Hands-on Lab

XL-320 NXC Programming – "Hello World (LED)" Example

This lab introduces NXC Programming of the Robotis XL-320 Dynamixel servo. The Lego NXT Brick's Port 4 features a serial interface (RS-485 protocol). This allows the Brick to communicate to the TTL-level serial port on the XL-320. Changing the XL-320's LED color is a "Hello World" example to introduce RS-485 programming and writing instructions to the XL-320.

Preliminary: Hardware connections and explanation

Hardware Connections

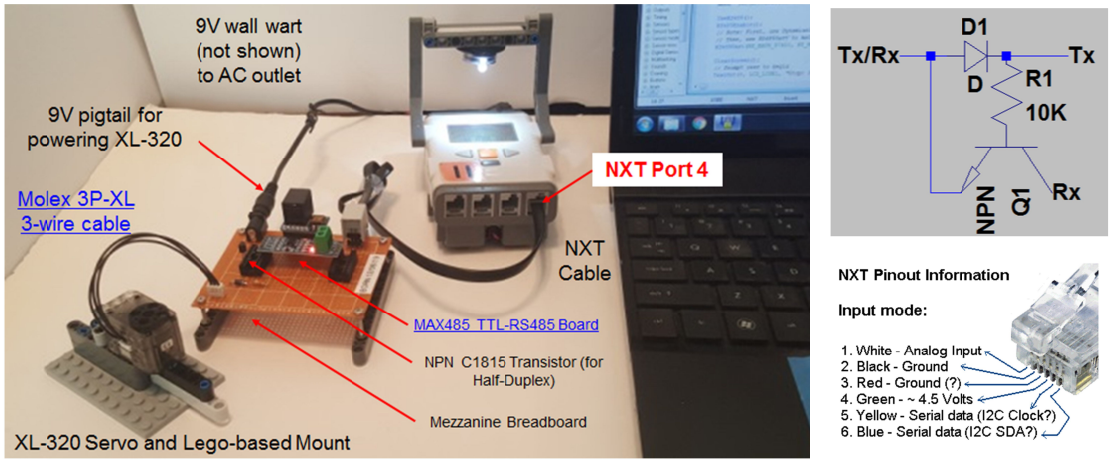


Figure A: NXT-to-XL320 connections (left). NPN Transistor for half-duplex (top right) and NXT cable wire description (bottom right)

Figure A shows the hardware connections. RS-485 protocol digitizes at -7 to +12 Volts. However, the XL-320 uses transistor-to-transistor logic (TTL) to convert bytes digitally (+5V and Ground). As such, a converter is needed. The Maxim MAX485 is a popular chip for such conversion. Its popularity is underscored by $2 boards complete with supporting passive components. One caveat of the XL-320 is that it uses half-duplex RS-485 interfacing. The Molex connector has wires for power, ground, and data. Thus only 1-wire is used to read and/or write bytes (i.e. half-duplex). As such, a NPN transistor is used to implement half-duplexing (**Figure A top right**). Lastly, the NXT cable's Yellow (YLW) and Blue (BLU) wires (**Figure A bottom right**) serve serial purposes when RS485 is invoked.

Dynamixel Protocol 2.0 and XL-320 Firmware

As introduced earlier, the XL-320 is a *smart* servo; firmware is embedded. Firmware is used to permanently hold device information like identifiers (e.g. model or ID number, communication settings like baud rate, and firmware version). Firmware often consists of Random Access Memory (*RAM*) and/or Electronically Erasable Programmable Read-Only Memory (*EEPROM*). RAM holds temporary information like encoder positions and LED states. EEPROM stores more permanent information like read/write instructions.

Robotis' information on the XL-320 is comprehensive, albeit cryptic:

1. XL-320 specifications http://emanual.robotis.com/docs/en/dxl/x/xl320/
2. EEPROM and RAM Control Table
   http://emanual.robotis.com/docs/en/dxl/x/xl320/#control-table
3. Robotis Protocol 2.0 Instruction and Status packets and Packet Processing
   http://emanual.robotis.com/docs/en/dxl/protocol2/

The XL-320 has powerful features and the above links are needed to exploit them.

Concept 1 Create Definition Header File (H-File) `xl320-defines1_0a.h`

Step 1: Create Definition Header File (H-File) – EEPROM Area

A cursory view of links underscores many details of the XL-320. It is helpful to first create a definition header file. This file will `#define` constants that will be needed to read and/or write Instruction and Status packets. Such packets reference the firmware to command the XL-320.

Section 2.2 of http://emanual.robotis.com/docs/en/dxl/x/xl320/ details the EEPROM Control Table and shown in **Figure 1A**.

