Lesson 13: Infrared Transmitters

The Big Idea:

In Lesson 12 the ability to detect infrared radiation modulated at 38,000 Hertz was added to the Arduino™. This lesson brings the ability to generate modulated infrared. The two can be used in combination to make a system that can detect nearby objects and obstacles. IR is transmitted, and the IR sensor looks for reflections.

Lesson 15 uses a pair of generators and sensors to add the ability to turn away from (or toward) an obstacle. But this has many other uses. Model railroad enthusiasts, for example, can detect trains. Elevators use such combinations to determine the precise position of the door relative to the floor.

Figure 13-1 shows a pair of infrared transmitter/receivers used by some Sears garage door openers to detect the presence of small children and pets.

Figure 13-1. Garage door obstacle detectors

Background:

About infrared transmission

By now the light-emitting diode (LED) is familiar. It is simply a semiconductor device that can emit electromagnetic radiation in the frequencies of visible light. Semiconductor technology makes up most of what is referred to as electronics. In addition to diodes, this family of devices includes transistors and integrated circuits, the building blocks of computers including the ATmega 328 that is the heart of the Arduino™ Uno.

An infrared transmitter is almost exactly the same as an LED. It, too, is a diode that emits electromagnetic radiation. But the frequencies of this radiation are just below the red end of the visible light spectrum. This is the range named infrared. The term "infra" is Latin meaning "below."

Because of the similarity to the LED, the infrared diode is referred to as an IRED.
The human eye does not respond to infrared. But some inexpensive digital cameras of the type found in older cell phones can. The circuit shown in Figure 13-2 will produce infrared light easily seen by one of these cameras.

<table>
<thead>
<tr>
<th>+5Volts</th>
<th>220 Ohm</th>
<th>Infrared Diode</th>
</tr>
</thead>
</table>

An infrared diode is electrically almost the same as a light-emitting diode. The difference is that the light emitted is not visible to the human eye. This is why infrared diodes are used for television remote controls.

To accurately mimic what the eye can see, better digital cameras filter out infrared light. But older cell phone cameras lacked this filter.

Figure 13-2. IRED circuit diagram

Recall from Lesson 12 that infrared is very common. For this reason, infrared used in remote control electronics is modulated at 38,000 Hz. This modulation is unlikely to be found in nature.

**About modulation**

Lesson 12 introduced the terms modulation, frequency, and the Hertz. This lesson will show how an Arduino™ can be programmed to transmit modulated pulses of infrared via an IRED. To do so, however, a little more needs to be understood about this modulation.

The IRED must be turned on for a short time and then turned off for a short time; this must happen 38,000 times per second. This pair of events is called a **cycle**. The length of time consumed by one cycle is called the **period**.

The units of time used for the Arduino™ are the second, the millisecond, and the microsecond. The relationships are: 1000 microseconds equals one millisecond, and one thousand milliseconds equals one second.

**Table 13-1. Arduino™ C-language time units**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Relation to second</th>
<th>Programming example</th>
</tr>
</thead>
<tbody>
<tr>
<td>second</td>
<td>s</td>
<td>1:1</td>
<td>none</td>
</tr>
<tr>
<td>millisecond</td>
<td>ms</td>
<td>1 ms = 10⁻³</td>
<td>delay(&lt;ms&gt;)</td>
</tr>
<tr>
<td>microsecond</td>
<td>µs</td>
<td>1 µs = 10⁻⁶ seconds</td>
<td>delayMicroseconds(&lt;µs&gt;)</td>
</tr>
</tbody>
</table>

Figure 13-3 shows a few of the cycles within a portion of a pulse. One of these cycles is identified; the duration in microseconds shown.
One cycle of a pulse modulated at 38,000 Hz is composed of the IRED being set on for 13 microseconds then off for 13 microseconds. The period of one cycle is 26 microseconds.

**Figure 13-3. Cycle within a portion of a pulse**

These modulation cycles may be thought of as pulses within pulses. The IR pulse itself is the time the IRED is on. When it is on it is actually turning on and off very rapidly. This rapid on and off is the modulation.

Why 13 microseconds on, followed by 13 microseconds off? It's math.

<table>
<thead>
<tr>
<th>Step 1: Calculate the time for one cycle:</th>
<th>[ t_{cycle} = \frac{1}{38000 \text{ cycles / second}} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ = 2.6 \times 10^{-5} \text{ seconds / cycle} ]</td>
</tr>
<tr>
<td></td>
<td>[ = 26 \times 10^{-6} \text{ seconds / cycle} ]</td>
</tr>
<tr>
<td></td>
<td>[ = 26 \mu s / cycle ]</td>
</tr>
</tbody>
</table>

| Step 2: Assume a symmetrical cycle. This means the time on must equal the time off. | \[ t_{on} = 13 \mu s \] and \[ t_{off} = 13 \mu s \] |

