ENGDUINO EXAMPLES: COMMUNICATIONS

Humans have been using simple communications systems for centuries to send messages over a long distance, much further than the human voice can reach. Information was sent between signal towers on the Great Wall of China by using flashing lanterns at night or smoke signals during the day. Around 100 BC a Roman historian called Polybius invented a way of using numbers to represent the alphabet using a grid.

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I/J	L
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Table 1- Polybius' Grid

Words can be spelled out using the grid reference of the letters so 'A' is 1,1 and 'S' is 4,3. The message "RED ROSE" is: (4,2) (1,5) (1,4) (4,2) (3,4) (4,3) (1,5). Polybius invented this scheme so that people could send a message using 2 torches. Imagine the sender holding a torch in each hand. Flashing the torch in the right hand to indicates the row of the letter in the grid. Flashing the torch in the left hand to indicates the column of the letter.

With another person:

One of you should encode a short, secret message using the Polybius Grid. Find a way of communicating your message to the other person without speaking. Let the other person decode the message.

IN THIS PROJECT

You will learn how to send and receive secret messages using two Engduinos. You will transmit messages using the LEDs on one device and receive messages using the light sensor on the other. The point of this project is to:

- learn about the way information is stored in a computer using binary,
- explore how information is transmitted between one computer and another.

GETTING STARTED

Computers only understand a very simple and limited language composed of '0's and '1's. This is a binary number system which you have probably learned about already. It is very easy to generate signals to represent this two-digit language, after all, we can think of the digit 0 as **OFF** and the digit 1 as **ON**. The 0 and 1 are known as 'bits'.

Programming the Engduino to send and receive binary messages is quite complex but we can make it easier by breaking the project down into stages:

Stage 1. Make one Engduino talk by sending a binary signal.

Stage 2. Make the other Engduino listen by receiving a binary signal.

Stage 3. Synchronise the sender and receiver.

Stage 4. At the sender, convert a character in a message to a binary signal.

Stage 5. At the receiver, convert binary signal to a character in a message.

You are going to complete this project by writing your own code and by using some code we have written for you.

Download the file containing the sketches you need from the Engduino website. The file is called `SendReceiveSketches.zip`. Save it on the computer's Desktop or somewhere inside your Documents folder.

To 'unzip' the file, right click on it and choose option "Extract All ….."

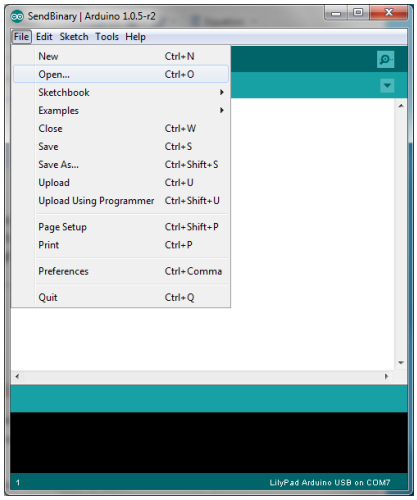
STAGE 1: SEND A BINARY SIGNAL WITH THE ENGDUINO

Complete the Traffic Lights tutorial before attempting this part of the project.

We have seen that you can communicate quite complex messages using a simple coding system like the Probius Grid. Our first task is to program one of the Engduinos to send a binary signal: '0' (OFF) and a '1' (ON) using the LEDs. We can use some of the code from the Traffic Lights tutorial to do this.

SEND A SIGNAL

Start the Arduino IDE. Open the sketch `SendBinary.ino`. Use File -> Open like this:



You can find this file in the `SendBinary` sketch in the folder you downloaded.

Now write some code to turn all the LEDs on for, say 1 second and then turn the LEDs off for 1 second repeatedly. You should add a definition for the length of time it takes to transmit a bit. Add this *immediately after the #include directives* in the following way

```
#define BIT_DURATION 1000 //Length of time (milliseconds) to transmit a bit
```

Then later in the main `loop()`, you can use this value like this:

```
delay(BIT_DURATION);
```

Don't forget to save your program intermittently. Upload the program to the Engduino that will be the sender in your communications system.

Your Engduino is now transmitting a series of bits like this:

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

STAGE 2: RECEIVE A BINARY SIGNAL WITH THE LIGHT SENSOR

You should work through the Light Sensor worksheet.

The next step is to write some code for the Engduino receiver that will interpret the binary signal. The question we need to answer is: are the light sensor readings significantly different when the LEDs on the sending device are on and off?

RECEIVE A SIGNAL

Open the file `ReceiveBinary.ino` which is in the `ReceiveBinary` folder.

Write some code to read the values from the light sensor and print out a message to the serial monitor repeatedly. You will need to upload your program to the second Engduino, the one you are using as a receiver. Make sure it is plugged into a USB port and that you have the correct COM port selected in the Arduino IDE.

The input from the light sensor ranges, in theory, from 0-1023. In practice it will rarely reach 1023 and, if you're just using the illumination in the room, the maximum might only be, say, 700 (you'll need to measure this). Likewise, the minimum might not be zero when it's fully covered – try it. Shine the Engduino with the flashing LEDs directly at the light sensor on the receiver.