2. 2. Control Table of EEPROM Area

Address	Size (Byte)	Data Name	Description	Access	Initial Value	Min	Max
0	2	Model Number	Model Number	R	350	-	-
2	1	Firmware Version	Firmware Version	R	-	-	-
3	1	ID	DYNAMIXEL ID	RW	1	0	252
4	1	Baud Rate	Communication Speed	RW	3	0	3
5	1	Return Delay Time	Response Delay Time	RW	250	0	254
6	2	CW Angle Limit	Clockwise Angle Limit	RW	0	0	1023
8	2	CCW Angle Limit	Counter-Clockwise Angle Limit	RW	1023	0	1023
11	1	Control Mode	Control Mode	RW	2	1	2
12	1	Temperature Limit	Maximum Internal Temperature Limit	RW	65	0	150
13	1	Min Voltage Limit	Minimum Input Voltage Limit	RW	60	50	250
14	1	Max Voltage Limit	Maximum Input Voltage Limit	RW	90	50	250
15	2	Max Torque	Maximun Torque	RW	1023	0	1023
17	1	Status Return Level	Select Types of Status Return	RW	2	0	2
18	1	Shutdown	Shutdown Error Information	RW	3	0	7

Figure 1A: Addresses (in Decimal) for each Data Name in **EEPROM**. This table can be found in Section 2.2 (Control Table) of the Robotis XL-320 E-Manual.

In the C programming language, all-uppercase is conventionally used to define constants. For this and future labs, the prefix `EEPROM_`, `RAM_`, and `INSTRUCTION_` will be used before the

`Data Name`. Also, the underscore character will be used between each word. Example, for `Model Number` (top line in **Figure 1A**) would be represented in the H-file as:

```
#define EEPROM_MODEL_NUMBER   0x00 // 2 bytes; motor's model number
```

Following this conventional, the EEPROM defines for the H-file (Yellow highlight) would look like **Figure 1B**.

```
// FILE: xl320-defines1_0a.h
// AUTH: P.Oh
// DATE: 09/19/19 12:24
// VERS: 1.0a: XL-320 motor defines in Control Table; no functions in this file
// DESC: Refers to Section 2
//       http://emanual.robotis.com/docs/en/dxl/x/xl320/#control-table
//       Section 2.2 is EEPROM Control Table defines
//       Section 2.3 is the RAM Control Table defines
// REFS: F:\nationalInstruments\nxcProjects\rs-485\dynamixel\Dynamixel XL-320\
//       paulOhDynamixelXl320HeaderFile-1.0d.h

// Instruction related Defines

#define HEADER_1                    0xFF // For Instruction Packet Header 1
#define HEADER_2                    0xFF // For Instruction Packet Header 2
#define HEADER_3                    0xFD // For Instruction Packet Header 3
#define RESERVED                    0x00 // For Instruction Packet Reserved

// EEPROM Address related Defines
// See Robotis Section 2.2 http://emanual.robotis.com/docs/en/dxl/x/xl320/

#define EEPROM_MODEL_NUMBER         0x00 // 2 bytes; motor's model number
#define EEPROM_FIRMWARE_VERSION     0x02 // 1 byte; motor's firmware version
#define EEPROM_ID                   0x03 // 1 byte; motor's ID number [0-252]
#define EEPROM_BAUD_RATE            0x04 // 1 byte; baud [0-3]
#define EEPROM_RETURN_DELAY_TIME    0x05 // 1 byte; instruction packet send time
#define EEPROM_CW_ANGLE_LIMIT       0x06 // 2 bytes; minimum value of Goal Position
#define EEPROM_CCW_ANGLE_LIMIT      0X08 // 2 bytes; maximum value of Goal Position
#define EEPROM_CONTROL_MODE         0x0B // 1 byte; Wheel (1) or Joint (2) modes
#define EEPROM_TEMPERATURE_LIMIT    0x0C // 1 byte; overheat shutdown value [0-100]
#define EEPROM_MIN_VOLTAGE_LIMIT    0x0D // 1 byte; minimum operational voltage
#define EEPROM_MAX_VOLTAGE_LIMIT    0x0E // 1 byte; maximum operational voltage
#define EEPROM_MAX_TORQUE           0x0F // 2 bytes; maximum torque value
#define EEPROM_STATUS_RETURN_LEVEL  0x11 // 1 byte; how to send status packet
#define EEPROM_SHUTDOWN             0x12 // 1 byte; when to shutdown motor
```

Figure 1B: `#defines` for the **EEPROM** Control Table constants

Step 2: Create Definition Header File (H-File) – RAM Area

Section 2.3 of http://emanual.robotis.com/docs/en/dxl/x/xl320/ details the RAM Control Table and shown in **Figure 1C**. Following the aforementioned naming convention, **Figure 1D** shows the #defines to be added to the H-file in **Figure 1B**.

