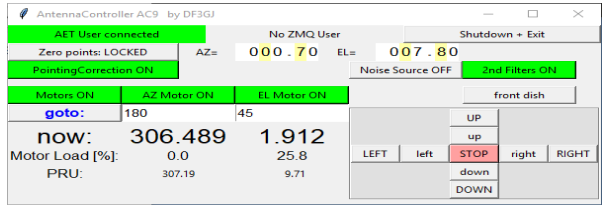
- spelled zero-em-queue, ØMQ, ZeroMQ, short ZMQ
- invented 2007 by Martin Sustrik, Pieter Hintjens (iMatix corp.)
- 'gilded' 2011 by CERN study for best middleware
- DESY (Hamburg) has positive experience
- Language bindings for many languages
- Message queuing without central message broker
- Delivery guarantes:
  - whole messages
  - ordered delivery
  - delivery, (limited) internal caching
- many peers
- automatic (re-)connects

==> *a socket on steroids*

- quite fast
- tolerable overhead

AntennaController AC9

Written in python by Joachim Köppen DF3GJ

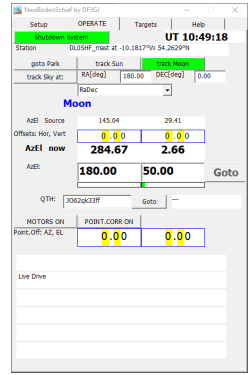


In: Current postition via ZMQ from Beaglebone, Target pos.
Antenna pointing control
Antenna pointing correction[1]
Out: Motor drive via ZMQ-KEB-Gateway to motor inverters

[1] J. Köppen 2014-11, *Pointing Correction for the DL0SHF 24 GHz Antenna* in www.dl0shf.de → Technische Berichte
`https://sat-sh.lernnetz.de/pdf/PointingCorrection24GHz.pdf`

Data source 'NewBodenSchief'

Written in C++ by Joachim Köppen DF3GJ



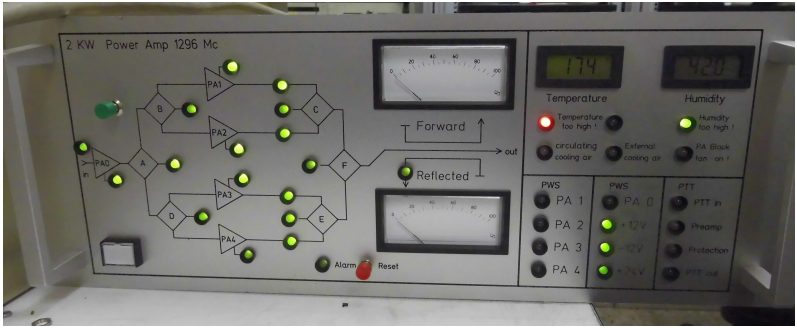
Radio astronomy targeting
Amateur radio pointing (astro objects, beacons, QTH-locators)
sends position-to-track to "AntennaControl AC9"

Amplifier control panel for 1 GHz PA

Hardware solution:

Amplifier control panel for 1 GHz PA

Hardware solution:



