



MINDSTORMS®
NXΤ'ΡΕΜΕ

LEGO® MINDSTORMS® NXT

Hardware Developer Kit

TABLE OF CONTENTS

TABLE OF CONTENTS	2
HARDWARE SPECIFICATION FOR THE NXT BRICK.....	3
NXT TECHNICAL OVERVIEW.....	4
OUTPUT PORTS.....	5
INPUT PORTS.....	6
Active sensorS	7
Passive sensorS.....	7
Digital sensors	7
High-speed communication port	8
I²C COMMUNICATION.....	9
Device memory arrangement.....	10
DISPLAY	11
BLUETOOTH®	12
Bluetooth® functionality within the NXT Brick	12
Interfacing with the BlueCore™ chip.....	13
UART interface between the ARM7 and the BlueCore chip.....	14
SOUND.....	15
DEBUGGING INFORMATION	16
Interfacing with the ARM7 microcontroller	16
Interfacing with the AVR microcontroller	17
Firmware requirements:	17
AVR TO ARM COMMUNICATION.....	18
Data sent from the ARM7 microcontroller	18
Data received from the AVR microcontroller	20
Communication scheme	20
Power management.....	21
Battery testing within the LEGO MINDSTORMS NXT.....	21
BACKWARDS COMPATIBILITY	23
LINKS	24
APPENDIX	25

HARDWARE SPECIFICATION FOR THE NXT BRICK

The LEGO® MINDSTORMS® NXT brick uses various advanced electronics to yield its broad functionality. To view the hardware schematics of the LEGO® MINDSTORMS® NXT, see Appendix 1 and 2; for hardware schematics of the LEGO MINDSTORMS® NXT® sensors, see Appendices 3-6.

Here is a summary list of hardware specifications for the NXT brick:

Main processor:	Atmel® 32-bit ARM® processor, AT91SAM7S256 - 256 KB FLASH - 64 KB RAM - 48 MHz
Co-processor:	Atmel® 8-bit AVR processor, ATmega48 - 4 KB FLASH - 512 Byte RAM - 8 MHz
Bluetooth wireless communication	CSR BlueCore™ 4 v2.0 +EDR System - Supporting the Serial Port Profile (SPP) - Internal 47 KByte RAM - External 8 MBit FLASH - 26 MHz
USB 2.0 communication	Full speed port (12 Mbit/s)
4 input ports	6-wire interface supporting both digital and analog interface - 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
3 output ports	6-wire interface supporting input from encoders
Display	100 x 64 pixel LCD black & white graphical display - View area: 26 X 40.6 mm
Loudspeaker	Sound output channel with 8-bit resolution - Supporting a sample rate of 2-16 KHz
4 button user-interface	Rubber buttons
Power source	6 AA batteries - Alkaline batteries are recommended - Rechargeable Lithium-Ion battery 1400 mAH is available
Connector	6-wire industry-standard connector, RJ12 Right side adjustment

NXT TECHNICAL OVERVIEW

This section provides a graphical overview of how different functions are connected and controlled within the intelligent brick. The figure only includes higher-level units within the NXT. For detailed information on how individual elements are connected, see the hardware schematic in Appendix 1 and 2.

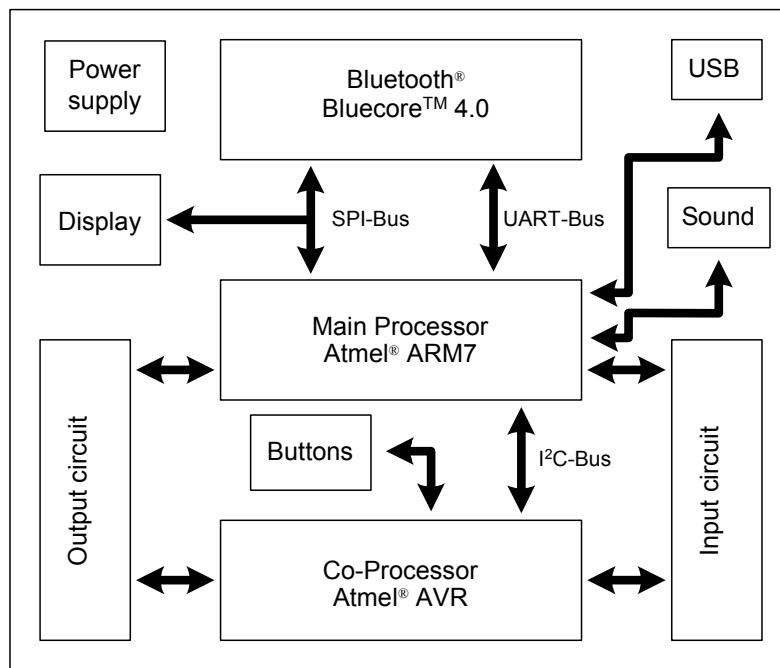


Figure 1: Hardware block diagram of the NXT brick

OUTPUT PORTS

The LEGO® MINDSTORMS® NXT has three output ports used for controlling actuators connected to the NXT brick.

A 6-wire digital user interface on the output ports was implemented so that output devices could send information back to the NXT brick without having to use up an input port as well.

The figure below shows the schematic details behind port A of the brick. The schematics for ports B and C are identical.

MotorA	
J1	
1	MA0
2	MA1
3	GND
4	POWERMA
5	TACHOA0
6	TACHOA1
Modular	
	Pin 1, MA0 PWM output signal for the actuators
	Pin 2, MA1 PWM output signal for the actuators
	Pin 3, GND Ground signal related to the output supply
	Pin 4, POWERMA 4.3 Volt output supply
	Pin 5, TACHOA0 Input value that includes Schmitt trigger functionality
	Pin 6, TACHOA1 Input value that includes Schmitt trigger functionality

MA0 and MA1 are output signals for controlling actuators. These signals are controlled by an internal motor driver which can supply a continuous 700 mA to each output port and a peak current of approximately 1 A. The output signal is a PWM signal, which can be controlled to either break or float between the signals. The motor driver has thermal protection built-in, which means that if too much power is continually drawn from the brick, the motor driver will automatically adjust the output current.

The output power (POWERMA) is connected internally to all of the power outputs in the output and input ports. The maximum output current that can be drawn from this supply is approximately 180 mA. This means that each port has approximately 20 mA. If more power is drawn, the total output current will be decreased automatically without further warning. If the power signal is short circuited to ground, the NXT brick will reset.

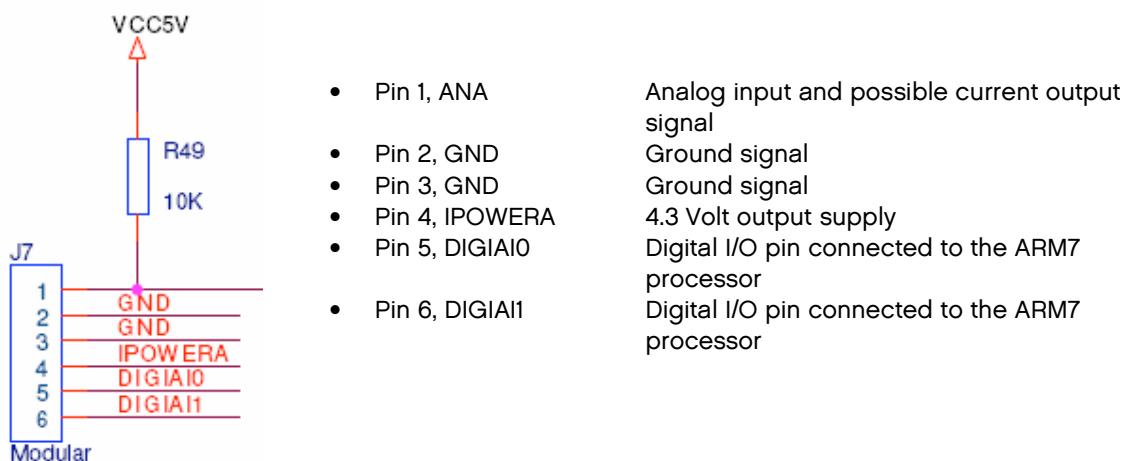
The TACHOA0 and TACHOA1 are input ports that have a Schmitt trigger mounted between the ports and the input pins on the ARM7 processor. These two signals allow the possibility of having a quadrature detector within the system. Within the standard firmware these two signals are used to count the numbers of tacho pulses from the motors and detect whether the motor is running clockwise or counterclockwise.

INPUT PORTS

The LEGO® MINDSTORMS® NXT has four input ports that allow the NXT to measure different parameters in the physical world (depending on the connected sensor).

The 6-wire digital user interface on the input ports enables having both an analog and digital interface within each connector. This allows the possibility of developing both analog and digital sensors for the NXT brick.

The figure below shows the schematic details behind port 1 on the brick. Ports 2, 3, and 4 have identical schematics. Behind port 4, the digital pins (DIGIxI0 and DIGIxI1) are connected to a RS485 controller that handles high-speed communication.



The input pin (ANA) is the analog input pin that is connected to a 10-bit A/D converter within the AVR processor. This is also connected to the current generator which is used for generating power for the active LEGO® MINDSTORMS® Robotic Invention System sensors. (See the section about active sensors in this chapter.) The A/D input signals are sampled with the same sampling rate for all analog sensors. As described in the active sensor documentation, analog sensors need 3 mS of supply power output before any measurements can occur. The sampling rate used for all analog sensors is 333 Hz.

Output power (IPOWERA) is connected internally to all of the power outputs in the output and input ports. The maximum output current that can be drawn from this supply is approximately 180 mA. This means that each port has approximately 20 mA available. If more power is drawn, the total output current will be decreased automatically without further warning. If the power signal is short circuited to ground, the NXT brick will reset.

The digital I/O pins (DIGIAI0 & DIGIAI1) are used for digital communication implemented as I²C compliant communication running at 9600 bit/s. The NXT can only function as a master in relation to I²C communication and requires that external devices have pull-up resistors included on their communication pins. See the I²C Communication chapter for further details.

In addition, the I/O pin on the ARM7 that is connected to DIGIXI1 can be set up to function as an analog input pin (although this is not supported directly within the standard firmware). This allows the possibility of implementing an analog input pin with a higher sampling rate.

ACTIVE SENSORS

To ensure backwards compatibility with LEGO® MINDSTORMS® Robotic Invention System sensors (developed for the RCX intelligent brick), a current generator has been added to the NXT brick to yield correct power and measurement intervals for these older sensors. Together with the standard LEGO firmware the current generator yields the same functionality as is available within the RCX intelligent brick. The current generator provides approximately 18 mA of output current.

The generator controls the power delivery to active sensors. It supplies the sensor with power for 3 mS and then measures the analog value during the following 0.1 mS.

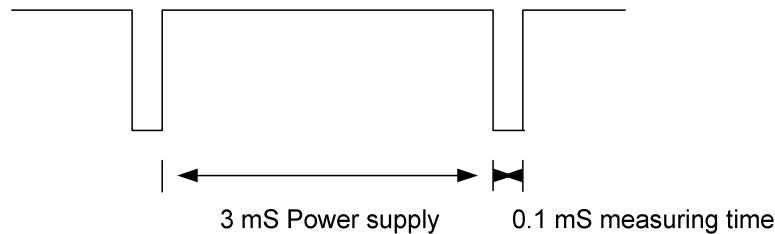


Figure 2: Timing diagram for the A/D input signal pin when using active sensors

The following sensors from the LEGO® MINDSTORMS® Robotics Invention system are active sensors:

- Light sensor
- Rotation sensor

PASSIVE SENSORS

All sensors that do not need the special power/measurement timing mentioned above are called passive sensors. These sensors are also sampled every 3 mS because sampling using the A/D converter is simultaneous and therefore, must uphold the timing required by the active sensors.

The following sensors are passive sensors:

- Touch sensor (both the RCX and NXT versions)
- Light sensor supplied in the LEGO MINDSTORMS® NXT sets
- Sound sensor
- Temperature sensor

DIGITAL SENSORS

All sensors that use I²C communication are named digital sensors because they include an external micro-controller that handles the sampling of the physical environment.

The following sensors are digital sensors:

- Ultrasonic sensor

HIGH-SPEED COMMUNICATION PORT

Port 4 on the NXT intelligent brick can function as a high-speed communication port. A RS485 communication chip is implemented behind the normal input circuit. This allows the implementation of high-speed bi-directional data communication on a multipoint data line over wider distances. Currently LEGO® has not developed any devices that need this communication functionality. However, if future devices are developed that require higher communication speeds, the NXT brick is prepared. For such future devices, LEGO may use the P-Net communication protocol (www.P-net.org), which enables multipoint data communication.

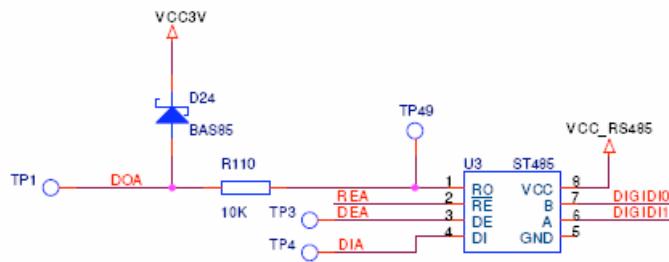


Figure 3: Hardware schematic for the RS485 chip behind port 4 on the NXT brick

Note: The RS485 chip is using 5 volts as its supply voltage and the ARM7 processor is using 3.3 volts. For this reason a level shifter has been applied between the RS485 chip and the ARM7 processor. See the schematics for further details.

The following communication parameters are set up for high-speed communication within the standard firmware:

- Communication speed: 921.6 Kbit/s
- Data bits: 8 bit
- Stop bit: 1 bit
- Parity: 0 bit

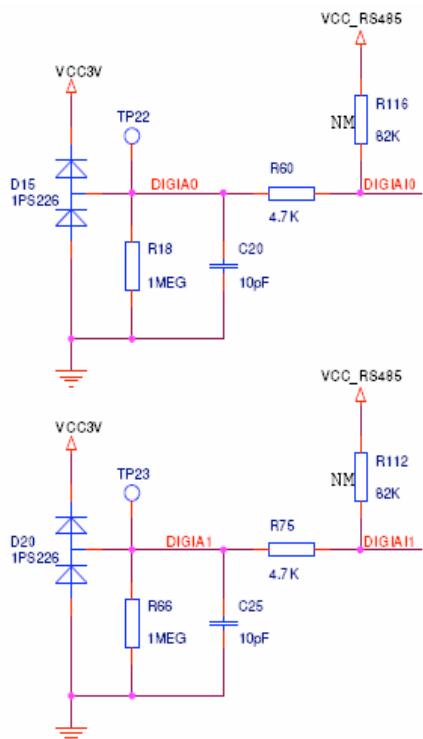
I²C COMMUNICATION

Within the NXT brick, a digital interface has been implemented using the I²C protocol. I²C is an industrial communication standard that was developed by Philips Semiconductors in the early 1980s. It has since been used in many different industrial components where simple digital communication is required.

I²C communication functions as the digital interface for external devices which needs to communicate with the NXT. Having a digital interface enables external devices to perform functionality individually and then only send the result back to the NXT or receive new information from the NXT.

The NXT brick has four I²C communication channels, one for each of the four input ports. The I²C digital communication is implemented as “master only” functionality, which means that the NXT is controlling the dataflow in each of the communication channels.

An important aspect to allowing I²C communication between two devices is the hardware set-up within each of the devices. The figure below shows the hardware schematic internally for input port 1 in the NXT. The schematic for input ports 2, 3 and 4 are the same with respect to I²C communication.



Important parameters to notice:

- There is a 4.7 k resistor in serial with the signal line.
- There is no pull-up resistor mounted internally in the NXT. This needs to be mounted in the external device. We recommend using a 82 K resistors as pull-up resistors on both the data and clock lines.
- DIGIx0 (Pin 5 within the connector) is the CLK signal and DIGIx1 (Pin 6 within the connector) is the DATA signal for I²C communication.
- The digital I/O pins on the NXT cannot be set to open drain directly. Therefore the NXT will drive the digital I/O pins either high or low depending on the situation, i.e., the NXT uses Push-Pull. When the NXT should not control the I/O lines, it will be set as input (e.g., when reading data from a device or when reading acknowledgement).
- The I²C communication is running at 9600 bit/s.
- Each channel has a 16 byte input buffer and a 16 byte output buffer. Therefore a maximum of 16 bytes can be sent and received during each data communication cycle.
- If multiple sensors are connected in sequence to the same sensor port, the resulting pull-up resistor needs to be of 82 k. For this reason, some consideration is needed when multiple sensors are connected in sequence to the same port.

Digital devices have some advantages compared to analog devices. Digital devices can include device names and can reference various individual parameters that are device specific (like calibration values, startup times, and so on). To be able to distinguish different digital device from each other, LEGO® has started an addressing scheme for its sensors that will expanded as the company develops new digital devices or approves third-party devices. Currently, the applicable list includes only the Ultrasonic sensor, which has been given address 1 (within a 7 bit context). This address is sent together with the communication direction bit, and as with all other data bytes, the address is transferred with the most significant bit first.

DEVICE MEMORY ARRANGEMENT

When using I²C communication, there are multiple ways to define and implement the functionality for reading and writing data to and from an external device.