2. 3. Control Table of RAM Area

Address	Size (Byte)	Data Name	Description	Access	Initial Value	Min	Max
24	1	Torque Enable	Motor Torque On/Off	RW	0	0	1
25	1	LED	Status LED On/Off	RW	0	0	7
27	1	D Gain	Derivative Gain	RW	0	0	254
28	1	I Gain	Integral Gain	RW	0	0	254
29	1	P Gain	Proportional Gain	RW	32	0	254
30	2	Goal Position	Desired Position	RW	-	0	1023
32	2	Moving Speed	Moving Speed(Moving Velocity)	RW	-	0	2047
35	2	Torque Limit	Torque Limit(Goal Torque)	RW	-	0	1023
37	2	Present Position	Present Position	R	-	-	-
39	2	Present Speed	Present Speed	R	-	-	-
41	2	Present Load	Present Load	R	-	-	-
45	1	Present Voltage	Present Voltage	R	-	-	-
46	1	Present Temperature	Present Temperature	R	-	-	-
47	1	Registered	If Instruction is registered	R	0	-	-
49	1	Moving	Movement Status	R	0	-	-
50	1	Hardware Error Status	Hardware Error Status	R	0	-	-
51	2	Punch	Minimum Current Threshold	RW	32	0	1023

Figure 1C: Addresses (in Decimal) for each Data Name in **RAM**. This table can be found in Section 2.2 (Control Table) of the Robotis XL-320 E-Manual.

```
// RAM Address related Defines
// See Robotis Section 2.3 http://emanual.robotis.com/docs/en/dxl/x/xl320/

#define RAM_TORQUE_ENABLE           0x18 // 1 byte; turns on/off torque control
#define RAM_LED                     0x19 // 1 byte; changes motor's LED color
#define RAM_D_GAIN                  0x1B // 1 byte; motor's derivative gain
#define RAM_I_GAIN                  0x1C // 1 byte; motor's integral gain
#define RAM_P_GAIN                  0x1D // 1 byte; motor's proportional gain
#define RAM_GOAL_POSITION           0x1E // 2 bytes; destination position value
#define RAM_MOVING_SPEED            0x20 // 2 bytes; Wheel or Joint dependent
#define RAM_TORQUE_LIMIT            0x23 // 2 bytes; maximum torque limit value
#define RAM_PRESENT_POSITION        0x25 // 2 bytes; motor's present position
#define RAM_PRESENT_SPEED           0x27 // 2 bytes; Wheel or Joint mode dependent [0-2047]
#define RAM_PRESENT_LOAD            0x29 // 2 bytes; currently applied load value is [0-2047]
#define RAM_PRESENT_VOLTAGE         0x2D // 1 byte; present supply voltage
#define RAM_PRESENT_TEMPERATURE     0x2E // 1 byte; motor's internal temperature in Celsius
#define RAM_REGISTERED              0x2F // 1 byte; REG_WRITE instruction received or not
#define RAM_MOVING                   0x31 // 1 byte; Goal Position completed or in-progress
#define RAM_HARDWARE_ERROR_STATUS   0x32 // 1 byte; present hardware error status
#define RAM_PUNCH                   0x33 // 2 bytes; minimum current to drive motor
```

Figure 1D: `#defines` for the **RAM** Control Table constants

Step 3: Create Definition Header File (H-File) – Instruction Area

The XL-320 is one of about a dozen different Dynamixel servos. Their EEPROM and RAM constants may differ but instructions are common. Robotis calls this Protocol 2.0. Section 2.5 of http://emanual.robotis.com/docs/en/dxl/protocol2/#instruction-packet details the Instruction Packet and given in **Figure 1E**.

2. 5. Instruction

The field that defines the type of command.

