Ada User Guide for LEGO MINDSTORMS NXT

Peter J. Bradley, Juan A. de la Puente, Juan Zamorano.
Universidad Politécnica de Madrid, Madrid, Spain.
http://polaris.dit.upm.es/str

Abstract

The purpose of this guide is to introduce the robotics kit LEGO MINDSTORMS NXT to the Ada community. All the steps required to complete a working Ada application running under the LEGO MINDSTORMS NXT are covered.

Keywords:

LEGO, MINDSTORMS, Ada, Ravenscar, Real-Time, Embedded, Robotics.

1 Introduction

The LEGO MINDSTORMS NXT (from now on NXT) is a simple and flexible robotics kit that allows Ada programmers to develop applications that interact with the "outside world" by means of sensors, actuators, etc. The dynamic features associated to this interaction with the physical environment require that the actions of the control software are executed at a specified time rate. Therefore, real-time constraints must be generally met. Ada's concurrency and real-time integrated features together with the use of the Ravenscar profile [1] makes it the ideal language for the NXT.

This guide is organised as follows. The first section is this introduction. Then, the second section shows some fundamental aspects of the NXT hardware that should be kept in mind for NXT Ada development. Section three briefly introduces Ada programming for the NXT taking into consideration the Ravenscar compliant NXT runtime system and the NXT Ada drivers library. The fourth section gives an overview of the development environment with a description of the tools required to work with the NXT. As an example, the development of a prototype vehicle, is presented in section five. Finally, section six describes how the internal JTAG interface of the NXT is accessed and used to debug Ada programs.

Throughout this guide the AdaCore GNAT GPL for LEGO MINDSTORMS NXT 2011 hosted in GNU/Linux for x86 (available from `http://polaris.dit.upm.es/str/projects/mindstorms`) will be used but note that the Windows version is also available (`http://libre.adacore.com/libre/tools/mindstorms`).

2 MINDSTORMS NXT

2.1 Architecture overview

The NXT kit comes with a programmable controller, also called Intelligent Brick. This Brick (see figure 1 for its block diagram) features a 32-bit ARM main processor (AT91SAM7S256) with 64 KB of RAM and 256 KB of Flash memory that runs at 48 MHz. To assist the main processor an 8-bit AVR co-processor (ATmega48) is also included. Main processor and co-processor periodically communicate through an I^2C bus.

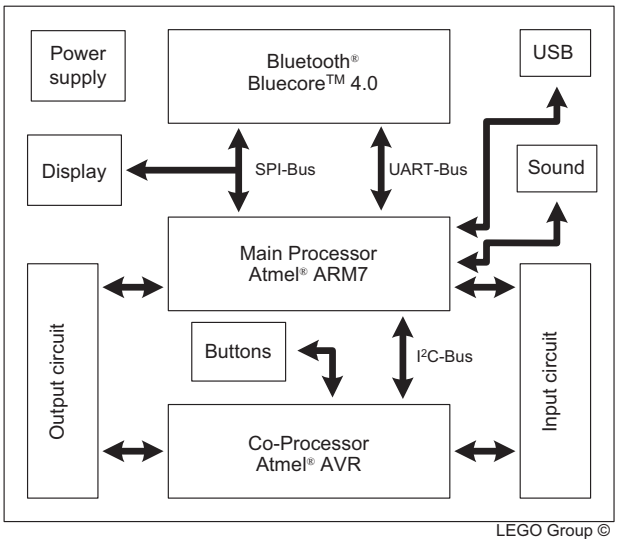

Figure 1: NXT block diagram.

It also has three output ports, which are bidirectional, to connect and control actuators such as electrical motors or linear actuators and four input ports that support both digital and analog sensors.

Communications with the Brick are possible using either USB, via a full-speed USB 2.0 port, or Bluetooth, available through a CSR BlueCore 4 chip that is connected to the ARM's USART. The USB 2.0 port is usually used to connect to a PC and Bluetooth to communicate with other NXT Bricks or any other Bluetooth-enabled devices such as smartphones, tablets, etc.

On the top of the Brick there is a 100 x 64 pixel LCD display connected to the main processor via a SPI bus (serial peripheral interface bus), and four rubber buttons, controlled by the co-processor, for interacting with the Brick.

The NXT Brick also comes with an audio amplifier,

connected to the ARM PWM (pulse-width modulation) controller, and a 16 Ω speaker with a bandwidth of 2 - 16 KHz.

For schematics and further information refer to *LEGO MINDSTORMS NXT Hardware Developer Kit* [2].

2.2 Processor and co-processor

The AVR co-processor handles the following low-level tasks for the main processor:

- **Power management**. Turns the NXT Brick off and wakes it up when the center orange button is pressed. It also monitors the battery status sending information to the ARM processor.
- **PWM generation**. Generates pulses for the three output ports at a frequency of 8 KHz with the duty cycle specified by the ARM processor.
- **A/D conversion**. Performs a 10 bit digital conversion of the analog signals at the input ports every 3 ms.
- **Button decoding**. Decodes the buttons so that the main processor is able to tell which buttons are pressed and which are not. Note that the co-processor does not carry out any button debouncing. If it is not handled at driver level the programmer should take care of it.