What is the light level when the LEDs are off? | |

What is the light level when the LEDs are on? | |

Add some more code to the receiver. If the light level is high then print a message to the serial monitor to show that you have received a '1'. Otherwise, if the light level is low then print a message to the serial monitor to show that you have received a '0'.

Congratulations! You have just created a communications system with a sender and a receiver.

STAGE 3: SYNCHRONISE THE SENDER AND RECEIVER

You should work through the Button worksheet.

Imagine that you have sent this binary message.

1 0 0 0 1 1 0 1 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 0 0 1 1

As you can see, there are some repeated digits. In some places there are two, three or even more repeated 0's or 1's. How will the receiver know how many 0's or 1's it has received?

In a computer system, all binary signals are synchronised using an internal clock. The hardware components in a computer use the clock to tell when to send a bit and when to receive a bit.

Our problem is more difficult to solve, we have two devices, a sender and receiver, and they don't share a clock. We need to synchronise the sender and receiver so that:

1. The sender and receiver know when to start transmitting and receiving.
2. The sender and receiver know how long an LED will flash for each binary digit.

We are going to have to synchronise the sender and receiver using some kind of manual technique. One way that we can do this is to tell the Engduinos when to start by pressing the button on the back of each device. This may not be very accurate but let's see if it's accurate enough.

ADDING A START SIGNAL

Add some code to the sender and the receiver so that they don't start sending or receiving until the button is pressed.

Now add a delay to the receiver so that it checks the light level every 1 second. Remember, the sender is sending a bit every second.

You are now at the point where you can send and receive a signal fairly reliably. Let's do one more thing before we finish this stage. At the moment, our sender and receiver are synchronised as closely as they can be. Both start sending and receiving as soon as we press the button and both wait 1 second before sending and receiving again.

The diagram in Figure 1 shows what happens when the sender and receiver are synchronised accurately enough.

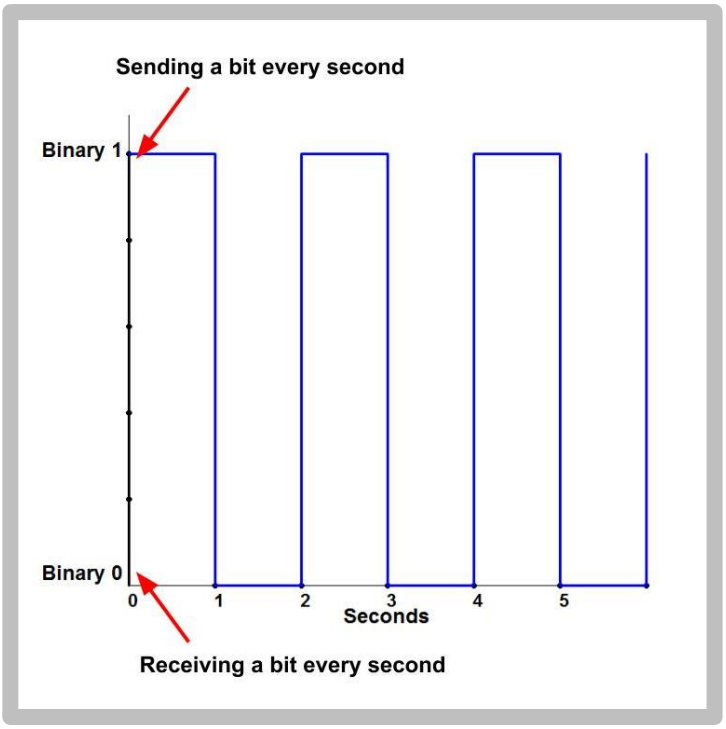
The 

Figure 1 - Sending and Receiving a Binary Digit

You can see that if there is a small delay at the receiver, things might be different. The receiver might not read the correct bit. To reduce the possibility of this happening you can add a small delay of 500 milliseconds to the receiver. Do this now, add the delay right after the code for waiting for the button press. This will ensure that the receiver reads the light sensor roughly in the middle of the transmission.

Next, we have to make that signal mean something – our message.

STAGE 4: TRANSLATE A LETTER OF THE ALPHABET INTO A BINARY SIGNAL

Transferring bits from one Engduino to the other is all very well but humans do not communicate in 0's and 1's; when we write things down we use an alphabet. Our next task is to write some code to translate letters of the alphabet into binary at the sender so that they are ready for transmission.

The coding system which is most often used to represent the alphabet is known as ASCII which stands for the **American Standard Code for Information Interchange**. At the time it was invented, ASCII was used to encode 128 characters: 0-9, a-z and A-Z and some punctuation symbols. Nowadays, ASCII has been superseded by UTF which stands for **U**niversal Character Set, **T**ransformation **F**ormat. UTF can accommodate many different alphabets and symbols such as Arabic and Kana, which is a Japanese script.

In ASCII, each character is represented as 7 or even 8 bits of binary. In this program we will use 7 bits. Table 1 contains the binary representations of the characters 'A' through to 'E'.

