netpp node v0.1 quick reference

A Network Property Protocol evaluation module

Abstract

The *netpp node* is a FPGA based networking module with a default firmware supporting access of registers through a UDP network connection, based on *netpp* – the network property protocol. By default, the netpp node acts as a slave device that is accessed by a netpp master client, such as a process monitor, a Python script or a simple command line utility.

The netpp node can be reprogrammed to have a different personality and be adapted to a custom application. This can either happen on a software/application specific level or on a pure hardware level by using a different set of peripherals.

The netpp node is aware of its properties, meaning, all netpp hardware variations can be queried for their capabilities, i.e. a property list consisting of named items, such as 'Temperature'.

1 Highlights

- 32 Bit processor 'ZPUng' v1.1, three stage pipeline @54 Mhz, 'dagobert' configuration (see Table 1)
- Remote access UART, GPIOs or other exported hardware registers over the network
- Analog backend options:,
  - Up to six channel 10 bit ADC
  - Three channel differential 16 bit Sigma-Delta ADC
- Add-on options (not part of evaluation kit):
  - SDK with GCC and simple downloader
  - High speed UDP streaming support (RTP)
  - Hardware debugging (integrated JTAG ICE)
  - Safety watchdog functionality for critical applications

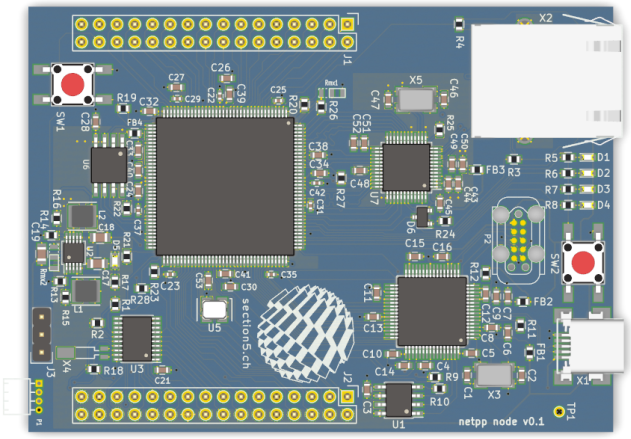


Fig. 1: netpp node top view

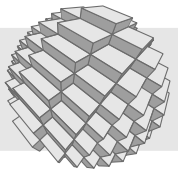
2 Introduction

2.1 Hardware

Table 2 shows the relevant user I/O connectors and interfaces

The J3 pin header (left bottom) determines whether the board is powered from the USB (Jumper position [1-2]) or the expansion connector (J2/VEXT, Position [2-3]). The headers can be mounted either upside (base board configuration) or downside (plug-on configuration). When using as USB-powered base board with a custom piggy back module, make sure to respect the maximum ratings of Table 40.

The msp430 analog and real time clock extension (when present) is powered by the separate VAA pin. When using as a standalone eval board, you can use a jumper on pin [19-20] on J2. For industrial applications,



Tab. 1: netpp node CPU configuration

dagobert SoC specifications		
Memory configuration		
ROM	256kB	Overlay program memory (SPI) read only
L1RAM	16kB	Level one program and data segment
L1CACHE	16kB	Program memory or ROM cache (configureable)
SRAM	2kB	Dedicated Stack Memory
SPAD	2x4kB	Two ScratchPad RAM buffers for peripheral data I/O
Peripherals		
SCACHE	1	SPI flash cache
SPI	2	SPI port
SIC	1	System Interrupt Controller with six peripheral channels, four priority levels
PWM	3	Simple PWM and timer module
DMAA	2	Autobuffer DMA for high speed RX/TX
UART	1	UART console
TWI	1	Two wire interface (I2C compatible)
EMAC	1	Extended Media Access Control Core (MII/MDIO), 10/100M

Tab. 2: netpp node user I/O

Designator	Description
J1, J2	Two expansion headers. For the pinout, please see Section 8.2
SW1, SW2	Push button switches (Reboot/Reset, User button)
D1..D4	LEDs (green, red, yellow, blue)
X1	USB B mini connector (to PC)
X2	100MBit RJ45 connector

there is the option to use a separate, decoupled voltage supply such as a battery.

When the board is powered, LED D5 lights up and some output should appear on the debug console (USB serial, see Section 2.4).

2.1.1 Networking

The network ethernet interface at X2 is configured by default according to Table 3. On the evaluation netpp node firmware ('NetppNode*'), these can be altered using the Python script `cfg_network.py`. See also Section 5.1 for changing network parameters over netpp. Depending on the firmware loaded, a netpp server listens on the indicated port.

Tab. 3: Default network properties

Property / description	Default
Network.IPaddr	192.168.0.5
Network.MACaddr	00:00:de:ad:be:ef
UDP port 'golden' Firmware	2016
UDP port 'deploy' Firmware	2008 (netpp default)
Protocol support	ARP, ICMP ping, UDP

When holding the push button SW2 at boot-up, the default 'golden' firmware is activated and the IP address as well as the MAC address is set to the default. A netpp node can then always be accessed using the URL

```
netpp UDP:192.168.0.5:2016
```

All communication with the netpp node happens over a netpp stack using UDP for transport.

Important: Make sure you have understood the UDP basics and why a packet may not arrive at the peer!

If a reliable connection is required, a specific UDP driver or TCP implementation must be considered as option.

2.2 netpp basics

netpp allows a uniform netpp client to query a netpp capable device's properties. The properties are an abstraction of register values, single bits or even function calls, referred to by names, like `ADC.Value`. When querying a netpp device, the properties represent as a tree, i.e. there is a root node being the device with children nodes that can be traversed recursively.

The important thing to remember:

- Property names use a C style notation
- Elements (children) are accessed by a dot '.' separator, like: `UART.RESET`
- Properties are data type aware and may require strict specification

Further netpp details are found in the netpp HOWTO. For a first start, it is recommended to experiment with the netpp command line interface in Section 3.2.

Some, but not all registers are directly exposed to the netpp server running on the netpp node. In this configuration, there is **no session management**, i.e. several clients can manipulate register bits at the same time.

2.2.1 netpp connections

A netpp device always has a netpp agent (UDP, TCP or other server) listening on some communication interface. For network connections, the agent listens on a specific port. Depending on whether you have an eval kit or a custom firmware edition, the ports may differ, see Table 3.