We characterize LEGO external devices as external memory areas from which we can read data or to which we can write data. By knowing the specifics behind each of the data memory locations in external devices, it's possible to control the device and read the desired data. The example memory map below shows how memory can be divided into areas that enable easier read and write access:

Transmitted from NXT				
Command	Byte 0	Byte 1	Byte 2	Length
Addr.				
Constants				
Read version	Device address	0x00	R + 0x03	8
Read product ID	Device address	0x08	R + 0x03	8
Read sensor type	Device address	0x10	R + 0x03	8
Read factory zero (Cal 1)	Device address	0x18	R + 0x03	1
Read factory scale factor (Cal 2)	Device address	0x19	R + 0x03	1
Read factory scale divisor	Device address	0x1A	R + 0x03	1
Read measurement units	Device address	0x1B	R + 0x03	7
Variables				
Read variable 1	Device address	0x40	R + 0x03	1
Read variable 2	Device address	0x41		
...				
Commands				
Command 1	Device address	0x80	0xXX	
Command 2	Device address	0x81		
...				

Figure 4: Example of a memory map for an external digital device

By building up the memory with the trailing commands as shown above, it is much easier to read data from the sensors and potentially set multiple parameters in one read/write cycle. For details on how the memory is divided up for the Ultrasonic sensor, see Appendix 7.

DISPLAY

A dot matrix display has been added to the NXT brick to improve the user interface. This display is a black and white graphical LCD display with a resolution of 100 x 64 pixels. The display has a viewing area of 26 x 40.6 mm. The LCD-controller used to control the display is an UltraChip 1601. See the Links section for detailed data sheets on the display's LCD-controller.

There is an SPI interface from the ARM7 microcontroller to the display's LCD-controller UltraChip 1601. The SPI interface is running at 2 MHz within the standard LEGO® firmware and two full memory maps are set aside within the firmware for updating the display. The display is continuously updated in a line sequence requiring 17 mS for a total display update.

LCD data is allocated within memory as a two dimensional array, Normal[8][100] (Normal[Y / 8][X]). Data is sent to the LCD controller in the following order: the first byte controls the first 8 pixel vertical (starting at (0,0)) and the second byte controls the next 8 vertical pixel horizontal.

The pixels within the display are allocated as follows:

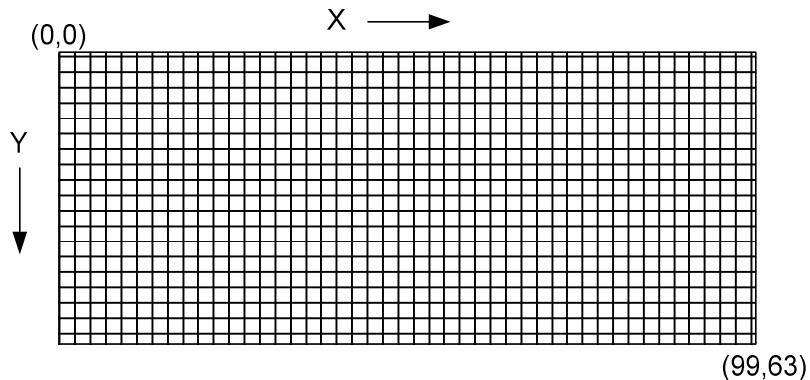


Figure 5: Bitmapping within the display

Technical specifications for the display:

- Format: 100 x 64 dots
- LCD mode: STN / Positive Reflective Mode / Gray
- Viewing direction: 6 o'clock
- Driving scheme: 1/65 duty cycle, 1/9 bias
- Power supply voltage (V_{DD}): 3.0V
- LCD driving voltage (VLCD): 9.0V (adjustable for best contrast)

BLUETOOTH®

The NXT brick supports wireless communication using Bluetooth® by including a CSR BlueCore™ 4 version 2 chip. The NXT brick can be connected wirelessly to three other devices at the same time but can only communicate with one device at a time. This functionality has been implemented using the Serial Port Profile (SPP), which can be considered a wireless serial port. The NXT brick can communicate with Bluetooth devices that can be programmed to communicate using the LEGO® MINDSTORMS® NXT Communication Protocol and that support the Serial Port Profile (SPP). It's possible to send programs and sound files between NXT bricks and to use wireless communication to send and receive information between bricks during program execution. To reduce the power consumption used by Bluetooth, the technology has been implemented as a Bluetooth® Class II device, which means that it can communicate up to a distance of approximately 10 meters.

BLUETOOTH® FUNCTIONALITY WITHIN THE NXT BRICK

The Bluetooth® functionality within the NXT brick is set up as a master/slave communication channel. This means that one NXT within the network needs to function as the master unit and that other NXT bricks communicate through it if they need to. The figure below shows which NXT devices can communicate directly within a network.

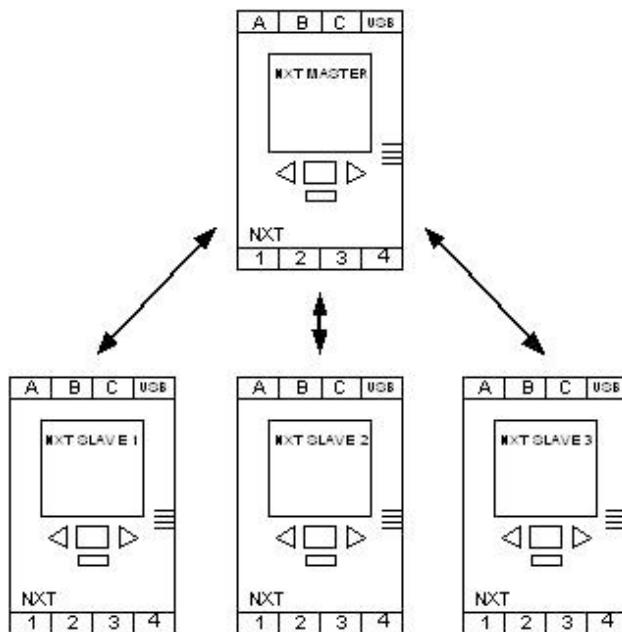


Figure 6: NXTs communicating using Bluetooth®

As shown in the figure above, the master NXT can be connected to three other Bluetooth® devices at the same time. The master NXT can only communicate with one of the slave units during a given moment, meaning that if the master NXT is communicating with NXT Slave 1 and NXT Slave 3 starts sending data to the master NXT, the master NXT will not evaluate the received data until it switches to NXT slave 3.

An NXT is not able to function as both a master and slave device at the same time because this could cause lost data between NXT devices.

Connections to other Bluetooth devices occur through channels. The NXT has four connection channels used for Bluetooth communication. Channel 0 is always used by slave NXT devices in communicating with the master NXT (i.e., towards the master NXT) while channels 1, 2, and 3 are used for outgoing communication from the master device to the slave devices.

INTERFACING WITH THE BLUECORE™ CHIP

Bluetooth® functionality within the NXT is implemented using a standalone chip, a CSR BlueCore™ 4 with an external 8 Mbit FLASH memory. The Bluetooth chip from CSR contains all the necessary hardware to run a self-contained Bluetooth node. A 16-bit integrated processor runs the Bluetooth stack implemented by CSR, called Bluelab. Bluetooth® is implemented within the NXT using version 3.2 of Bluelab. The firmware within the BlueCore integrates a user programmable VM-task allowing us to control and run small amounts of application code. A command interpreter is integrated within the VM that is able to decode and respond to commands received through the UART interface from the ARM7 processor.

The VM has a full implementation of both the Bluetooth SPP-A and SPP-B profiles. The SPP-A profile is used when the local BlueCore is the connection initiator (MASTER device) while the SPP-B profile is used when another Bluetooth device initiates the connection (SLAVE device). The BlueCore uses what is referred to as “stream-mode” to exchange data at a rate of ≤ 220 K baud after a connection is established. When BlueCore is not in “stream-mode,” it is in “command-mode” which is used to control the VM application within BlueCore and by extension, the Bluetooth functionality within the NXT. Which communication type the UART includes is controlled by two interface signals (ARM7_CMD & BC4_CMD).

For a detailed description of the communication protocol used between the ARM7 processor and the BlueCore chip, see Appendix 8.

The figure below shows the interface between the ARM7 processor and the BlueCore chip. (Its functionality is explained below the figure.) For a detailed description of the pin layout, see the hardware schematics for the NXT brick.

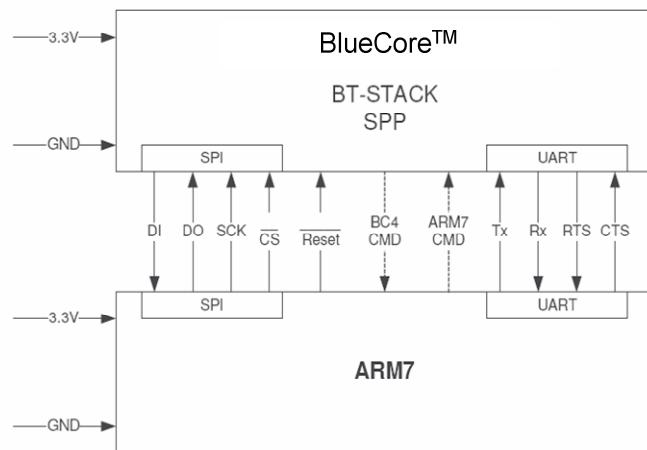


Figure 7: Interface between the ARM7 processor and the BlueCore chip

The SPI interface allows the possibility of updating the BlueCore chip. It is not in use during normal operation of the NXT brick. The SPI interface is shared with the display within the NXT brick.

The Reset pin is used at startup to re-initialize the chip correctly and to disable Bluetooth functionality.

BC4-CMD: Indication from the BlueCore to the ARM7 as to which data type the BlueCore expects to send to the ARM7.

ARM7-CMD: Indication from the ARM7 to the BlueCore as to which data type the ARM7 expects to send to the BlueCore.

UART communication is used for both data and command communication between the BlueCore and the ARM7 processor.

UART interface between the ARM7 and the BlueCore chip

The UART within the BlueCore chip is initialized for communication with the ARM7 using the following set-up (both for stream-mode and command-mode):

Communication speed:	460.8 K bit/s
Data bits:	8 bits
Parity:	No parity bits
Stop bit:	One stop bit
Flow control:	Hardware handshake signals (RTS & CTS)

SOUND

The NXT brick includes a sound amplifier chip to improve the sound output level and quality. The sound output is a PWM output signal that is controlled by the ARM7 microcontroller. The filters introduced before the amplifier will reduce the over-sampling noise in the signal.

The sound driver (SPY0030A) is a differential sound amplifier chip from SUNPLUS that can have a maximum gain of 20. For detailed information about the sound amplifier, see the data sheet for the SUNPLUS sound driver.

The loudspeaker within the NXT is a 16 ohm speaker with a diameter of 21 mm. The table below shows the current and power consumption when sounds are played at two different frequencies.

Sound current consumption

Frequency	Current mA	Power mW
440 Hz	102	169
4 KHz	78	97

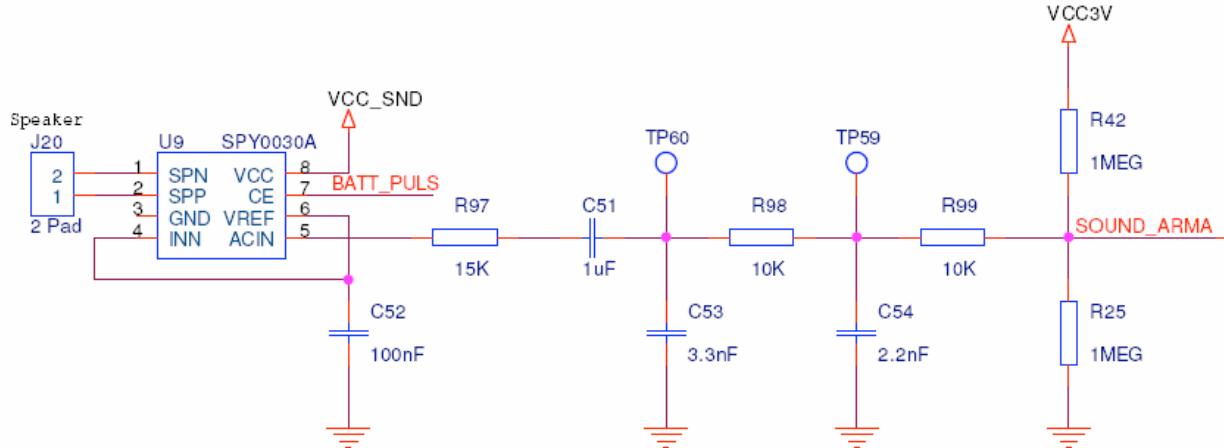


Figure 8: Schematic for sound output within the NXT

DEBUGGING INFORMATION

This section briefly describes how users can interface with the two microcontrollers within the NXT using JTAG interfaces. (Important note: When the NXT is disassembled or when third party firmware is used with the NXT, all warranties are rendered invalid.)

When developing third party firmware for the NXT, developers must be careful when addressing the hardware because incorrect initialization can destroy hardware components. Study the hardware schematic carefully before developing new firmware.

The main processor within the NXT brick is the ARM7 processor that handles all the user-specific functionality. The AVR microcontroller handles lower-level functionality, like controlling the motor PWM, power management, and A/D conversion. To connect a JTAG interface to either the ARM7 or AVR microcontroller, the NXT needs to be disassembled.

INTERFACING WITH THE ARM7 MICROCONTROLLER

The connection points to the ARM7 processor have not been removed from the NXT brick but the necessary hardware to make a connection have not been mounted on the main PCBA to save cost and mounting time during production. Therefore, to interface with the ARM7 processor, a 10-pin connector (single row using 1.27 pitch) must be mounted to the NXT and a debugging wire connected from the JTAG to the NXT.

The J-tag connector (J17) has the following schematic layout:

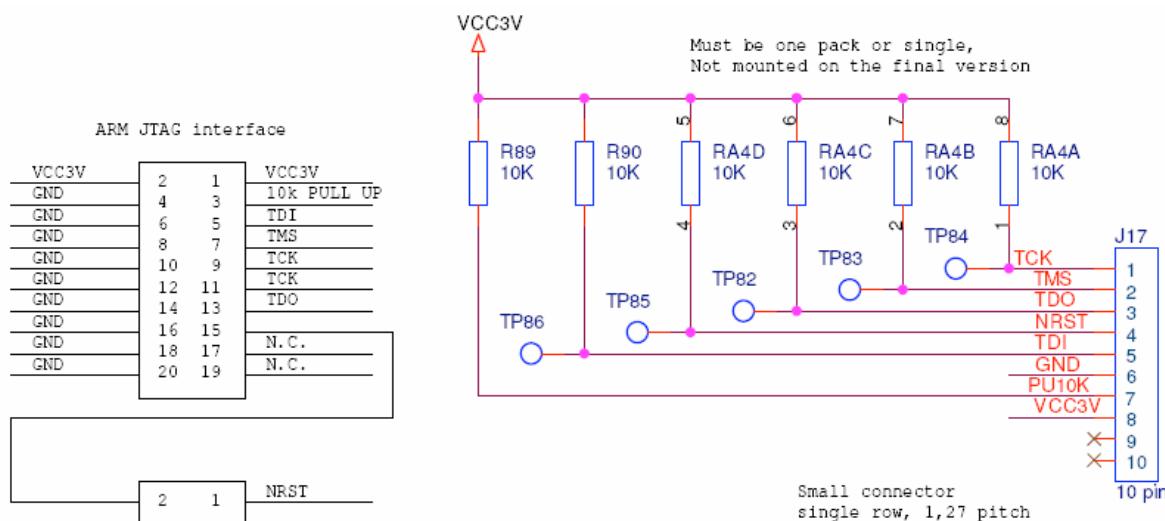


Figure 9: Hardware schematic for interfacing with the ARM7 microcontroller

INTERFACING WITH THE AVR MICROCONTROLLER

The connection points to the AVR processor have not been removed from the brick but the necessary hardware to make a connection has not been mounted on the main PCBA to save cost and mounting time during production. Therefore, to interface with the AVR processor, a 8-pin connector (single row using 1.27 pitch) must be mounted to the NXT and a debugging wire connected from the JTAG to the NXT.

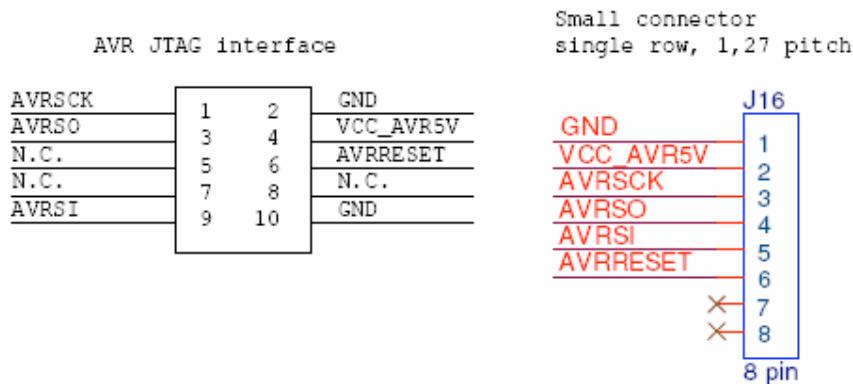


Figure 10: Hardware schematic for interfacing with the AVR microcontroller

FIRMWARE REQUIREMENTS:

The NXT is made up of multiple processors and there are some minor requirements that need to be upheld for the system to function correctly. Some of these requirements are related to the hardware setup; others are implemented in areas that can be changed by the user. This section will describe the requirements that an alternative firmware version must uphold.