English Alphabet	Binary
A	1000001
B	1000010
C	1000011
D	1000100
E	1000101

Table 2 - ASCII to Binary Conversion

It would take a while to type in the ASCII conversion table into our program so we will use some code that exists already. As programmers, we use and re-use code regularly, after all there is no point in writing code yourself if someone else has already done it.

You have been given two extra sketches with your Sender and Receiver sketches called `ASCII2Binary.ino` and `Binary2ASCII.ino`. Take a look at the files. Even if you don't understand the code yet, you will see something that looks like a lookup table near the top of each one.

These files contain some simple conversion utilities. As their names suggests, one will convert an ASCII character into seven binary digits. The other will convert seven binary digits into an ASCII character.

Before we can use these utilities, we will have to learn how to store our binary digits in an array. Let's learn what an array is.

ASIDE: ARRAYS

An array is a data structure used in just about all programming languages. In our case, it's a numbered collection of things of the same type with a fixed maximum size. In essence it's a set of boxes into which we can put values – each box has a number, starting at 0 and going up.

Say we needed to store 7 binary digits, then we create an array of length 7. Let's say we store these values in an array of integers and we want the name of the array to be 'binaryValue'. We declare the array *immediately after the #include directives* in the following way:

```
#define ARRAYSIZE 7
integer binaryValue[ARRAYSIZE];
```

This creates an array of 7 boxes (labelled 0, 1, 2, 3, 4, 5, 6) in which we can store integer numbers.

0	1	2	3	4	5	6
1	**1**	**0**	**0**	**0**	**1**	**0**

The letter 'b' in binary

At the sender, we are going to translate an ASCII character to a set of binary digits then we are going to make the Engduino flash on and off for each digit.

To do this, we need to go through each cell in the array in turn. For this we need another variable that will tell us which of those spaces, or boxes, contains the digit we'll transmit next – we'll call this the index variable.

We just need to organise that the index variable always points to the next box that we want to transmit. To achieve this, add the following after the above two lines – it creates a variable 'index' and sets its value to be 0 (the first location in buffer).

```
int index = 0;
```

Now let's look at how to use this.

BACK AGAIN

In the `loop()` section of your sketch, you will need something that looks a bit like the code below:

```
void loop() {


  //
  // Wait until the button has been pressed.
  //

  EngduinoButton.waitUntilPressed();


  char nextCharacter = 'a';       // a character to transmit


  //
  // We have the next letter in our message, now lookup the binary value
  // using the method ASCII2Binary.
  //

  ASCII2Binary(nextCharacter, binaryValue);

  for (int index = 0; index < ARRAYSIZE; index++) {

       if (binaryValue[index] == 1){
                EngduinoLEDs.setAll(WHITE);
       }

       delay(BIT_DURATION);
       EngduinoLEDs.setAll(OFF);
  }
```

Give this a go…. add some print statements to your code using the `Serial.print()` function so that you can see what it is doing using the Serial monitor.

SENDING A MESSAGE CONTAINING SEVERAL CHARACTERS

Now that you know how to use arrays you can create one in which to store your simple message:

```
#define MESSAGESIZE 5

char message[MESSAGESIZE]={'H', 'E', 'L', 'L', 'O'};
```

This time, our array contains characters. Add an array containing a message to your program. Use a `for loop` to go through each character in the message and translate it to a binary value.

STAGE 5: TRANSLATE A BINARY SIGNAL TO A LETTER OF THE ALPHABET

The final stage of this project is to collect 7 bits of binary at the receiver and translate them into an ASCII character.

Go back to editing the code for the receiver. In the same way as you did before, declare an array of integers, length 7. Let's say we we want the name of the array to be 'binaryValue', again, we declare the array *immediately after the #include directives* in the following way:

```
#define ARRAYSIZE 7
integer binaryValue[ARRAYSIZE];
```

Now edit your code so that you are using the array to store the binary signals you are collecting. Your code should look something like this:

```
void loop() {

  int lightValue;

  // Wait until the button has been pressed.

  EngduinoButton.waitUntilPressed();

  delay(BIT_DURATION / 2)                  // Wait for a short time

  //
  // Detect 7 bits
  //

  for (int i = 0; i < ARRAYSIZE; i++) {

      lightValue = EngduinoLight.lightLevel();

      if ( lightValue >= LIGHT_ON )      // Assume 1 received
          binaryValue[i] = 1;

      else
          binaryValue[i] = 0;            // Assume 0 received

      delay(BIT_DURATION);
  }
```

CONVERTING THE BINARY DIGITS TO A CHARACTER

Look at the sketch called Binary2ASCII, how do you think you should use it? Give it a go …..

WHAT NEXT?

In this tutorial you have covered the basic elements of all digital communications systems but there are several ways in which our communications system could be improved. You have probably noticed that sometimes the receiver does not receive the bit correctly. You could think about using a parity bit for this, take a look at the Section on Error Detection and Correction in the A-Level Computing Wikibook by Peter Kemp.