To handle all of the above it is necessary for main processor and co-processor to periodically exchange information. The communication between the two microcontrollers is set up as two memory allocations that, on the original LEGO firmware, are updated on both microcontrollers every 2 ms. The communication interface operates at 380 Kbit/s using the I^2C hardware interface in both microcontrollers with the ARM main processor functioning as master.

2.3 Output ports

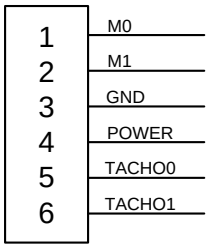

Figure 2: Output port generic schematic.

All of the three output ports work in the same manner, see figure 2. They have a ground (GND) and a 4.3 V supply output (POWER). Two output signals (M0 & M1)that come from an internal H-bridge motor driver that controls the motor standby, forward, reverse or brake modes. This motor driver is governed by the PWM pulses generated by the co-processor. It also has two input signals (TACHO0 & TACHO1) that are connected to the main processor's parallel input/output controller (PIO) using a Schmitt trigger for noise suppression. Within the Ada drivers these two last signals are used for the motor encoder. The encoder has a resolution of 360 counts per revolution. When the motor rotates the ARM processor receives an interrupt in order to update the encoder counter through the parallel I/O controller. Notice that clockwise and counterclockwise operation is detected by the counter's increments or decrements.

2.4 Input ports

Depending on the type of sensor connected to the NXT Brick the input ports behave differently. The input ports allow both digital and analog interfaces, see figure 3.

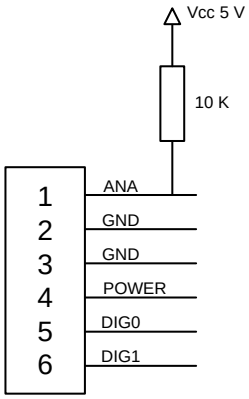

Figure 3: Input NXT generic schematic.

LEGO considers three types of sensors:

- **Active sensors**. These kind of sensors belong to the previous version of LEGO MINDSTORMS, the RCX. They require an NXT adapter cable. NXT firmware provides the same functionality available in the RCX Bricks by using an extra current source. This current source delivers power (approximately 18 mA) to the active sensors. It supplies power to the sensor through the analog pin (ANA) during 3 ms and then measures the analog value during the following 0.1 ms. The AVR sends the 10 bit digital conversion of the analog value to the main processor using the scheme presented in section 2.2.

  When using these kind of sensors (e.g. RCX light sensor, RCX rotation sensor) be sure to set the appropriate input power settings by calling `Set_Input_Power( sensor_id , RCX_9V)` from `NXT.AVR` driver package where `sensor_id` is the input port used for the active sensor.

- **Passive sensors**. These kind are analog sensors that do not need the special power/measurement timing of the active sensors. The power needs of these sensors are not covered via the analog pin (ANA) but via a specific pin (POWER). Note that the sampling of all the AVR A/D converters occurs simultaneously so active and passive sensors must be sampled at the same rate, 333 Hz.

All of the sensors packed with the LEGO MINDSTORMS NXT are passive with the exception of the ultrasonic sensor.

- **Digital sensors**. These sensors contain all the necessary logic and processing resources to work independently. Thus, they perform their function autonomously and send or receive information to/from the ARM via an I^2C channel (DIGI0 & DIGI1) running at 9600 bit/s where the ARM functions as master. These sensors are mapped as external memory areas from/to which the programmer can read or write to control the behaviour of the sensor and harvest data. For a memory arrangement that optimizes read and write access refer to *LEGO MINDSTORMS NXT Hardware Developer Kit* [2].

  The ultrasonic sensor is the only digital sensor packed in the NXT kit.

If a higher sampling rate is required by an analog input the hardware allows configuring DIGI1 as an analog input.

Port 4 can also function as a high-speed communication port. It has a RS485 IC that allows for high-speed-bi-directional multipoint communications.

2.5 Bluetooth features

The NXT Brick can be connected using Bluetooth to any other Bluetooth device that implements the Serial Port Profile (SPP), a serial cable emulation profile. The effective working Bluetooth range for the NXT Brick is approximately 10 m (Bluetooth Class II device).

The NXT Brick provides a master/slave communication scheme with four channels. Channel 0 is used when working as slave and the other three when working as master. The NXT Brick can either work as master or slave. This means that when the NXT Brick works as master it can communicate with three more devices.

The CSR BlueCore 4 firmware is implemented as a virtual machine with an integrated command interpreter. Thus, communication between the main ARM processor and the Bluetooth chip is handled by a set of defined commands and data streams that are exchanged through the USART channel. Refer to *LEGO MINDSTORMS NXT ARM7 Bluetooth Developer Kit* [3] for a full specification.