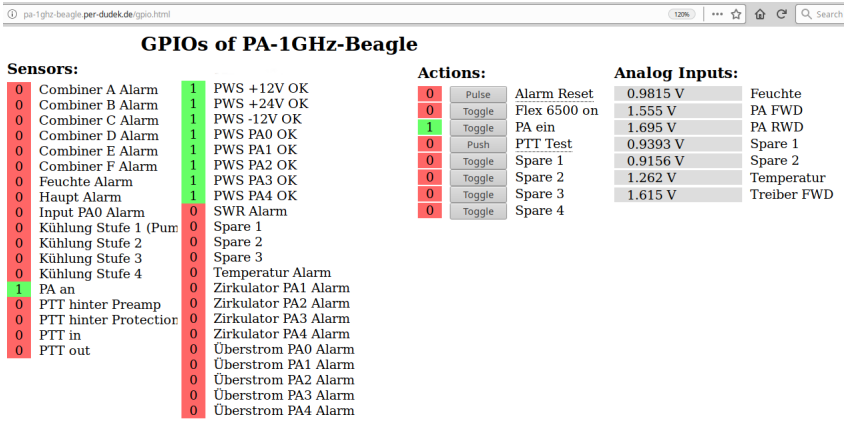
1 GHz Amplifier GPIOs

Beaglebone GPIO data (via ZMQ):


```
{"Action":
  {"Alarm␣Reset": {
   "_active": 1, "_duration": 0.5, "_initial": 0,
   "_type": "Pulse", "val": 0},
       "Flex␣6500␣on": 0,
       "PA␣ein": 0,
       "PTT␣Test": {"_active": 1, "_addend": 820265503, "_initial": 0,
            "_key": 1128659581, "_modulus": 1327218127, "_timeout":
           0.5, "_type": "Momentary", "val": 0},
       "Spare␣1": 0,
       "Spare␣2": 0,
       "Spare␣3": 0,
       "Spare␣4": 0},
       "Analog␣Input": {
        "Feuchte": {"_precision": 4, "_unit": "V", "val": "0.9499"},
        "PA␣FWD": {"_precision": 4, "_unit": "V", "val": "1.594"},
        "PA␣RWD": {"_precision": 4, "_unit": "V", "val": "1.716"},
        "Temperatur": {"_precision": 4, "_unit": "V", "val": "1.236"},
        "Treiber␣FWD": {"_precision": 4, "_unit": "V", "val": "1.661"
            },
```

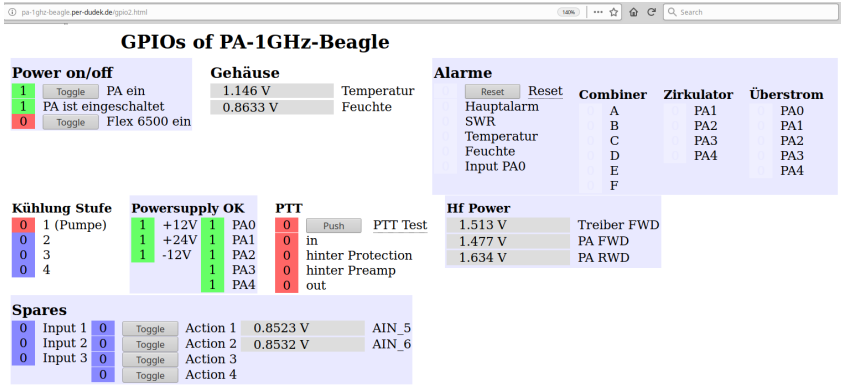
1 GHz Amplifier GPIOs

Simple webbrowser interface:



1 GHz Amplifier GPIOs

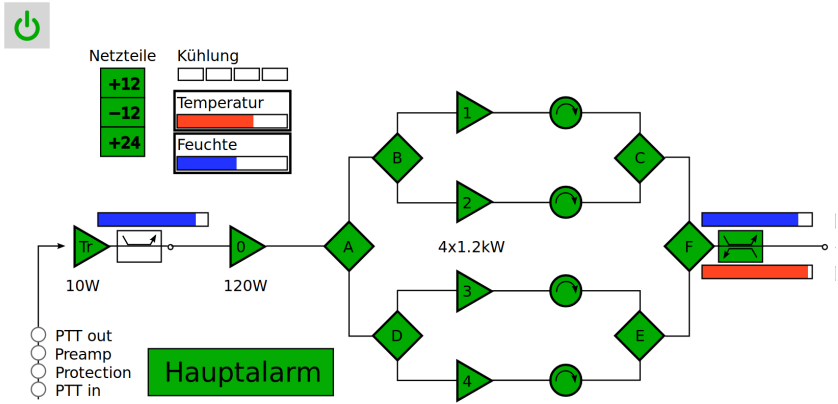
Intermediate webbrowser interface:



1 GHz Amplifier Panel

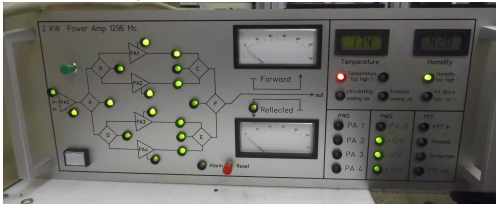
Advanced webbrowser interface:

PA control at PA-1GHz-Beagle

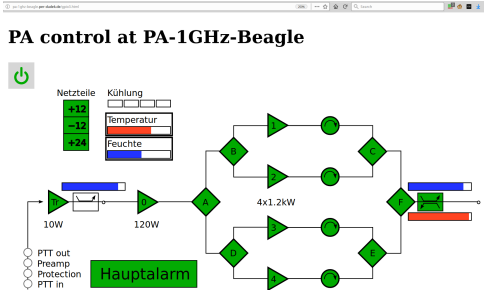


1 GHz Amplifier Panel

Hardware panel:



Software panel:



Controlling other devices

- HF switches on 60m tower (9 bands)

