**Hands-on Lab**

**Lego Sensing – Analog-to-Digital Basics**

The Lego NXT contains a 10-bit analog-to-digital (ADC) convertor. This lab will develop sensors. This is important because sensors are a critical component for any robot.

**Concept 1 – NXT ADC:** Homemade touch sensor

Ports 1 to 4 on an NXT Brick are connected to 10-bit ADC. First, the port’s connector uses a 6-line ribbon cable. The cable can used to connect sensors (i.e. input) or actuators (i.e. output). Since we are interested in the ADC, each wire’s role is defined in **Figure 1A**.



**Figure 1A:** An NXT cable has six wires with roles assigned above

Ports 1 to 4 each are connected to a 10 kilo-ohm resistor and 5 Volt supply which go into a 10-bit ADC (see Figure 1B).

**Figure 1B:** When Pins 1 and 2 are open, then, the ADC will read +5V (left). If the switch closes (right), then Pins 1 and 2 are shorted; the path of least resistance forces the ADC to read 0V.





**Step 1:** Create a circuit that reflects Figure 1B (right).

A solderless breadboard is perhaps the easiest method to construct the circuit. The switch can be simply made with some wire.

**Step 2:** Write the following NxC program and execute

// FILE: touch1\_0.nxc

// DATE: 08/18/16 01:17

// AUTH: P.Oh

// DESC: Homemade touch sensor; sensor port 1

// VERS: 1.0

task main() {

 int touchSensorValue;

 string strTouchSensorValue; // store integer value of touch sensor as string

 string strMessageAndValue; // To display touch sensor value

 SetSensorTouch(IN\_1); // homemade touch sensor on Brick Port 1

 do {

 touchSensorValue = Sensor(IN\_1);

 strTouchSensorValue = NumToStr(touchSensorValue);

 strMessageAndValue = StrCat("Touch reads:", strTouchSensorValue);

 TextOut(10, LCD\_LINE4, strMessageAndValue);

 Wait(100);

 } while(true); // endless do-while loop

 StopAllTasks();

} // end main

**Code Explanation:** The NxC statement SetSensorTouch(IN\_1)prepares Port 1 for inputs – by setting Pins 1 (White) and 2 (Black) for reading. The Sensor(IN\_1) statement then reads Port 1 and returns a value. This value is stored in the variable touchSensorValue. If the value is 1, in means Pins 1 and 2 are shorted (i.e. switch is closed). If the value is 0, then the two pins are not connected (i.e. switch is open).

**Exercise 1:** In NxC create programs for the following:

* 1. Brick displays “”Touch sensor is: “, with “ON = 1” when the switch is closed and “OFF = 0” when the switch is open. If the switch is closed, then play a tone. Use statements like TextOut and PlayTone

**Concept 2 – Voltage Divider:** Homemade ohmmeter

Expanding upon Figure 1B, one can create insert a resistor between Pins 1 and 2. This is shown in **Figure 2A**.



**Figure 2A:** Insert a random resistor $R$in between Pins 1 and 2.

Recall, **Figure 2A** is a voltage divider where we have the voltage across the resistor $R$as:

$$V\_{R}=\frac{R}{10000 kΩ+R}V\_{N}$$

**Step 1:** Build the circuit given in Figure 2A.

**Step 2:** Write and execute the following NxC program

// FILE: ohm1\_0.nxc

// DATE: 08/18/16 02:07

// AUTH: P.Oh

// DESC: Homemade ohm sensor; sensor port 1

// Uses Brick's Port 1's WHITE (AN) and BLACK (GND) lines

// Display value of unknown resistor connected between WHITE and BLACK lines

// Treats WHITE and BLACK lines as input into Brick's internal 10-bit ADC

// VERS: 1.0 - simple program

task main() {

 int touchSensorRawValue; // a number between 0 and 1023 (10-bit ADC)

 float ohmValue;

 SetSensorTouch(IN\_1); // homemade touch sensor on Brick Port 1

 do {

 TextOut(0, LCD\_LINE1, "Raw value:");

 touchSensorRawValue = SensorRaw(IN\_1); // read raw value at port

 TextOut(0, LCD\_LINE2, FormatNum("%d", touchSensorRawValue));

 ohmValue = ((10000)\*touchSensorRawValue) / (1023-touchSensorRawValue);

 TextOut(0, LCD\_LINE3, "Ohm value is:");

 TextOut(0, LCD\_LINE4, FormatNum("%3.3f", ohmValue));

 Wait(100);

 ClearScreen();

 } while(true); // endless do-while loop

 StopAllTasks();

} // end main

**Code Explanation:** To read the actual ADC value (called $raw$), one uses the NxC statement touchSensorRawValue = SensorRaw(IN\_1). Recall that we have a 10-bit ADC, so the raw value will range from 0 to $2^{10}-1=1023.$ Thus, we can calculate the unknown resistor that lies between Pins 1 and 2 with the formula

$$R=\frac{10000}{1023-raw}raw$$

So, this homemade ohmmeter can detect resistances between $≈9Ω$ and 10,220,000$Ω$.