3 Ada programming for NXT

3.1 NXT run-time system

The AdaCore GNAT GPL for LEGO MINDSTORMS NXT 2011 cross-compiler toolchain relies on a Ravenscar small footprint run-time system (Ravenscar SFP). It is really a superset of the zero footprint profile. It adds the specification of a secondary stack mechanism for unconstrained objects and the Ravenscar tasking features to the zero footprint profile. This means that Ada applications for the NXT should comply with the Ravenscar profile for tasking purposes. Also, as it is targeted for use with embedded systems, it uses a sequential Ada subset where not all language features are available. For example, attributes 'Image and 'Value are not included. Moreover, there is no exception propagation. Unhandled exceptions jump to a "last chance handler" that can be reprogrammed as desired as long as the application then terminates (it must not return to the caller). Note that you must explicitly include the package NXT.Last_Chance, using a with-clause, for it to be part of your application. If you do not, a default handler is included that only displays an address for the exception on the NXT LCD screen. For a full description of the Ravenscar SFP profile refer to *GNAT User's Guide "Supplement for High-Integrity Edition Platforms"* [4].

The purpose of the Ravenscar profile is to restrict the use of many tasking facilities so that the outcome of the program is predictable. For this purpose, the profile is restricted to a fixed priority and pre-emptive scheduling. With fixed priority pre-emptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to be executed. Also, the Immediate Ceiling Priority Protocol (ICPP) is enforced by the Ravenscar profile. This means that when a task locks the resource, its priority is temporarily raised to the priority ceiling of the resource, thus no task that may lock the resource is able to get scheduled. This allows execution of a low priority task deferring execution of higher-priority tasks, thus minimizing priority inversion. More information can be found in *Annex D: Real-Time Systems* of the *Ada 2005 Reference Manual* [5].

When writing an Ada application for NXT you should bear in mind that only the Ada subset defined by the Ravenscar profile can be used for tasking. These are some of the restrictions:

- requeue statement.
- abort statements.
- Task entries.
- Dynamic priorities.
- Relative delays.
- Protected types with more than one entry.
- Protected entries with barriers other than a single boolean variable declared within the same protected type.
- Entry calls to a protected entry with a call already queued.
- select statements.
- Task termination.

For a full and detailed list refer to *Guide for the use of the Ada Ravenscar Profile in high integrity systems* [1].

3.2 NXT Ada drivers

The NXT drivers developed by AdaCore are completely coded in Ada. These drivers are based on those of the LeJOS Project. The LeJOS Project is a tiny Java virtual machine ported to the NXT Brick in 2006 (`http://lejos.sourceforge.net`).

These drivers have undergone major updates in the last two versions of GNAT GPL for MINDSTORMS (2010 & 2011) so 2010 programs might not compile with the 2011 compiler. Unfortunately, AdaCore does not supply API documentation with the drivers. It is convenient to revise the drivers' code to understand how they work. A full description of the drivers is out of the scope of this guide.

For every Ada NXT program the NXT.AVR package must always be imported even if its functions are not required. The body of this package contains a periodic task called Pump, with the highest priority, executed every 20 ms, that handles the co-processor communications (explained in subsection 2.2) using a circular buffer. By adding a with-clause to the main program and importing NXT.AVR the execution of this task within the program is guaranteed. It is also advisable to import NXT.Display and NXT.Last_Chance for exception handling.

High-level access to motors and sensors is available through a series of object oriented interfaces that provide a tagged type, a constructor function and some operations. NXT.Motors and NXT.I2C_Sensors packages provide abstract types and primitive operations. This object oriented structure eases extending the code with new drivers for third-party sensors. For AVR connected peripherals (analog sensors, motors, buttons, etc.) the low-level package NXT.AVR can also be used.

Note that these drivers provide user-transparent button debouncing through the NXT.Filtering package.

Both AVR and Bluetooth interfaces perform checksum analysis for all data exchanged with the main processor to discard inconsistent data.

When using the concurrency features available with the Ravenscar profile it must be considered that the display and AVR drivers do not implement a thread-safe environment. LCD data and the circular buffer with the outgoing messages to the AVR are defined as global variables with no access control. For concurrent access to the display the NXT.Display.Concurrent package provided can be used. For AVR concurrent access a thread-safe solution must be provided by the user to avoid race conditions when calling Power_Down, Set_Power and Set_Input_Power procedures. Notice, that because of the periodic task that handles ARM - AVR communications, every time a motor is used or a power down to the NXT is set, race condition issues are present. The 2010 GNU/Linux GNAT version provided modified drivers that addressed this issue but since the 2011 GNU/Linux version changed its interface the solution has not yet been adapted.

4 Development Environment

4.1 Tools overview

A cross-compiler toolchain is a set of tools (essentially a compiler, an assembler and a linker) that create executable code for a platform, in this case the NXT main processor (ARMv3 architecture), other than the one on which the tools run, that is, GNU/Linux x86. Cross-compiler toolchains are used to compile code for a platform upon which it is not feasible to do the compiling. AdaCore has ported the GNAT compiler toolchain to the ARM architecture by porting part of the LEON-based Open Ravenscar Real-Time Kernel (ORK+)[1] developed by a team of the Department of Telematics Engineering from the Technical University of Madrid (DIT/UPM) [6].

4.2 Compiling a program

The NXT's original firmware for the main processor is completely removed (this invalidates the warranty) and replaced by a binary image of the user's Ada application that is executed from RAM. Flash memory is not used. This means that every time a program is executed it must first by uploaded to RAM.