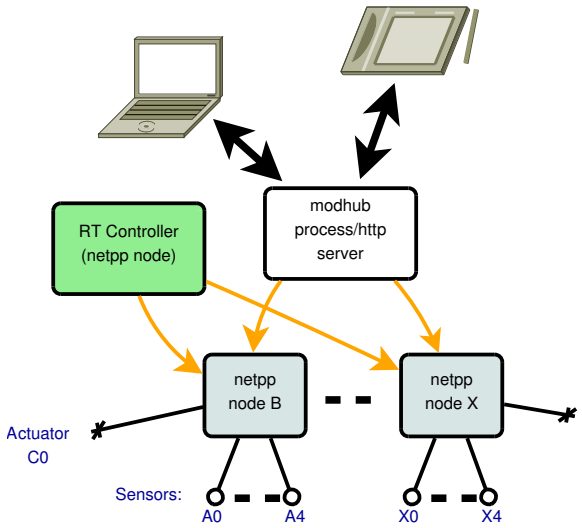


Fig. 2: netpp node mesh/sensor network

2.2.2 Typical network application

A typical netpp node setup is shown in Fig. 2. netpp nodes are slaves by default and are accessed repeatedly by a controller. For plain simple process monitoring of sensor values, a process server polls and possibly caches netpp node values and processes them for display on a normal PC or hand held device. This can also work via a classic http connection using a HTML5 gateway. The orange arrows depict UDP connections with well defined timeouts and may be implemented as a separate, protected network.

For advanced applications, the netpp node can take a master role as well and push data to a **modhub** data distribution daemon (custom application, not part of the eval kit).

When implementing a sensor network of more complexity, note that network layout planning is important and that it is recommended to clearly separate the networks according to traffic and routing expectations.

The netpp node eval kit **does not** have a registered MAC address vendor class, therefore it is recommended to not plug in a netpp node to a running network without verifying that all MAC addresses on the physical LAN are unique. The MAC address can be reprogrammed by the user.

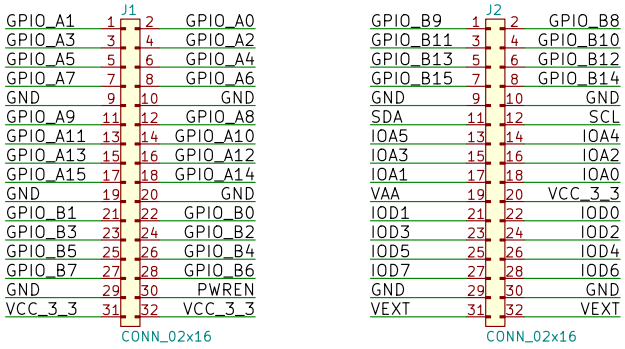


Fig. 3: Pinout netpp node rev0.1

2.3 Pin map

The connectors J1 and J2 (see Fig. 3) provide the I/O signals for expansion hardware:

GPIO_A[0..15]
: General purpose I/O pin port A. Controlled by `GPIO[0]` property.

GPIO_B[0..15]
: General purpose I/O pin port B, multiplexed. Controlled by `GPIO[1]` property.

IOA[0..5]
: Analog input channels

IOD[0..7]
: Digital auxiliary pins. Exposure to properties is custom specific.

SCL, SDA
: I2C bus for external devices

The IO* signals are only relevant when U3 is populated. Depending on the deployed chip variant, functionality is different, see also Section 2.5.

2.3.1 Port multiplexing

The pin functionality of ports A and B is defined by a corresponding MUX bit in the SysCtrl container, as shown in Table 4. Fig. 4 and Fig. 5 show the pin layout in a pseudo register schematic when the corresponding MUX bits are set. The LCDIO multiplexing is not supported in the default eval firmware.

When the SysCtrl::UART_?X_SEL (see Table 14) flags are set, the console UART transmit functionality is deactivated and the pins are routed to the corresponding pins on the Port B connector. However, the receive functionality on the console is always present. When the system is configured for two UARTs, UART0 always remains the console and UART1 is rerouted instead. The peripheral configuration is obtained from Table 1 and can also be

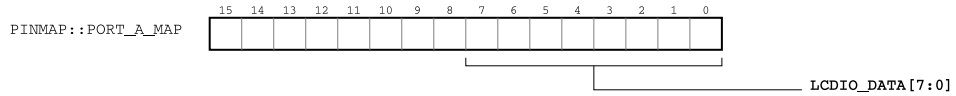


Fig. 4: Pin map GPIO_A

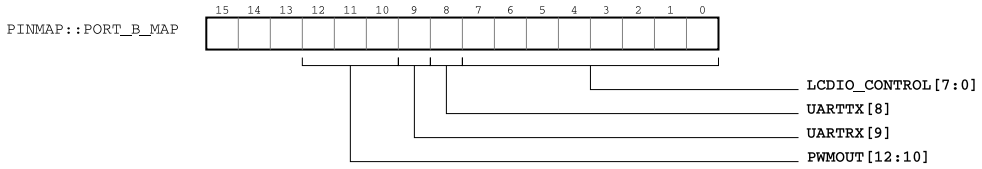


Fig. 5: Pin map GPIO_B

polled from the UART property, if it is an array, `UART.Size` will show you the number of UARTs implemented.

2.4 Debug console

The debug console (UART0) is mapped by default to the USB-to-serial channel, accessible at X1. When plugged in to a PC, a virtual COM device is typically present which can be accessed by a terminal program (via the second TTY port of the device), for example minicom (Linux) or teraterm (Windows). The default communication parameters are: `115200 8N1`. Programming of a new firmware also occurs via the USB connection.

Upon start and a valid ethernet link, the console should output something the text below:

```
------
netpp OS '$VERSION_TAG'
(c) 2004-2019 section5.ch
Reserve Hub 'UDP#1' with 1 ports
Reserve Hub 'UDP#2' with 2 ports
IPaddr: 192.168.0.9:2016
Link active
```

The debug console can be turned off by setting the Property `SysCtrl.ConsoleOn` to 0 (false).

2.5 Optional analog inputs

U3 on the PCB can be populated with a few msp430 variants that are programmed via the P1 port. Depending on the model, extra functionality is available. Table 5 shows the supported configurations. By default, TWI communication is implemented towards the msp430 units, unless noted otherwise. In the default configuration, the msp430 are just used as intelligent ADCs without any filtering, however they can be reprogrammed by a developer for special purposes. The sampling value query is limited by the number of asynchronous I2C requests.

The effective maximum sample rate towards the netpp node SoC is determined by the communication interface speed. See also interface specifications in Table 6.

Tab. 6: Interface specifications

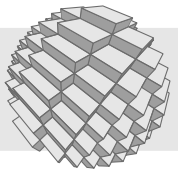
Interface	Typical sample rate limit
SPI (msp430 master)	200 ksps (10 bit)
I2C (msp430 slave)	1 ksps (10 bit)
UART (bidirectional)	200 ksps (8 bit)

If a msp430 extension is present, it can be probed through the field below. The analog extension type is encoded in the netpp node device name. For supported netpp node configurations, see also Section 7.1.1.