Controlling other devices

- HF switches on 60m tower (9 bands)
- 2 masts (28m high) for short wave

Controlling other devices

- HF switches on 60m tower (9 bands)
- 2 masts (28m high) for short wave
- 6 cameras (az,el,zoom,focus,iris)

Controlling other devices

- HF switches on 60m tower (9 bands)
- 2 masts (28m high) for short wave
- 6 cameras (az,el,zoom,focus,iris)
- Moon beacon `http://moonbeacon.dl0shf.de`

Controlling other devices

- HF switches on 60m tower (9 bands)
- 2 masts (28m high) for short wave
- 6 cameras (az,el,zoom,focus,iris)
- Moon beacon `http://moonbeacon.dl0shf.de`
- PA 10 GHz: 50W Transistor + 600W TWT

Controlling other devices

- HF switches on 60m tower (9 bands)
- 2 masts (28m high) for short wave
- 6 cameras (az,el,zoom,focus,iris)
- Moon beacon `http://moonbeacon.dl0shf.de`
- PA 10 GHz: 50W Transistor + 600W TWT

Conclusion:


+

+


+

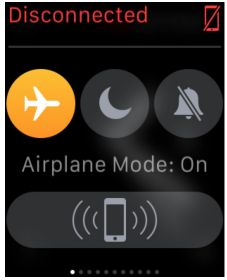

+

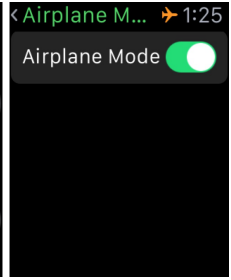
+ WebSocket + JavaScript = versatile

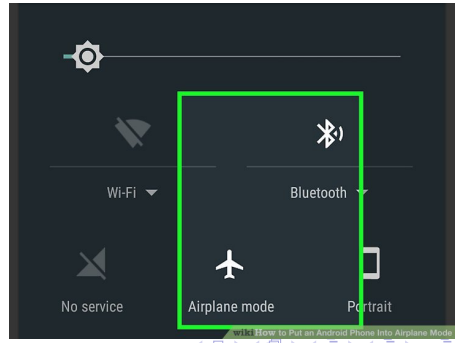
Live Demo

Please turn off all WiFi and mobile data. We need all bandwidth to run our demo. Thank you.

Please turn on “Airplane Mode”. Bitte „Flugmodus“ aktivieren.







Thank you for listening.

Thank you for listening.

We are willing to share techniques and software.
Please ask.

Cpu usage for antenna control

```
root@Spg-1GHz-Beagle:~# uptime
 19:54:53 up 314 days,  1:04,  1 user,  load average: 0,10, 0,13, 0,10
```

10 months without reboot and counting.

```
root@Spg-1GHz-Beagle:~# cat /sys/devices/system/cpu/cpu0/cpufreq/stats/
    time_in_state
300000 2707389289
600000 5774576
720000 11004
800000 6622
1000000 161727
```

Using only 10% of system running at 300 (max: 1000) MHz for $>$ 99.8% of the time

PRU process statistics

How much data to process ?

```
root@Spg-1GHz-Beagle:~# cat /proc/interrupts  | grep pruss_evt0
201:   540847970       INTC  20 Level     pruss_evt0
```

540 847 970 PRU interrupts $\approx$ 20 int/sec for 314 days $\widehat{=}$ 10 Hz SSI data rate.

```
root@Spg-1GHz-Beagle:~# ps wuax
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        568  5.3  2.3  35560 11620 ?        S<sl  2017 24395:05
    python3.4 /opt/SsiPru/loader.py -q
```

SsiPru loader/network data pump runnig for 24395 minutes of 452 160 minutes uptime (314 days) == 5.4% cpu time.

==> Python is efficient!

GPIO process statistics

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        295  0.0  2.7  55012 13944 ?        Ssl   2017 327:12 /usr/
    bin/python3.4 /opt/LichtTimer/LichtTimer.py -q
root        296  0.1  2.9  56628 15112 ?        Ssl   2017 627:50 /usr/
    bin/python3.4 /opt/BeagleZmqGpio/WebGpioZmq.py -q
root        445  0.1  2.2  34728 11272 ?       540847970 Ssl   2017
    573:27 /usr/bin/python3.4 /opt/BeagleZmqGpio/BeagleZmqGpio.py
root        316  0.0  0.4 146656  2276 ?        Ssl   2017   5:26 /usr/
    bin/weborf -x -b /var/www -p 80 -u www
```

GPIO uses negligible cpu-time, quite a lot of RAM (RSS!)