Instead of using the widespread ELF as executable file format the EABI format is used by the GNAT cross-toolchain. EABI has been created as a common binary interface so that object code and libraries created with one toolchain can be linked to a project created with a different one.

To generate an executable NXT file from the user's Ada application the GNAT cross-toolchain needs first to compile and then link to RAM all compiled code using `kernel_samba.ld` linker script. The code that needs to be compiled is the user's Ada code, the run-time system, the Ada NXT required drivers, nxt_main() C function (`main.c`), a low-level routine to initialise the system (`init.s`), a low-level interrupt handler routine (`irq.s`), a vector table that is remapped to RAM (`vectors.s`) by `init.s` and the elaboration code generated by the GNAT binder.

A *GNU make* script (`Makefile.inc`) is in charge of building the binary image that is uploaded. This script compiles the run-time libraries every time since precompiled library units are not used.

4.3 Uploading a program

With no firmware, when the orange button of the NXT Brick is pressed the ARM main processor executes the default Boot Program (SAM-BA Boot Assistant) located in the first two sectors of the Flash memory. The SAM-BA Boot Assistant supports serial communications through the USB Device Port.

LibNXT is a utility library for communicating with the NXT Brick from a POSIX-compliant host computer using USB. When the ARM processor is in SAM-BA mode,

[1] ORK+ is an open source real-time kernel that implements the Ravenscar profile for the GNAT compiler system on a bare LEON2 processor.

LibNXT is able upload the binary image file of the NXT executable to RAM and then execute it. For Windows host platforms the Atmel SAM-BA software is available.

5 Vehicle Prototype

This section describes the steps to have a working NXT vehicle prototype using Ada [2].

5.1 Functionality

The vehicle has a front castor wheel, free to turn, and two back wheels, each driven by an independent motor. To control the vehicle a hardwired joystick made with a touch sensor to start/stop drive and a motors encoder to control operation is used. Depending on the angle of the joystick encoder, different speed commands are sent to the vehicle motors, thus controlling vehicle motion, see figure 4.

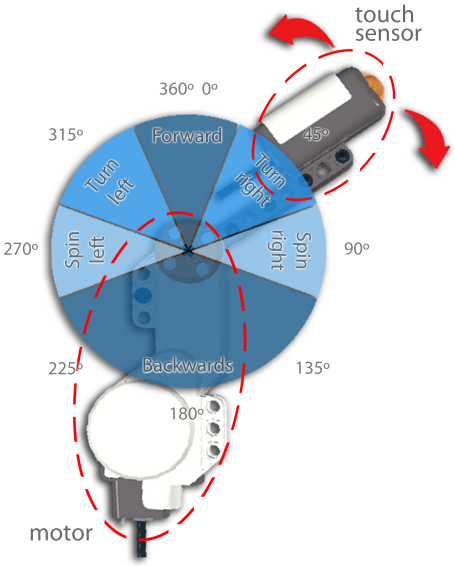

Figure 4: Vehicle's joystick.

5.2 Design and assembly

Next step is to assemble a prototype that achieves the above mentioned functionality. The best way to do so, especially if dealing with a complex design, is to model it using a CAD tool. LEGO offers a freeware software to develop NXT models, *LEGO Digital Designer*[3] [8]. The vehicle prototype for this guide was modelled with *LDD*, see figure 5.

Although it can initially be somehow frustrating, using these kind of tools decreases assembly time by allowing the development of several prototypes. It lists the bricks used and generates a step-by-step building guide for the

[2]Example modified from Bradley et al. [7].

[3]This software is available for Windows and Mac OS. *LDraw* and *LeoCAD* are other CAD software alternatives.

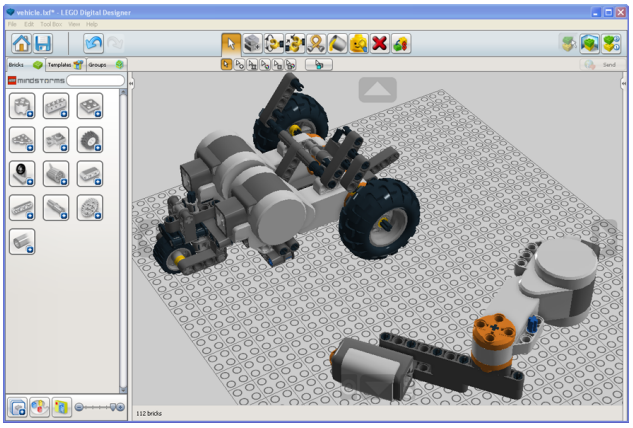

Figure 5: *LDD* model for the vehicle prototype.

model. Figure 6 shows the vehicle prototype fully assembled using the generated building guide from the *LDD* model.

5.3 Software Architecture

The following are the tasks involved in the software architecture of the vehicle prototype:

- **Control_Task**: Periodic task that executes every 20 ms. It checks if the touch sensor is pressed (a 20 ms period to detect a man operated touch sensor is considered sufficient). In case it is, it gets the value of the joystick motor encoder to determine the speed commands that are then stored in the circular buffer. These speed commands are later sent to the AVR by the Pump task. The task takes the position of the joystick motor at the beginning of its execution as reference point. It also checks if the orange button is pressed to switch off the NXT Brick.

