**Hands-on Lab**

**Lego Data Logging and Actuation**

We’ve explored the Lego NXT’s input port and ADC to measure resistances and voltages. This lab will turn the NXT into a data logger. Also, the lab will explore using the output port as a voltage source.

**Concept 1 – NXT File Saving Review:**

As a quick review, the NXT’s ability to save data will be explored.

# The program displaySquareAndSquareRoot1\_0.nxc displayed an integer, its square and square root on the Brick’s LCD. This program used the for-loop to iterate the integer from 1 to 10. Building on this example, a program is written to save the values to a file. The file will then be imported into an Excel worksheet. Once one has a worksheet, the data can be manipulated and/or plotted.

**Step 1:** Click File – Open and load displaySquareAndSquareRoot1\_0.nxc. Click File – Save As with the name “displaySquareAndSquareRoot2\_0.nxc”.

**Step 2:** Define global variables that serve for file handling. Add the following code to above your task main routine.

// File: displaySquareAndSquareRoot2\_0.nxc

// Date: 10/01/12 15:43

// Desc: Display number, its square and square root save to file

// Vers: 2.0

// Refs: displaySquareAndSquareRoot1\_0.nxc

// Global variables (for file writing)

unsigned int result; // flag returned when handling files

byte fileHandle; // handle to the data file

short bytesWritten; // number of bytes written to the file

string fileHeader; // column header for data in the file

int fileNumber, filePart; // integers to split up data file names

string fileName; // name of the file

string strFileNumber; // file number e.g myDataFile 1, 2, 3

string strFilePart; // file part e.g. myDataFile1-1, 1-2, 1-3

string text; // string to be written to file i.e. data values

task main ()

**Step 3:** Compose a function to initiate a file. Add the following code above task main:

string strFilePart; // file part e.g. myDataFile1-1, 1-2, 1-3

string text; // string to be written to file i.e. data values

// Create and initialize a file

void InitWriteToFile() {

 fileNumber = 0; // set first data file to be zero

 filePart = 0; // set first part of first data file to zero

 fileName = "squareData.csv" ; // name of data file

 result=CreateFile(fileName, 1024, fileHandle);

 // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535

 // returns file handle (unsigned int)

 // check if the file already exists

 while (result==LDR\_FILEEXISTS) // LDR\_FILEEXISTS returns if file pre-exists

 {

 CloseFile(fileHandle);

 fileNumber = fileNumber + 1; // create new file if already exists

 fileName=NumToStr(fileNumber);

 fileName=StrCat("squareData" , fileName, ".csv");

 result=CreateFile(fileName, 1024, fileHandle);

 } // end while

 // play a tone every time a file is created

 PlayTone(TONE\_B7, 5);

 fileHeader = "x, x^2, sqrt(x)" ; // header for myData file

 WriteLnString(fileHandle, fileHeader, bytesWritten);

 // NXT Guide Section 6.59.2.43 pg. 554

 // Write string and new line to a file

 // bytesWritten is an unsigned int. Its value is # of bytes written

} // end InitWriteToFile

task main ()

**Step 4:** Compose a function to write to file. Add the following code above task main:

} // end InitWriteToFile

void WriteToFile(string strTempText) {

 // strTempText stores the text (i.e. ticks and motorRpm to be written to file

 // write string to file

 result=WriteLnString(fileHandle, strTempText, bytesWritten);

 // if the end of file is reached, close the file and create a new part

 if (result==LDR\_EOFEXPECTED) // LDR\_EOFEXPECTED is flagged when end-of-file

 { // close the current file

 CloseFile(fileHandle); // NXT Guide Section 6.59.2.1 pg. 535

 // Closes file associated with file handle

 // create the next file name

 filePart = filePart + 1;

 strFileNumber = NumToStr(fileNumber);

 strFilePart = NumToStr(filePart);

 fileName = StrCat("squareData" , strFileNumber,"-", strFilePart ,".csv");

 // delete the file if it exists

 DeleteFile(fileName); // NXT Guide Section 6.59.2.5 pg. 537

 // Delete the file specified by the string input

 // create a new file

 CreateFile(fileName, 1024, fileHandle);

 // play a tone every time a file is created

 PlayTone(TONE\_B7, 5);

 WriteLnString(fileHandle, strTempText, bytesWritten);

 } // end if

} // end WriteToFile

task main ()

**Step 5:** Next, compose a function that closes the file. Add the following code above task main:

} // end WriteToFile

// Close the file

void StopWriteToFile() {

 // close the file

 CloseFile(fileHandle);

} // end StopWriteToFile

task main ()

Add this function

At this point, save your NxC program. To recap, Step 2 declared the variables needed for file handling and Steps 3 to 5 created functions to respectively initialize (i.e. create), write string data and close a file.