For various reasons, overall power distribution within the system is controlled by the AVR microcontroller. A small startup sequence has been implemented to protect the product from misuse. This sequence controls the power to the ARM7 processor and will cause it to be turned off if the ARM7 processor doesn't send back the following message to the AVR microcontroller over I²C communication within 5 minutes of start-up:

"\xCC""Let's samba nxt arm in arm, (c)LEGO System A/S"

This means that the following data should be sent to the AVR:

```
[0xCC,0x4C,0x65,0x74,0x27,0x73,0x20,0x73,0x61,0x6D,0x62,0x61,0x20,0x6E,0x78,0x74,0x20,0x61,0x72,0x6D,0x20,0x69,0x6E,0x20,0x61,0x72,0x6D,0x2C,0x20,0x28,0x63,0x29,0x4C,0x45,0x47,0x4F,0x20,0x53,0x79,0x73,0x74,0x65,0x6D,0x20,0x41,0x2F,0x53]
```

AVR TO ARM COMMUNICATION

The interface between the ARM7 microcontroller and the AVR microcontroller is implemented using the hardware I²C communication channel on both the ARM7 and AVR microcontroller. To enable both microcontrollers to run somewhat independently from each other, the communication interface between the two microcontrollers is set up as two memory allocations that are updated on both microcontrollers every 2 mS. The interface is set up to communicate at 380 Kbit/s using the hardware I²C interface within the microcontrollers.

The main tasks for the AVR microcontroller are power management, creating PWM output signals for the three motors, and performing A/D conversion from the input ports. Within the standard NXT firmware, data is sent to the AVR microcontroller through a struct implementation that is continuously updated in the ARM7 microcontroller to match the required functionality of the AVR microcontroller.

Because of limitations within the ARM7 microcontroller chip, the ARM7 can only function as the master within the I²C communication setup. For further details, see the data sheet for the Atmel AT91SAM7S256 microcontroller.

DATA SENT FROM THE ARM7 MICROCONTROLLER

```
typedef struct
{
    UBYTE Power;
    UBYTE PwmFreq;
    SBYTE PwmValue[NOS_OF_AVR_OUTPUTS];
    UBYTE OutputMode;
    UBYTE InputPower;
} ITOAVR;
```

Power	Command byte that is used during power down and firmware update mode. During normal communication, this byte should be set to zero.
PwmFreq	Holds the PWM frequency used by the PWM signal for the three outputs. It can have the following range: 1 – 32 KHz. Units are in KHz. The standard LEGO firmware uses 8 KHz.
PwmValue	Holds the power level for the individual output. The first element in the array relates to output A. It can range from -100 to +100 where -100 is full power clockwise and +100 is full power counterclockwise.
OutputMode	Holds the output mode that can be either float or break between PWM pulses. 0x00 means break and 0x01 means float.
InputPower	Holds the parameter used to define the 9V sensor supply. Input power is configured as bit fields where bit 0 and 1 is sensor 0, bit 2 and 3 is sensor 1, bit 4 and 5 is sensor 2, and bit 6 and 7 is sensor 3. 0: 9V is off 1: 9V is on unless measuring a sensor (when it is off temporarily each 3 mS) 2: 9V is always on

When controlling the power management and the firmware download mode, other data packages should be sent to the AVR microcontroller but through the struct interface described above.

When powering down the NXT, the Power byte should be set to 0x5A and the PwmFreq should be set to 0x00. This will cause the AVR to turn off the NXT brick and wake up when the “Select” button is pressed.

When downloading new firmware to the NXT brick, a special firmware mode is required within the brick. To enable this functionality, the Power byte should be set to 0xA5 and the PwmFreq should be set to 5A. This will cause the NXT to go into firmware update mode. Firmware update mode is a low-level mode that the ARM7 processor is sent into by the AVR microcontroller. Read more about firmware download mode in the data sheet for the ATMEL AT91SAM7S256 processor, called SAMBA mode.

DATA RECEIVED FROM THE AVR MICROCONTROLLER

```
typedef struct
{
    WORD AdValue[NOS_OF_AVR_INPUTS];
    WORD Buttons;
    WORD Battery;
} IOFROMAVR;
```

AdValue	Holds the raw value from the 10 bit A/D converter. The first element in the array relates to sensor input 1 on the NXT brick.
Buttons	Holds the status of the buttons. Button 1, 2 and 3 are returned as a 10 bit AD value. Button 0 adds 0x7FF to this. AD values for button thresholds can be calculated from the resistor values on the button PCB schematics.
Battery	Holds information about the measured battery level, whether an Accu pack has been inserted, and the AVR firmware version. 16 bits as follows:
Bit 15	0 = AA batteries 1 = Accu pack inserted
Bit 13 - 14	0..3 = major version
Bit 10 - 12	0..7 = minor version
Bit 0 - 9	0..1023 (multiply with 13.848 to calculate actual mV)

COMMUNICATION SCHEME

As the ARM can only function as a master, it will be the initiator in both receiving and transmitting data to and from the AVR. In normal operation, transmission and reception are interleaved each second millisecond. The only restriction to this communication is that it complies with the transfer data structures mentioned above as these are the only ones supported by the AVR.

The data structures can be transferred at any desired time, meaning that there are no critical dependencies in the firmware. However, sensor and motor timing will be affected by this, i.e., the rotation sensor will not work correctly if data from the AVR is not transferred at a minimum rate of 3 millisecond intervals.

POWER MANAGEMENT

Power within the NXT brick comes from 6 AA batteries or a rechargeable lithium ion battery (that also includes a AC power plug to the LEGO Transformer). Power management within the NXT brick consists of a switch mode power supply, which generates a 5-volt supply from the batteries and from the 5-volt supply, another 3.3-volt supply is generated for the ARM7 processor and the BlueCore chip.

To protect the power supply within the NXT, a poly switch is connected at the beginning of the power circuit. The poly switch has a hold current of 1.85 A and will be triggered at approximately 3.3 A.

Current consumption measurement:

The effects are based on a battery voltage of 9 volts.

Supply voltage	Current		Effect (Battery = 9 Volts)	
	Max [mA]	Normal [mA]	Max [mW]	Normal [mW]
No load on motors				
9 Volt	339	114	5184	1422
5 Volt	271	112	1744	448
3.3 Volt	72	38	410	216
Load on motors				
9 Volt	2901	848	26109	7632
5 Volt	271	112	1142	307
3.3 Volt	72	38	410	137
Standby	46 uA assumed standby current due to brown out detection			

Figure 11: Current measurement on the MINDSTORMS NXT brick

Battery testing within the LEGO MINDSTORMS NXT

The NXT brick's performance depends on the batteries used and the load applied to the brick. The two figures shown below illustrate the performance of the NXT brick while using 6 standard alkaline batteries and while using the LEGO® MINDSTORMS® Lithium Ion rechargeable battery. The test was performed with two LEGO® MINDSTORMS® NXT motors attached to the NXT with the motors reversing direction every 5 sec while running at full speed.

From this test, it's clear that the NXT brick performs well when running on alkaline batteries, rechargeable lithium ion, and rechargeable Ni-MH batteries. The LEGO Lithium Ion battery is the only solution that allows the NXT to be powered continuously while the battery is connected to A/C power through a transformer. If the load is more than approximately 500mA, the battery will not be recharged.

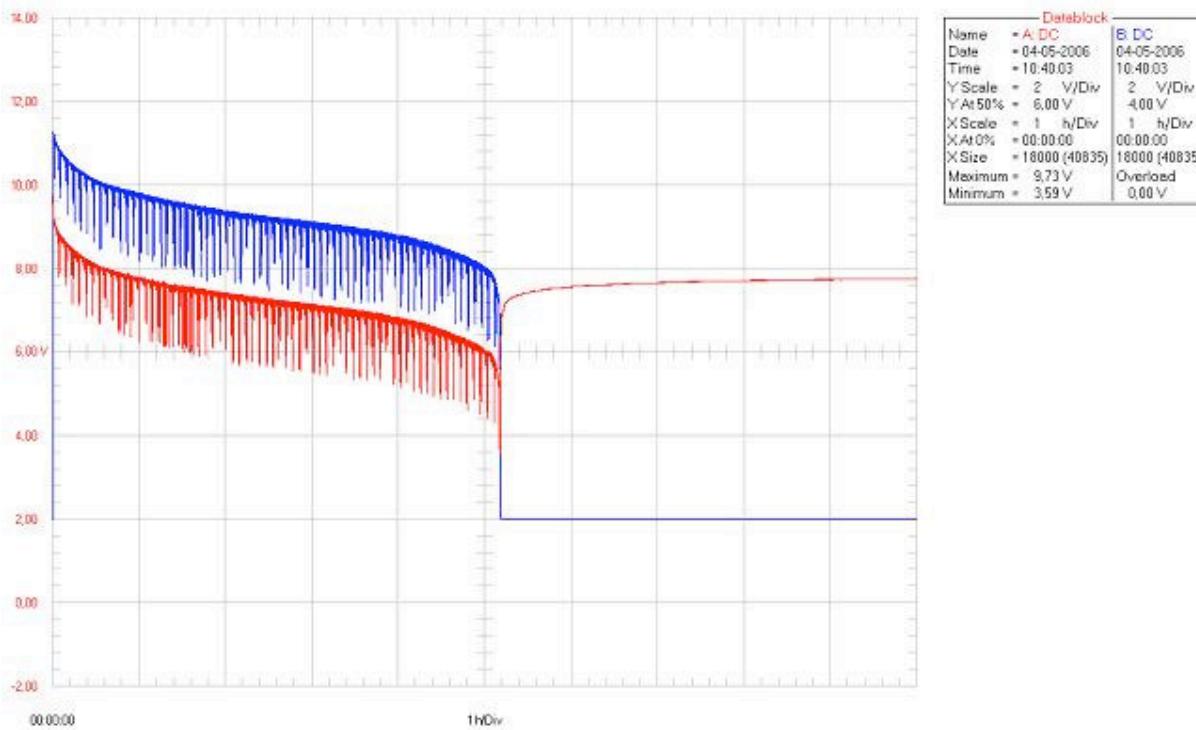


Figure 12: Load on the NXT when using standard alkaline batteries

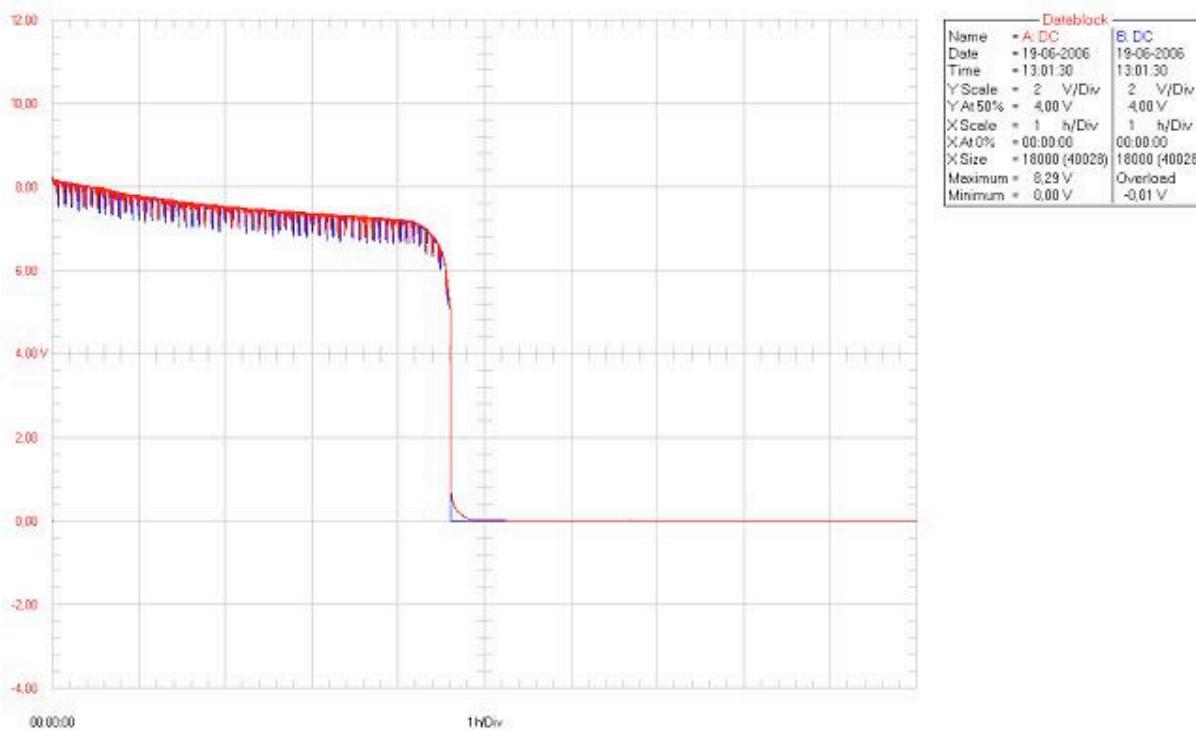


Figure 13: Load on the NXT when using the LEGO MINDSTORMS Lithium Ion rechargeable battery

BACKWARDS COMPATIBILITY

The NXT brick is backwards compatible in the sense that it is possible to use LEGO® MINDSTORMS® Robotic Invention System motors and sensors if they are connected to the NXT brick with a converter cable. The converter cable does not transform any of the signals but it does need to be connected to the correct pins of the input and output ports.

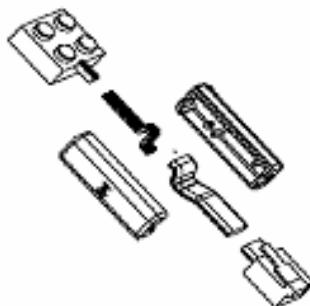


Figure 14: LEGO MINDSTORMS NXT converter cable

On the NXT's display (within the view menu), older sensors and motors have icons proceeded by an “**” icon.

Within the converter cable, pins 1 and 2 in the RJ12 connector are connected to the two connection points of the LEGO® 2x2 el-plate connector. When pins 1 and 2 are used, the converter cable can be used for both motors and sensors.