Value	Instructions	Description
0x01	Ping	Instruction that checks whether the Packet has arrived to a device with the same ID as Packet ID
0x02	Read	Instruction to read data from the Device
0x03	Write	Instruction to write data on the Device
0x04	Reg Write	Instruction that registers the Instruction Packet to a standby status; Packet is later executed through the Action command
0x05	Action	Instruction that executes the Packet that was registered beforehand using Reg Write
0x06	Factory Reset	Instruction that resets the Control Table to its initial factory default settings
0x08	Reboot	Instruction to reboot the Device
0x10	Clear	Instruction to reset certain information
0x55	Status(Return)	Return Instruction for the Instruction Packet
0x82	Sync Read	For multiple devices, Instruction to read data from the same Address with the same length at once
0x83	Sync Write	For multiple devices, Instruction to write data on the same Address with the same length at once
0x92	Bulk Read	For multiple devices, Instruction to read data from different Addresses with different lengths at once
0x93	Bulk Write	For multiple devices, Instruction to write data on different Addresses with different lengths at once

Figure 1E: Addresses (in Hex) for each Instruction. This table can be found in Section 2.5 (Instruction) of the Robotis Protocol 2.0 E-Manual.

Continuing with the naming convention, the #defines in **Figure 1F** can be added to the H-file.

```
// Instruction related Defines
// See Section 2.5
// http://emanual.robotis.com/docs/en/dxl/protocol2/#instruction-packet

#define INSTRUCTION_PING           0x01 // checks if arriving packet ID is same as packet ID
#define INSTRUCTION_READ           0x02 // read data from device
#define INSTRUCTION_WRITE          0x03 // write data to device
#define INSTRUCTION_REG_WRITE      0x04 // registers instruction packet to set for standby
#define INSTRUCTION_ACTION         0x05 // executes packet by INSTRUCTION_ REG_WRITE
#define INSTRUCTION_FACTORY_RESET  0x06 // reset Control Table to factory default
#define INSTRUCTION_REBOOT         0x08 // reboot device
#define INSTRUCTION_CLEAR          0x10 // reset certain information
#define INSTRUCTION_STATUS_RETURN  0x55 // return instruction for the Instruction packet
#define INSTRUCTION_SYNC_READ      0x82 // multiple devices: read all devices
#define INSTRUCTION_SYNC_WRITE     0x83 // multiple devices: write all devices
#define INSTRUCTION_BULK_READ      0x92 // multiple devices: read different devices
#define INSTRUCTION_BULK_WRITE     0x93 // multiple devices: write different devices
```

Figure 1F: `#defines` for the Robotis Dynamixel Protocol 2.0 Instruction Packet

Step 4: Create Definition Header File (H-File) – Packet Headers

Section 3 of Protocol 2.0 http://emanual.robotis.com/docs/en/dxl/protocol2/#status-packet provides details of packets. The XL-320's firmware uses packets to read and/or write instructions. The packets are delivered via RS-485. A packet is information, typically in the form of bytes. Each byte and its location within the packet, connotes instructions. **Figure 1G** shows the packet form:

3. Status Packet

Header1	Header2	Header3	Reserved	Packet ID	Length1	Length2	Instruction	ERR	PARAM	PARAM	PARAM	CRC1	CRC2
0xFF	0xFF	0xFD	0x00	ID	Len_L	Len_H	Instruction	Error	Param 1	...	Param N	CRC_L	CRC_H

Figure 1G: Section 3 of the Robotis Dynamixel Protocol 2.0 illustrates the packet format

The first 4 bytes do not change and hence included in the H-file. See (the un-highlighted section) of **Figure 1B**.

Concept 2 Create Definition Header File (H-File) `xl320-functions1_0c.h`

Preamble:

The XL-320's firmware communicates via packets that have the form of **Figure 1G**. Code is simplified if one creates a separate H-file consisting of functions. The main C program can then call these functions. Function naming is not unique. Defining a naming convention would make code more readable; one could more easily recognize a function specifically written for the XL-320. For this Concept (and future ones), the convention used is the prefix `XL320_` and then a name (first word lower case and subsequent words' first letter capitalized). For example, for lighting up the XL-320's LED, the function would be:

```
void XL320_setLed(unsigned char, unsigned char XL320_ledColor)
```

Packet Structure: Figure 2A annotates **Figure 1G** with more details.