**Step 6:** File data is stored as strings. As such, strings must be declared for each integer and float. Also, to create a file, one must initialize one. Add the following within task main:

task main ()

{

 int x; // integers from 1 to 10

 int xSquared; // square of x

 float xSquareRoot; // square root of x

 string strX;

 string strXSquared;

 string strXSquareRoot;

 // Create a new file that captures time and motor speed

 InitWriteToFile();

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

 xSquared = x\*x;

 xSquareRoot = sqrt(x);

Declare string versions of integers and floats. Also, create a file.

**Step 7:** In the for-loop, the program iterates from 1 to 10, calculating the square and square root. We can use the FormatNum function to create a string version of numbers (i.e. integers and floats). Add the following within the for-loop:

 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);

 // Create string version of calculated values

 strX = FormatNum("%d" , x);

 strXSquared = FormatNum("%d" , xSquared);

 strXSquareRoot = FormatNum("%3.3f" , xSquareRoot);

 } // end of for loop

} // end of main

FormatNum is akin to ANSI-C’s sprintf() function. It creates strings from numbers.

**Step 8:** Finally, one should write the 3 strings (strX, strXSquared and strXSquareRoot) to the file. To do so efficiently, one can employ the ANSI-C strcat function which concatenates multiple strings into a single one. Finally, write the string to file. Add the following code within the for-loop

// Create string version of calculated values

 strX = FormatNum("%d" , x);

 strXSquared = FormatNum("%d" , xSquared);

 strXSquareRoot = FormatNum("%3.3f" , xSquareRoot);

 // Concatenate the 3 strings into a single one.

 // Write resulting string to file. The text will be end with a EOL

 text=StrCat(strX, "," , strXSquared, "," , strXSquareRoot, "," );

 WriteToFile(text);

 } // end of for loop

} // end of main

Use strcat to combine strings. Write resulting string to file

**Step 9:** After the program has generated the data (i.e. completed the for-loop), one terminates the program gracefully by closing the file. One can also add an LCD message and beep to let the user know the program is done. Add the following after the for-loop and before the end of main.

 // Concatenate the 3 strings into a single one.

 // Write resulting string to file. The text will be end with a EOL

 text=StrCat(strX, "," , strXSquared, "," , strXSquareRoot, "," );

 WriteToFile(text);

 } // end of for loop

 // Finished computing square and square root, so clean up and quit

 ClearScreen();

 TextOut(0, LCD\_LINE2, "Quitting", false);

 StopWriteToFile();

 PlaySound(SOUND\_LOW\_BEEP); // Beep to signal quitting

 Wait(SEC\_2);

} // end of main

Add this alert user of termination and close file

**Step 10:** Save, compile and execute the resulting program. The program should iterate from 1 to 10, displaying the integers, its square and square root. Additionally, in the background, the Brick stores the data to file named: squareData.csv.

To view this data file, after the program completes, select Tools – NXT Explorer (see **Figure 1A**). A pop-up box should display the files stored within your NXT Brick (as shown **in Figure 1B**). Click-and-drag the file squareData.csv from the left pane (i.e. Brick’s directory) to the right one (your PC’s drive).



**Figure 1B:** Click-and-drag the data file squareData.csv to your PC.



**Figure 1A:** Launch the NXT Explorer to view Brick’s files

**Step 11:** Double-click on the version of squareData.csv that is saved on your PC. Excel should already be configured to open CSV (comma-separated files), resulting in **Figure 1C. Figure 1D** shows the resulting scatter plot of the first 2 columns.



**Figure 1D:** Scatter plot of first 2 columns of data reveal the expected parabolic curve resulting from computing the square of values.



**Figure 1C:** Excel opens the resulting squareData.csv file. One can then select data for a scatter plot.

**Code Explanation:** displaySquareAndSquareRoot2\_0.nxc iterates from 1 to 10 using a for-loop. Within this loop, the square and square root is also computed. To save any values to a file, one must first declare (Step 2) and initialize (Step 3) one. File data is stored as strings (i.e. a collection of alphanumeric characters). As such, string versions of any computation are needed and the strcat function is used (Steps 6 and 7) along with the file writing function created in Step 4. After computations are finished (i.e. for-loop terminates), the file should be closed (Step 9) using the function created in Step 5. Steps 10 and 11 show the instructions for using NXT Explorer within the BricxCC IDE to export any files saved on the Brick’s memory, to one’s PC.

