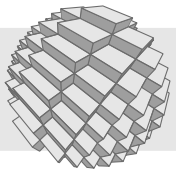


netpp node [PRELIMINARY reference]

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'.

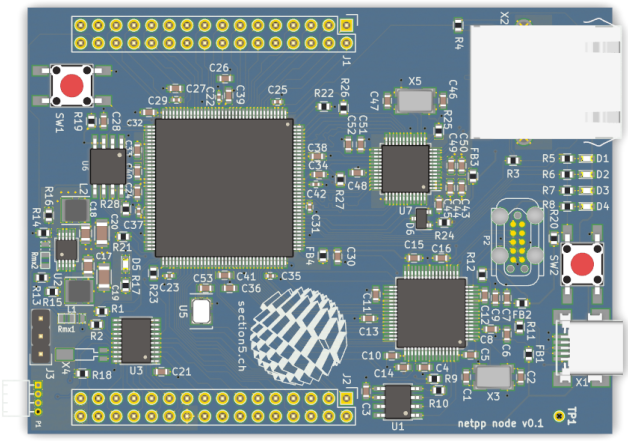


Fig. 1: netpp node top view

1 Highlights

- 32 Bit processor 'ZPUng' v1.1, three stage pipeline @54 Mhz, 'dagobert' configuration (see Table 1)
- Remote access UART, i2c or other exported hardware registers over the network
- Up to 1200 netpp requests per second
- High speed UDP streaming possibilities (RTP)
- Add-on: SDK with GCC, gdb (integrated USB-JTAG-Debugger)
- Separated processing of network and safety relevant circuitry
- Analog backend options: 5-channel 10 bit ADC, 3-channel differential 16 bit Sigma-Delta ADC

2 Introduction

2.1 Hardware

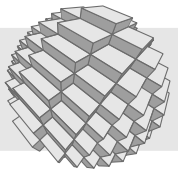
Table 2 shows the relevant user I/O connectors and interfaces

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 34.

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

2.1.1 Networking

The network ethernet interface at X2 is configured by default according to Table 3. On the evaluation netpp node firmware ('NetppNodev0'), these can not be altered.



Tab. 1: netpp node CPU configuration

dagobert SoC specifications		
Memory configuration		
ROM	256kB	Overlay program memory (SPI) read only
L1RAM	32kB	Level one program and data segment
L1CACHE	32kB	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
ADC-7x	1	(external ADC interface) *Optional configuration*
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

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

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.

Tab. 2: netpp node user I/O

Designator	Description
`J1, J2`	Two expansion headers. For the pinout, please see Section 7.2
`SW1, SW2`	Push button switches (Reboot/Reset, User button)
`D1..D4`	LEDs (green, red, blue, yellow)
`X1`	100MBit RJ45 connector
`X2`	USB B mini connector (to PC)

Tab. 3: Default network properties

Property	Default
`Network.IPaddr`	192.168.0.5
`Network.MACaddr`	00:00:de:ad:be:ef
Protocol support	ARP, ICMP ping, UDP

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 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.

2.3 Pin map

The connectors J1 and J2 provide most of the relevant I/O signals:

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..4]
: Analog input channels (only with populated U3). See Section 6.3.1.

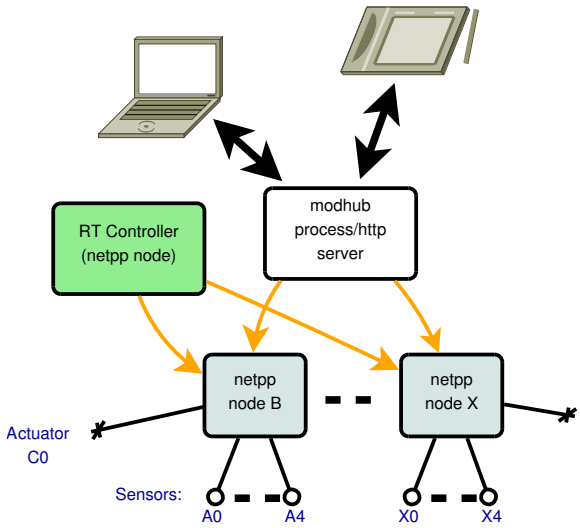


Fig. 2: netpp node mesh/sensor network

IOD[0..7]
: Digital auxiliary pins (only with populated U3). Exposure to properties is custom specific.