`SysCtrl.MSP430Type`
: Read-only 'STRING' property listing the identifier of the installed msp430

2.6 Real time clock

When X4 is populated with a 32768 Hz quartz and the `RTC` property container is present, the system can run a real time clock (optionally battery powered) via the VAA pin. The system can be altered by modifying the Rmx2 resistor configuration such that the external PWREN pin can be used by the real time clock logik to turn on power at specific occasions as well as soft powerup/powerdown functionality.



Tab. 4: Pin map GPIO_B

Pin	MUX = 0	MUX = 1	MUX Property
J1[22..24]	GPIO[1][0..2]	PWM[0..2] (out)	PWMxOutEn
J2[2]	GPIO[1][8]	UART0_RX (in)	UART_RX_SEL
J2[1]	GPIO[1][9]	UART0_TX (out)	UART_TX_SEL

Tab. 5: Supported MSP430 types

msp430	ADC/Ch	ADC type	I2C	SPI	UART
F2012	5	10 bit SAR, 200 ksps	y	(y)	n
F2013	3(4)	16 bit sigma delta	y	(y)	n
G2553	5	10 bit SAR, 200 ksps	y	y	y

3 Software

3.1 Installation

When using the section5 netpp node SDK docker container, all relevant netpp utilities are preinstalled. However, there are also specific clients available as installer package.

3.1.1 Windows 32 bit

Please follow the instructions from http://section5.ch/index.php/2017/09/12/netpp-for-windows-quick-start/

Note that a property transaction may fail using UDP (default firmware). This can happen more often when:

- The netpp node is not directly connected to the PC (i.e. a router is in between)
- The network traffic is high
- The netpp client is run from within a virtual machine

3.2 netpp command line interface

The netpp command line interface is a session based console tool to manipulate remote netpp devices.

1. Run the netpp command line interface:

```
netpp-cli UDP:192.168.0.5:2016
```

2. Type **?** to see all properties:

```
Connected to 'NetppNodeSD16' device class
Child: (u) [20000001] 'ADCValue'
Child: (S) [20000002] 'SD16'
Child: (S) [00000001] 'SysCtrl'

...
```

3. Type a property name to see its value or possible children:

```
netpp> Flash
'Flash' not in cache, querying...
Type : {Struct} [RW.]
Child: (c) [0000000b] 'EraseSector'
Child: (u) [0000000c] 'Addr'
Child: (B) [0000000d] 'Buffer'
Child: (m) [0000000e] 'Mode'
Child: (u) [0000000f] 'CRC'
```

The properties have specific data types and follow a classic namespace scheme found in most programming languages. Properties that have children can be containers whose members are accessed by the dot notation, like `SysCtrl.RESET`. In particular, there are the following complex types:

STRUCT {Struct}
: This container has arbitrary children. They are simply accessed using a '.' notation, like `Flash.Addr`.

ARRAY {Array}
: This container always has a Size property. The second property is a prototype of the actual array items. When accessing an item instance, the indexing notation '[i]' is used, like: `GPIO[0].Dir`. The prototype property itself is no more specified in the index notation.

For a detailed explanation of the netpp master command line interface, see the netpp HOWTO (Section 9).

In Section 7 you find a reference of all properties existing on this netpp target.

3.2.1 Device classes

Depending on the installed analog backend, different device variants exist with a possibly different property

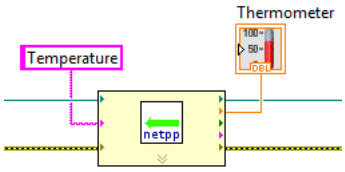


Fig. 7: netpp for LabView 'get property'

set. The device name represents the corresponding device class ('NetppNodeSD16' in the above example). See Section 7.1.1 for supported devices. The base device class properties (Section 7.2) are common to all netpp node devices.

3.3 Python

For scripted remote control, a Python API is included in the SDK.

3.4 HTML5 gateway add-on

For embedded targets with Python and TCP support, a web interface can be dynamically derived from a the netpp property list, running with a tiny memory footprint. This enables remote control from a mobile device via a Raspberry Pi or some industrial embedded Linux solution, for instance. HTML code can also be embedded into the netpp node using simple string template properties.

3.5 Third party tool integration

3.5.1 process view server (pvhub)

For process monitoring or control using a master PC, a server utility is available to display a graphical user interface for the netpp node. This can be accessed through the publicly available 'pvbrowser'. It is also included in the SDK docker container. An example is provided for the default stock firmware.

3.5.2 NI Labview

Based on the OpenG Python scripting VI plugin, a netpp VI wrapper is available that allows access from within Labview. For example, a value query works simply by the schematic in Fig. 7.

3.5.3 OpenLab

This is an OpenSource Labview-like environment. Netpp Java wrappers are available for this project for interested Java Developers under an Open Source license.

3.6 msp430 firmware development

For msp430 firmware development, there is no 'built-in' support. Experienced users however have the possibility to flash the on-board msp430 using a SBW adapter such as the msp430 LaunchPad or a TI FET430UIF. Table 7 below shows the driver names supported by the mspdebug tool (Linux SDK).

Tab. 7: msp430 SBW support table

msp430 type	Driver
G2553	rf2500
F201x	uif

4 Usage

This section covers the basic functionality and property manipulation sequences for the built-in peripherals.

4.1 LED test

- Set `SysCtrl.TEST_EN` to enable LED testing
- Set the `LED` properties
- Clear `SysCtrl.TEST_EN` to activate system status LED mode

Note that when the system is compiled in debug mode, the internal state machine modifies the Status LEDs, thus possibly interfering with user settings. See for the different LED roles in dependence of the operation mode.

Tab. 8: LED role

Dbg	TEST_EN	D1[G]	D2[R]	D3[Y]	D4[B]
*	0	off	Break	Emu	IRQ
No	1	Green	Red	Yellow	Blue
Yes	1	Link up	Init..	Illegal Pkt	Blue

When UDP packets are sent to the netpp node other than to the netpp service, D3 will light up in Debug mode and turn off when the next valid packet is received.

4.2 Analog inputs (ADC10)

This section covers the ADC10 variant of the netpp node only.