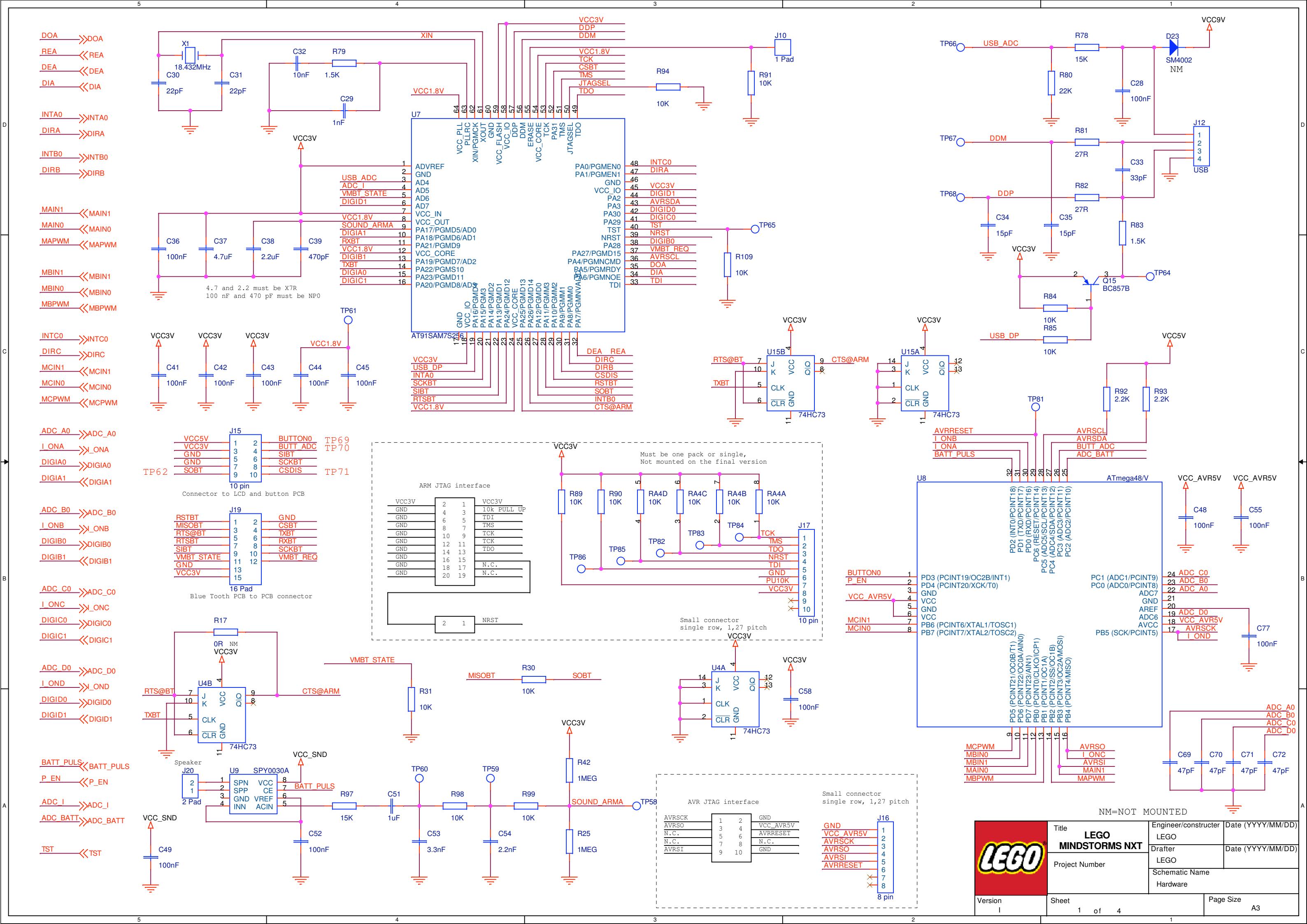
LINKS

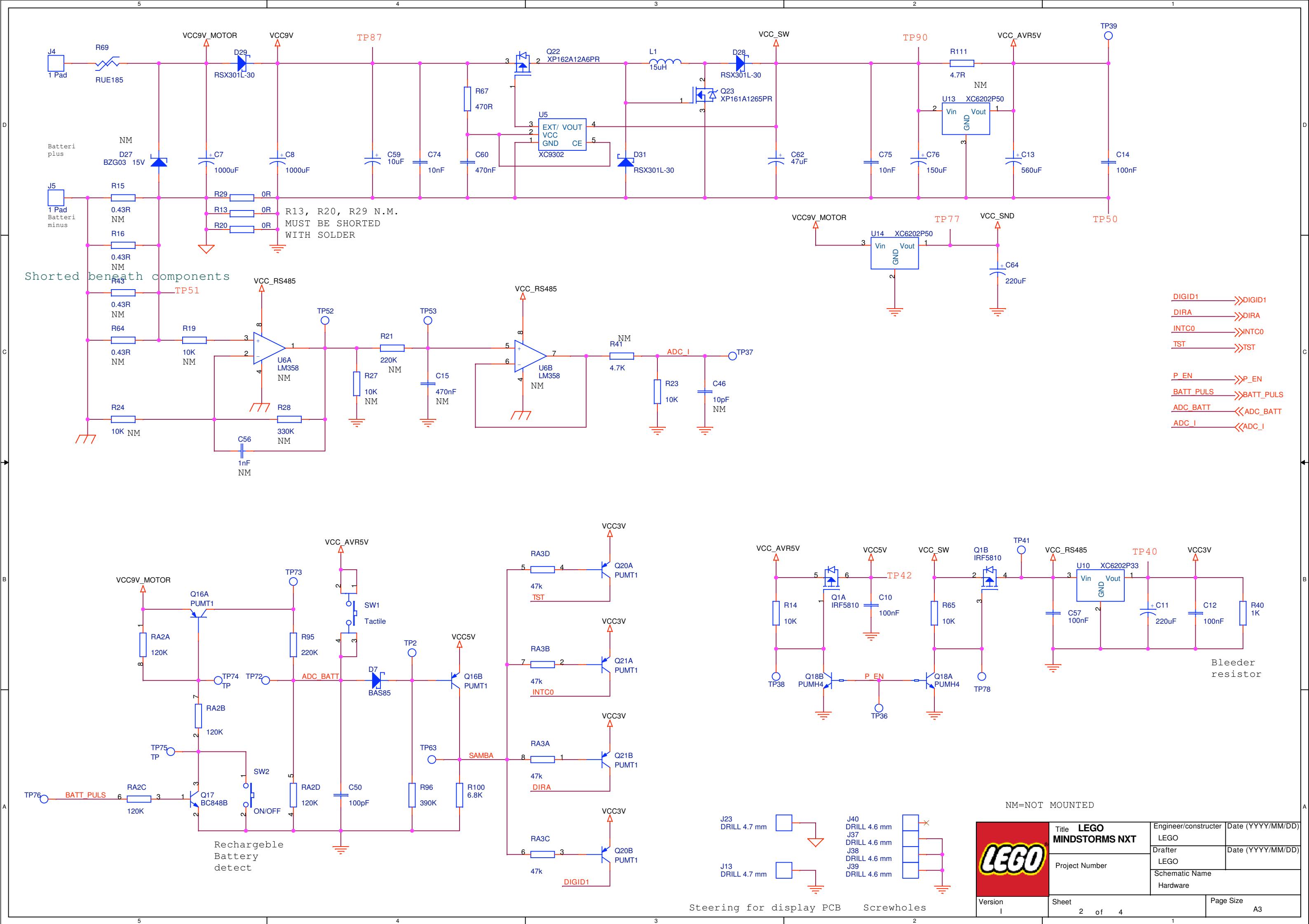
Visit these websites for more detailed documentation on the respective components and protocols:

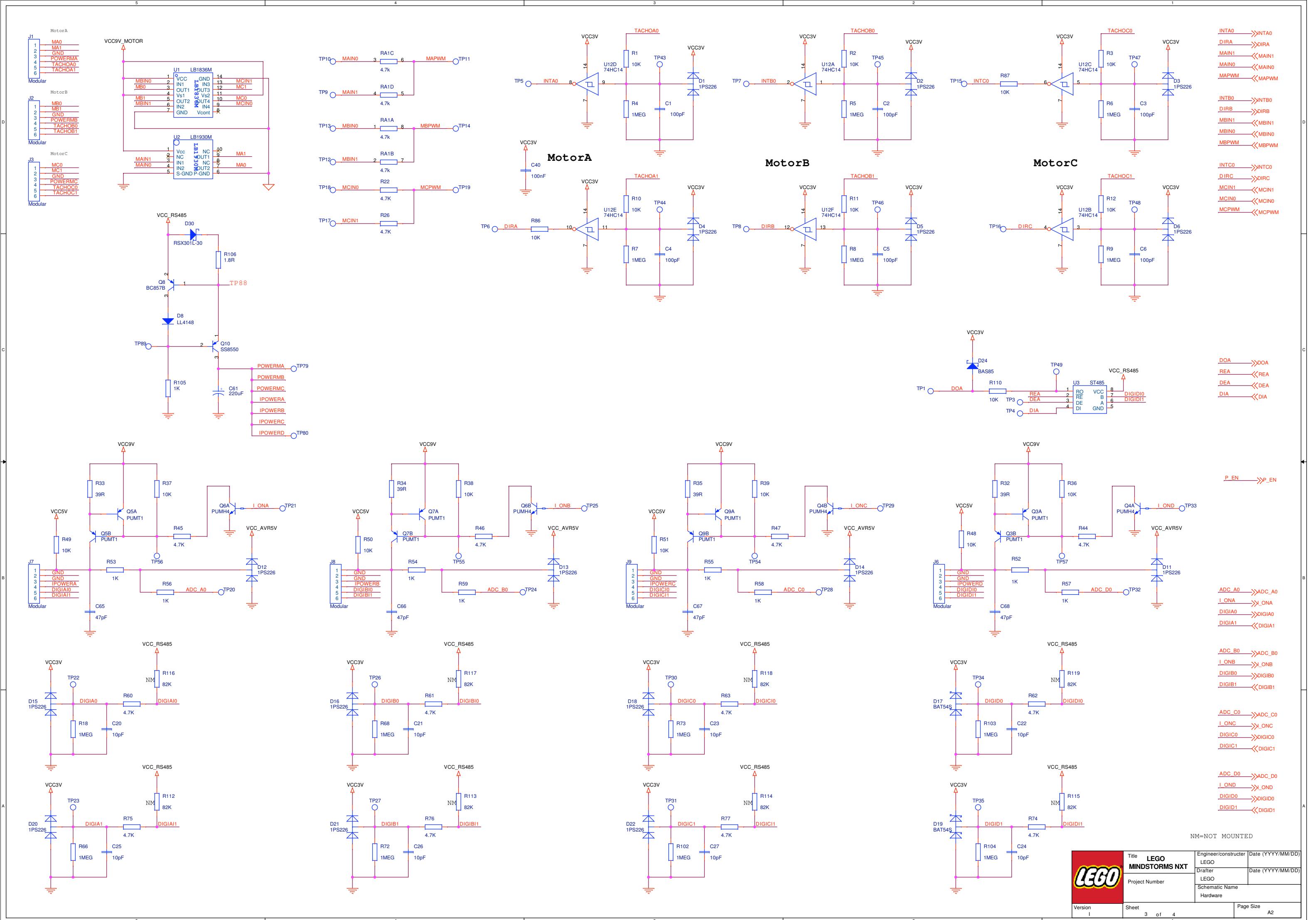
- www.at91.com
- www.atmel.com
- <http://www.standardics.philips.com/literature/books/i2c/pdf/i2c.bus.specification.pdf>
- www.P-NET.org
- www.ultrachip.com

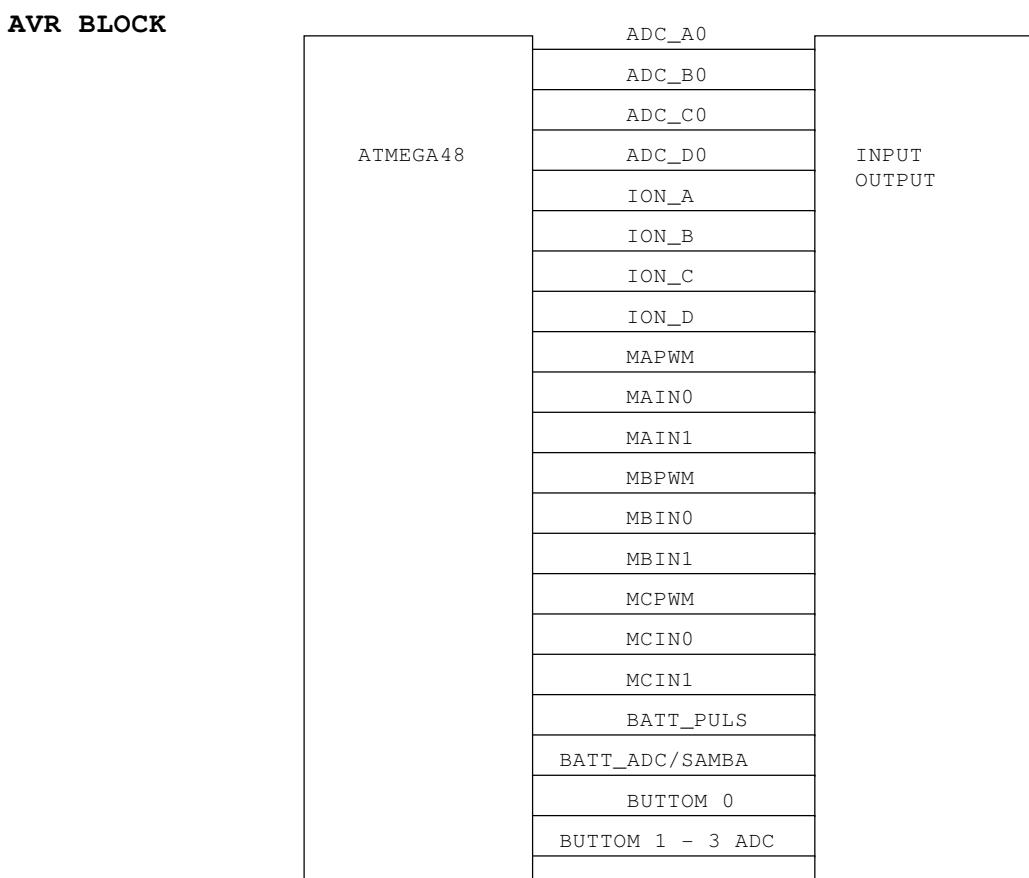
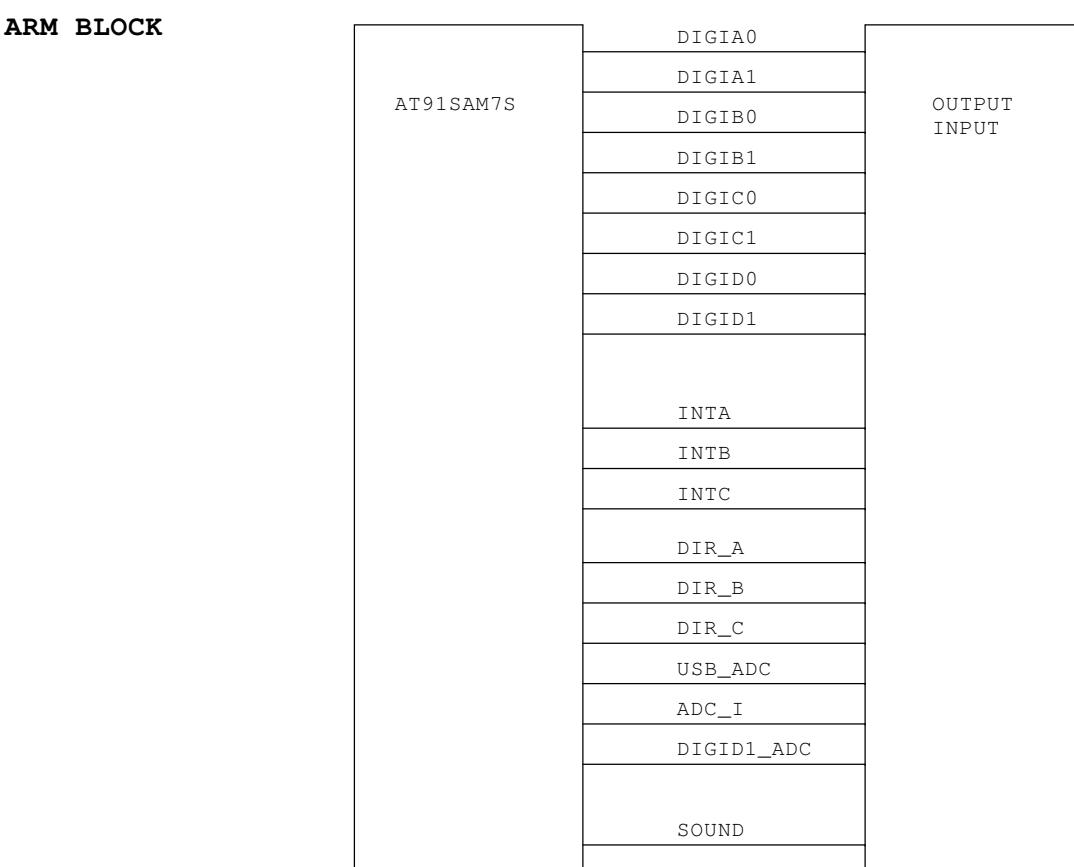
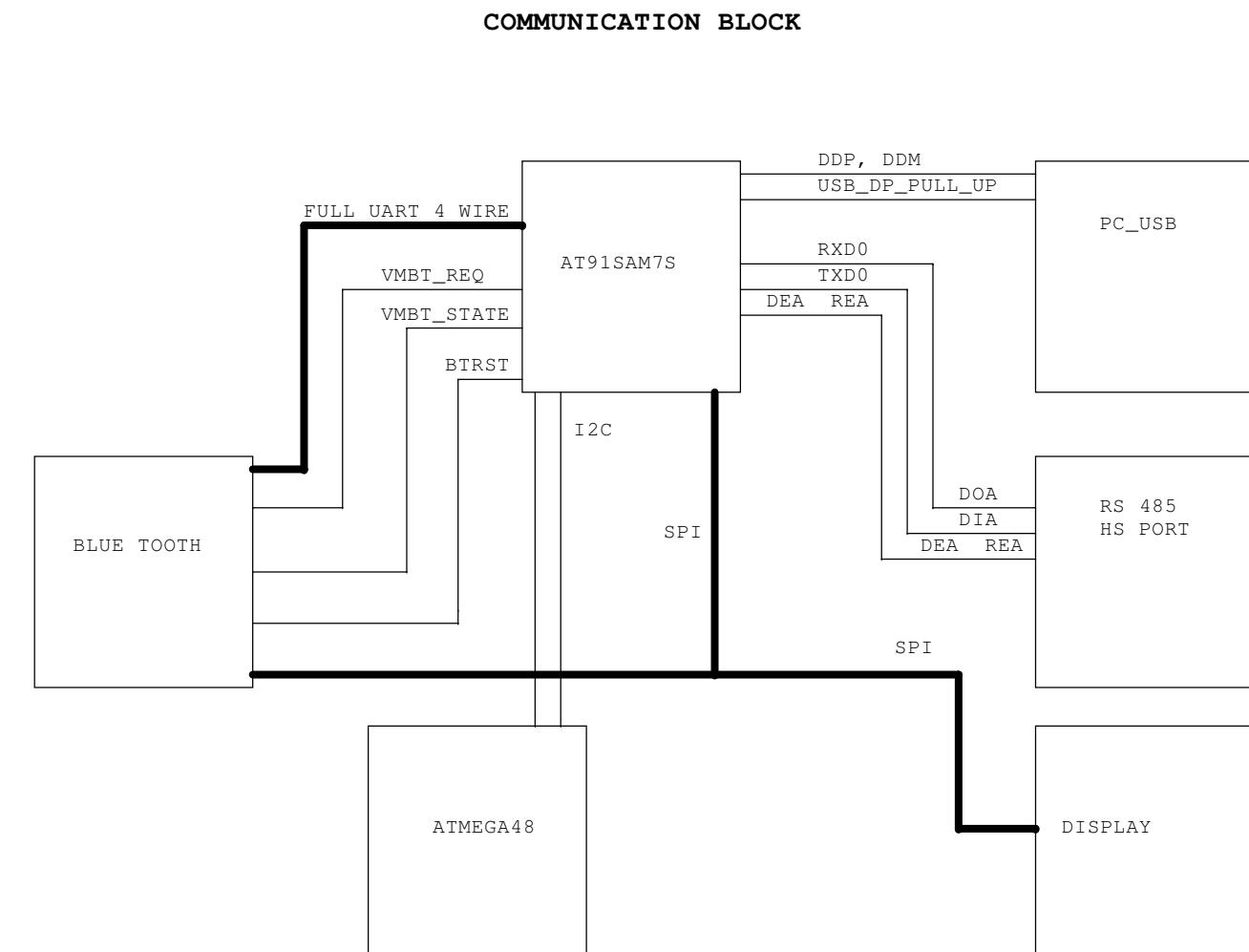
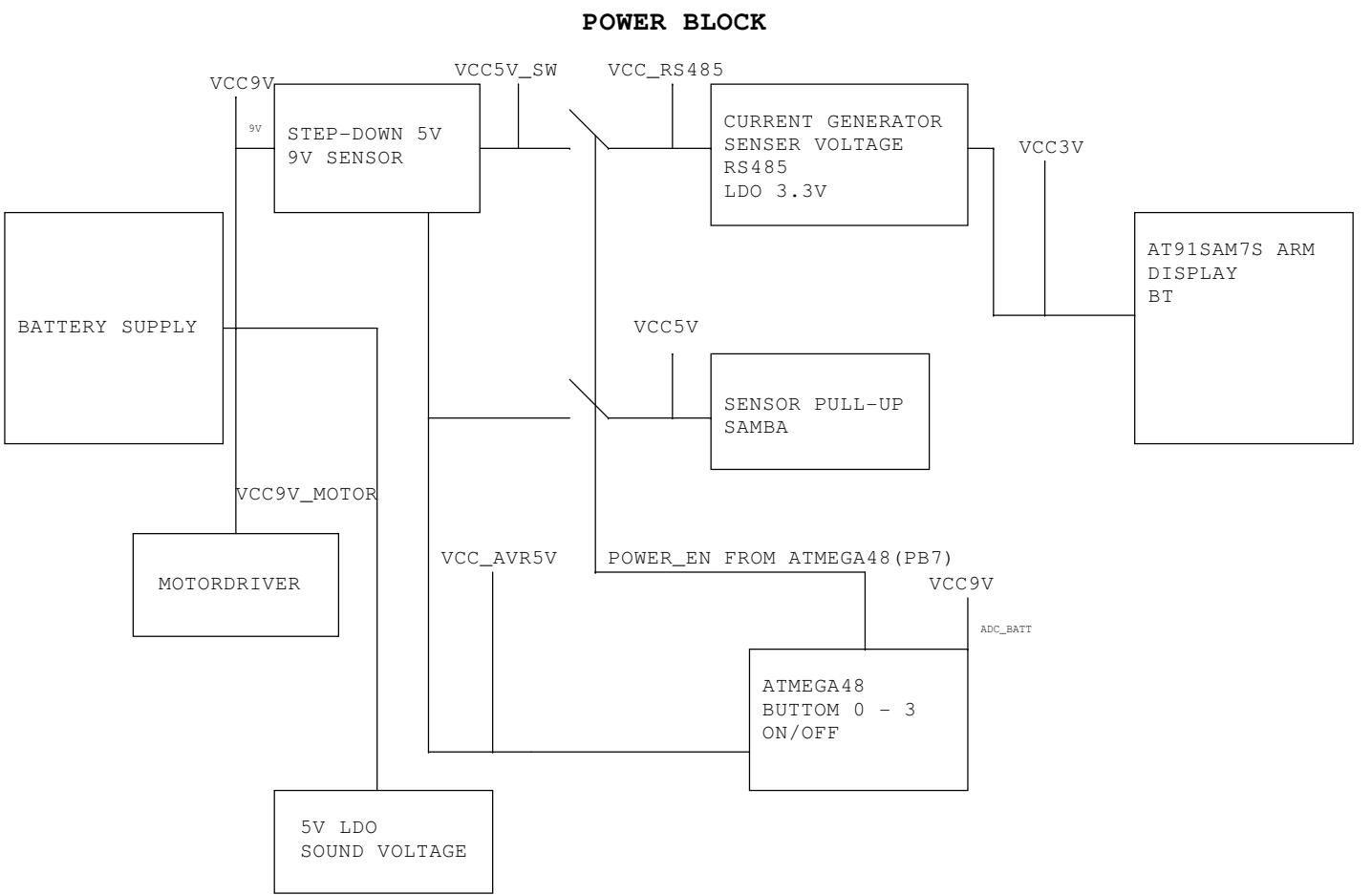
APPENDIX