The pin functionality on Port B is defined by a corresponding MUX bit in the SysCtrl container, as shown in Table 4. For the PWM, the MUX property is found in the PWM[0..2] array containers.

Tab. 4: Pin map GPIO_B

Pin	MUX = 0	MUX = 1	Property
0..3	GPIO[1][0..3]	PWM[0..2] (out)	PWMxOutEn
8	GPIO[1][8]	UART0_RX (in)	UART_RX_SEL
9	GPIO[1][9]	UART0_TX (out)	UART_TX_SEL

When the SysCtrl::UART_?X_SEL (see Table 18) 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 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 X2. 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.

The debug console can be turned off via the Property `ConsoleOn`.

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 6.1.1.

`MSP430Type`
: Read-only 'MODE' property listing the identifier of the installed msp430

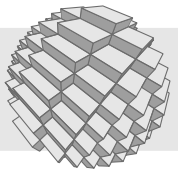
2.6 Real time clock

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.

3 Software

3.1 Installation

When using the section5 netpp node SDK docker container, all relevant netpp utilities are preinstalled. How-



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

ever, 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/

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 'NetppNodev0' device class
   Child: (u) [00000001] 'HWRevision'
   Child: (s) [00000002] 'RevisionTag'
   Child: (s) [00000003] 'MSP430Type'

   ...
   ```

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

   ```
   netpp> GPIO
   'GPIO' not in cache, querying...
   Type : {Array} [RW.]
   Child: (i) [0000001d] 'Size'
   Child: (S) [00000011] 'Pin'
   ```

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.Lock`.

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 8).

In Section 6 you find a reference of all properties existing on this netpp target.

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 OpenLab

This is an OpenSource Labview-like environment. Netpp Java wrappers are available for this project.

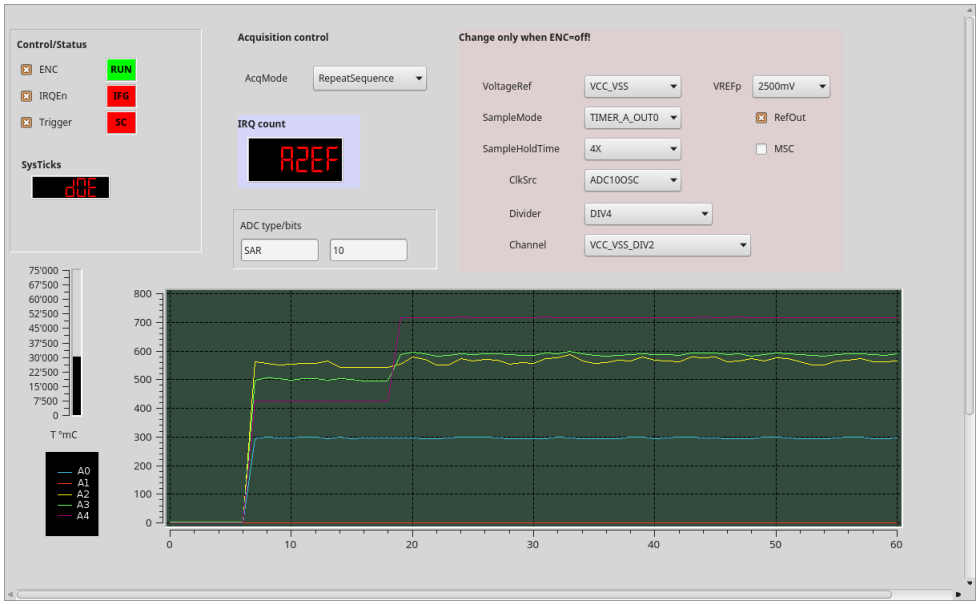


Fig. 3: pvbrowser screen shot

3.6 Firmware 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
```

3.7 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.

Note that msp430GXXXX types are not supported by a EZ430 FET!

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

4.2 Analog inputs (ADC10)

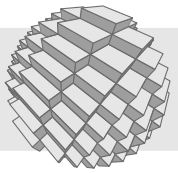
This section covers the ADC10 variant of the netpp node only.

The analog inputs [A0..A1] are by default available via the IOA0..IOA4 pins as shown in the schematic Fig. 5. Prior to sampling, the ADC is configured as follows:

- Turn `ADC.ENC` off
- 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 7. When `RefOut` is set, the pins are turned into outputs of the `RefVoltage`.