- **Display_Task**: Periodic task that executes every 500 ms with a lower priority than Control_Task. This task shows the joystick's position, the execution time and the battery's mV on the LCD screen.

- **Background** procedure: This is just a background procedure that executes every time the ARM processor is free.

Although the application performs as expected, the circular buffer global variable used for the ARM - AVR communications is not thread-safe and a race condition exist. This race condition may or may not happen, and if it happens, it does not necessarily mean the performance of the vehicle will be affected. Nevertheless, it is not a good programming practice to rely on non controlled access to a global variable.

There is a thread-safe vehicle version using the 2010 modified AVR drivers that can be downloaded from `http://polaris.dit.upm.es/str/projects/mindstorms/2010`.

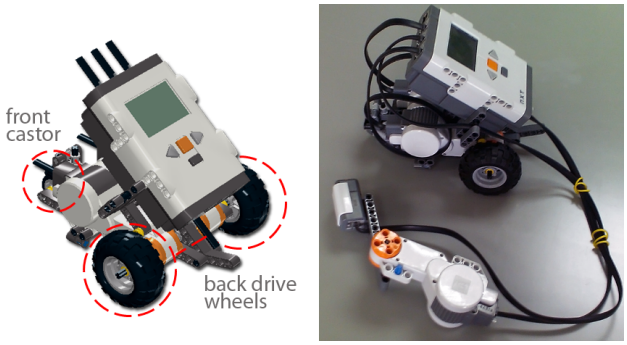

Figure 6: Vehicle prototype fully assembled.

5.4 Software Implementation

Three compilation units are used for the Ada vehicle application: The main procedure (`vehicle.adb`) that calls Background procedure, a package declaration (`tasks.ads`) and its body (`tasks.adb`). The Tasks package includes the two control tasks (Control_Task and Display_Task), the empty procedure (Background) and some auxiliary functions. Listing 1 shows a fragment of `tasks.adb` containing the declaration of the two tasks and the background procedure. When declaring a task, besides using pragma Priority to establish the static priority, pragma Storage_Size is used. Pragma Storage_Size specifies the amount of memory to be allocated for the task stack. Notice that this pragma is required because of the small amount of memory available, 64KB of RAM memory. The stack size must not be exceeded. If it does, a `Storage_Error` will be raised. If this Storage_Size pragma is not used, a compiling error about RAM overflowing could be prompted.

It must be remembered that the clock resolution defined by the run-time system is of 1 ms.

Listing 1: Specification of tasks.

```
-----------------------
-- Background task --
-----------------------
procedure Background is
begin
   loop
      null;
   end loop;
end Background;

--------------
-- Tasks --
--------------
task Control_Task is
   pragma Priority
      (System.Priority'First + 2);
   pragma Storage_Size (4096);
end Control_Task;

task Display_Task is
   pragma Priority
      (System.Priority'First + 1);
   pragma Storage_Size (4096);
end Display_Task;
```

6 Debugging Solution

A remote debugger is an extremely useful tool for an embedded system developer. It can drastically decrease development time. There is no open source Ada/C debugging solution for the NXT. In this section we describe a way to remotely debug Ada/C programs for the NXT using the *GNU debugger* (*GDB*) and the ARM EmbeddedICE (In-circuit Emulator) technology. The ARM EmbeddedICE is a JTAG[4]-based debugging channel available on the ARM main processor. Debugging the NXT from a host computer through the available JTAG interface is therefore possible. RAM and Flash programming is also available using this method.

This solution has been adapted to work on GNU/Linux x86 hosts but it could be easily ported to a Windows platform.

6.1 Overview

The JTAG-based debugging channel provides real-time access to memory addresses and data dependent watchpoints, step-by-step execution, full control of the central processing unit and other related debugging features. It requires no use of memory unlike debugging monitor solutions.

The ARM featured EmbeddedICE-compatible macrocell from the NXT includes an ARM7 core, a small amount of control logic, a TAP[5] (Test Access Port) controller for the JTAG interface and an EmbeddedICE macrocell, see figure 7. This EmbeddedICE macrocell has two real-time watchpoint registers as well as control and status registers. Each watchpoint register can be configured as a watchpoint for data access or a breakpoint for instruction fetch. If a match occurs between the values programmed into the watchpoint registers and the values of the address bus and data busses or some specific control signal, the ARM7 core ceases to read instructions from the data bus and isolates itself from the memory system entering debug state. Access to the processor's state and memory system is then possible through the JTAG interface using the TAP controller.

GDB provides the remote serial protocol (RSP) for remote debugging. RSP is a *GDB* protocol used to send debugging commands through a serial or ethernet link. Using a localhost TCP connection on the developer's host computer an *OpenOCD* daemon processes the commands issued by *GDB*.

OpenOCD (*The Open On-Chip Debugger*) is an open source tool initially developed by Dominic Rath as part

[4] JTAG, as defined by the IEEE Std.-1149.1 standard, is an integrated method for testing interconnects on printed circuit boards (PCBs) that are implemented at the integrated circuit (IC) level.