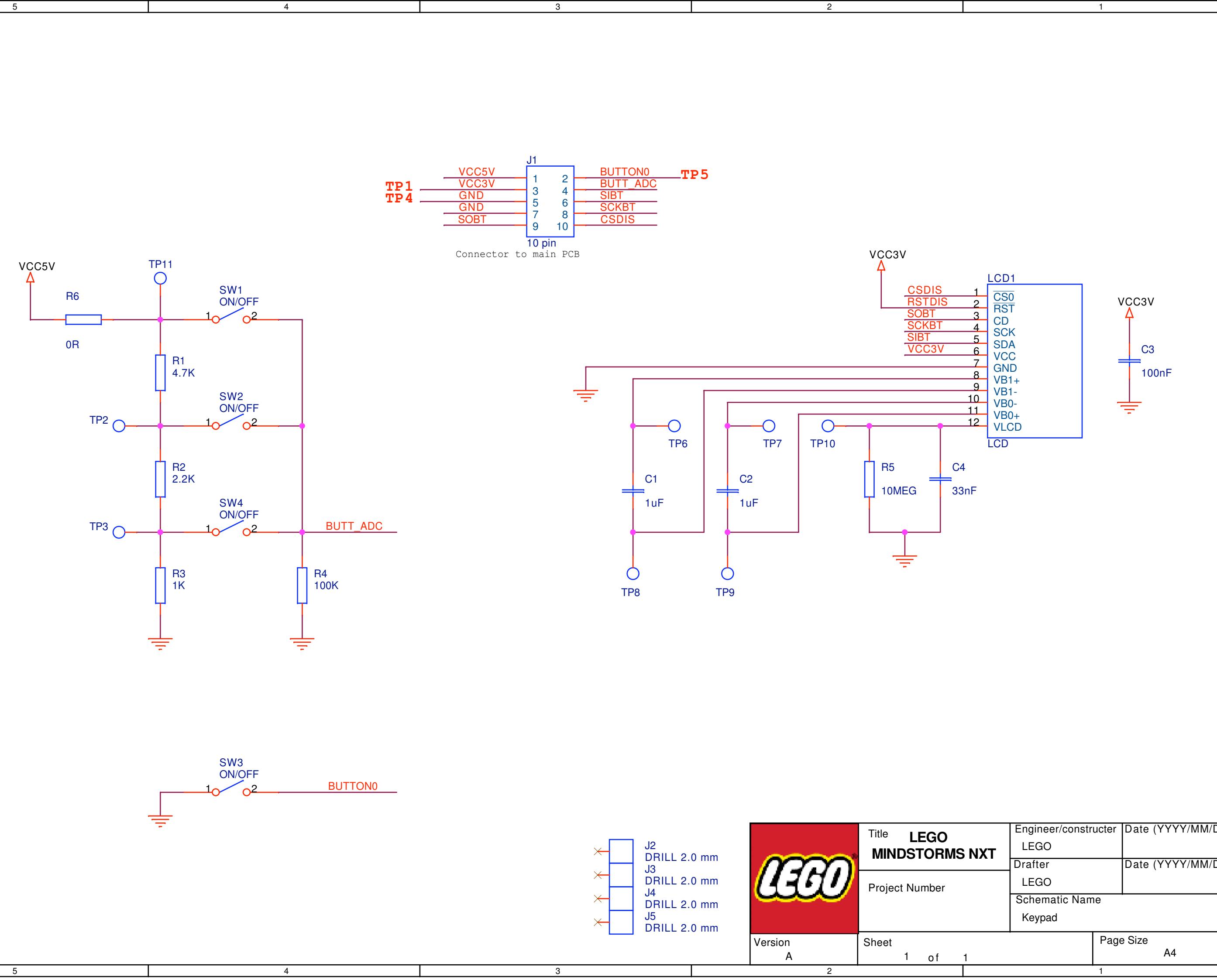
1. LEGO® MINDSTORMS® NXT hardware schematic
2. LEGO® MINDSTORMS® NXT hardware schematic
3. LEGO® MINDSTORMS® NXT Ultrasonic Sensor hardware schematic
4. LEGO® MINDSTORMS® NXT Light Sensor hardware schematic
5. LEGO® MINDSTORMS® NXT Sound Sensor hardware schematic
6. LEGO® MINDSTORMS® NXT Touch Sensor hardware schematic
7. LEGO® MINDSTORMS® NXT Ultrasonic Sensor I²C communication protocol
8. LEGO MINDSTORMS NXT ARM7 Bluetooth® interface specification

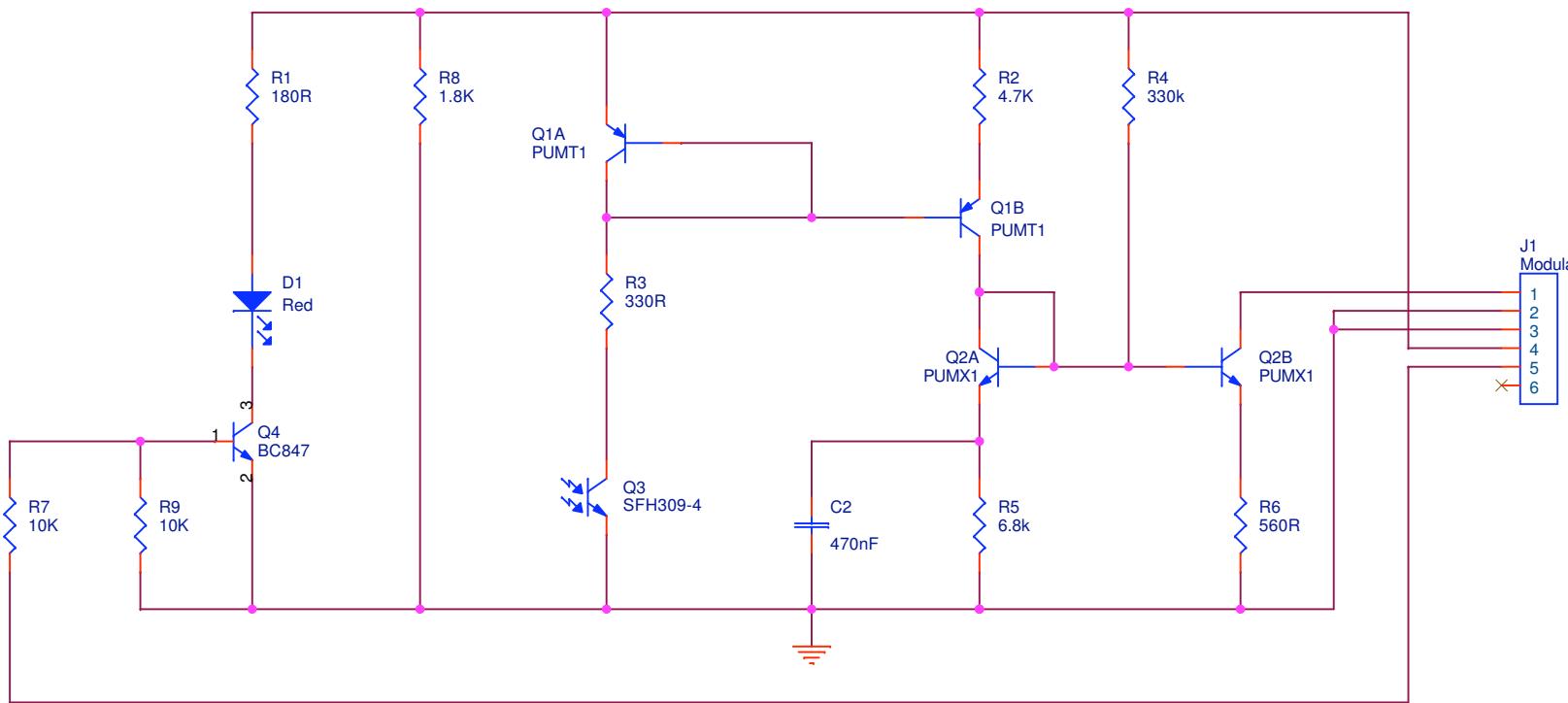












Drills for
LED fix



Title **LEGO
MINDSTORMS NXT**

Project Number

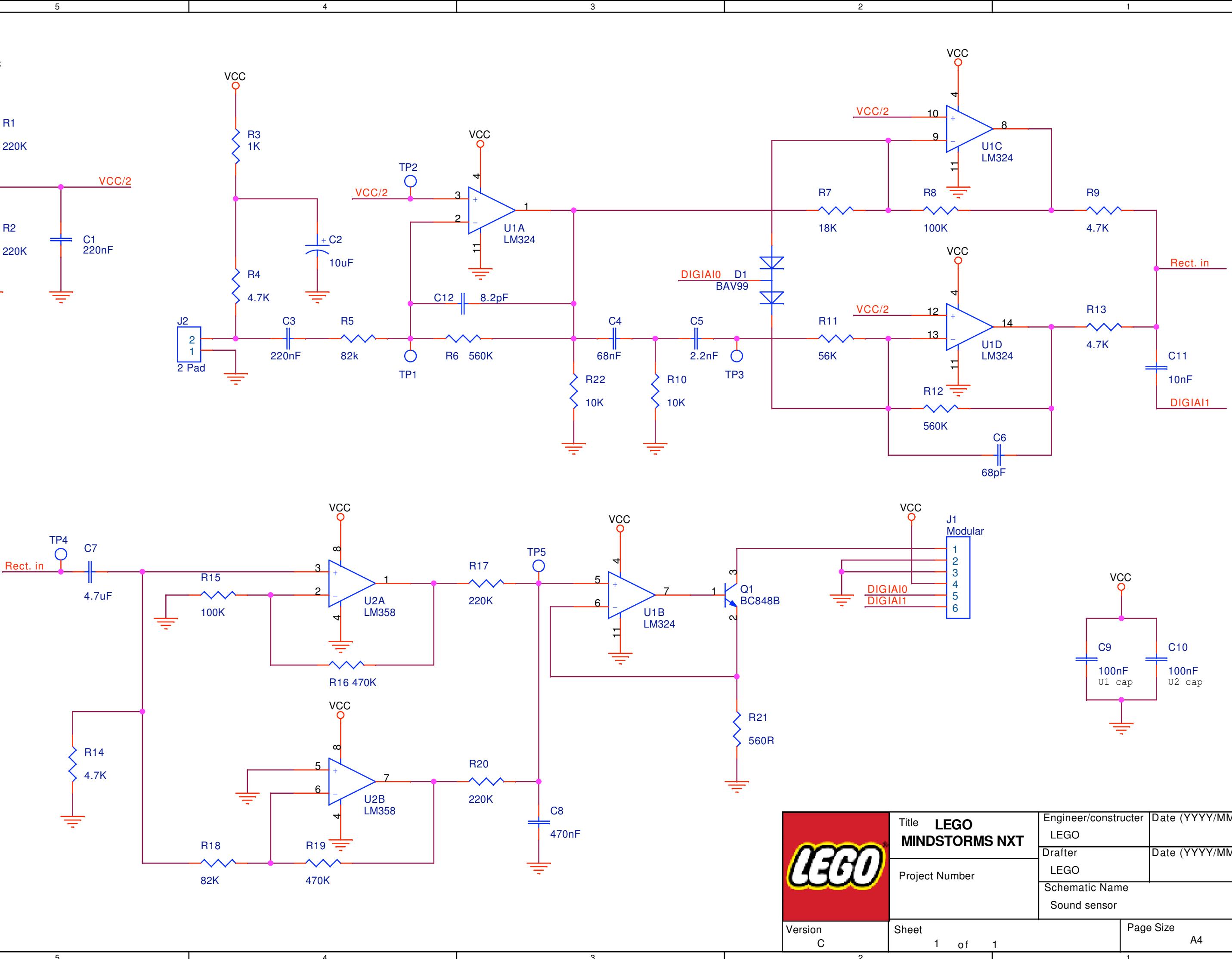
Version F

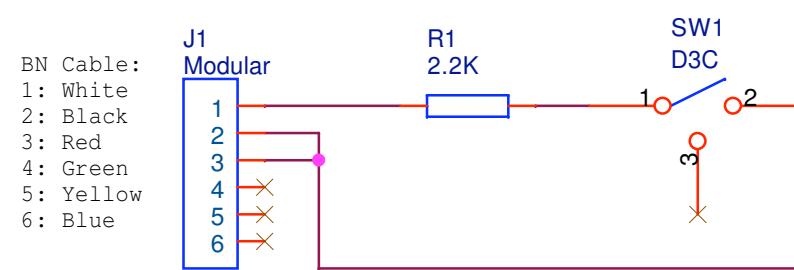
Engineer/constructor LEGO Date (YYYY/MM/DD)