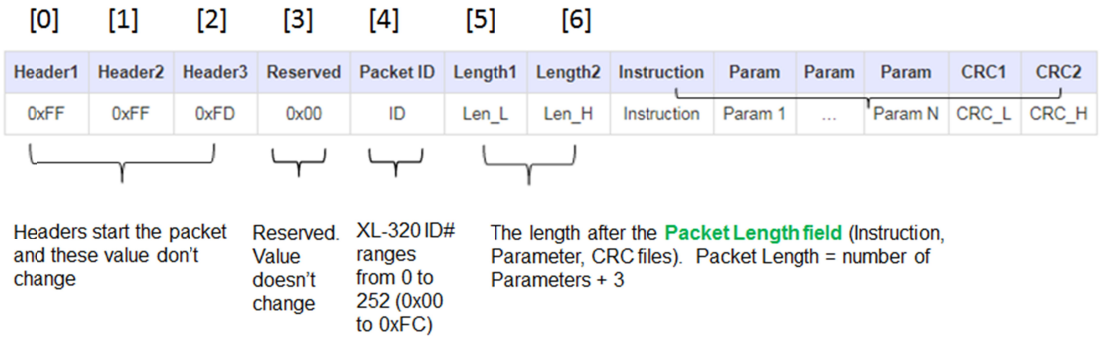
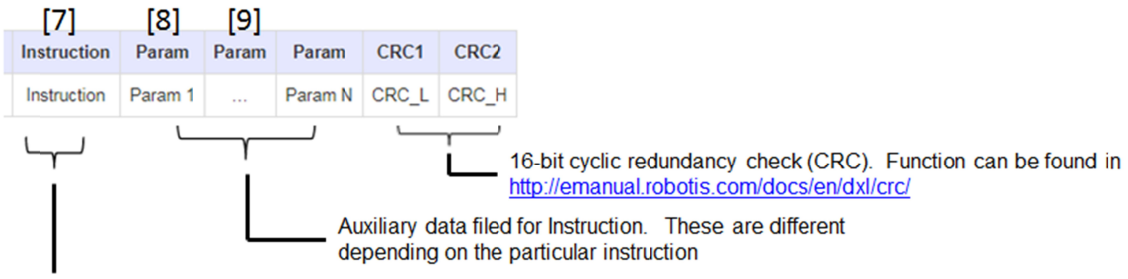


Figure 2A: Annotated explanation of each byte in a packet



Value	Instructions	Description
0x01	Ping	Instruction that checks whether the Packet has arrived to a device with the same ID as Packet ID
0x02	Read	Instruction to read data from the Device
0x03	Write	Instruction to write data on the Device
0x04	Reg Write	Instruction that registers the Instruction Packet to a standby status; Packet is later executed through the Action command
0x05	Action	Instruction that executes the Packet that was registered beforehand using Reg Write
0x06	Factory Reset	Instruction that resets the Control Table to its initial factory default settings
0x08	Reboot	Instruction to reboot the Device
0x10	Clear	Instruction to reset certain information
0x55	Status(Return)	Return Instruction for the Instruction Packet
0x82	Sync Read	For multiple devices, Instruction to read data from the same Address with the same length at once
0x83	Sync Write	For multiple devices, Instruction to write data on the same Address with the same length at once
0x92	Bulk Read	For multiple devices, Instruction to read data from different Addresses with different lengths at once
0x93	Bulk Write	For multiple devices, Instruction to write data on different Addresses with different lengths at once

Figure 2A continued – Table with Values is from **Figure 1E**

One notices that a packet length is not fixed. Some instructions require more information. As such, the `Param` field (see [9] in **Figure 2A**) could be multiple bytes. One references **Figure 1C** to determine how many bytes (and hence Parameters) will be needed.

Writing the **`XL320_setLed`** function

Step 1: Calculate lengths `Len_L` and `Len_H` for the packet (see **Figure 2A** [5] and [6])

The XL-320's LED

Figure 1C shows that the XL-320's LED has an address of 25 Decimal (DEC). Also the size is 1-byte which can take a value from 0 to 7. Figure 2A shows that the packet length is the number of parameters + 3. One thus has:

Number of Parameters = Reserved Byte + Packet ID Byte + Value Byte + 3 Bytes = 6

The XL-320 uses 16-bit integer values. Thus to represent as two 8-bit numbers (i.e. 2 bytes), Protocol 2.0 employs a Little Endian format. Little Endian means that the lower significant bits are stored in the first byte, and the higher ones in the second byte. Hence to express 6 DEC (or 0x06) one uses sets `Len_L` to `0x06` and `Len_H` to `0x00`. The yellow highlight in **Figure 2B** shows this expressed using the naming convention.

```
// Variables to set Length 1 and Length 2
unsigned char XL320_setLedLength_L;
unsigned char XL320_setLedLength_H;

// Variables to set up packet array
unsigned char tempPacket[];   // temporary packet
unsigned char finalPacket[];  // final packet to transmit

// Variables for checksum CRC
unsigned short setLed_CRC;
byte CRC_L, CRC_H;

// 1. Calculate lengths
// Recall that Length 1 and Length 2 = number of parameters + 3
// Setting LED requires only 3 parameters: address, 0x00 and output color
// Hence number of parameters + 3 is 3 + 3 = 6 Dec = 0x06

XL320_setLedLength_L = 0x06;
XL320_setLedLength_H = 0x00;
```

Figure 2B: Len_L (or `XL320_setLedLength_L`) and Len_H (or `XL320_setLedLength_H`) set to 0x06 and 0x00 respectively, represents a packet length of 6 DEC

Step 2: Construct first part of the Packet

Thus far, the packet for setting the LED looks like **Figure 2C**:

Header1	Header2	Header3	Reserved	Packet ID	Length1	Length2	Instruction	Param	Param	Param	CRC1	CRC2
0xFF	0xFF	0xFD	0x00	ID	0x06	0x00	Instruction	Param 1	...	Param N	CRC_L	CRC_H

Figure 2C: From left to right: the first 4 bytes do not change.