[5] a TAP is the core of the JTAG standard. It is a finite state machine that controls JTAG operations.

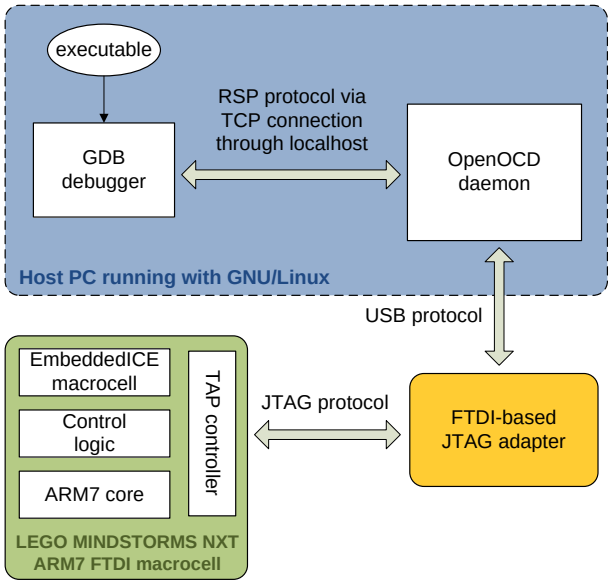

Figure 7: ICE debugging solution for NXT.

of his diploma thesis at the University of Applied Sciences Augsburg [9]. This software provides debugging, in-system programming and boundary-scan testing for embedded targets such as the NXT. *OpenOCD* essentially allows *GDB* to talk through a JTAG adapter to the EmbeddedICE-compatible macrocell on the NXT.

A JTAG adapter is a piece of hardware that connects the host computer with the JTAG interface of the remote target. The JTAG adapter is in charge of adapting the serial electric signalling received from *OpenOCD*, using, in this case, an FTDI[6] chip, to send the JTAG operations to the TAP controller. Figure 7 shows the debugging scheme.

6.2 Modifying the NXT Brick

To connect *GDB* in the host computer with the JTAG interface of the NXT a JTAG adapter is required. Also, The NXT Brick PCB has the provision for mounting a JTAG connector but this has not been mounted to save cost. The NXT Brick must be opened in order to access the JTAG interface. Note that by performing this modification warranty will be lost.

6.2.1 FTDI-based JTAG adapter

An FTDI-based JTAG adapter that is both compatible with *OpenOCD* and the main processor of the NXT (AT91SAM7S256) is required. For this guide the ARM-USB-TINY-H adapter by Olimex (`http://www.olimex.com`) was used. *Open On-Chip Debugger: OpenOCD User's Guide* [10] offers other vendor options.

6.2.2 Tools and materials

- Small Philips head screwdriver.
- Fine wire cutter.
- Wire stripper.
- Soldering iron with a fine tip and solder.
- De-soldering pump.
- Magnifying glass.
- Drill with 4 mm diameter bit.
- Digital multimeter.
- 20 pin 2.54 pitch ribbon cable male connector (ARM JTAG connector).
- 30 SWG single core polyurethane insulated cable.

6.2.3 NXT Brick disassembly

Take out the battery pack or batteries to gain access to the four Philips head screws. Unscrew them and remove the front cover. Remove the silicon rubber buttons' assembly.

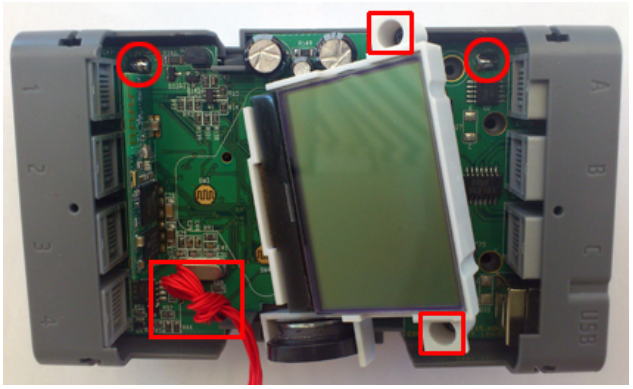

Figure 8: NXT without front cover.

Find the two screws, that hold down the LCD display, located on each side of it (the two small squares of figure 8). Loosen these screws and carefully lift the LCD display to get access to the battery terminals that are soldered to the main PCB. Note that the LCD display cannot be removed from the PCB board on some models.

Once the two display screws have been removed the two battery terminals must be de-soldered (the two small circles of figure 8). To do this, remove the solder with the soldering iron and the de-soldering pump. When the terminals are free of solder separate the PCB from the battery case and remove the input and output connector supports. Note that there is a small silicone rubber push-button between the battery case and the PCB.

6.2.4 JTAG connection

Since there was no short delivery 1.27 pitch connectors at the time, The hard-wired option presented below was used.

Cut 8 equal lengths, at least 100 mm, of the single core cable and strip 3 mm of insulation on one side. Identify both ends with an indelible marker. The JTAG interface (J17 on the PCB) is located below the loudspeaker beside the quartz crystal (the big square of figure 8). Pin 1 has

[6] Hardware solution to interface with USB peripherals.

a square pad and the remaining pins have round pads. Insert one by one the stripped ends of the 8 cables in pins 1 - 8 and solder them to the board. This type of wire is used because, unlike PVC insulation, it supports high temperatures (155 °C) and makes soldering easy. With the magnifying glass inspect each solder for bridges between pins. See left picture from figure 9 for the final result.