Drafter LEGO Date (YYYY/MM/DD)

Schematic Name Light sensor

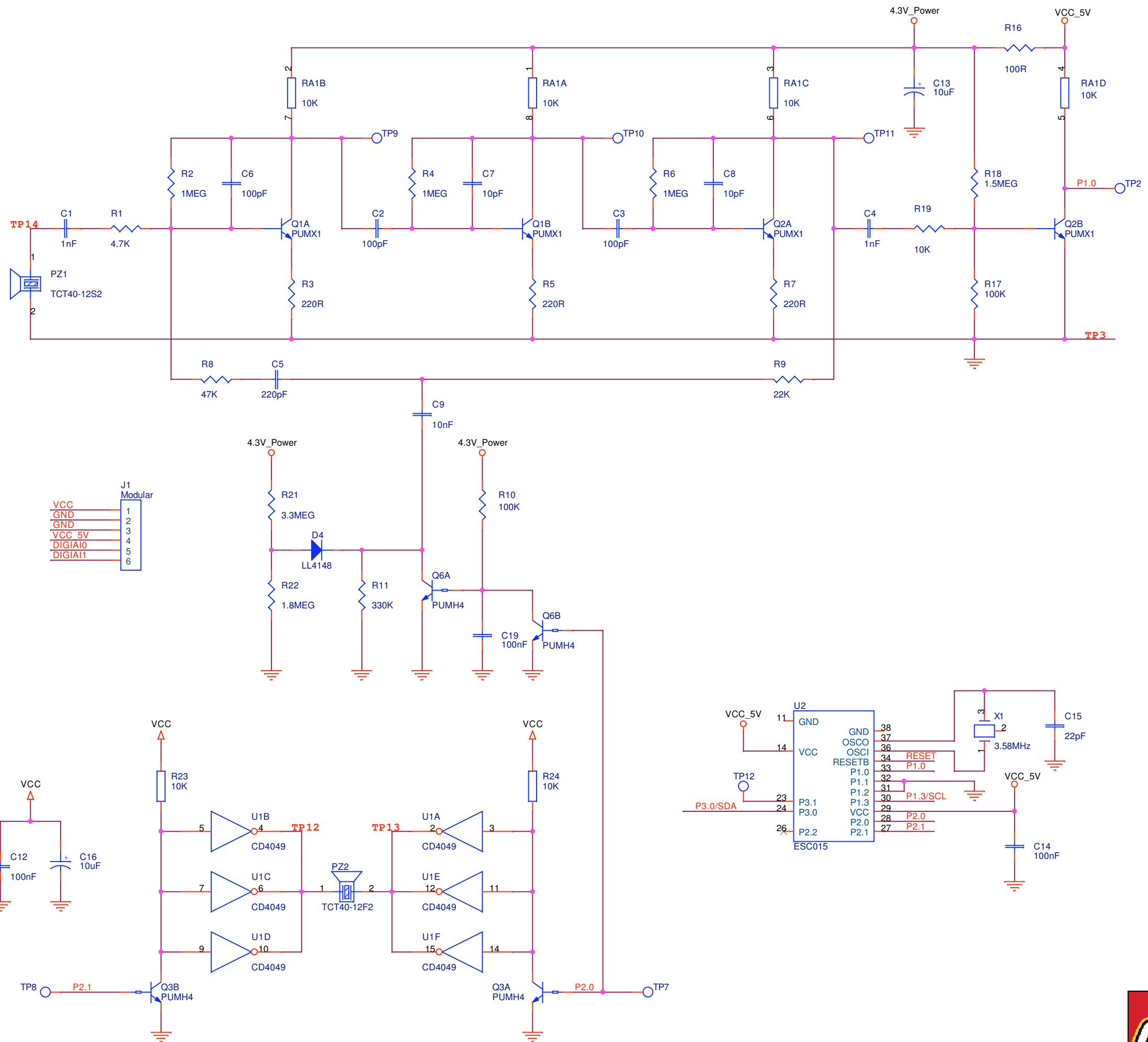
Sheet 1 of 1 Page Size A4





Title	LEGO MINDSTORMS NXT
Project Number	
Schematic Name	Touch sensor
Version	A
Sheet	1 of 1
Page Size	A4

Engineer/constructor	Date (YYYY/MM/DD)
LEGO	
Drafter	Date (YYYY/MM/DD)
LEGO	
Schematic Name	
Touch sensor	



Title	LEGO	Engineer/constructor	Date (YYYY/MM/DD)
	MINDSTORMS NXT	LEGO	
Project Number	Drafter	Date (YYYY/MM/DD)	
	LEGO		
Schematic Name			
Ultrasonic sensor			
Sheet 1	of 1	Page Size A3	



LEGO® MINDSTORMS® NXT

Ultrasonic Sensor

I²C Communication Protocol

The table documents the low-speed communication protocol which is used for communicating with the LEGO® MINDSTORMS® NXT Ultrasonic Sensor. Not all of these commands are accessible within the software, but the ultrasonic sensor itself supports these commands and functionalities.

Command	Transmitted from NXT				Received in NXT									Comments
	Byte 0	Byte 1	Byte 2	Length	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	
Addr.														
Constants														
Read version	0x02	0x00	R + 0x03	8	0x56	0x31	0x2E	0x30	0x00					V1.0
Read product ID	0x02	0x08	R + 0x03	8	0x4C	0x45	0x47	0x4F	0x00					LEGO
Read sensor type	0x02	0x10	R + 0x03	8	0x53	0x6F	0x6E	0x61	0x72	0x00				Sonar
Read factory zero (Cal 1)	0x02	0x11	R + 0x03	1	0x00									
Read factory scale factor (Cal 2)	0x02	0x12	R + 0x03	1	0x01									
Read factory scale divisor	0x02	0x13	R + 0x03	1	0x0E									
Read measurement units	0x02	0x14	R + 0x03	7	0x31	0x30	0x45	0x2D	0x32	0x6D	0x00			10E-2m
Variables														
Read continuous measurements interval	0x02	0x40	R + 0x03	1	Interval									
Read command state	0x02	0x41	R + 0x03	1	Command state									
Read Measurement Byte 0	0x02	0x42	R + 0x03	1	Result 1									
Read Measurement Byte 1	0x02	0x43	R + 0x03	1	Result 2									
Read Measurement Byte 2	0x02	0x44	R + 0x03	1	Result 3									
Read Measurement Byte 3	0x02	0x45	R + 0x03	1	Result 4									
Read Measurement Byte 4	0x02	0x46	R + 0x03	1	Result 5									
Read Measurement Byte 5	0x02	0x47	R + 0x03	1	Result 6									
Read Measurement Byte 6	0x02	0x48	R + 0x03	1	Result 7									
Read Measurement Byte 7	0x02	0x49	R + 0x03	1	Result 8									
Read actual zero (Cal 1)	0x02	0x50	R + 0x03	1	0x00									

Read actual scale factor (Cal 2)	0x02	0x51	R + 0x03	1	0x01
Read actual scale divisor	0x02	0x52	R + 0x03	1	0x0E

Commands

Off Command	0x02	0x41	0x00
Single shot command	0x02	0x41	0x01
Continuous measurement command (default)	0x02	0x41	0x02
Event capture command	0x02	0x41	0x03
Request warm reset	0x02	0x41	0x04
Set continuous measurement interval	0x02	0x40	"Interval"
Set actual zero (Cal 1)	0x02	0x50	"Value"
Set actual scale factor (Cal 2)	0x02	0x51	"Value"
Set actual scale divisor	0x02	0x52	"Value"

Single shot command:

In this mode the ultrasonic sensor will only make a new measurement every time the command byte is send to the sensor. The sensor will measure distances for up to 8 objects and save the distances within the “Read measurement byte 0 – 7”.

Continuous measurement command:

This is the default mode, where the sensor continuously makes new measurement with the specified interval.

Event capture command:

Within this mode the sensor will measure whether any other ultrasonic sensors are within the vicinity. With this information a program can evaluate when it is best to make a new measurement which will not conflict with other ultrasonic sensors.



MINDSTORMS®
NXΤ'ΡΕΜΕ

LEGO® MINDSTORMS® NXT ARM7 Bluetooth® Interface Specification

TABLE OF CONTENTS

TABLE OF CONTENTS	2
HARDWARE INTERFACE	4
Control signals	4
SPI interface	4
UART interface	4
UART INTERFACE STATES	5
Stream mode.....	5
VM Command mode	5
COMMAND MESSAGES BETWEEN BLUECORE™ & ARM7	6
Message diagram	6
BlueCore™ state diagram	7
OPERATING MODES	8
STREAM_BREAKING_MODE	8
DONT_BREAK_STREAM_MODE	8
COMMAND MESSAGE CODING	9
Message structure	9
Message Wrapping	9
COMMAND MESSAGES (ARM7 => BLUECORE™)	10
00 BeginInquiry	10
01 CancelInquiry	10
02 Connect	10
03 OpenPort	11
04 LookupName	11
05 AddDevice.....	11
06 RemoveDevice	11
07 DumpList	12
08 CloseConnection.....	12
09 AcceptConnection	12
0A PinCode.....	12
0B OpenStream.....	13
0C StartHeart.....	13
1C SetDiscoverable	13
1D ClosePort.....	13
21 SetFriendlyName	14
23 GetLinkQuality	14
25 SetFactorySettings	14
27 GetLocalAddr	14
29 GetFriendlyName.....	14
2A GetDiscoverable	15
2B GetPortOpen.....	15
2F GetVersion	15
33 GetBrickStatusbyte.....	15
34 SetBrickStatusbyte.....	15
35 GetOperatingMode	16
36 SetOperatingMode.....	16
38 GetConnectionStatus.....	16
3A GotoDFUMode	16
RESULT MESSAGES (BLUECORE™ => ARM7)	17
0D Heartbeat	17

0E InquiryRunning.....	17
0F InquiryResult.....	17
10 InquiryStopped	17
11 LookupNameResult	18
12 LookupNameFailure.....	18
13 ConnectResult	18
14 ResetIndication.....	18
15 RequestPinCode	19
16 RequestConnection	19
17 ListResult.....	19
18 ListItem.....	20
19 ListDumpStopped	20
1A CloseConnectionResult.....	20
1B PortOpenResult.....	21
1E ClosePortResult.....	21
1F PinCodeAck	21
20 SetDiscoverableAck	21
22 SetFriendlyNameAck.....	22
24 LinkQualityResult	22
26 SetFactorySettingsAck	22
28 GetLocalAddrResult.....	22
2C GetFriendlyNameResult	23
2D GetDiscoverableResult.....	23
2E GetPortOpenResult.....	23
30 GetVersionResult	23
31 GetBrickStatusbyteResult.....	23
32 SetBrickStatusbyteResult.....	24
37 OperatingModeResult	24
39 ConnectionStatusResult	24
C-CODE STANDARD FOR MESSAGE ID.....	25
Enumeration.....	25

HARDWARE INTERFACE

The Bluetooth® chip from CSR, named BlueCore™, contains all the necessary hardware to run a completely self-contained Bluetooth node. A 16-bit integrated processor runs the BT-Stack implementation from CSR called BlueLab. This firmware integrates a user programmable VM-task enabling us to control the BT node and run small amounts of application code. We have integrated a command interpreter in the VM that decodes and responds to commands received through the UART.

Our VM-code is a full implementation of both the BT SPP-A and SPP-B profiles. The two SPP profiles differ in the way a connection is established with remote BT-nodes: SPP-A is used when the local BlueCore™ chip is the connection initiator while SPP-B is used when the remote node initiates the connection.

BlueCore™ uses “stream mode” to exchange data at a rate of <= 220K baud after a connection is established. This effectively emulates a serial cable between the two connected BT-nodes. The UART is used in both stream mode and command mode (which is used to control the VM application within BlueCore™ and by extension, the Bluetooth functionality within the NXT).

The state of the UART channel is controlled by two interface signals (ARM7_CMD & BC4_CMD). The program runs on BlueLab version 3.2.

The main NXT processor that controls our user interface provides drivers for peripherals and runs user code.

CONTROL SIGNALS

- Reset: Active low signal initiated by the ARM7 that resets the BlueCore™ chip.
ARM7_CMD: Active high signal that signals the state of the UART channel seen from the ARM7. Input at BlueCore™ PIO(11).
BC4_CMD: Active high signal that signals the state of the UART channel seen from BlueCore™. Output at BlueCore™ PIO(10).

For further details, see the UART Interface States section below.

SPI INTERFACE

The SPI enables firmware updates of the BlueCore™ chip through CSR's DFU-algorithm. The SPI signals are shared with the display (except for the active low control signal).

UART INTERFACE

High-speed full-duplex interface with handshake signals (RTS & CTS).

UART settings used both in stream and command modes:

- 460.8K Baud
- 8 Bit
- No Parity
- One stop bit

UART INTERFACE STATES

STREAM MODE

BlueCore™ is fully transparent although it is controlled by UART in stream mode. It is, therefore, up to the higher firmware-levels running in the ARM7 and the remote node to pack, unpack, and interpret the data.

VM COMMAND MODE

Command mode is initiated by request from either the ARM7 or BlueCore™ chips. All data is packed, unpacked, and interpreted in accordance with the protocol definition.

State transitions

Below is shown how the ARM7 transitions the BlueCore™ chip and the UART interface from stream mode to command mode and back and vice-versa.

** 1. Create stream connection **

INIT: BC4_CMD low and ARM_CMD low
ARM: OpenStream
BC4: Open stream ARM->RADIO
BC4: Set BC4_CMD high
ARM: Set ARM_CMD high
BC4: Open stream RADIO->ARM

** 2. BC4 closes stream **

INIT: BC4_CMD high and ARM_CMD high
BC4: Close stream RADIO->ARM
BC4: Set BC4_CMD low
ARM: Set ARM_CMD low
BC4: Close stream ARM->RADIO
BC4: Send Telegram

** 3. ARM close stream **

INIT: BC4_CMD high and ARM_CMD high
ARM: Set ARM_CMD low
BC4: Close stream RADIO->ARM
BC4: Close stream ARM->RADIO
BC4: Set BC4_CMD low
ARM: Send Telegram

COMMAND MESSAGES BETWEEN BLUECORE™ & ARM7

Known Bluetooth addresses and their user-friendly names are stored in the BlueCore™ chip, enabling fast connections.

Figures 1 and 2 below show message and state diagrams for command messages sent between the BlueCore™ and ARM7 processors. All of the command messages and their respective replies are described in further detail later in this document.