The Packet ID is the servo ID number (which one sets under Dynamixel Wizard). The Length1 and Length2 bytes were calculated in the previous step.

In the definition H-file (**`xl320-defines1_0a.h`**) one used #define for constants (see **Figure 2D**):

```
#define HEADER_1                    0xFF // For Instruction Packet Header 1
#define HEADER_2                    0xFF // For Instruction Packet Header 2
#define HEADER_3                    0xFD // For Instruction Packet Header 3
#define RESERVED                    0x00 // For Instruction Packet Reserved
:
:
#define RAM_LED                     0x19 // 1 byte; changes motor's LED color
:
:
#define INSTRUCTION_WRITE           0x03 // write data to device
```

Figure 2D: Snippets of xl320-defines1_0a.h created in Concept 1

In NXC Programming, **Figure 2C** can be represented by the code snipped in **Figure 2E**. Here, the NXC function `ArrayBuild` is used to create a multi-byte packet called `tempPacket`.

```
// 2. Construct first part of packet
ArrayBuild(tempPacket, HEADER_1, HEADER_2, HEADER_3, RESERVED, XL320_servoId,
  XL320_setLedLength_L, XL320_setLedLength_H, INSTRUCTION_WRITE,
  RAM_LED, 0x00, XL320_ledColor);
```

Figure 2E: Code snippet of **xl320-functions1_0c.h**

In yellow-highlight are variables that are passed to the XL320_setLed function, namely, the servo's ID number and desired color.