The analog inputs [A0..A5] are by default available via the IOA0..IOA5 pins as shown in the schematic Fig. 10. Prior to sampling, the ADC is configured as follows:

- Turn `ADC.ENC` off

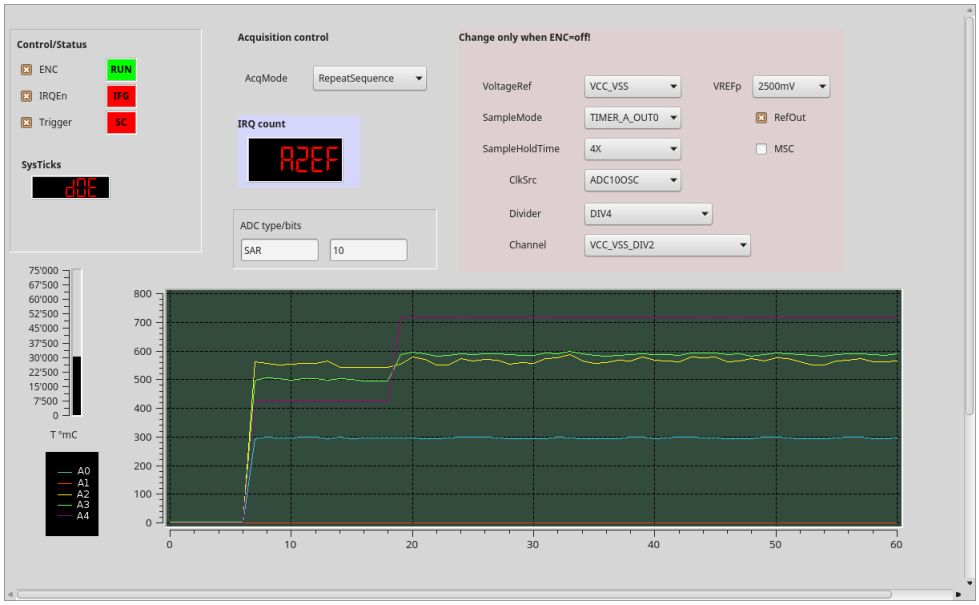


Fig. 6: pvbrowser screen shot

- Configure conversion mode: `ADC.SHS`.
- Choose reference voltage `RefVoltage` and reference voltage mode `VrefMode`.
- Turn on `ADC.ENC` to start ADC

The `ADC.SHS` property determines the trigger behaviour:

MANUAL_TRIGGER
: In this mode, a `Trigger` toggle is required to start a conversion

TIMER_A modes
: The timer A determines the timing of the sample conversion. It is preset at approx. 5kHz in the default firmware and has limited timing accuracy. In the demo firmware, there is no difference between the TIMER_A modes, timed sampling is not supported.

4.2.1 Channels and References

Apart from the external analog inputs (`A0..A4`), internal voltages can also be selected for testing and calibration. This is done by setting the `ADC.Channel` property in Debug mode (`ADC.AcqMode` = RepeatSingle). Example scenarios:

- When `ADC.VrefMode` is 0 ('VCC_VSS') and channel 11 is selected, the measured ADC value should be around 512.
- When in 'VREFp_VSS' mode and `ADC.RefVoltage` = '2500mV', the value should be around 725.

For external voltage reference modes, pins (IO)A3 and A4 take an alternate role as displayed in Table 9. When `RefOut` is set, the pins are turned into outputs of the `RefVoltage`. A5 can be used as an option on HW rev 0.1 when R28 is not populated and the msp430 is not configured for SPI mode.

Tab. 9: VRef roles

Pin	VeREF[m,p] mode	RefOut
A3	VeREF- (input)	VREF- (output of RefVoltage)
A4	VeREF+ (input)	VREF+ (output of RefVoltage)

Detailed information for direct register access is found in the msp430g2553 datasheet at the Texas Instruments website www.ti.com and the family datasheet `slau144j.pdf`.

4.3 Analog inputs (SD16)

This section covers the SD16 variant of the netpp node.

The analog inputs [A0..A1] are by default available via the IOA0..IOA4 pins as shown in the schematic Fig. 10.

For a quick start, follow the steps below:

1. Select channel 5 ('AVcc_AVssby11') in `SD16.Channel`.
2. Make sure `SD16.SSEL` is set to 0 ('MCLK') or 1 and `SD16.REFON` is set.
3. Turn on the `SD16.SC` ('Convert')

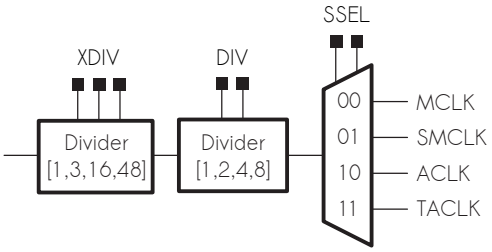


Fig. 8: Clock selection and division

4. Poll the `ADCValue` property

Unlike the ADC10 units, the SD16 uses differential inputs according to Table 10.

Tab. 10: Channel functions

Channel	Input pin pair	Comment
0	IOA0+, IOA1-	-
1	IOA2+, IOA3-	-
2	IOA4+, IOA5-	Not usable on HW revision 0.0, see also Section 8.3.3
3	IOA6+, IOA7-	Not supported (I2C)
4	IOA1+, IOA2-	-

In order to avoid artefacts, sampling according to a specific input clock may be required by the application. Fig. 8 shows the input clock schematic and mode selection.

For detailed function of the analog input pins, please see SD16 documentation in the family datasheet `slau144j.pdf`.

4.4 PWM

The dagobert SoC has three simple PWM controllers built in. PWM0 has a special role and is connected to the system timer. The PWM property is an array of a property structure where each members is a direct proxy to the hardware registers. A short stepwise example:

1. Set `Period` to period length in cycles minus one
2. Set `Width` to pulse width (must be <= `Period`)
3. Set `PWMStartAll` to start PWMs simultaneously
4. Poll `Count` to verify the PWM is running
5. Stop single PWM (except unit 0) using `PWM1Stop`, `PWM2Stop`

On the default firmware, the timer unit 0 (corresponding to PWM[0]) also functions as system timer. It is therefore recommended to not change the `PWM[0].Period` value, as this would change the timeout behaviour of the netpp main loop and protocol handler.

4.5 UART I/O

The UART0 is by default used as console I/O over the built-in USBserial (`/dev/ttyUSB1` on Linux). To turn off the console and use it for raw I/O, disable the `SysCtrl.ConsoleOn` property. Also, you might want to reroute the UART to the I/O pins using the port muxer, see Section 2.3.

- Pull UART into reset using `UART.UART_RESET`.
- Configure `UART.Bps`
- Send data by setting the `UART.TxData` property. The UART buffer has a limited size, when the error `DCERR_PROPERTY_SIZE_MATCH` ('Property size mismatch ') is received, you will have to break up the buffer in chunks.
- Receive data by querying the `UART.RxData` property. This returns the currently available bytes in the buffer up to a maximum length. Read repeatedly until you receive an empty string.
- Check for possible errors (readonly-Flags of the UART container property) when getting a bad return code

4.6 Push buttons

The SW1 button is hard wired to the FPGA reset. The SW2 user button function is determined by the firmware and is by default configured as custom configuration bypass. In case an IP address was configured and can not be determined via the debug console, holding SW2 during reboot uses the default network properties from Table 3.

4.7 I2C interface

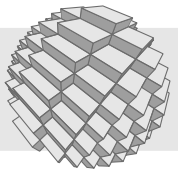
The I2C port's SCL and SDA pins are exposed on the J2 header (see Fig. 3). When attaching an external device, you should run through this check list:

- Pullups required? (see Section 8.3.3). When a msp430 (U3) is present, the internal pullups are normally sufficient and R1, R2 are not populated.
- i2c slave address not equal 0x10 (when msp430 unit populated)?
- netpp description for this device present?