Tab. 7: 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. 5.

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')
4. Poll the `ADCValue` property

Unlike the ADC10 units, the SD16 uses differential inputs according to Table 8.

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

Tab. 8: Channel functions

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

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:

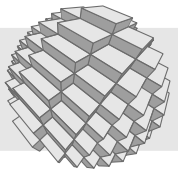
1. Set `Period` to period length in cycles minus one
2. Set `Width` to pulse width (must be <= `Period`)
3. Set `PWMStartAll` to start PWMs simultaneously
4. Poll `Count` to verify the PWM is running
5. Stop single PWM (except unit 0) using `PWM1Stop`, `PWM2Stop`

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 `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

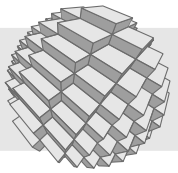


5 Troubleshooting

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

System no longer responding and Red LED on
: System entered a **BREAK** condition. In the eval version, this can happen on illegal UDP packets. 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.

Communication errors and slow response
: Check the blue D3 LED for high packet/IRQ activity, likewise check UDP traffic right at the connector LED or use Wireshark to monitor packet rates.



6 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 'NetppNodev0' base class,

Device Revision: 0.0

Make sure to check against the device revision (`RevisionTag` 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. 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).

6.1 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
```

6.1.1 Derived device classes

The default netpp node device 'NetppNodev0' 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 6.3.

NetppNodeSD16
: 16 bit sigma delta analog extension, see Section 6.4.

6.2 Base class properties

The base class (NetppNodev0) properties

6.2.1 System configuration

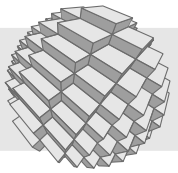
Tab. 9: Property group 'System configuration'

Property	Type	Flags	Description
`HWRevision`	REGISTER	RO	Hardware version register
`RevisionTag`	STRING	RO	Hardware revision tag
`MSP430Type`	STRING	RO	MSP430 type identifier
`ConsoleOn`	BOOL	RW	Set to enable console (Default = on)
`Reboot`	COMMAND	WO	Schedules cold boot procedure (flash reload) from next time slot within main loop