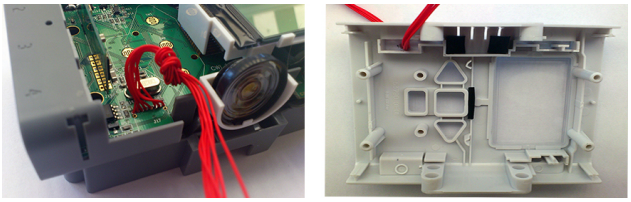

Figure 9: Soldered JTAG interface & front cover drilled hole.

Drill a 4 mm hole on the front cover of the NXT Brick directly above the J17 connection as shown in the right picture of figure 9. As a strain relief bundle the eight wires together and tie a knot with them 20 mm from the PCB. Take them through the hole of the front cover and cut them to length for the connection to the ribbon cable connector according to figure 10. As the wire used has a smaller gauge than the connector it is advisable to solder the connections after inserting them. Therefore, strip the wires, insert them and solder them. Try to use as little solder as possible to allow inserting the header in the connector.

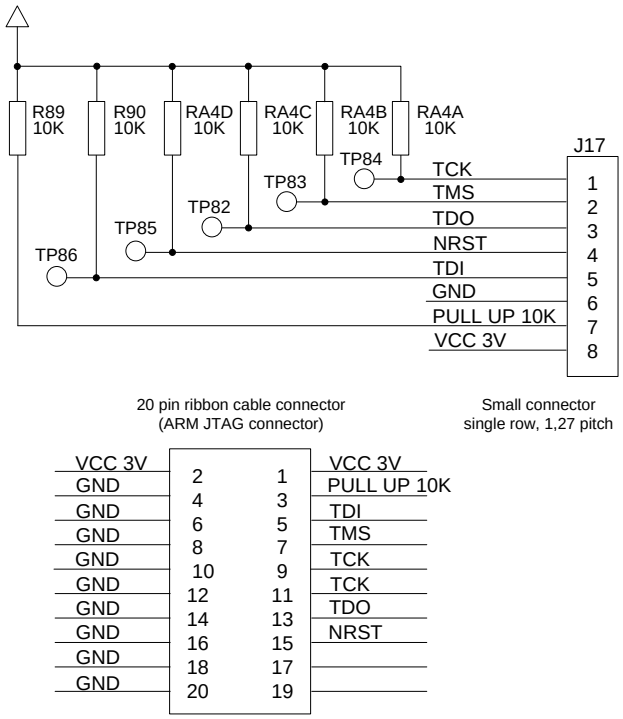

Figure 10: NXT JTAG hardware schematic.

Note that the GND connection is only connected to pin 6 because the JTAG adapter used has all the GND pins internally connected.

6.2.5 Testing the connections

Locate on the NXT Brick PCB resistor R89, check for continuity with the multimeter in Ω between the top of R89 and pin 2 of the ribbon cable connector (VCC 3V). Check that the other end of R89 is connected to pin 3 of the ribbon cable connector (PULL UP 10K). Next, check the GND connection between pin 6 of the ribbon cable connector and the negative battery terminal PCB connection (J5). Locate test points TP82 - TP86 on the solder side of the PCB and check with the multimeter for continuity between them and the corresponding pins of the ribbon cable connector. Also check for short-circuits between connections.

Finally, once the connections have been checked, reassemble the NXT Brick.

For a more graphical guide on the modification of the NXT Brick refer to *Installing the JTAG connector* [11].

6.3 A debugging session

In order to remotely debug programs under GNU/Linux `libusb-0.1`, `libusb-dev`, `libftdi1` and `libftdi-dev` are required. The FTDI module with the JTAG adapter information will probably have to be loaded also, once it is plugged in:

```
$ sudo modprobe -v ftdi_sio
  vendor=0x... product=0x...
```

When the NXT has no firmware the orange button must be pressed. Then, when a clicking sound is heard, the JTAG adapter must be plugged to the NXT. Next, `arm-eabi-openocd` must be run with a specific configuration script:

```
$ arm-eabi-openocd -f debug-ram.cfg
```

This configuration file is a setup for *OpenOCD* that establishes communications with the NXT EmbeddedICE macrocell. The script usually contains the daemon configuration that establishes communications with *GDB*, the configuration for the adapter, the board, the target and some init commands. JTAG adapter vendors usually provide this *OpenOCD* script and in case they do not, the `share/openocd/scripts` folder from the install directory contains generic configuration files. For further information refer to *Open On-Chip Debugger: OpenOCD User's Guide* [10].

When *OpenOCD* handshakes with the NXT successfully *GDB* must be run with the executable as parameter, not with the binary image:

```
$ arm-eabi-gdb executable_name
```

Any breakpoints should be added at this point. After, the `gdbinit` script, see listing 2, must be run:

```
gdb> source gdbinit
```

Cross-debugging is now possible.