Note that there is no default support in the eval firmware for external devices.

4.8 Flash access

The SPI flash can be written using simple property accesses. Follow these steps to write data to the user flash area (see Table 12) using the Flash container properties:



1. Set Flash.Addr to the sector to erase (if erasing required)
2. Set Flash.EraseSector command to any value
3. Set Flash.Buffer with the data. Flash.Addr will autoincrement by the size of the written buffer.

Likewise, data can be read by setting Addr and reading the Buffer property.

See also flash.py script (Section 5.2.2).

5 Configuration

This section covers the boot-up configuration of the netpp node and firmware updates.

5.1 Boot Configuration

Some internal (as well as netpp-exported) parameters have a user shadow instance in a non-volatile flash (NV) area. If the SysCtrl.StoreNV command property is set to 1, the current configuration is stored into the NV area and is active on the next Reboot. See Table 11 for the configuration properties. In the firmware source, more non-volatile configuration variables can be added by specific C compiler attributes.

Examples:

5.1.1 Change IP address

To change the IP address, use this property setting sequence, for example using netpp-cli:

```
$ netpp-cli UDP:192.168.0.26:2016
netpp> Network.IPaddr 0xc0a80009
netpp> SysCtrl.StoreNV 1
netpp> SysCtrl.Reboot 0
```

Upon reboot, all further commands will cause a communication timeout and a new connection will have to be initiated.

5.1.2 Enable custom firmware

Assume, you have loaded your custom firmware using flash.py (see Section 5.2.2) into the user area and tested it by setting SysCtrl.Reboot to 1. To permanently boot this firmware, use this sequence:

```
netpp> SysCtrl.BootMode 1
netpp> SysCtrl.StoreNV 1
netpp> SysCtrl.Reboot 0
```

5.2 Firmware update

5.2.1 Local update

To download a firmware (Bit-File) onto the target, you typically run the **make download** command in your platform project directory of the SDK, like $(SDK)/syn/xilinx/netpp_node/. This is currently supported in Linux only.

When the SDK is not present, you will need a customized **papilio-prog** executable. Then you can download a bit file for testing:

```
papilio-prog -f netpp_node_fw.bit
```

For permanent installation, you need to write the bit file into flash using

```
NETPP_NODE=$MASOCIST_SDK/syn/xilinx/netpp_node
papilio-prog -b $(NETPP_NODE)/bscan_spi_lx9.bit \
 -s a -f netpp_node_fw.bit
```

5.2.2 Network update

Using the Flash property, sectors on the SPI flash can be modified, like for a secondary image to boot from. This secondary image is a standard FPGA binary generated from a bit file. This file is written to the UserROM area of the SPI flash (see Table 12). Upon setting the SysCtrl.Reboot property to 1 (see also BootMode choices), the system will warm-boot into this new image. If the flash was not written correctly, the boot procedure will fall back into the default BootROM image (which should never be modified). Setting SysCtrl.Reboot to 0 will reboot into the default BootROM image, provided that the property is existing in the current design.

If the system fails due to an invalid firmware image, it can be restored to default bootloader mode by holding USER_RESET (SW2) upon reset (SW1). It is then accessible by the default network configuration.

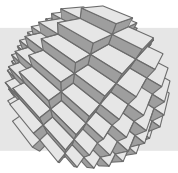
Note that on the netpp node eval platform, Flash access to all areas is enabled by default.

An example on how to upgrade and boot into a custom image is found in the SDK docker container:

```
cd $HOME/src/netpp/devices/netpp_node
make all DEPLOY=y # Build deployment version of firmware
make image        # Build binary
python flash.py   # Write into flash
```

6 Troubleshooting

When SysCtrl.TEST_EN is not enabled, the LEDs D1..D4 display the current system Status, see also LED property. To test for errors during development, make sure TEST_EN is off.



Tab. 11: Configuration properties

Property	Description
Network.IPaddr	IPv4 address
Network.MACaddr	Hardware address, immutable from Bootloader
SysCtrl.BootMode	Boot mode (0: boot loader, 1: deployment firmware)

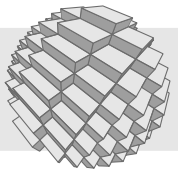
Tab. 12: SPI flash map

ID[Size]	Addr	Access	Description
BootROM	0x000000	RO	Boot ROM area (FPGA image)
CodeROM	0x054000	RO	Code ROM
ConfigBase	0x0C0000	RW	Configuration area
UserROM	0x0D0000	RW	User ROM area (second FPGA image)

System no longer responding and Red LED on
: System entered a **BREAK** condition. When the system is in BREAK state, the cause can be only determined by the hardware debugger interface. Otherwise, reset the system using the SW1 reset button. This condition is ONLY to be expected in experimental, non-verified evaluation firmware or when a hardware mismatch is detected.

Communication errors and slow response
: Check the blue D3 LED for high packet/IRQ activity, likewise verify the LEDs on the connector or use Wireshark to monitor packet rates.

Green LED on X2 connector blinking fast, no response from target
: The system was reset and is not booting correctly. If a manual reset does not recover, the firmware may have been damaged and reprogramming may be required.



7 Property reference

This section contains a list of the available properties, listed by group. The property naming follows a particular style, for example:

`BIT_SPECIFICATION`
: A property that maps directly to a register bit (field)

`MoreAbstractProperty`
: A more abstracted property that does not necessarily match a hardware register

This property reference is valid for the 'NetppNode*' classes with:

Device Revision: 0.4

Make sure to check against the device revision (`DeviceRevision` property), if you find a mismatch between documentation and device properties.

Note that with 0.x version (developer) releases, the properties can change without warning and the DeviceRevision property is not present. If you are developing a GUI application based on a specific property set, make sure to handle non-existing properties or synchronize explicitely using the builtin checksum property (See `TOKEN_CHECK` in the netpp programmer API documentation).

7.1 Reading the property reference

As the Property interface is hierarchy aware, there are so called 'top level properties' inside a device's property list query that is normally occuring at the start of a session.

All top level properties seen from a query are listed in groups in this reference. If a top level property is more complex (like a STRUCT, ARRAY, or MODE property) or has documented children in general, it may refer to another table containing the description of its children. In the PDF file of this documentation, simply jump to the table of the referred property by clicking on the property (possibly marked as a hyperlink in some PDF viewers).