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

1-1: Iterate integers from -10 to +10 incrementally by 1. Compute the square and cube and save to a file named “squareAndCube.csv”. Export the data file and plot the curves in Excel.

1-2: Capture all file handling functions into a header file named fileSavingFunctions.h. Rewrite a new program called displaySquareAndSquareRoot3\_0.nxc that includes this header file. This new program should run like displaySquareAndSquareRoot2\_0.nxc – and just serves as a sanity check that file saving works.

1-3: Prove that you know how to name data files and save data in desired formats, and can import these into Excel.

**Concept 2 – NXT Timing Review:**

As a quick review, the NXT’s timer capabilities will be explored. Here, the open-loop step response of an NXT motor, connected to Port A, will be captured.

**Step 1:** Open BricxCC, click File – New and save your program as nxtMotorOlsr1\_0.nxc. Write the following code.

// File: nxtMotorOlsr1\_0.nxc

// Auth: P.Oh

// Date: 10/14/12 15:33 - works!

// Vers: 1.0: Uses MotorRotationCount which reports encoder count in degrees

// and program calculates difference over delta tic counts

// Desc: NXT motor on Port A, save open-loop step response data to file

#include "fileSavingFunctions.h"

#define MOTOR OUT\_A // set constant MOTOR for Port A

#define FULL\_SPEED 75 // 75 percent of possible motor speed

#define DEG2RPM 166.667 // deg/msec to RPM

#define RPM2RADPERSEC 0.105; // RPM to rad/sec

task main() {

 // Motor related variables

 long prevAngleInDegrees; // placeholder for degree read by motor encoder

 long curAngleInDegrees; // current motor angle [DEG]

 long deltaAngleInDegrees; // change in motor angle [DEG]

 string strDeltaAngleInDegrees; // string form of deltaAngleInDegrees

 float motorRpm; // motor speed [RPM]

 string strMotorRpm; // store integer value of motorRpm as string

 float motorRadPerSec; // motor speed [rad/s]

 string strMotorRadPerSec; // string form of motorRadPerSec

// Timing related variables

 long prevTick;

 long curTick; // current timer value

 long deltaT; // For calculating time between ticks

 string strDeltaT; // string form of deltaT

 float elapsedTimeInSeconds; // time in seconds

 string strElapsedTimeInSeconds; // string form of elapsed time

**Code description:** task main begins by declaring motor related variables. As one will see later, the NxC function MotorRotationCount(MOTOR) will be used. This function reports the position (not the velocity) of the motor connected to the Brick’s MOTOR port (which happens to be Port A). Also, time related variables are declared. As one will encounter later, the NxC function CurrentTick() will be used to poll the Brick’s current clock (called a tick counter). Like a stopwatch, variables curTick and prevTick are used to calculate the time that has elapsed and store the resulting difference in the variable elapsedTimeInSeconds.

**Step 2:** Continue adding code to your program nxtMotorOlsr1\_0.nxc

// Button related variables

 bool orangeButtonPushed, rightArrowButtonPushed;

 // Create a new file that captures time and motor speed

 InitWriteToFile();

 // Initialize variables

 elapsedTimeInSeconds = 0.0; // set elapsed time to zero

 prevAngleInDegrees = 0; // motor initially motionless so set angle to zero

 // Prompt user to begin step input

 TextOut (0, LCD\_LINE1, "Right Btn starts" );

 do { // wait until user hits right button

 rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);

 } while(!rightArrowButtonPushed);

// Begin step response

 prevTick = CurrentTick();

 TextOut (0, LCD\_LINE1, "Orange Btn quits" );

 // Command motor to move i.e. step input

 OnFwd(MOTOR, FULL\_SPEED); // turn on motor at 100% full speed

**Code description:** The program declares two button related Boolean variables: rightArrowButtonPushed and orangeButtonPushed. These variables are used to detect which buttons the user pushes on the Brick. The code above then initializes a new file by calling the InitWriteToFile() function where data (like time and motor velocity) will be saved.