**Table 13-2. Vocabulary**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>A data type that may have only two states: true and false. First encountered in the lesson on input from the serial port as the result of a test of the number of characters waiting to be read.</td>
</tr>
<tr>
<td>cycle</td>
<td>A single pulse followed by a period of no pulse just before another pulse. With infrared the period of one cycle is 26 microseconds.</td>
</tr>
<tr>
<td>flow chart</td>
<td>A method of diagramming process flow, frequently used by programmers to describe how a portion of an Arduino™ sketch is to work.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>infrared-emitting diode (IRED)</td>
<td>A semiconductor device similar to an LED except that the electromagnetic radiation frequency is just below visible red light.</td>
</tr>
<tr>
<td>period</td>
<td>The elapsed time of one cycle. With infrared, the period of one cycle is 26 microseconds.</td>
</tr>
<tr>
<td>second, millisecond, microsecond</td>
<td>Measures of time, each 1/1000 of the preceding measure.</td>
</tr>
<tr>
<td>programming statement execution time</td>
<td>The time taken by the Arduino™ to perform a programming statement. Execution time must be allowed for in portions of a sketch where timing is important, such as generating a pulse.</td>
</tr>
</tbody>
</table>

**Discussion:**

This lesson implements an IRED in combination with an infrared sensor to construct an obstacle sensor, much as might be used by a rolling robot to find its way through a maze. The programming statements to do this fit neatly into a helper method. This method has three basic steps: light the IRED for a short time; determine if the IR sensor saw any infrared reflections from an obstacle; and return the results to the method that called it.

![Flow chart of helper method that detects obstacles by flashing modulated infrared light](image)

If an obstacle is present, some of this infrared light will be reflected where it can be seen by the IR sensor.

After flashing the infrared light, the program code reads the digital pin to which the output of the IR sensor is attached.

If that pin is LOW, then infrared was detected. Otherwise, infrared was not detected.

Figure 13-4. Flow chart of helper method that detects obstacles by flashing modulated infrared light
About flow charting
The chart in Figure 13-4 is an example of flow charting, a process commonly used by engineers to diagram processes. Figure 13-4 shows how the IR transmitter and sensor can be used together to detect an obstacle. It also serves to guide the writing of the C-language program to make this happen. Each shape has a specific meaning.

Table 13-3. Shapes used in flow charting in this lesson

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Start / End</td>
<td>Identifies the beginning of a process and the end. A process may have more than one end but not more than one beginning. In C, the beginning might be the name of a method.</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>A specific task. In C, this is a collection of programming statements that do one thing. An example is using a for loop to light and modulate an IRED.</td>
</tr>
<tr>
<td></td>
<td>Decision</td>
<td>Selects what is to happen next based on the state of some variables. In C, this is usually represented by if and if-else.</td>
</tr>
</tbody>
</table>

Program code
Because the task of attempting to detect an obstacle is self-contained, it fits well into its own method. For illustration, suppose that method is named `lookForReflection()`. The flow chart says either an obstacle is detected or it is not. This is a true or a false, where a returned value of true means yes, an obstacle reflected some infrared light.

Boolean data type
C has a data type called `boolean`. A `boolean` variable is one that can have as a value only true or false. These are, in fact, the words used to represent these values. Example 13-1 shows the declaration of a variable of type `boolean` and the setting of the value of true.

**Example 13-1. Declaring variable type of boolean and value of true**
```c
boolean gameOver;  // declares variable of type boolean
gameOver = true;  // gameOver is now true
```

The helper method `lookForReflection()` is to return true if an obstacle reflects infrared. Otherwise it is to return false. So, for this reason, the variable is written as having a return type of `boolean`. 
**Example 13-2.**
```java
boolean lookForReflection(){
    // programming statements
    // go here
}
```

**Generating one pulse cycle**

One single infrared pulse is **HIGH** for 13 microseconds, then **LOW** for 13 microseconds. Assume `IRTransmitterPin` is initialized to the pin number of the digital port to which the IRED is attached. The programming statements shown in Example 13-3 generate one period of an infrared pulse.

**Example 13-3.**
```java
digitalWrite(IRTransmitterPin, HIGH);
delayMicroseconds(13);
digitalWrite(IRTransmitterPin, LOW);
delayMicroseconds(13);
```

**Generating a modulated infrared pulse**

The period of one cycle is 26 microseconds. To generate an IR pulse of a specific duration a `for` loop is used. The number of times the loop runs is determined by the duration of the pulse. Typically, an infrared pulse is one millisecond. The general formula for the number of periods is shown in Example 13-4.

**Example 13-4.**
```java
numberOfPeriods = (38,000 periods / sec) * (duration in seconds)
```

One millisecond is $10^{-3}$ seconds. The number of periods, then, is as shown in Example 13-5.

**Example 13-5.**
```java
numberOfPeriods = (38,000 periods / sec) * (10^{-3} sec)
numberOfPeriods = 38 periods
```

So, the final `for` loop to generate a modulated pulse of 1 millisecond duration is shown in Example 13-6.