The full property identication for a specific property represents the entire node hierarchy, for example:

```
PWM[0].Invert
```

Specific for mode properties, legal values can be queried from the 'choice' children of a MODE property, for example in netpp-cli:

```
netpp> ADC.AcqMode.Single
'ADC.AcqMode.Single' not in cache, querying...
Type : Mode [R..]
Value: #0
netpp> ADC.AcqMode 0
```

7.1.1 Derived device classes

The default netpp node device 'NetppNodev1' is considered the 'Base class', as all netpp nodes have these properties. When an analog extension is present, the devices have different names and additional properties, as found in the corresponding sections:

NetppNodeADC10
: 10 bit SAR analog extension, see Section 7.3.

NetppNodeSD16
: 16 bit sigma delta analog extension, see Section 7.4.

7.2 Base class properties

The base class (NetppNodev1) properties

7.2.1 System configuration

Tab. 13: Property group 'System configuration'

Property	Type	Flags	Description
`SysCtrl`	STRUCT	RW	System control container
`Network`	STRUCT	RW	Network parameters, non-volatile
`Flash`	STRUCT	RW	Direct low level flash access. Experimental.

Configuration and status query
See Table 13

Tab. 17: Mode `SysCtrl.BootMode` – possible values

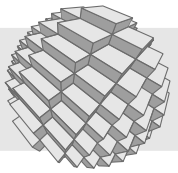
Value	Mode name	Description
0	BOOTLOADER	Boot into bootloader (primary firmware)
1	SECONDARY	Boot into secondary firmware

7.2.2 GPIO

Port input/output control properties that are non-system specific.
See Table 18

7.2.3 Low Level register access

Properties for direct low level register access
See Table 22



Tab. 14: Struct SysCtrl

Property	Type	Flags	Description
TEST_EN	BOOL	RW	Enable LED testing (user access)
HWRevision	REGISTER	RO	Hardware version register
Reboot	COMMAND	WO	Issues reboot (0: bootrom, 1: secondary image)
ConsoleOn	BOOL	RW	Set to enable console (Default = on)
MSP430Type	STRING	RO	MSP430 type identifier
ReleaseTag	STRING	RO	System release version tag
BootMode	MODE	RW	Boot mode (non-volatile)
StoreNV	COMMAND	WO	Store configuration in Flash

Tab. 15: Struct Network

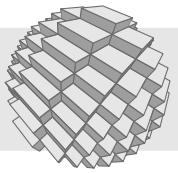
Property	Type	Flags	Description
MACaddr	BUFFER	RO	MAC (hardware ethernet) address
IPaddr	REGISTER	RW	IPv4 address, in 32 bit big endian format

Tab. 18: Property group 'GPIO'

Property	Type	Flags	Description
UartTxEn	BOOL	RW	Enable UartTX on Port B
UartRxEn	BOOL	RW	Enable UartRX on Port B
PWM0OutEn	BOOL	RW	When set, enable PWM function on GPIO
PWM1OutEn	BOOL	RW	When set, enable PWM function on GPIO
PWM2OutEn	BOOL	RW	When set, enable PWM function on GPIO
PWM1Stop	COMMAND	WO	Stop PWM1
PWM2Stop	COMMAND	WO	Stop PWM2
PWMStartAll	COMMAND	WO	Start all PWMs synchronously
UART	STRUCT	RW	UART I/O handling for primitive buffer based remote control
GPIO	ARRAY	RW	The GPIO ports
PWM	ARRAY	RW	The PWM unit array property

Tab. 22: Property group 'Low Level register access'

Property	Type	Flags	Description
LED	STRUCT	RW	User LEDs D1..D4



Tab. 16: Struct Flash

Property	Type	Flags	Description
EraseSector	COMMAND	WO	Erase sector at address 'Addr'
Addr	REGISTER	RW	Flash address pointer
Buffer	BUFFER	RW	Flash data buffer. Reads/writes a chunk of bytes from/to 'Addr'.
Count	INT	WO	Count of bytes for some operations

Tab. 19: Array item GPIO[i]

Property	Type	Flags	Description
In	REGISTER	RO	GPIO input register
Dir	REGISTER	RW	GPIO direction register, set '1' for output
Out	REGISTER	RW	GPIO output register
Set	REGISTER	WO	Atomic GPIO SET register, write one to set corresponding bit
Clr	REGISTER	WO	Atomic GPIO CLEAR register, write one to clear corresponding bit

Tab. 20: Array item PWM[i]

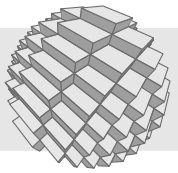
Property	Type	Flags	Description
Invert	BOOL	RW	When set, invert PWM output
Count	REGISTER	RO	Current counter value
Width	REGISTER	RW	PWM pulse width register, should be smaller than Period
Period	REGISTER	RW	PWM period register
Running	BOOL	RO	Set when the PWM/Timer is running

Tab. 21: Struct UART

Property	Type	Flags	Description
RXREADY	BOOL	RO	Set when data ready in RX FIFO
TXREADY	BOOL	RO	Set when TX FIFO ready for data
TXBUSY	BOOL	RO	'1' when TX in progress
FRERR	BOOL	RO	Sticky framing error. Set when stop bit not null. Reset by UART_RESET.
TXOVR	BOOL	RO	Transmitter FIFO overrun. Cleared by UART_RESET.
RXOVR	BOOL	RO	Receiver FIFO overrun. Cleared by UART_RESET.
UART_RESET	BOOL	RW	'1': Reset UART. Clear to run.
Bps	INT	RW	Bit per second. When writing this value, running transfers might end up with broken data.
TxData	BUFFER	WO	Buffered write to UART_TX
RxData	BUFFER	RO	Buffered read from UART_RX

Tab. 23: Struct LED

Property	Type	Flags	Description
Green	BOOL	RW	Green: Ready
Red	BOOL	RW	Red: Error/Break
Yellow	BOOL	RW	Yellow: Warning
Blue	BOOL	RW	Blue: Activity.



7.3 ADC10 extension

The additional properties for the 'NetppNodeADC10' device

7.3.1 ADC10 Analog I/O

Tab. 24: Property group 'ADC10 Analog I/O'

Property	Type	Flags	Description
ADC	STRUCT	RW	ADC parameters
ADCBuf	ARRAY	RW	ADC channel value array

Properties for analog I/O based on the msp430 extension

See Table 24

Tab. 26: Mode ADC.VrefMode – possible values

Value	Mode name	Description
0	VCC_VSS	Range V_CC to V_SS
1	VREFp_VSS	V_REF+ to V_SS
2	VeREFp_VSS	V_eREF+ (external) to VSS
3	VeREFpbuf_VSS	Buffered V_eREF+ to VSS
4	VCC_VREFm	V_CC to VREF-
5	VREFp_VREFm	V_REF+ to V_REF-
6	VeREFp_VREFm	V_eREF+ (external) to VREF-
7	VeREFpbuf_VREFm	V_eREF+ (external, buffered) to VREF-