We will perform a step response on the NXT motor. As such, we initialize the motor’s position at 0 degrees (prevAngleInDegrees = 0;) and say that 0 seconds has elapsed (elapsedTimeInSeconds = 0.0;).

The program then displays a message on the Brick’s LCD and enters a do-while loop that simply polls for the right button to be pressed with NxC’s ButtonPressed function.

Once the user pushed the right arrow button, the Brick’s internal counter is polled (i.e. the stopwatch is pushed to start) with the prevTick = CurrentTick(); statement. The LCD displays a message to prompt the user to hit the orange button to terminate the program. The program then commands the NXT motor with the NxC function OnFwd(MOTOR, FULL\_SPEED);

**Step 3:** Continue adding code to calculate motor velocity and elapsed time

do {

 // Read change in motor angle

 curAngleInDegrees = MotorRotationCount(MOTOR); // get relative position

 deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees;

 strDeltaAngleInDegrees = FormatNum("deltaAngle = %ld", deltaAngleInDegrees);

 // Measure elapsed time and hence motor RPM

 curTick = CurrentTick(); // read timer value

 deltaT = curTick - prevTick; // measure time elapsed between angle reads

 motorRpm = deltaAngleInDegrees \* DEG2RPM / deltaT;

 strMotorRpm = FormatNum("%5.2f" , motorRpm);

 motorRadPerSec = motorRpm \* RPM2RADPERSEC;

 strMotorRadPerSec = FormatNum("%5.3f" , motorRadPerSec);

 elapsedTimeInSeconds = elapsedTimeInSeconds + (deltaT/1000.0); // in sec

 strElapsedTimeInSeconds = FormatNum("%5.3f" , elapsedTimeInSeconds);

 // Display motor actual speed and elapsed time

 TextOut(0, LCD\_LINE4, FormatNum("RPM = %5.2f" , motorRpm));

 TextOut(0, LCD\_LINE6, FormatNum("Time = %5.3f s" , elapsedTimeInSeconds));

 // Write text data to file. The text will be end with a EOL

 text=StrCat(strElapsedTimeInSeconds, "," , strMotorRadPerSec, "," , strMotorRpm,"," );

 WriteToFile(text);

 // Update current tic value and angle

 prevTick = curTick;

 prevAngleInDegrees = curAngleInDegrees;

 Wait(100); // update loop every 100 milliseconds

 // Check if user wants to quit

 orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);

 } while( !orangeButtonPushed && (FreeMemory()>=2000) );

**Code description:** The program begins a do-while loop which exits when the user pushed the Brick’s orange button or when the Brick’s memory storage runs out.

The statement curAngleInDegrees = MotorRotationCount(MOTOR); polls the motor’s internal encoder hardware. The encoder reports values from 0 to 360 degrees and saves the value in the variable curAngleInDegrees. Because the encoder reports relative changes, the change in angle needs to be subtracted from the previously polled value. This is achieved through the deltaAngleInDegrees = curAngleInDegrees - prevAngleInDegrees; statement. For file saving purposes, the numeric data is converted into a string of alphanumeric characters using NxC’s FormatNum function.

Unfortunately NxC does not have a function that reports motor velocities directly. As such, velocity is calculated by polling sequential motor angles and dividing by the elapsed time. The motor velocity in both radians per second and RPM are calculated and saved to file.

Before the loop iterates, the Brick’s “stopwatch” and angle encoder are updated with their last polled values i.e. prevTick = curTick; prevAngleInDegrees = curAngleInDegrees;

Finally, the program invokes the Wait(100); statement. This essentially sets the sampling time of the program to 100 milliseconds. The program loops back unless it detected the user pushing the orange button (or the Brick’s free memory exceeds 2 KB).

**Step 4:** Continue adding code to complete the program

// Orange button pressed, so command 0 speed to motor and quit

 ClearScreen();

 TextOut(0, LCD\_LINE2, "Quitting", false);

 // Stop motor

 OnFwd(MOTOR, 0);

 StopWriteToFile();

 PlaySound(SOUND\_LOW\_BEEP); // Beep to signal quitting

 Wait(SEC\_2);

} // end of main

**Step 5:** Lastly, modify your header file fileSavingFunctions.h

In your InitWriteToFile() function define the file name to store data as well as the first row that labels your data:

void InitWriteToFile() {

 fileNumber = 0; // set first data file to be zero

 filePart = 0; // set first part of first data file to zero

 fileName = "nxtMotorData.csv" ; // name of data file

 result=CreateFile(fileName, 1024, fileHandle);