MESSAGE DIAGRAM

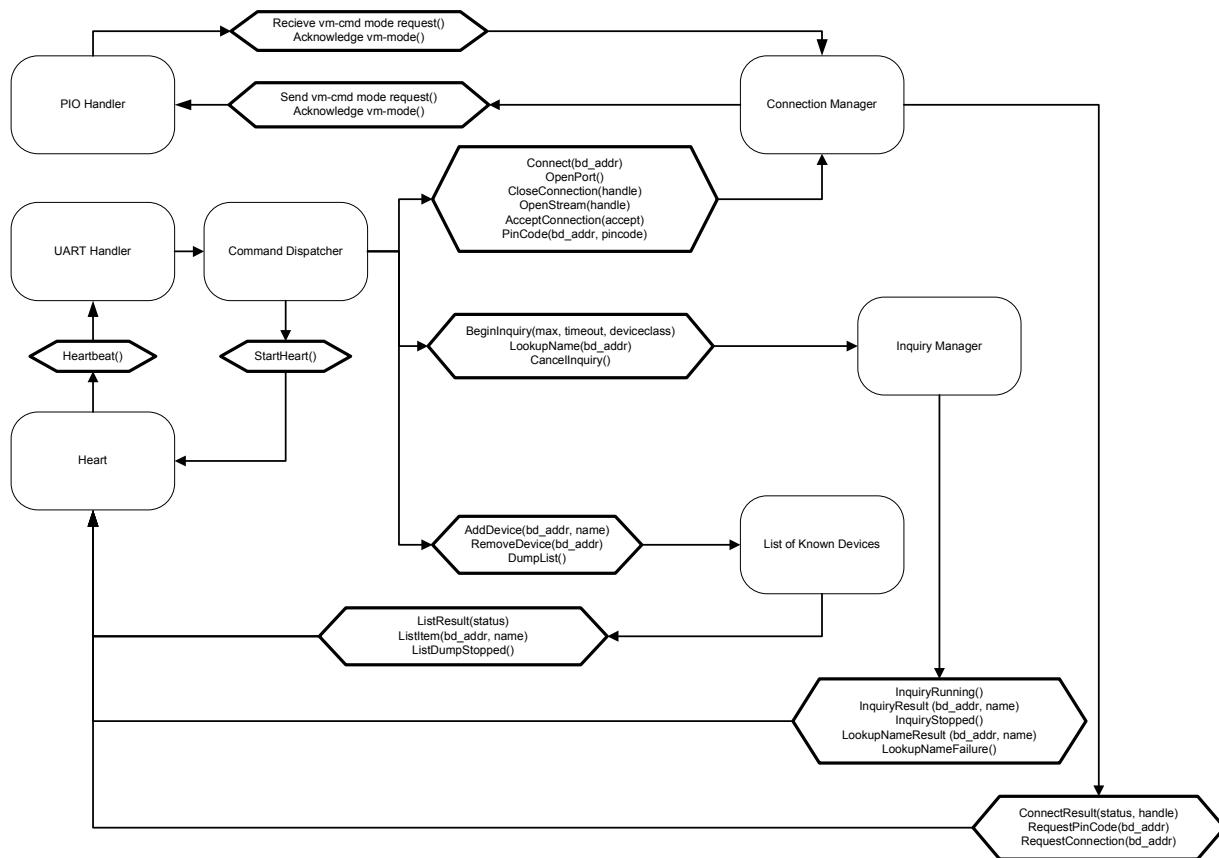
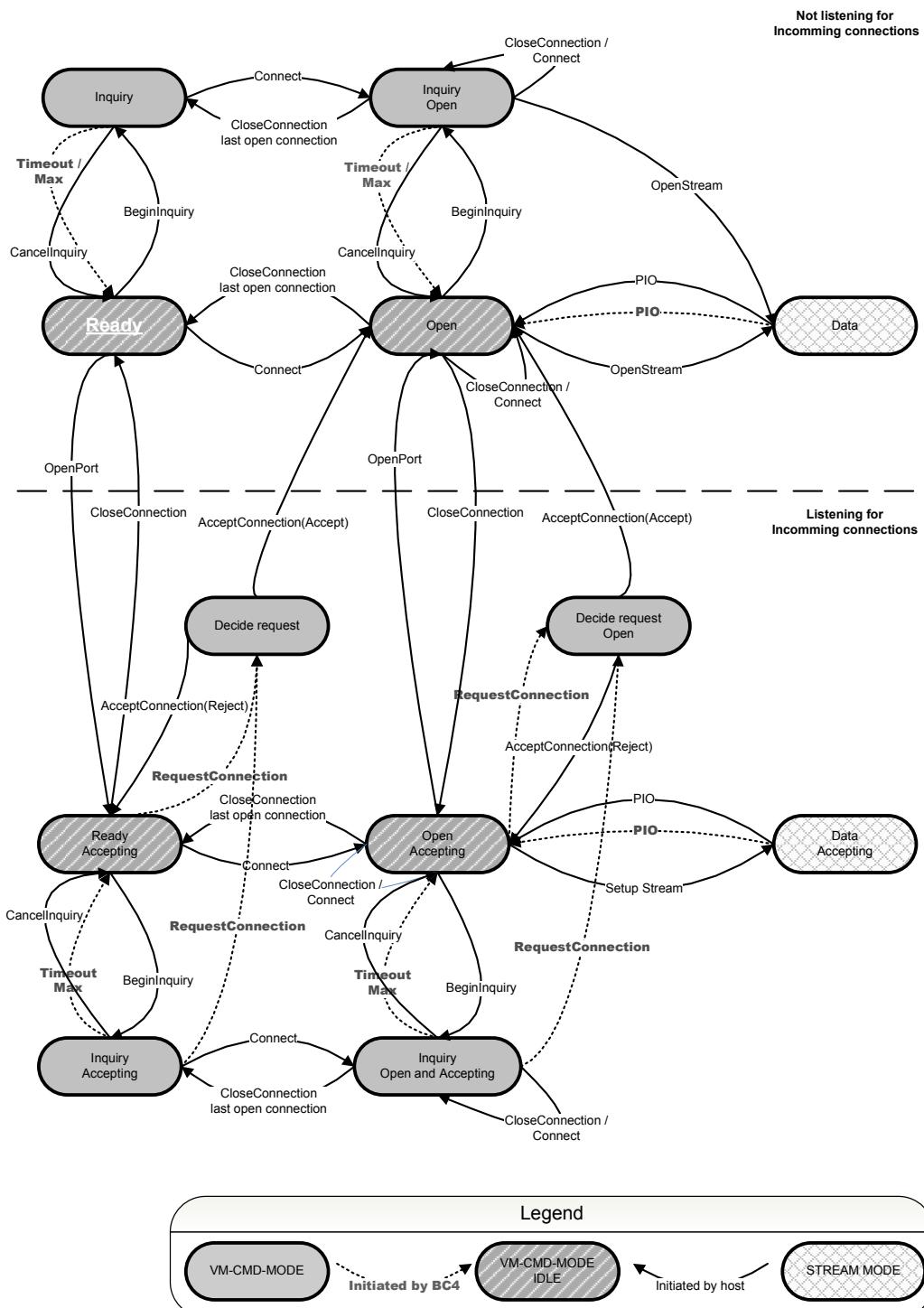


Figure 1: Block diagram for communication between the BlueCore™ and ARM7 processors

BLUECORE™ STATE DIAGRAM

Figure 2: State diagram for communication between the BlueCore™ and ARM7 processors

OPERATING MODES

Two operating modes are defined:

- STREAM_BREAKING_MODE
- DONT_BREAK_STREAM_MODE

The modes specify how incoming events are handled. Events that should be handled by the BlueCore™ chip include:

- connection requests,
- pairing requests, and
- disconnection

The operating mode is set using the SetOperatingMode telegram while the current setting can be read using the GetOperatingMode telegram. The setting is saved in the persistent storage and is auto-applied at start-up. The SetFactorySettings telegram also resets this setting.

STREAM_BREAKING_MODE

In this mode, an open stream will be closed by the BlueCore™ chip in the event of an incoming event.

DONT_BREAK_STREAM_MODE

In this mode, the BlueCore™ chip will not break any streams unless the connection currently streaming is closed.

	STREAM_BREAKING_MODE	DON'T_BREAK_STREAM_MODE
Connection request	1) If in stream_mode then break the stream 2) Send a ConnectionRequest telegram	1) If in stream_mode then reject the request 2) Else if remote device in device list then send ConnectionRequest telegram 3) Else reject the request
Pairing request	1) If in stream_mode then ignore the request 2) Else send PinCodeRequest telegram	1) If in stream_mode then ignore the request 2) Else send PinCodeRequest telegram
Disconnection	1) If in stream_mode then break the stream 2) Send CloseConnectionResult telegram	1) If in stream_mode with current stream then close stream and send CloseConnectionResult telegram 2) Else if in stream_mode then wait for cmd_mode and send CloseConnectionResult 3) Else send CloseConnectionResult

COMMAND MESSAGE CODING

MESSAGE STRUCTURE

Encoding and decoding of command mode telegrams is handled at byte level. This is necessary because of the different interpretation of variables longer than 8 bits by the two processors. The two also use different endians: ARM7 uses little endian and BlueCore™ big endian.

MESSAGE WRAPPING

|Length|Message type|Message content|SUM High byte|SUM Low byte|

Length: Ubyte, Length of the complete telegram (excluding length)

Message type: Ubyte, Defined by enumeration of all message types

Message content: Defined for each message in the entries below

SUM: Uint16, Negated sum of all previous bytes => Message ID + Message content + SUM = 0

COMMAND MESSAGES (ARM7 => BLUECORE™)

00 BEGININQUIRY

Parameters: uint8 max_devices, uint16 timeout, uint32 class_of_device

In reply to: None

Return messages: InquiryRunning, InquiryResult and InquiryStopped

This command starts the inquiry process. It is acknowledged by an InquiryRunning message from the BlueCore™ chip. The inquiry can be cancelled by sending a CancelInquiry message. The parameters are transferred directly to the corresponding parameters of the BlueLab inquiry command. The following is copied from the BlueLab documentation:

"The time the inquiry is performed for is in fact timeout * 1.28 seconds. The allowed values of timeout are in the range 0x01 to 0x30. This corresponds to an inquiry timeout range of 1.28 to 61.44 seconds."

For every device found, an InquiryResult message is sent to the host. The inquiry will run until max_devices devices have been found or timeout is reached. An InquiryStopped message indicates that the inquiry has ended. Switching to stream mode will cancel inquiry without indication.

Telegram:

```
| 10 | MSG_BeginInquiry | max_devices | timeout [Hi] | timeout[Lo] | class_of_device[hi] |
| class_of_device[hi-1] | class_of_device[hi-2] | class_of_device[lo] | SUM[Hi] | SUM[Lo] |
```

01 CANCELINQUIRY

Parameters: None

In reply to: None

Return messages: InquiryStopped

This command stops the inquiry process. It is acknowledged by an InquiryStopped message from the BlueCore™ chip. It cannot be guaranteed that no InquiryResult messages will be sent between the CancelInquiry and the InquiryStopped message but they will be kept at a minimum. Switching to stream mode will cancel inquiry without indication.

Telegram:

```
| 31 MSG_CancelInquiry | SUM[Hi] | SUM[Lo] |
```

02 CONNECT

Parameters: bdaddr device_address

In reply to: None

Return messages: ConnectResult, RequestPinCode

This message indicates to the BlueCore™ chip that the host wants to connect to a remote device. The parameter specifies the Bluetooth device address of the remote device. In reply to this message a ConnectResult message is returned indicating the success or failure of the connect operation. After a successful operation an OpenStream message can be used to switch to stream mode. To close a connection the host can send a CloseConnection message.

Telegram:

```
| 101 MSG_Connect | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[lo] | bdaddr.uap |
| bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |
```

03 OPENPORT

Parameters: None
 In reply to: None
 Return messages: OpenPortResult

To begin accepting connections from the outside, send this message to the BlueCore™ chip. This command is acknowledged by an OpenPortResult message indicating success or failure. After a successful operation, the BlueCore™ chip can send RequestConnection messages. Sending a ClosePort message to the BlueCore™ chip closes the port.

Telegram:
 | 3I MSG_OpenPort | SUM[Hi] | SUM[Lo] |

04 LOOKUPNAME

Parameters: bdaddr device_address
 In reply to: None
 Return messages: LookupNameFailure and LookupNameResult

Tells the BlueCore™ chip to look up and return the friendly name of a remote device. The result of this command will be returned in a LookupNameResult message or in case of failure, a LookupNameFailure.

Telegram:
 | 10I MSG_LookupName | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
 bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |

05 ADDDEVICE

Parameters: bdaddr device_address, char [16] name, uint32 class_of_device
 In reply to: None
 Return messages: ListResult

Adds or updates a device entry in the list of known devices. A ListResult message indicates success or failure of the operation.

Telegram:
 | 30I MSG_AddDevice | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] | bdaddr.uap |
 bdaddr.nap[Hi] | bdaddr.nap[Lo] | char [16] name | class_of_device[hi] | class_of_device[hi-1] |
 class_of_device[hi-2] | class_of_device[lo] | SUM[Hi] | SUM[Lo] |

06 REMOVEDevice

Parameters: bdaddr device_address
 In reply to: None
 Return messages: ListResult

This message can be sent to erase a device from the list of known devices. A ListResult message acknowledges the operation.

Telegram:
 | 10I MSG_RemoveDevice | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
 bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |

07 DUMPLIST

Parameters: None
In reply to: None
Return messages: ListItem and ListDumpStopped

Send this message to retrieve the list of known devices. The items on the list will be sent one by one in ListItem messages. When the last item has been sent, a ListDumpStopped message is sent.

Telegram:
I 3I MSG_DumpList I SUM[Hi] I SUM[Lo] I

08 CLOSECONNECTION

Parameters: uint8 handle
In reply to: None
Return messages: CloseConnectionResult

This message closes a connection an active connection or an open port. The handle is given by a ConnectResult message sent to the host in reply to Connect messages. The success or failure of the command is returned in a CloseConnectionResult message.

Telegram:
I 4I MSG_CloseConnection I handle I SUM[Hi] I SUM[Lo] I

09 ACCEPTCONNECTION

Parameters: uint8 accept
In reply to: RequestConnection
Return messages: ConnectResult and RequestPinCode

This message is used to indicate whether the BlueCore™ chip should accept an incoming connection. The message should be sent in reply to a RequestConnection message. The accept parameter should be set to 1 if the connection is accepted and 0 if it is not. A ConnectResult or RequestPinCode message will be sent in response to this message.

Telegram:
I 4I MSG_AcceptConnection I accept I SUM[Hi] I SUM[Lo] I

0A PINCODE

Parameters: bdaddr bd_addr, char [16] pin_code
In reply to: RequestPinCode
Return messages: PinCodeAck

This message is used to send a pin code entered by the user on to the BlueCore™ chip. This message should be used in response to a RequestPinCode message. The pin_code parameter is to be null-terminated if the pin_code is shorter than 16 chars. If none of the 16 chars is null, the pin code is assumed to be 16 chars long.

Telegram:
I 26I MSG_PinCode I bdaddr.lap[hi] I bdaddr.lap[hi-1] I bdaddr.lap[hi-2] I bdaddr.lap[Lo] I bdaddr.uap I bdaddr.nap[Hi] I bdaddr.nap[Lo] I char [16], pin_code I SUM[Hi] I SUM[Lo] I

0B OPENSTREAM

Parameters: uint8 handle

In reply to: None

Return messages: A switch to stream mode is signaled on the PIO-pins

This message will set up a stream to the connection indicated by the handle parameter. After this call, the UART will go into stream mode. The only way to break stream mode is to signal on the PIO-pins. If the call fails, we are in trouble. The host should reset the BlueCore™ chip after a timeout period. Note that this call cancels a running inquiry.

Telegram:

```
| 4| MSG_OpenStream | handle | SUM[Hi] | SUM[Lo] |
```

0C STARTHEART

Parameters: None

In reply to: None

Return messages: Heartbeat

This message indicates that the host wants to receive heartbeat signals. The Heartbeat message will be sent in reply and again every time the UART has been idle for X msec.

Telegram:

```
| 3| MSG_StartHeart | SUM[Hi] | SUM[Lo] |
```

1C SETDISCOVERABLE

Parameters: uint8 visible

In reply to: None

Return messages: SetDiscoverableAck

This message will enable or disable inquiry scanning. If the visible parameter is set to 1, the BlueCore™ chip will answer incoming inquiries. If visible is set to 0, the BlueCore™ chip will not answer, rendering the BlueCore™ chip invisible to inquiries. This does not affect the ability to accept incoming connections.

Telegram:

```
| 4| MSG_SetDiscoverable | visible | SUM[Hi] | SUM[Lo] |
```

1D CLOSEPORT

Parameters: uint8 handle

In reply to: None

Return messages: ClosePortResult

This message will close the port. Until it is removed, the handle should always be 03...

Telegram:

```
| 4| MSG_ClosePort | handle | SUM[Hi] | SUM[Lo] |
```

21 SETFRIENDLYNAME

Parameters: char [16] name
In reply to: None
Return messages: SetFriendlyName Ack

This message is used to set the friendly name of the local device. The name parameter is to be null-terminated if the name is shorter than 16 chars. If none of the 16 chars is null, the name is assumed to be 16 chars long.

Telegram:

```
| 19 | MSG_SetFriendlyName | char [16] name | SUM[Hi] | SUM[Lo] |
```

23 GETLINKQUALITY

Parameters: uint8 handle
In reply to: None
Return messages: LinkQualityResult

This message requests a reading of the HCI link quality of a connection.

Telegram:

```
| 41 | MSG_GetLinkQuality | handle | SUM[Hi] | SUM[Lo] |
```

25 SETFACTORYSETTINGS

Parameters: None
In reply to: None
Return messages: SetFactorySettingsAck

This message is sent to clear the settings in the persistent storage. The BlueCore™ chip should be restarted after calling this function. Otherwise old values can be floating around the BlueCore™ chip causing unexpected behavior.

Telegram:

```
| 31 | MSG_SetFactorySettings | SUM[Hi] | SUM[Lo] |
```

27 GETLOCALADDR

Parameters: None
In reply to: None
Return messages: GetLocalAddrResult

This message will fetch the local Bluetooth device address.

Telegram:

```
| 31 | MSG_GetLocalAddr | SUM[Hi] | SUM[Lo] |
```

29 GETFRIENDLYNAME

Parameters: None
In reply to: None
Return messages: GetFriendlyNameResult

This message will fetch the friendly name of the local Bluetooth device.

Telegram:

```
| 31 | MSG_GetFriendlyName | SUM[Hi] | SUM[Lo] |
```

2A GETDISCOVERABLE

Parameters:

None

In reply to:

None

Return messages:

GetDiscoverableResult

This message will fetch the status of the discoverable local Bluetooth devices.

Telegram:

| 3I MSG__GetDiscoverable | SUM[Hi] | SUM[Lo] |

2B GETPORTOPEN

Parameters:

None

In reply to:

None

Return messages:

GetPortOpenResult

This message will fetch the status of the local Bluetooth device port.

Telegram:

| 3I MSG__GetPortOpen | SUM[Hi] | SUM[Lo] |

2F GETVERSION

Parameters:

None

In reply to:

None

Return messages:

GetVersionResult

This message will fetch the version of the BlueCore™ code.

Telegram:

| 3I MSG__GetVersionOpen | SUM[Hi] | SUM[Lo] |

33 GETBRICKSTATUSBYTE

Parameters:

None

In reply to:

None

Return messages:

GetBrickStatusbyteResult

This message will fetch the status bytes from persistent storage.

Telegram:

| 3I MSG__GetBrickStatusbyte | SUM[Hi] | SUM[Lo] |

34 SETBRICKSTATUSBYTE

Parameters:

uint8 byte1, uint8 byte2

In reply to:

None

Return messages:

SetBrickStatusbyteResult

This message set the status bytes in the persistent storage.