Listing 2: *GDB* init script

```
# Init command
target remote localhost:3333

# OpenOCD command to halt the processor
# and wait
monitor soft_reset_halt

# OpenOCD command to select the core state
monitor arm core_state arm

# set flash wait state (AT91C_MC_FMR)
monitor mww 0xffffff60 0x00320100

# watchdog disable (AT91C_WDTC_WDMR)
monitor mww 0xfffffd44 0xa0008000

# enable main oscillator (AT91C_PMC_MOR)
monitor mww 0xfffffc20 0xa0000601

# wait 100 ms
monitor sleep 100

# set PLL register (AT91C_PMC_PLLR)
monitor mww 0xfffffc2c 0x00480a0e

# wait 200 ms
monitor sleep 200

# set master clock to PLL (AT91C_PMC_MCKR)
monitor mww 0xfffffc30 0x7

# wait 100 ms
monitor sleep 100

# enable user reset AT91C_RSTC_RMR
monitor mww 0xfffffd08 0xa5000401

# force a peripheral RESET AT91C_RSTC_RCR
monitor mww 0xfffffd00 0xa5000004

# toggle the remap register to place RAM
# at 0x00000000
monitor mww 0xffffff00 0x01

# set the PC to 0x00000000
monitor reg pc 0x00000000

# enable use of software breakpoints
monitor gdb_breakpoint_override soft
monitor arm7_9 dbgrq enable

# upload the application
load

# resume execution from reset vector
continue
```

This *GDB* script basically sets the ARM processor to execute the application and set some debugging features. The script used is a modified version of that presented in *Using Open Source Tools for AT91SAM7S Cross Development* by James P. Lynch [12].

7 Conclusions

This guide shows the basics for Ada development using LEGO MINDSTORMS NXT. The Ravenscar profile run-time system offers concurrency Ada programming while making possible a schedulability analysis of the system. Ada development on the NXT presents a whole perspective of an embedded system with real-time constraints.

At a reasonable price the NXT kit offers all kinds of sensors and mechanisms to work with, even custom-made sensors can be developed.

Development and sharing of Ada projects with the NXT would be of great interest, in the same way as other complex models like Rubik's cube solvers, Segway robots, scanners, etc. have been developed using other programming languages and shared.

The Ada community is encouraged to use this development platform that, besides the fun, can be an interesting teaching asset.

It is important to note that all of the tools used, except *LDD*, are open source and therefore there is no dependance on software vendors. All of the source code is available and can by modified.

Acknowledgements

The authors would like to thank AdaCore for their work adapting the Ravenscar run-time system and developing the Ada drivers for the LEGO MINDSTORMS NXT platform.

References

[1] Burns A, Dobbing B, Vardanega T. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Ada Lett. 2004 June;XXIV:1–74. Available from: `http://doi.acm.org/10.1145/997119.997120`.

[2] LEGO. LEGO MINDSTORMS NXT Hardware Developer Kit;. Version 1.00. Available from: `http://mindstorms.lego.com`.

[3] LEGO. LEGO MINDSTORMS NXT ARM7 Bluetooth Developer Kit;. Version 1.00. Available from: `http://mindstorms.lego.com`.

[4] AdaCore. GNAT Pro User's Guide, Supplement for High-Integrity Edition Platforms; 2011. The GNAT Ada Compiler. GNAT GPL Edition, Version 2011 Document revision level 175263.

[5] Std. 8652:1995/Amd 1:2007 — Ada 2005 Reference Manual. Language and Standard Libraries; 2007. Published by Springer-Verlag, ISBN 978-3-540-69335-2.

[6] de la Puente JA, Ruiz JF, Zamorano J. An Open Ravenscar Real-Time Kernel for GNAT. In: Proceedings of the 5th Ada-Europe International Conference on Reliable Software Technologies. Ada-Europe '00. London, UK: Springer-Verlag; 2000. p. 5–15. Available from: http://portal.acm.org/citation.cfm?id=646579.697613.

[7] Bradley PJ, de la Puente JA, Zamorano J. Real-time system development in Ada using LEGO MINDSTORMS NXT. In: Proceedings of the ACM SIGAda annual international conference on SIGAda. SIGAda '10. New York, NY, USA: ACM; 2010. p. 37–40. Available from: http://doi.acm.org/10.1145/1879063.1879077.

[8] LEGO. LEGO Digital Designer 4.1 User Manual; 2011. Available from: http://ldd.lego.com.

[9] Rath D. Open On-Chip Debugger. Design and Implementation of an On-Chip Debug Solution for Embedded Target Systems based on the ARM7 and ARM9 Family. University of Applied Sciences Augsburg; 2005.

[10] Brownell D. Open On-Chip Debugger: OpenOCD User's Guide; 2011. Available from: http://openocd.berlios.de.

[11] IAR. Installing the JTAG connector. IAR KickStart for LEGO MINDSTORMS NXT; 2009. Available from: http://www.iar.com/website1/1.0.1.0/1483/1.

[12] Lynch JP. Using Open Source Tools for AT91SAM7S Cross Development. Grand Island, New York, USA; 2007. Revision C.