`Flash`	STRUCT	RW	Direct low level flash access. Experimental.

Configuration and status query
See Table 9

Tab. 10: Mode `Flash.Mode` – possible values

Value	Mode name	Description
0	READ	Read-only mode
1	WRITE	Write mode



Tab. 11: 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'.
Mode	MODE	RW	Flash mode, currently unused.
CRC	REGISTER	RO	Calculates CRC from sector at address 'Addr'

6.2.2 GPIO

Tab. 12: Property group 'GPIO'

Property	Type	Flags	Description
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

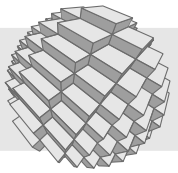
Port input/output control properties that are non-system specific.
See Table 12

6.2.3 Low Level register access

Tab. 16: Property group 'Low Level register access'

Property	Type	Flags	Description
LED	STRUCT	RW	User LEDs D1..D4
SysCtrl	STRUCT	RW	System control container

Properties for direct low level register access
See Table 16



Tab. 13: 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. 14: 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

6.3 ADC10 extension

The additional properties for the 'NetppNodeADC10' device

6.3.1 ADC10 Analog I/O

Tab. 19: 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 19

Tab. 20: 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. 21: 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. 22: Mode ADC.SHT – possible values

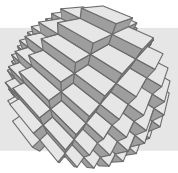
Value	Mode name	Description
0	4X	Four clock cycles
1	8X	Eight clock cycles
2	16X	..
3	64X	

6.4 ADC_SD16 extension

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

6.4.1 SD16 Analog I/O

Properties for analog I/O based on the msp430 extension
See Table 26



Tab. 15: 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. 17: Struct LED

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

Tab. 23: Mode ADC.RefVoltage – possible values

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

Tab. 24: 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. 26: Property group 'SD16 Analog I/O'

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

Tab. 27: 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. 28: Mode SD16.Channel – possible values

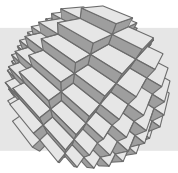
Value	Mode name	Description
0	A0p_A1m	Inputs A0+,A1-
1	A2p_A3m	Inputs A2+,A3-
2	A4p_A5m	Not available in this HW revision
3	A6p_A7m	N/A with i2c interface