**Exercise 2:**

2-1: Derive the equation above and calculate the min and max resistances that can detected

2-2: Replace a fixed resistor with a potentiometer and show with a real ohmmeter, that your NxC program works

**Concept 3 – ADC Voltages:** Build a voltmeter

Blah: Maybe make this an exercise.

The for statement is part of the ANSI-C standard and is used to repeatedly execute a block of code. Modify the program in Concept 2 with the following for statement. Execution reveals that the variable x increments from 1 to 10, displaying the square and square root.

for (x = 1; x <=10; x++) {

 xSquared = x\*x;

 xSquareRoot = sqrt(x);

 // TextOut (xPosition, yPosition, string) put string on LCD's x,y position

 // NB: x = y = 0 is lower left corner of LCD; +x goes rights, +y goes up

 // FormatNum is a string with sprintf syntax

 TextOut (10, LCD\_LINE4, FormatNum("x = %d" , x));

 TextOut (10, LCD\_LINE5, FormatNum("xSquared = %d" , xSquared));

 TextOut (10, LCD\_LINE6, FormatNum("sqrt(x) = %3.3f" , xSquareRoot));

 Wait (SEC\_2);

 }

**Code Explanation:** The variable x begins with 1 and executes the statements between its parentheses. These statements compute the square and square root and display their values using the TextOut statement at various rows of the LCD. The program waits 2 seconds, checks if x less than or equal to 10. If it is x increments and again goes through computation and display. If x is greater than 10, then the loop is bypassed – which ultimately leads the termination of main.

**Concept 2C – do-while Statement:** Repeat execution of code

The Do-While statement is also a part of the ANSI-C standard for looping. Do-While is often used to execute code until some condition is met. Write the following program, save as displaySquareAndSquareRoot1\_1.nxc, and execute.

**Code Example:** displaySquareAndSquareRoot1\_1.nxc

// File: displaySquareAndSquareRoot1\_1.nxc

// Date: 08/31/12 13:04

// Desc: Display number, its square and square root

// Vers: 1.0 - works!

// 1.1: Push orange button to begin - works!

task main ()

{

 int x; // integers from 1 to 10

 int xSquared; // square of x

 float xSquareRoot; // square root of x

 bool buttonPushed;

 TextOut (0, LCD\_LINE1, "Push Orange" );

 TextOut (0, LCD\_LINE2, "button to start" );

 do {

 // nothing if Orange center button is not pushed

 buttonPushed = ButtonPressed(BTNCENTER, FALSE);

 } while(!buttonPushed);

 // exit loop when BTNCENTER = true i.e. pushed

 for (x = 1; x <=10; x++) {

 xSquared = x\*x;

 xSquareRoot = sqrt(x);

 // TextOut (xPosition, yPosition, string) put string on LCD's x,y position

 // NB: x = y = 0 is lower left corner of LCD; +x goes rights, +y goes up

 // FormatNum is a string with sprintf syntax

 TextOut (10, LCD\_LINE4, FormatNum("x = %d" , x));

 TextOut (10, LCD\_LINE5, FormatNum("xSquared = %d" , xSquared));

 TextOut (10, LCD\_LINE6, FormatNum("sqrt(x) = %3.3f" , xSquareRoot));

 Wait (SEC\_2);

 }

} // end main

**Code Explanation:** The do-while statement polls the ButtonPressed statement. In the NxC manual, ButtonPressed is used to check if a specified button is pressed. Int his case, the defined constant BTNCENTER refers the Brick’s orange button. When pressed, the Boolean variable buttonPushed becomes TRUE. The exclamation mark (!) in ANSI-C refers to negation. In other words, the do-while keeps polling ButtonPressed while buttonPushed remains not-TRUE (i.e. FALSE).

**Concept 2D – if Statement:** Conditional statement

The if statement is the ANSI-C standard to conditionally execute a statement. To appreciate this, write the following program, save as displaySquareAndSquareRoot1\_1.nxc and execute.

**Code Example:** displaySquareAndSquareRoot1\_1.nxc

// File: displaySquareAndSquareRoot1\_2.nxc

// Date: 09/04/12 11:14

// Desc: Display number, its square and square root

// Vers: 1.0 - works!

// 1.1: Push orange button to begin - works!

// 1.2: Arrows to increment, grey button to quit - works!

task main ()

{

 int x; // integers from 1 to 10

 int xSquared; // square of x

 float xSquareRoot; // square root of x

 bool orangeButtonPushed;

 bool greyButtonPushed;

 bool leftArrowButtonPushed;

 bool rightArrowButtonPushed;

 TextOut (0, LCD\_LINE1, "Push Orange" );

 TextOut (0, LCD\_LINE2, "button to start" );

 do {

 // nothing if Orange center button is not pushed

 orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);

 } while(!orangeButtonPushed);

 // exit loop when BTNCENTER = true i.e. pushed

 ClearScreen(); //clear screen

 TextOut (0, LCD\_LINE1, "Arrows increment" );