Step 3: Perform Checksum

Checksum is often employed to verify packets were correctly transferred between devices. The XL-320 uses the Cyclic Redundancy Check (CRC) method to perform the checksum as seen in http://emanual.robotis.com/docs/en/dxl/crc/. The Robotis XL-320 E-Manual provides this function in C as shown in **Figure 2F**.

```
unsigned short update_crc(unsigned short crc_accum, unsigned char *data_blk_ptr, unsigned short data_blk_size)
{
    unsigned short i, j;
    unsigned short crc_table[256] = {
        0x0000, 0x8005, 0x800F, 0x000A, 0x801B, 0x001E, 0x0014, 0x8011,
        0x8033, 0x0036, 0x003C, 0x8039, 0x0028, 0x802D, 0x8027, 0x0022,
        0x8063, 0x0066, 0x006C, 0x8069, 0x0078, 0x807D, 0x8077, 0x0072,
        0x0050, 0x8055, 0x805F, 0x005A, 0x804B, 0x004E, 0x0044, 0x8041,
        0x80C3, 0x00C6, 0x00CC, 0x80C9, 0x00D8, 0x80DD, 0x80D7, 0x00D2,
        0x00F0, 0x80F5, 0x80FF, 0x00FA, 0x80EB, 0x00EE, 0x00E4, 0x80E1,
        0x00A0, 0x80A5, 0x80AF, 0x00AA, 0x80BB, 0x00BE, 0x00B4, 0x80B1,
        0x8093, 0x0096, 0x009C, 0x8099, 0x0088, 0x808D, 0x8087, 0x0082,
        0x8183, 0x0186, 0x018C, 0x8189, 0x0198, 0x819D, 0x8197, 0x0192,
        0x01B0, 0x81B5, 0x81BF, 0x01BA, 0x81AB, 0x01AE, 0x01A4, 0x81A1,
        0x01E0, 0x81E5, 0x81EF, 0x01EA, 0x81FB, 0x01FE, 0x01F4, 0x81F1,
        0x81D3, 0x01D6, 0x01DC, 0x81D9, 0x01C8, 0x81CD, 0x81C7, 0x01C2,
        0x0140, 0x8145, 0x814F, 0x014A, 0x815B, 0x015E, 0x0154, 0x8151,
        0x8173, 0x0176, 0x017C, 0x8179, 0x0168, 0x816D, 0x8167, 0x0162,
        0x8123, 0x0126, 0x012C, 0x8129, 0x0138, 0x813D, 0x8137, 0x0132,
        0x0110, 0x8115, 0x811F, 0x011A, 0x810B, 0x010E, 0x0104, 0x8101,
        0x8303, 0x0306, 0x030C, 0x8309, 0x0318, 0x831D, 0x8317, 0x0312,
        0x0330, 0x8335, 0x833F, 0x033A, 0x832B, 0x032E, 0x0324, 0x8321,
        0x0360, 0x8365, 0x836F, 0x036A, 0x837B, 0x037E, 0x0374, 0x8371,
        0x8353, 0x0356, 0x035C, 0x8359, 0x0348, 0x834D, 0x8347, 0x0342,
        0x03C0, 0x83C5, 0x83CF, 0x03CA, 0x83DB, 0x03DE, 0x03D4, 0x83D1,
        0x83F3, 0x03F6, 0x03FC, 0x83F9, 0x03E8, 0x83ED, 0x83E7, 0x03E2,
        0x83A3, 0x03A6, 0x03AC, 0x83A9, 0x03B8, 0x83BD, 0x83B7, 0x03B2,
        0x0390, 0x8395, 0x839F, 0x039A, 0x838B, 0x038E, 0x0384, 0x8381,
        0x0280, 0x8285, 0x828F, 0x028A, 0x829B, 0x029E, 0x0294, 0x8291,
        0x82B3, 0x02B6, 0x02BC, 0x82B9, 0x02A8, 0x82AD, 0x82A7, 0x02A2,
        0x82E3, 0x02E6, 0x02EC, 0x82E9, 0x02F8, 0x82FD, 0x82F7, 0x02F2,
        0x02D0, 0x82D5, 0x82DF, 0x02DA, 0x82CB, 0x02CE, 0x02C4, 0x82C1,
        0x8243, 0x0246, 0x024C, 0x8249, 0x0258, 0x825D, 0x8257, 0x0252,
        0x0270, 0x8275, 0x827F, 0x027A, 0x826B, 0x026E, 0x0264, 0x8261,
        0x0220, 0x8225, 0x822F, 0x022A, 0x823B, 0x023E, 0x0234, 0x8231,
        0x8213, 0x0216, 0x021C, 0x8219, 0x0208, 0x820D, 0x8207, 0x0202
    };

    for(j = 0; j < data_blk_size; j++)
    {
        i = ((unsigned short)(crc_accum >> 8) ^ data_blk_ptr[j]) & 0xFF;
        crc_accum = (crc_accum << 8) ^ crc_table[i];
    }

    return crc_accum;
}
```

Figure 2F: Section 1 of the CRC Calculation from Robotis E-Manual

The details of how CRC operates are beyond the scope of this lab. Essentially, an NXC equivalent of **Figure 2F** is created as a function `update_crc` in the `xl320-functions1_0c.nxc` file.

The function is called with the following NXC code:

```
// 3. Perform checksum, see Section 1.2 of http://emanual.robotis.com/docs/en/dxl/crc/
unsigned int packetLength = (XL320_setLedLength_H >> 8) + XL320_setLedLength_L;

// See last bullet in Section 1.2 "Packet Analysis and CRC Calculation"
setLed_CRC = update_crc(0, tempPacket, 5 + packetLength);
CRC_L = (setLed_CRC & 0x00FF);
CRC_H = (setLed_CRC >> 8) & 0x00FF;
```

Figure 2G: CRC values are called using the `update_crc` function call

Note that the logic `AND` operator and bit-wise operator `>>` are used to create the Little Endian forms of the low and high bytes for `CRC_L` and `CRC_H` respectively.

Step 4: Concatenate final packet and transmit

Now that the CRC bytes have been calculated, the final form of the packet can be built, transmitted via a NXC `RS485Write` call and confirmed via a NXC `waitForMessageToBeSent` call (see **Figure 2H**).

```
  // 4. Concatenate into final packet and sent thru NXT RS485
  ArrayBuild(finalPacket, tempPacket, CRC_L, CRC_H);
  RS485Write(finalPacket);

  // 5. Call inline function
  waitForMessageToBeSent();

}; // end XL320_setLed function
```

Figure 2H: Final packet creation, transmission and confirmation

Concept 3 Create NXC Main program `xl320-helloLed1_0a.nxc`

With firmware constants defined (Concept 1) and the packet formation for the LED (Concept 2) H-files ready, one can write a simple NXC program to light the XL-320's LED.