4	A1p_A2m	Inputs A1+,A2-
5	AVcc_AVssby11	A5 - (AVcc - AVss)/11
6	TEMP	Temperature channel
7	PGAOFFS	PGA offset measurement

Tab. 29: Mode SD16.BufMode – possible values

Value	Mode name	Description
0	OFF	
1	SLOW	
2	MEDIUM	
3	HIGH	

Tab. 30: Mode SD16.OSR – possible values

Value	Mode name	Description
0	OSR256	
1	OSR128	
2	OSR64	
3	OSR32	



Tab. 18: Struct SysCtrl

Property	Type	Flags	Description
TEST_EN	BOOL	RW	Enable LED testing (user access)
UART_TX_SEL	BOOL	RW	Enable UartTX on Port B
UART_RX_SEL	BOOL	RW	Enable UartRX on Port B

Tab. 25: Struct ADC

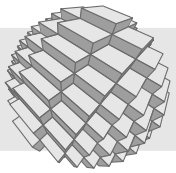
Property	Type	Flags	Description
Busy	BOOL	RO	When set, ADC is running
VrefMode	MODE	RW	Voltage reference selection
MSC	BOOL	RW	When set, auto-repeat conversion regardless of timer frequency. Turn off by default.
SHS	MODE	RW	Conversion trigger mode
SHT	MODE	RW	Sample&Hold timing
ENC	BOOL	RW	Encoding when 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.

Tab. 31: Mode SD16.ClkDiv0 – possible values

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

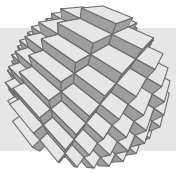
Tab. 32: Mode SD16.SSEL – possible values

Value	Mode name	Description
0	MCLK	
1	SMCLK	
2	ACLK	
3	ExtTACLK	



Tab. 33: Struct SD16

Property	Type	Flags	Description
Gain	MODE	RW	Preamplifier gain
Channel	MODE	RW	Analog input channel
BufMode	MODE	RW	
SingleConv	BOOL	RW	Single conversion
OSR	MODE	RW	OversamplingRate
Unipolar	BOOL	RW	
IFG	BOOL	RO	
IE	BOOL	RW	
REFON	BOOL	RW	Enable 1.2V reference. Must be enabled for conversion.
VMIDON	BOOL	RW	Enable VRef output on IOA3
ClkDiv0	MODE	RW	
SSEL	MODE	RW	
LSBacc	BOOL	RW	
LSBtog	BOOL	RW	
SC	BOOL	RW	Single conversion trigger ACK bit. Cleared after successful single trigger action.



7 Technical specifications

7.1 Electrical

Table 34 shows the electrical characteristics and maximum ratings.

Tab. 34: 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

7.2 Drawings

- Top level schematic: Fig. 5
- Mechanical outline: Fig. 4

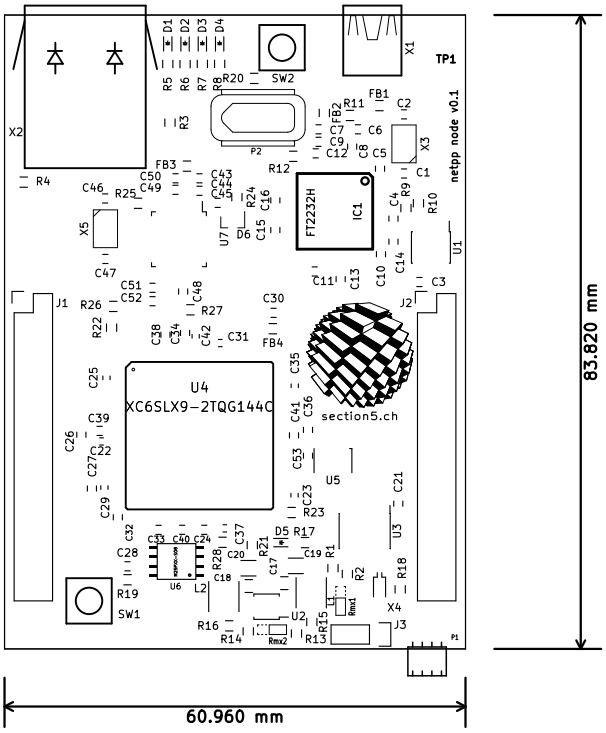


Fig. 4: Mechanical outline

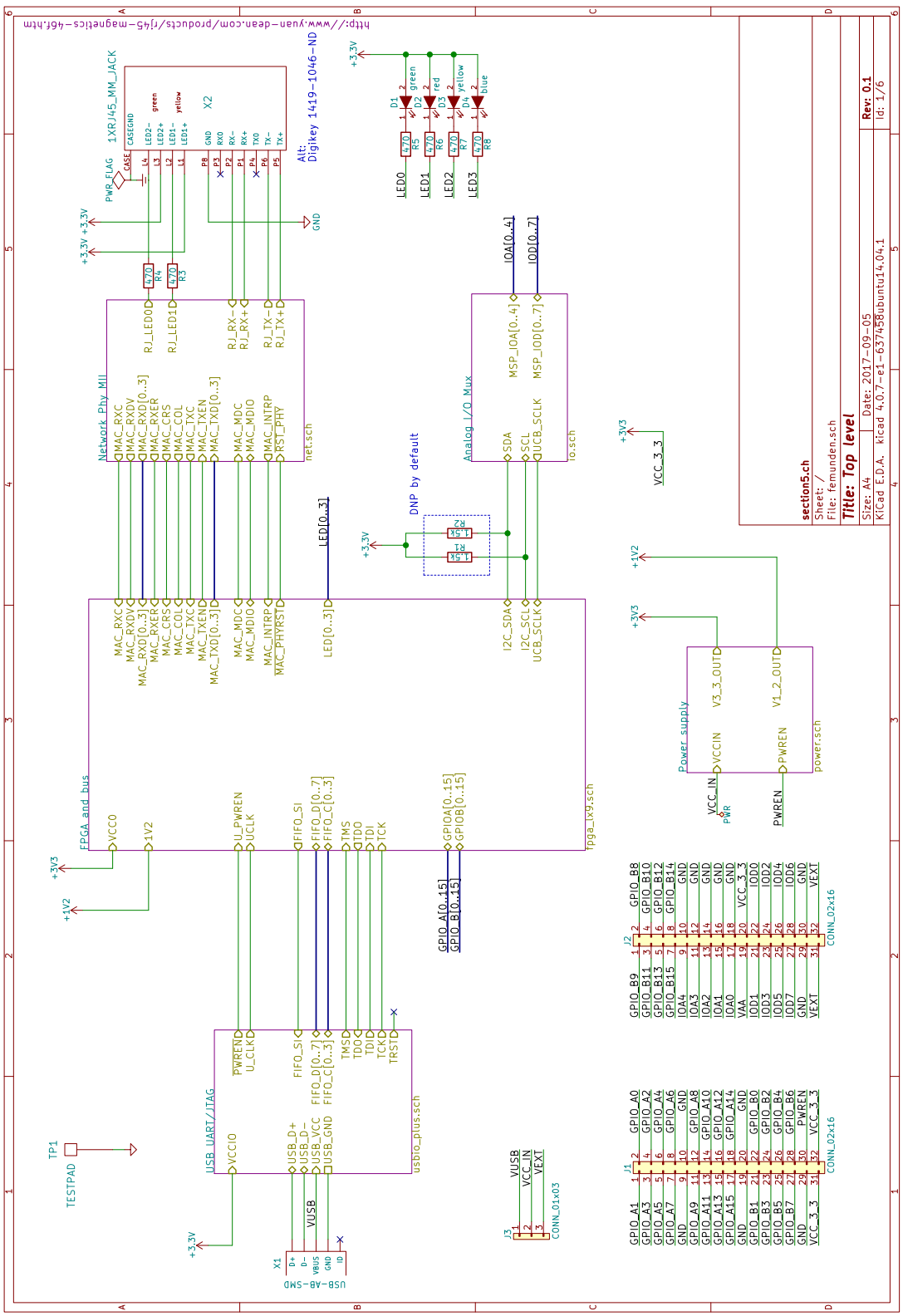
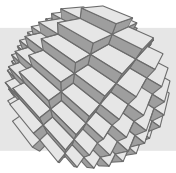


Fig. 5: Top level schematic



8 Further pointers

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

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

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

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