Telegram:

| 5I MSG__SetBrickStatusbyte | byte1 | byte2 | SUM[Hi] | SUM[Lo] |

35 GETOPERATINGMODE

Parameters: None
In reply to: None
Return messages: OperatingModeResult

This message gets the operating mode of the brick. See “36 SetOperatingMode” for a description of the modes.

Telegram:

I 3 I MSG_GetOperatingMode I SUM[Hi] I SUM[Lo] I

36 SETOPERATINGMODE

Parameters: uint8 mode
In reply to: None
Return messages: OperatingModeResult

This message sets the operating mode of the brick. The mode should be one of:

```
typedef enum {
    STREAM_BREAKING_MODE,
    DONT_BREAK_STREAM_MODE
} OperatingMode;
```

Telegram:

I 4 I MSG_SetOperatingMode I mode I SUM[Hi] I SUM[Lo] I

38 GETCONNECTIONSTATUS

Parameters: None
In reply to: None
Return messages: ConnectionStatusResult

This message gets the connection status of the brick.

Telegram:

I 3 I MSG_GetConnectionStatus I SUM[Hi] I SUM[Lo] I

3A GOTODFUMODE

Parameters: None
In reply to: None
Return messages: None

This message will cause a warm reboot into DFU boot mode.

Telegram:

I 3 I MSG_GotoDFUMode I SUM[Hi] I SUM[Lo] I

RESULT MESSAGES (BLUECORE™ => ARM7)

0D HEARTBEAT

Parameters: None
In reply to: StartHeart, and a BlueCore™ initiated mode shift to command mode
Return messages: None

After a StartHeart message is sent, this message is sent periodically. See the description of the StartHeart message for details.

Telegram:
I 3I MSG_Heartbeat I SUM[Hi] I SUM[Lo] I

0E INQUIRYRUNNING

Parameters: None
In reply to: BeginInquiry
Return messages: None

Sent as acknowledgement of a BeginInquiry message. This message will be followed by zero or more InquiryResult messages and lastly by an InquiryStopped message.

Telegram:
I 3I MSG_InquiryRunning I SUM[Hi] I SUM[Lo] I

0F INQUIRYRESULT

Parameters: bdaddr device_address, char [16] name, uint32 class_of_device
In reply to: BeginInquiry
Return messages: None

For each device found in an inquiry, this message is sent from the BlueCore™ to the host. The message contains the device address of the device and the friendly name. If the friendly name is less than 16 chars long, the string is null-terminated; otherwise it is assumed to be 16 chars long. The class_of_device parameter contains the device class identifier as specified in the Bluetooth specifications.

Telegram:
I 30 I MSG_InquiryResult I bdaddr.lap[hi] I bdaddr.lap[hi-1] I bdaddr.lap[hi-2] I bdaddr.lap[Lo] I bdaddr.uap I bdaddr.nap[Hi] I bdaddr.nap[Lo] I char [16], name I class_of_device[hi] I class_of_device[hi-1] I class_of_device[hi-2] I class_of_device[lo] I SUM[Hi] I SUM[Lo] I

10 INQUIRYSTOPPED

Parameters: None
In reply to: BeginInquiry
Return messages: None

This message indicates that an inquiry has ended. This may be because a timeout or that the maximum number of found devices has been reached. An inquiry will also end if the UART enters stream mode but this will not generate an InquiryStopped message.

Telegram:
I 3I MSG_InquiryStopped I SUM[Hi] I SUM[Lo] I

11 LOOKUPNAMERESULT

Parameters: bdaddr device_address, char [16] name, uint32 class_of_device
 In reply to: LookupName
 Return messages: None

This message is sent in response to a LookupName command message. The message contains the device address and the friendly name of the device. If the friendly name is less than 16 chars long, the string is null-terminated; otherwise it is assumed to be 16 chars long.

The class of device is read from the device list. If the device is not on the list, zero is returned.

Telegram:

```
| 30| MSG__LookupNameResult | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
  bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | char [16], name | class_of_device[hi] |
  class_of_device[hi-1] | class_of_device[hi-2] | class_of_device[lo] | SUM[Hi] | SUM[Lo] |
```

12 LOOKUPNAMEFAILURE

Parameters: bdaddr device_address
 In reply to: LookupName
 Return messages: None

This message is sent as response to a LookupName command message in case of failure.

Telegram:

```
| 10| MSG__LookupNameFailure | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
  bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |
```

13 CONNECTRESULT

Parameters: uint8 status, uint8 handle
 In reply to: Connect and OpenPort
 Return messages: None

This message is sent in response to Connect and OpenPort messages. The status parameter is 1 if the Connect or OpenPort operation was a success and 0 if it is not.

Telegram:

```
| 5| MSG__ConnectResult | status | handle | SUM[Hi] | SUM[Lo] |
```

14 RESETINDICATION

Parameters: None
 In reply to: None
 Return messages: None

This message is sent to the host when the BlueCore™ chip is finished with its initialization.

Telegram:

```
| 3| MSG__ResetIndication | SUM[Hi] | SUM[Lo] |
```

15 REQUESTPINCODE

Parameters: bdaddr device_address
 In reply to: None
 Return messages: PinCode

This message is sent if a remote device is requesting a pin code. The host should prompt the user for a pin code and return it in a PinCode message. The device_address parameter contains the device address of the remote device.

Telegram:

```
| 10| MSG_RequestPinCode | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |
```

16 REQUESTCONNECTION

Parameters: bdaddr device_address
 In reply to: None
 Return messages: AcceptConnection

This message is sent to the host if a remote device wants to connect to the BlueCore™ chip. The host should respond by sending an AcceptConnection message indicating whether or not the connection should be accepted.

Telegram:

```
| 10| MSG_RequestConnection | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |
bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |
```

17 LISTRESULT

Parameters: uint8 status
 In reply to: AddDevice or RemoveDevice
 Return messages: None

The status parameter is given by the following enum:

```
enum {
    LR_SUCCESS = 0x50,
    LR_COULD_NOT_SAVE,
    LR_STORE_IS_FULL,
    LR_ENTRY_REMOVED,
    LR_UNKNOWN_ADDR
};
```

LR_SUCCESS: Indicates that the operation was successful.

LR_COULD_NOT_SAVE: The entry could not be written to persistent storage. The BlueCore™ chip must be reset to activate defragmentation of the flash.

LR_STORE_IS_FULL: There are no empty slots in the list to save an entry to. Use RemoveDevice to create an open slot.

LR_ENTRY_REMOVED: The entry was successfully removed.

LR_UNKNOWN_ADDR: The list does not contain an entry with the provided Bluetooth device address.

Telegram:

```
| 4| MSG_ListResult | status | SUM[Hi] | SUM[Lo] |
```

18 LISTITEM

Parameters: bdaddr device_address, char [16] name, uint32 class_of_device
 In reply to: DumpList
 Return messages: None

This message is sent from the BlueCore™ chip to the host for each device found on the list of known devices. The message contains the device address of the device and the friendly name. If the friendly name is less than 16 chars long the string is null-terminated; otherwise it is assumed to be 16 chars long.

Telegram:

```
| 26| MSG_ListItem | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] | bdaddr.uap |  
bdaddr.nap[Hi] | bdaddr.nap[Lo] | char [16] name | class_of_device[hi] | class_of_device[hi-1] |  
class_of_device[hi-2] | class_of_device[lo] | SUM[Hi] | SUM[Lo] |
```

19 LISTDUMPSTOPPED

Parameters: None
 In reply to: DumpList
 Return messages: None

This message indicates that the list dump was completed.

Telegram:

```
| 3| MSG_ListDumpStopped | SUM[Hi] | SUM[Lo] |
```

1A CLOSECONNECTIONRESULT

Parameters: uint8 status, uint8 handle
 In reply to: CloseConnection
 Return messages: None

This message is sent in response to CloseConnection messages. The status parameter is given by the following enum:

```
typedef enum
{
    /*! Successful disconnection.*/
    spp_disconnect_success,
    /*! Unsuccessful due to the link being lost.*/
    spp_disconnect_link_loss,
    /*! Unsuccessful due to no service level connection.*/
    spp_disconnect_no_slc,
    /*! Unsuccessful due to time out.*/
    spp_disconnect_timeout,
    /*! Unsuccessful for some other reason.*/
    spp_disconnect_error
} spp_disconnect_status;
```

Telegram:

```
| 5| MSG_CloseConnectionResult | status | handle | SUM[Hi] | SUM[Lo] |
```

1B PORTOPENRESULT

Parameters: uint8 status, uint8 handle, uint8 ps_success
In reply to: OpenPort
Return messages: None

This message is the result of a PortOpen command. It will contain the status 1 if successful and 0 otherwise. Ps_Success is 0 if the port-open could not be written to persistent storage and 1 if it could. The port will be opened regardless of the persistent storage.

Telegram:

```
| 6 | MSG_PortOpenResult | status | handle | ps_success | SUM[Hi] | SUM[Lo] |
```

1E CLOSEPORTRESULT

Parameters: uint8 status, uint8 handle, uint8 ps_sucess
In reply to: ClosePort
Return messages: None

This message is the result of an CloseOpen command. It will contain the status 1 if successful and 0 otherwise. Ps_Success is 0 if the port-close could not be written to persistent storage and 1 if it could. The port will be closed regardless of the persistent storage.

Telegram:

```
| 6 | MSG_ClosePortResult | status | handle | ps_success | SUM[Hi] | SUM[Lo] |
```

1F PINCODEACK

Parameters: None
In reply to: PinCode
Return messages: None

This message is sent after a PinCode message is received. See the description of the PinCode telegram for details.

Telegram:

```
| 3 | MSG_PinCodeAck | SUM[Hi] | SUM[Lo] |
```

20 SETDISCOVERABLEACK

Parameters: uint8 success
In reply to: SetDiscoverable
Return messages: None

This message is sent after a SetDiscoverable message is received. See the description of the SetDiscoverable telegram for details. Success is 0 if the name could not be written to persistent storage and 1 if it could. The discoverability will be changed regardless of the persistent storage.

Telegram:

```
| 3 | MSG_SetDiscoverableAck | success | SUM[Hi] | SUM[Lo] |
```

22 SETFRIENDLYNAMEACK

Parameters: uint8 success
In reply to: SetFriendlyName
Return messages: None

This message is sent after a SetFriendlyName message is received. See the description of the SetFriendlyName telegram for details. Success is 0 if the name could not be written to persistent storage and 1 if it could. The name will be changed regardless of the persistent storage.

Telegram:

```
| 4 | MSG_SetFriendlyNameAck | success | SUM[Hi] | SUM[Lo] |
```

24 LINKQUALITYRESULT

Parameters: uint8 quality
In reply to: GetLinkQuality
Return messages: None

This message contains the result of a GetLinkQuality message. The quality is an octet value ranging from 0x00 to 0xFF. If the value is high, the link quality is better.

Telegram:

```
| 4 | MSG_GetLinkQuality | quality | SUM[Hi] | SUM[Lo] |
```

26 SETFACTORYSETTINGSACK

Parameters: None
In reply to: SetFactorySettings
Return messages: None

This message is sent when the settings in the persistent storage have been cleared.

Telegram:

```
| 3 | MSG_SetFactorySettingsAck | SUM[Hi] | SUM[Lo] |
```

28 GETLOCALADDRRESULT

Parameters: bdaddr addr
In reply to: GetLocalAddr
Return messages: None

This message returns the local Bluetooth device address.

Telegram:

```
| 10 | MSG_GetLocalAddrResult | bdaddr.lap[hi] | bdaddr.lap[hi-1] | bdaddr.lap[hi-2] | bdaddr.lap[Lo] |  
bdaddr.uap | bdaddr.nap[Hi] | bdaddr.nap[Lo] | SUM[Hi] | SUM[Lo] |
```

2C GETFRIENDLYNAMERESULT

Parameters: char name[16]
In reply to: GetFriendlyName
Return messages: None

This message returns the friendly name of the local Bluetooth device. If the name is shorter than 16 chars the name will be zero-padded.

Telegram:

```
| 9 | MSG__GetFriendlyNameResult | char [16] | SUM[Hi] | SUM[Lo] |
```

2D GETDISCOVERABLERESULT

Parameters: uint8 discoverable
In reply to: GetDiscoverable
Return messages: None

The discoverable parameter will be 1 if the device is discoverable and 0 otherwise.

Telegram:

```
| 4 | MSG__GetDiscoverableResult | discoverable | SUM[Hi] | SUM[Lo] |
```

2E GETPORTOPENRESULT

Parameters: uint8 portIsOpen
In reply to: GetDiscoverable
Return messages: None

The portIsOpen parameter will be 1 if the port is open and 0 otherwise.

Telegram:

```
| 4 | MSG__GetPortOpenResult | portIsOpen | SUM[Hi] | SUM[Lo] |
```

30 GETVERSIONRESULT

Parameters: uint8 major, uint8 minor
In reply to: GetVersion
Return messages: None

This message contains the version number of the firmware implemented within the BlueCore™ chip.

Telegram:

```
| 5 | MSG__GetVersionResult | major | minor | SUM[Hi] | SUM[Lo] |
```

31 GETBRICKSTATUSBYTERESULT

Parameters: uint8 byte1, uint8 byte2
In reply to: GetBrickStatusbyte
Return messages: None

This message contains the status bytes from persistent storage.

Telegram:

```
| 5 | MSG__GetBrickStatusbyteResult | byte1 | byte2 | SUM[Hi] | SUM[Lo] |
```

32 SETBRICKSTATUSBYTERESULT

Parameters: uint8 success
 In reply to: SetBrickStatusbyte
 Return messages: None

The success parameter is given by the following enum, also used in ListResult:

```
enum
{
    LR_SUCCESS = 0x50,
    LR_COULD_NOT_SAVE
};
```

LR_SUCCESS: Indicates that the operation was successful.

LR_COULD_NOT_SAVE: The entry could not be written to persistent storage. The BlueCore™ chip must be reset to activate defragmentation of the flash.

Telegram:

```
| 3| MSG_SetBrickStatusbyteResult | success | SUM[Hi] | SUM[Lo] |
```

37 OPERATINGMODERESULT

Parameters: uint8 mode
 In reply to: SetOperatingMode, GetOperatingMode
 Return messages: None

This message indicates the operating mode of the brick.

Telegram:

```
| 4| MSG_OperatingModeResult | mode | SUM[Hi] | SUM[Lo] |
```

39 CONNECTIONSTATUSRESULT

Parameters: uint8 status_handle0, uint8 status_handle1, uint8 status_handle2, uint8 status_handle3
 In reply to: GetConnectionStatus
 Return messages: None

This message indicates the status of the connection. Each status byte will contain a value in the enumeration:

```
typedef enum {
    CONN_READY,
    CONN_INITIALIZED,
    CONN_CONNECTED,
    CONN_CONNECTING,
    CONN_STREAM_OPEN
} ConnState;
```

Telegram:

```
| 10| MSG_ConnectionStatusResult |3 x RESERVED | h0 | h1 | h2 | h3 | SUM[Hi] | SUM[Lo] |
```

C-CODE STANDARD FOR MESSAGE ID

This enumeration defines the message id numbering. The first message type has id 0.

ENUMERATION

```
enum MSG_TYPES
{
    MSG_BeginInquiry,
    MSG_CancelInquiry,
    MSG_Connect,
    MSG_OpenPort,
    MSG_LookupName,
    MSG_AddDevice,
    MSG_RemoveDevice,
    MSG_DumpList,
    MSG_CloseConnection,
    MSG_AcceptConnection,
    MSG_PinCode,
    MSG_OpenStream,
    MSG_StartHeart,
    MSG_Heartbeat,
    MSG_InquiryRunning,
    MSG_InquiryResult,
    MSG_InquiryStopped,
    MSG_LookupNameResult,
    MSG_LookupNameFailure,
    MSG_ConnectResult,
    MSG_ResetIndication,
    MSG_RequestPinCode,
    MSG_RequestConnection,
    MSG_ListResult,
    MSG_ListItem,
    MSG_ListDumpStopped,
    MSG_CloseConnectionResult,
    MSG_PortOpenResult,
    MSG_SetDiscoverable,
    MSG_ClosePort,
    MSG_ClosePortResult,
    MSG_PinCodeAck,
    MSG_SetDiscoverableAck,
    MSG_SetFriendlyName,
    MSG_SetFriendlyNameAck,
    MSG_GetLinkQuality,
    MSG_LinkQualityResult,
    MSG_SetFactorySettings,
    MSG_SetFactorySettingsAck,
    MSG_GetLocalAddr,
    MSG_GetLocalAddrResult,
    MSG_GetFriendlyName,
    MSG_GetDiscoverable,
    MSG_GetPortOpen,
    MSG_GetFriendlyNameResult,
    MSG_GetDiscoverableResult,
    MSG_GetPortOpenResult,
    MSG_GetVersion,
```

```
MSG__GetVersionResult,  
MSG__GetBrickStatusbyteResult,  
MSG__SetBrickStatusbyteResult,  
MSG__GetBrickStatusbyte,  
MSG__SetBrickStatusbyte,  
MSG__GetOperatingMode,  
MSG__SetOperatingMode,  
MSG__SetOperatingModeResult,  
MSG__GetConnectionStatus,  
MSG__ConnectionStatusResult,  
MSG__gotoDFUMode  
};
```