Step 1: Start NXC code with comments and #includes to H-files

```
// FILE: xl320-helloLed1_0a.nxc - Works!
// DATE: 09/19/19 13:06
// AUTH: P.Oh
// DESC: Cycles thru XL-320 LED colors
// VERS: 1.0a:
// REFS: xl320-setLed1_0b.nxc; xl320-functions1_0a.h; xl320-defines.h
// NOTE: If factory default XL-320 used, then ID is 0x01
//       ID of 0xFE commands any and all XL-320 motors

#include "xl320-defines1_0a.h"   // XL-320 defines from Control Table
#include "xl320-functions1_0c.h" // P.Oh functions written for XL-320

#define ID_ALL_MOTORS 0XFE       // 0XFE commands all XL-320 motors
#define ID_MOTOR01    0X01       // Assumes Motor 1 configured with ID = 1
```

The yellow-highlights shows include the H-files from Concepts 1 and 2. The ID_MOTOR01 is defined as a constant. This should be the Servo ID number that was set in Dynamixel Wizard (e.g. 0x01). A servo ID number of 0xFE is defined by the XL-320 firmware as commanding any and all connected servos.

Step 2: Declare NXT buttons and establish RS-485 connections

```
task main() {

  byte ledColor;
  bool  orangeButtonPushed;      // Detect Brick Center button state
  bool  rightArrowButtonPushed;  // Detect Brick right arrow button state
  bool  leftArrowButtonPushed;   // Detect Brick left arrow button state

  UseRS485();
  RS485Enable();
  // Note: First, use Dynamixel Wizard to set XL-320 to desired baud rate
  // Then, use RS485Uart to match this baud rate e.g. 9600
  RS485Uart(HS_BAUD_57600, HS_MODE_8N1); // 57600 baud, 8bit, 1stop, no parity

  ClearScreen();
  // Prompt user to begin
  TextOut(0, LCD_LINE1, "Stop: Press ORG" );
```

The program will detect pushing of the NXT Brick's center button (orange-colored one) to commence cycling thru LED colors. The NXC RS-485 functions `UseRS485`, `RS485Enable`, and `RS485Uart` establish a 57600 baud rate and 8N1 (8-bits, no-parity, 1 stop-bit) protocol.

Step 3: Call XL320_setLed function to change LED color

Section 2.4.14 of the XL-320 E-manual http://emanual.robotis.com/docs/en/dxl/x/xl320/#led details the different LED colors on the servo (see **Figure 3A**)

2. 4. 14. LED(25)

The combination of bit changes the output color of XL-320.

Enabled Bit	Decimal Value	Output Color
NONE	0	OFF
0	1	Red
1	2	Green
2	4	Blue
0 + 1	3	Yellow
1 + 2	6	Cyan
0 + 2	5	Purple
0 + 1 + 2	7	White

Figure 3A: Section 2.4.14 of the XL-320 E-Manual shows LED color values

There are 8-states for the XL-320 LED: off or 7 different colors. A `switch` statement, in a `do-while` loop, cycles thru the colors.

```
  // See Section 2.4.14 LED http://emanual.robotis.com/docs/en/dxl/x/xl320/#led
  // LED values: 0 (Off); 1 (Red); 2 (Green); 3 (Yellow); 4 (Blue); 5 (Purple)
  //             6 (Cyan); 7 (White)

  ledColor = 0; // set LED to off first
  do {
     orangeButtonPushed = ButtonPressed(BTNCENTER, FALSE);
     XL320_setLed(ID_ALL_MOTORS, ledColor);
     switch(ledColor) {
       case 0: TextOut(0, LCD_LINE3, FormatNum("%d OFF" , ledColor));
               break;
       case 1: TextOut(0, LCD_LINE3, FormatNum("%d RED" , ledColor));
               break;
       case 2: TextOut(0, LCD_LINE3, FormatNum("%d GRN" , ledColor));
               break;
       case 3: TextOut(0, LCD_LINE3, FormatNum("%d YLW" , ledColor));
               break;
       case 4: TextOut(0, LCD_LINE3, FormatNum("%d BLU" , ledColor));
               break;
       case 5: TextOut(0, LCD_LINE3, FormatNum("%d PUR" , ledColor));
               break;
       case 6: TextOut(0, LCD_LINE3, FormatNum("%d CYA" , ledColor));
               break;
       case 7: TextOut(0, LCD_LINE3, FormatNum("%d WHT" , ledColor));
               break;
     }; // end switch

     Wait(1000);
     ledColor++;
     if(ledColor > 7) ledColor = 0;
  } while(!orangeButtonPushed);
  ClearScreen();

} // end main
```

The yellow-highlighted line shows the call to XL320_setLed, which passes the servo's ID number and desired color). The color name is displayed based on values in **Figure 3A**.

Congratulations! You can change the XL-320 LED color.