Tab. 27: Mode ADC.SHS – possible values

Value	Mode name	Description
0	MANUAL_TRIGGER	Software-Trigger (via 'Trigger' property)
1	TIMER_A_OUT1	TIMERA1 output
2	TIMER_A_OUT0	TIMERA0 output
3	TIMER_A_OUT2	TIMERA2 output

Tab. 28: Mode ADC.SHT – possible values

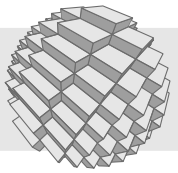
Value	Mode name	Description
0	4X	Four clock cycles
1	8X	Eight clock cycles
2	16X	..
3	64X	

Tab. 29: Mode ADC.RefVoltage – possible values

Value	Mode name	Description
0	1500mV	1.5V Reference voltage
1	2500mV	2.5V Reference voltage

Tab. 30: Mode ADC.ClkDiv – possible values

Value	Mode name	Description
0	DIV1	No division
1	DIV2	Clock divided by 2
2	DIV3	..
3	DIV4	
4	DIV5	
5	DIV6	
6	DIV7	
7	DIV8	



Tab. 25: Struct ADC

Property	Type	Flags	Description
Reset	BOOL	RW	
Busy	BOOL	RO	When set, ADC is running
VrefMode	MODE	RW	Voltage reference selection
MSC	BOOL	RW	When set, auto-repeat conversion regardless of timer frequency. Turn off by default.
SHS	MODE	RW	Conversion trigger mode
SHT	MODE	RW	Sample&Hold timing
ENC	BOOL	RW	Encoding when 1. Turn off to change ADC settings.
RefVoltage	MODE	RW	Selection of internal reference voltage
RefOut	BOOL	RW	Enable reference voltage output on A3/A4
ClkDiv	MODE	RW	Clock divider for analog conversion input clock
IRQen	BOOL	RW	IRQ enable. Upon completion of a conversion, the IFG property is set, when this bit is enabled, the corresponding IRQ routine is entered.
IFG	BOOL	RO	Interrupt flag. When IRQen=FALSE, it remains flagged until cleared by software routines.
Trigger	BOOL	WO	Software trigger to start conversion
SC	BOOL	RO	Single conversion trigger ACK bit. Cleared after successful single trigger action.

7.4 ADC_SD16 extension

These are the additional properties of the 'NetppNodeSD16' backend.

7.4.1 SD16 Analog I/O

Tab. 31: Property group 'SD16 Analog I/O'

Property	Type	Flags	Description
ADCValue	REGISTER	RO	ADC 16 bit value
SD16	STRUCT	RW	ADC parameters

Properties for analog I/O based on the msp430 extension

See Table 31

Tab. 33: Mode SD16.Gain – possible values

Value	Mode name	Description
0	X1	No amplification
1	X2	Gain x2
2	X4	..
3	X8	
4	X16	
5	X32	

Tab. 34: Mode SD16.Channel – possible values

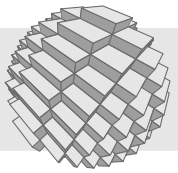
Value	Mode name	Description
0	A0p_A1m	Inputs A0+,A1-
1	A2p_A3m	Inputs A2+,A3-
2	A4p_A5m	Inputs A4+,A5-
3	A6p_A7m	N/A with i2c interface
4	A1p_A2m	Inputs A1+,A2-
5	AVcc_AVssby11	A5 - (AVcc - AVss)/11
6	TEMP	Temperature channel
7	PGAOFFS	PGA offset measurement

Tab. 35: Mode SD16.BufMode – possible values

Value	Mode name	Description
0	OFF	High impedance buffer off
1	SLOW	Slow speed/current
2	MEDIUM	Medium speed/current
3	HIGH	High speed/current

Tab. 36: Mode SD16.OSR – possible values

Value	Mode name	Description
0	OSR256	256x oversampling
1	OSR128	..
2	OSR64	
3	OSR32	



Tab. 32: Struct SD16

Property	Type	Flags	Description
Gain	MODE	RW	Preamplifier gain
Channel	MODE	RW	Analog input channel
BufMode	MODE	RW	High impedance buffer
SingleConv	BOOL	RW	Single conversion
OSR	MODE	RW	OversamplingRate
Unipolar	BOOL	RW	Unipolar measurement format (see SD16 chapter in slau144j.pdf)
IFG	BOOL	RO	Interrupt flag for ADC conversion
IE	BOOL	RW	Enable IRQ upon conversion result
REFON	BOOL	RW	Enable 1.2V reference. Must be enabled for conversion.
VMIDON	BOOL	RW	Enable VRef output on IOA3
DIV	MODE	RW	Clock division register bitfield 0
XDIV	MODE	RW	Clock division register second divider
SSEL	MODE	RW	Conversion clock selection
SC	BOOL	RW	Start Conversion (toggle for single conversion mode)

Tab. 37: Mode SD16.DIV – possible values

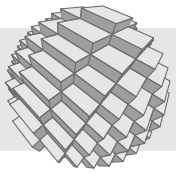
Value	Mode name	Description
0	DIV1	
1	DIV2	
2	DIV4	
3	DIV8	

Tab. 38: Mode SD16.XDIV – possible values

Value	Mode name	Description
0	DIV1	
1	DIV3	
2	DIV16	
3	DIV48	

Tab. 39: Mode SD16.SSEL – possible values

Value	Mode name	Description
0	MCLK	Main clock
1	SMCLK	Sub-Main clock
2	ACLK	Auxiliary clock (32768 Hz)
3	ExtTACLK	External TACLK (Pin IOA0)



8 Technical specifications

8.1 Electrical

Table 40 shows the electrical characteristics and maximum ratings.

Tab. 40: Electrical characteristics

Electrical characteristics		
	Standard operation	
V_{DD}	5V	Supply voltage (USB) or J2:VEXT pin
V_{IO}	3.3V	I/O voltage
I_{idle}	104mA	Power consumption with PHY disabled, CPU@54Mhz
I_{net}	154mA	Typical power consumption of bare netpp node (54 MHz, no peripherals) during network operation
	Maximum ratings	
I_{max}	500 mA	Maximum current that board including piggy back can draw from USB connector
V_{DD}	6 V	Absolute maximum supply voltage from J2:VEXT pin