 // NXT Guide Section 9.100 pg. 1812 and Section 6.59.2.2 pg. 535

 // returns file handle (unsigned int)

// check if the file already exists

 while (result==LDR\_FILEEXISTS) // LDR\_FILEEXISTS returns if file pre-exists

 {

 CloseFile(fileHandle);

 fileNumber = fileNumber + 1; // create new file if already exists

 fileName=NumToStr(fileNumber);

 fileName=StrCat("nxtMotorData" , fileName, ".csv");

 result=CreateFile(fileName, 1024, fileHandle);

 } // end while

 // play a tone every time a file is created

 PlayTone(TONE\_B7, 5);

 fileHeader = "Time [s], Motor Speed [rad/s], Motor Speed [RPM]" ; // header

 WriteLnString(fileHandle, fileHeader, bytesWritten);

 // NXT Guide Section 6.59.2.43 pg. 554

 // Write string and new line to a file

 // bytesWritten is an unsigned int. Its value is # of bytes written

} // end InitWriteToFile

In your WriteToFile() function, edit code so that data is saved to the desired file

void WriteToFile(string strTempText) {

 // strTempText stores the text (i.e. ticks and motorRpm to be written to file

:

:

:

// create the next file name

 filePart = filePart + 1;

 strFileNumber = NumToStr(fileNumber);

 strFilePart = NumToStr(filePart);

 fileName = StrCat("nxtMotorData" , strFileNumber,"-", strFilePart ,".csv");

:

 } // end if

} // end WriteToFile

**Step 6:** Save both nxtMotorOlsr1\_0.nxc and fileSavingFunctions.h files. Compile and execute. Once the program runs, hit the right arrow button to start the step response. After a few seconds (say 5 seconds), hit the orange button to terminate. Open NXT Explorer to retrieve your data file and plot in Excel.

In Excel, you should have a plot that resembles **Figure 2A**.



**Figure 2A:** Excel plot of nxtMotorData.csv

**Exercise 2:** In NxC create programs for the following:

2-1: Write a program that performs an open-loop step response but acquires the NXT motor’s position from 0 to 5 seconds (use a 100 millisecond sample time). Plot the curve. Why does this curve loop like a ramp?

2-2: Prove that you can capture data for desired duration and at a desired sampling time.

**Concept 3 – Aliasing**

Shannon’s sampling theorem puts a limit on the minimum sampling time. Suppose the phenomena you wish to you sample has a frequency $f\_{p}$. Shannon says that you must sample at least twice as fast. Failing to do so will result in aliasing – or essentially data that does not capture the phenomena.

**Step 1:** Prepare a 1 Hz sine wave with a function generator. The peak-to-peak voltage should be less than 5 Volts. Also, adjust the sine wave so that voltages are all positive, that is, have a DC offset of 5 Volts. Thus, the minimum voltage would be 0 Volts, and maximum voltage is 5 Volts (see **Figure 3A** top left).

**Step 2:** Use the oscilloscope to measure the sine wave’s frequency. Calculate the minimum sampling time (based on Shannon’s Theorem). Connect the function generator to Pins 1 and 2 of the NXT Brick via Port 1.



**Figure 3A:** Function generator set for a 1 Hz sine wave with a 2.5 V DC offset (top left) and 4.5 V peak-to-peak voltage (top right). Oscilloscope hooked up to the function generator (bottom left) and shows resulting sine wave (bottom right)

**Step 3:** Write an NxC program to capture the voltages from the function generator. Use a fast sampling frequency (e.g. 10 times the minimum). Capture data for about 5-seconds. Confirm that a plot of the captured data in Excel, matches that function generator’s sine wave.

**Step 4:** Repeat Step 3 but use a sampling frequency slightly above the minimum (e.g. 1.5 times the minimum). Show the resulting plot

**Step 5:** Repeat Step 4 using a sampling frequency at the minimum value. Show the resulting plot.

**Step 6:** Repeat Step 5 using a sampling frequency below the minimum value and show the resulting plot.

**Exercise 3:**

3-1: Provide Excel plots of the sine wave captured in Steps 3, 4, 5, and 6. See **Figure 3B** for an example. Explain which plots have aliasing and why this was the result.