 TextOut (0, LCD\_LINE2, "Grey quits" );

 x = 1; // start with x = 1

 do {

 // Stay in this loop until grey button pushed

 // Check which button pressed

 leftArrowButtonPushed = ButtonPressed(BTNLEFT, FALSE);

 rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);

 greyButtonPushed = ButtonPressed(BTNEXIT, FALSE);

 if(leftArrowButtonPushed) x = x - 1;

 if(rightArrowButtonPushed) x = x + 1;

 if(x < 0) x = 0; // don't go below 0

 xSquared = x\*x;

 xSquareRoot = sqrt(x);

 // TextOut (xPosition, yPosition, string) put string on LCD's x,y position

 // NB: x = y = 0 is lower left corner of LCD; +x goes rights, +y goes up

 // FormatNum is a string with sprintf syntax

 TextOut (10, LCD\_LINE4, FormatNum("x = %3d" , x));

 TextOut (10, LCD\_LINE5, FormatNum("xSquared = %3d" , xSquared));

 TextOut (10, LCD\_LINE6, FormatNum("sqrt(x) = %3.3f" , xSquareRoot));

 Wait (500); // wait 500 msec

 } while(!greyButtonPushed);

} // end main

**Code Explanation:** Like displaySquareAndSquareRoot1\_1.nxc given in Concept 2C, this version (1.2) adds Boolean variables to the Brick’s 3 other buttons (left and right arrows and the grey center button). The if statement is used to increment or decrement the value of the variable x, and consequently compute and display the x’s square and square root. The do-while statement continues to monitor the status of the grey center button, and if pushed, exits the program.

**Concept 3 – Motors:** An NxC program to command NXT motors to move

**Step 1:** Open a new file and save as “helloMotor.nxc”. Type the following and save

**Code Example:** helloMotor.nxc – Rotate Motors A and C then Counter-rotate them

// FILE: helloMotor1\_0.nxc

// AUTH: P.Oh

// DATE: 03/16/11

// DESC: Motors connected to Ports A and C. Command to rotate,

// and counter-rotate fixed amount

task main() {

OnFwd(OUT\_AC, 75);

 // NXC Guide P. 294 (6.36.2.27): OnFwd(byte outputs, char pwr)

 // outputs is OUT\_X where X = [A, B, C, AB, AC, BC, ABC]

 // pwr is from [0, 100]

 Wait(5000); // continues for specified milliseconds

 OnRev(OUT\_AC, 25);

 // NXC Guide P. 300 (6.36.2.37)

 Wait(2000);

 Off(OUT\_AC); // stop and end program gracefully

 StopAllTasks();

}

**Step 2:** Attached 2 NXT motors (to Ports A and C) on the Brick

**Step 3:** Save All, Compile, then Download and Run

Code Explanation: The NxC manual describes OnFwd and OnRev statements. These are specific to NxC and not part of the ANSI-C standard. Often, specific hardware (like NXT motors and sensors) dictate using non-ANSI standard statements. The OnFwd statement uses the defined constant OUT\_AC to reference Brick ports A and C and commands motors connected to these ports to run at 75% of maximum power. Similarly, the OnRev statement commands the motors to rotate in the opposite direction at 25% of maximum power. Lastly, Off and StopAllTasks are additional non-ANSI statements, to stop the motors and exit the program gracefully.

**Exercise 3:** In NxC create programs for the following:

* 1. The repeat statement (page 25 Section 3.3.3.7) of the NXC\_Guide.pdf document, describes looping. Add a repeat(3) statement below the task main() and encase the code with a pair of French braces. This should make your helloMotor code operate 3 times.
	2. Look up the RotateMotor statement (page 308 Section 6.36.2.255). Set Motor A to run at a power level of 75 and rotate to -180 degrees.

**Programming Exercises**

* 1. Write a program that converts your age (in years) and displays your age in the number of days. Don’t worry about fractional years and leap years. For example, if you are 10 years old, then the Brick should compute (365 days per year) and display 3650 days old. Hint: use int or long for variables
	2. Write a program that increments a foot from 1 feet to 10 feet and displays the equivalent in meters (0.3048 meters per foot). NB: only display the 2 digits after the decimal. For example, for 1 foot, display (1 ft = 0.30 meters)
	3. Define 3 string variables called firstName, middleInitial, and lastName that are assigned with your own name. For example firstName = “Paul”, middleInitial = “Y” and lastName = “Oh”. Have the Brick display:

Paul

Y

Oh

Paul Oh

Paul Y. Oh

Look up and use the strcat function to display the 4th and 5th lines above

* 1. Look up strlen and write a program that displays the number of characters for each line in program 1-3. For example, output should look like

Paul: 4

Yu: 2

Oh: 2

Paul Oh: 7

Paul Y. Oh: 10

NB: White spaces count as characters in ANSI-C

* 1. Write a program that uses the variable days to hold a value from 0 to 31. Use a for loop to display days and its equivalent in weeks and days. For example, 8 days would display 1 week and 1 day.
	2. Write a program that computes and displays all the prime numbers smaller than or equal to 55. For example the display should read: 1, 5, 11, 55.