8.2 Drawings

- Top level schematic: Fig. 10
- Mechanical outline: Fig. 9

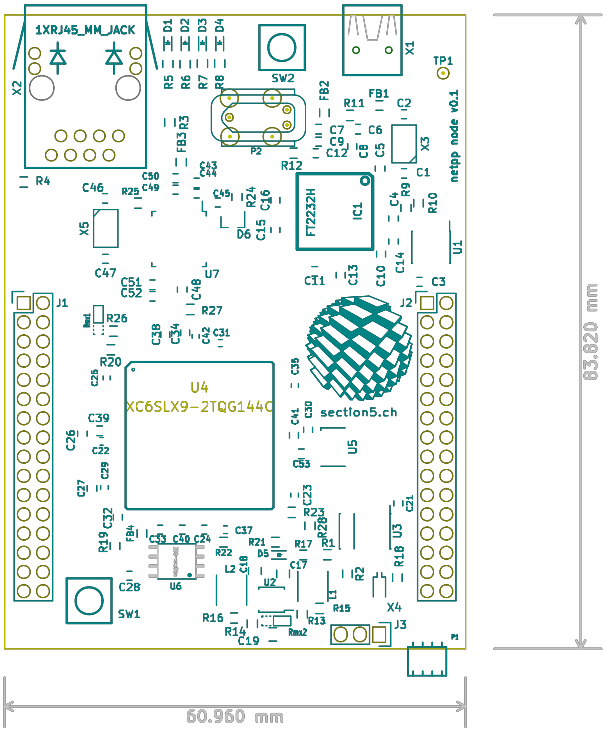


Fig. 9: Mechanical outline

8.3 Revision/changes

The HW revision tag is found on the bottom layer underneath X2. Do not confuse with the netpp node version listed on the top of the board near TP1.

8.3.1 Hardware Revisions

Rev 0.0proto First Prototype, limited releases

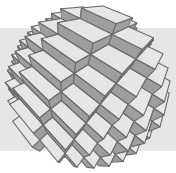
Rev0.1 Changes:

1. Improved capacitor buffering
2. PAD changes U5
3. IOA5 option
4. J2 layout changed (SCL and SDA exported)

8.3.2 dagobert SoC revisions

The dagobert CPU/System on chip revisions are determined by the `SysCtrl.HWRevision` property. The lower 16 bit word denotes the major:minor revision bytes, i.e. 0x0003 stands for Rev 0.3.

The complete errata and revision specification is found in the dagobert SoC manual (Section 9).



0.1 development revision First spartan6 backport, limited releases

0.2 'Improved IRQ' Removed JPEG I/O signals, improved network I/O

0.3 LCD engine support Added LCD I/O engine and port muxing for Port A

8.3.3 Population options

R1, R2
: Use 1.5k resistors for bus pullup when needed (no U2 installed)

R28
: Use 0-100 Ohm for SPI mode setup (custom FPGA image required)

Rmx1
: HSWAP behaviour, please refer to the Spartan6 datasheet

Rmx2
: Option for external power control (msp430 soft start/FPGA shutdown)

9 Further pointers

Web link
: http://section5.ch/index.php/2017/08/21/netpp-node-evaluation-platform/

netpp HOWTO
: http://section5.ch/index.php/dokumentation/

netpp node SDK evaluation docker container
: http://section5.ch/index.php/product/masocist-evaluation-docker-container/

Reference designs
: netpp node piggy back KiCAD example designs on request

dagobert Hardware Reference `soc-dagobert.pdf`
: On request only (NDA required)

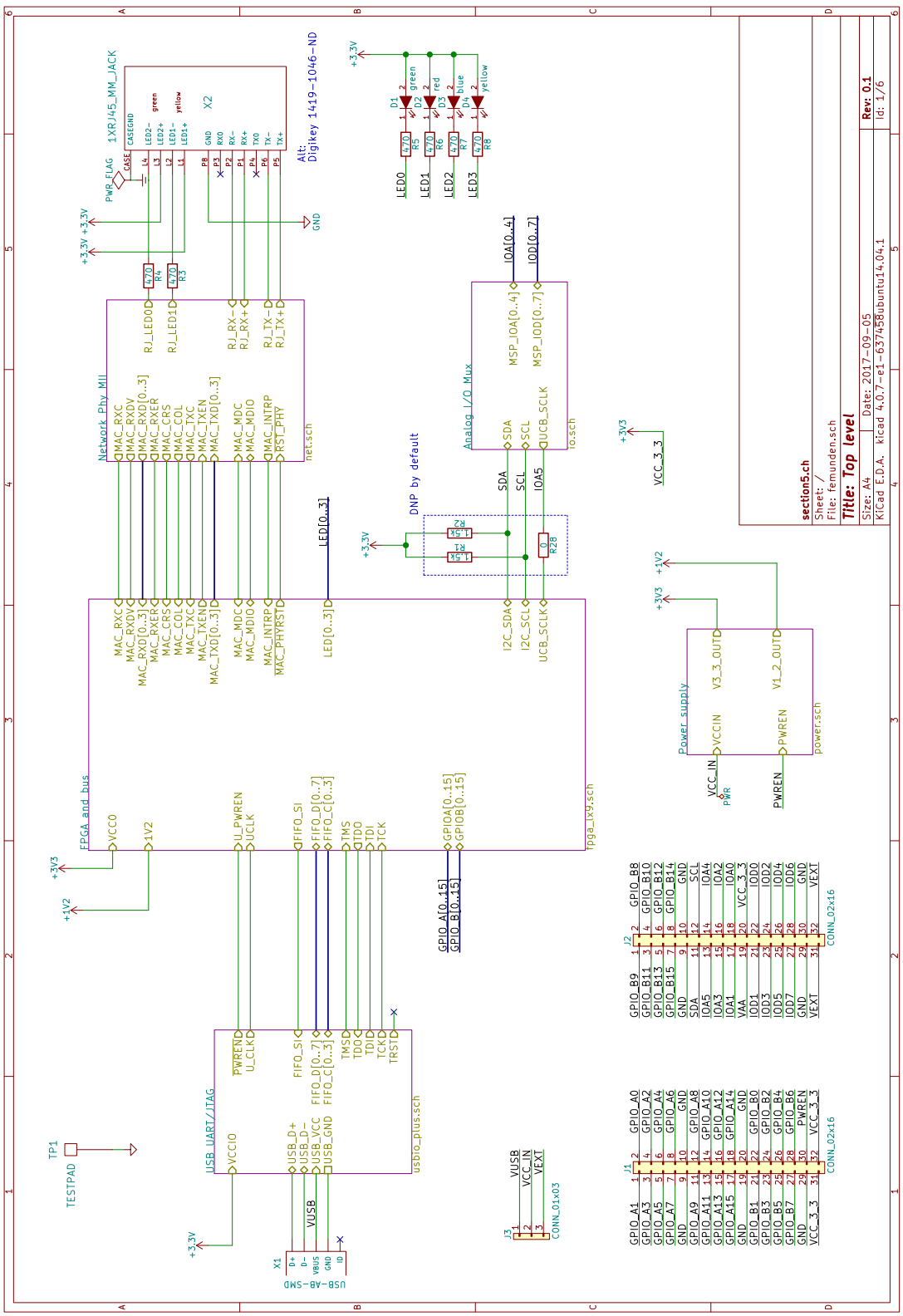


Fig. 10: Top level schematic