**Figure 3B:** Excel plots for Exercise 3

Note to self: XLS files (as seen in the plots above) suggest that my timing routines are off. Re-examine oscope1\_0.nxc for timing – especially sample time.

**Concept 4 – Data Logger – Temperature Sensing**

**Concept 5 – NXT Voltage Source**

Motors, relays, and pneumatic valves are examples of actuators. As such, actuators are critical components in any robot. Actuators need a power source to adjust motor speed, relay rates, and valve displacements. The NXT Brick’s Ports A, B, and C can be programmed as an adjustable power source so that the actuator’s state can be controlled. Recall the following pin-out



**Step 1:** Connect an NXT motor to Port A. Compose and run the following (sanity check) program

// FILE: voltOut1\_0.nxc - WORKS!

// DATE: 08/18/16 19:33

// AUTH: P.Oh

// DESC: Port A (Motor port) WHITE (Line 1) is M1. BLACK (Line 2) is M2

// Can attach NXT motor (sanity check); piezo buzzer; and E10 7.5V lamp

// VERS: 1.0 - simple program: voltage output (0 to 9V)

task main() {

 // button variables

 bool orangeButtonPushed, rightArrowButtonPushed, leftArrowButtonPushed;

 bool greyButtonPushed;

 int powerLevel; // 0 to 100 will be sent to Port A for corresponding 0V to 9V

 // Prompt user to begin

 TextOut (0, LCD\_LINE1, "Orange Btn starts");

 do {

 orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);

 } while(!orangeButtonPushed);

 ClearScreen();

 TextOut(0, LCD\_LINE1, "Grey Btn Stops");

 TextOut(0, LCD\_LINE3, "<- keys ->");

 powerLevel = 50; // middle of range... this is about 3.9V

 // 10 = about 0.6V; 70 = about 5V; 90 = about 7V

 do {

 leftArrowButtonPushed = ButtonPressed(BTNLEFT, FALSE);

 rightArrowButtonPushed = ButtonPressed(BTNRIGHT, FALSE);

 greyButtonPushed = ButtonPressed(BTNEXIT, FALSE);

 if(leftArrowButtonPushed) powerLevel = powerLevel - 10;

 if(rightArrowButtonPushed) powerLevel = powerLevel + 10;

 if(powerLevel <= 10) powerLevel = 10; // set saturation minimum

 if(powerLevel >= 90) powerLevel = 90; // set saturation maximum

 TextOut(0, LCD\_LINE4, FormatNum("Power = %3d", powerLevel));

 OnFwd(OUT\_A, powerLevel);

 Wait(250); // need some delay so that buttons can be read properly

 } while(!greyButtonPushed); // end do-while

 StopAllTasks();

} // end main

**Code Explanation:** The Brick’s keys are used to adjust the amount of power (i.e. voltage) being sourced out of Port A. Each time the left arrow button is pushed, the power level decrements by 10 and vice-versa for the right arrow button. This results in controlling the NXT motor’s rotational speed.

The power supply comes from the batteries inside the Brick. Thus, the amount of voltage the port can provide is determined by the voltage-levels of the Brick’s batteries. The maximum voltage would be 9 Volts (six 1.5 Volt AA batteries). But often, the batteries (especially rechargeable ones) will have lower voltages.

The NXT Brick uses pulse-width modulation (PWM) to adjust the voltage coming out of Ports A, B, and C. Brick specs say that the PWM cycle is 128 microseconds (or 7800 Hz). Port A can source 800 mA while Ports B and C can source 500 mA. PWM is the ratio of the times when a signal is on and off. This results in efficiency. Think about a bicyclist. The bike’s speed is a ratio of how the bicyclist pedals verses coasts. A bicyclist pedaling 100% of the time will likely get tired quickly. By contrast, pedaling 0% of the time, the bike won’t move.

**Exercise 5:**

5-1: Replace the motor with a voltmeter. Connect the voltmeter’s positive cable to Pin 1 and negative cable to Pin 2 of Port A. Rerun your voltOut1\_0.nxc program. What power level corresponds to +5V? What are the voltages at power levels 10 and 90?

5-2: Replace the voltmeter with a 7.2V lamp and/or a 3-28V piezo buzzer. What happens when you run voltOut1\_0.nxc?

5-3: Replace the voltmeter with an oscilloscope and run voltOut1\_0.nxc. What happens to the wave when you increment/decrement the power level? What is the frequency of the wave? What is the ratio of the on to off times when the power level is 10, 50, 90?