**Example 13-6.**
```java
// not quite correct. Does not account for the time of
// execution of the for loop statements
for(int counter = 0; counter < 38; counter++){
digitalWrite(IRTransmitterPin, HIGH);
```
As precise as this is, the code is still not quite right because of *programming statement execution time*. This is the program delay caused by the execution of program `for` statement, where the loop is ended, the counter incremented, and the test performed. So, the IRED is off longer than 13 microseconds.

\[ \text{timeOff} = 13 \text{ milliseconds} + \text{the statement execution time of processing the for loop.} \]

For the Arduino™ Uno, this is about 4 microseconds.

The corrected code subtracts this from the second delay. The revised loop is shown in Example 13-7.

**Example 13-7.**

```c
// includes latency for Arduino™ Uno
for(int counter = 0; counter < 38; counter++){
  digitalWrite(IRTransmitterPin, HIGH);
  delayMicroseconds(13);
  digitalWrite(IRTransmitterPin, LOW);
  delayMicroseconds(9);  // adjusted for statement execution time
}
```

**Reading the IR Receiver and Returning the Result**

Programming statements for reading an infrared sensor are covered in Lesson 12. To these are added the statements to return true or false. `IRDetectorPin` is assigned the number of the Arduino™ pin to which the output of the IR sensor is attached.

**Example 13-8.**

```c
boolean result;
if(digitalRead(IRDetectorPin) == LOW){
  result = true;
} else {
  result = false;
}
return result;
```

Putting all this together the completed helper method is shown in Example 13-9.
Example 13-9.

```cpp
boolean lookForReflection(){
    // Send IR for 1 ms
    int halfPulse = 13;
    for(int counter = 0; counter < 32; counter++){
        digitalWrite(IRTransmitterPin, HIGH);
        delayMicroseconds(halfPulse);
        digitalWrite(IRTransmitterPin, LOW);
        delayMicroseconds(halfPulse - 4);
    }

    boolean result;
    if( digitalRead(IRDetectorPin) == LOW){
        result = true;
    } else {
        result = false;
    }
    return result;
}
```

The final six lines of program code can be simplified as shown in Example 13-10.

Example 13-10.

```cpp
if( digitalRead(IRDetectorPin) == LOW){
    return true;
} else {
    return false;
}
```

Goals:
The goals of this lesson are:

1. Know that infrared diodes are connected to digital ports and are wired and programmed exactly the same way.

2. Be able to write an Arduino™ sketch that generates infrared light that can be detected by an infrared detector. This means that when on, the IRED is modulated at 38,000 Hz.

3. Be able to write an Arduino™ sketch that pairs an IRED with an IR sensor to form a system capable of detecting obstacles that reflect infrared light.
### Materials:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Part</th>
<th>Image</th>
<th>Notes</th>
<th>Catalog Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arduino™ Uno</td>
<td><img src="image" alt="Arduino Uno" /></td>
<td>Single-board computer. This board is delicate and should be handled with care. When you are not using it, keep it in a box or plastic bag.</td>
<td>3102</td>
</tr>
<tr>
<td>1</td>
<td>USB Cable</td>
<td><img src="image" alt="USB Cable" /></td>
<td>This is a standard USB adapter cable with a flat connector on one end and a square connector on the other.</td>
<td>2301</td>
</tr>
<tr>
<td>1</td>
<td>Computer with at least one USB port and access to the Arduino™ website, <a href="http://www.arduino.cc">http://www.arduino.cc</a></td>
<td><img src="image" alt="Computer" /></td>
<td>The operating system of this computer must be Windows, Macintosh OS/X, or Linux.</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>Bread-board</td>
<td><img src="image" alt="Bread-board" /></td>
<td>Used for prototyping.</td>
<td>3104</td>
</tr>
<tr>
<td>As req’d</td>
<td>Jumper wires</td>
<td><img src="image" alt="Jumper wires" /></td>
<td>Used with bread-boards for wiring the components.</td>
<td>3105</td>
</tr>
<tr>
<td>1</td>
<td>Light-emitting diode (LED)</td>
<td><img src="image" alt="Light-emitting diode" /></td>
<td>Single color (color doesn't matter), about 0.02 amps rated current, diffused.</td>
<td>1301</td>
</tr>
<tr>
<td>2</td>
<td>Resistors, 220 ohm</td>
<td><img src="image" alt="Resistors" /></td>
<td>1/4 watt, 5% tolerance, red-red-brown.</td>
<td>0102</td>
</tr>
<tr>
<td>1</td>
<td>Infrared headlight</td>
<td><img src="image" alt="Infrared headlight" /></td>
<td>Infrared-emitting diode with light-proof shrink wrap. See Note 1</td>
<td>1302</td>
</tr>
<tr>
<td>1</td>
<td>Infrared sensor</td>
<td><img src="image" alt="Infrared sensor" /></td>
<td>3-pin, 38kHz.</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: Headlight is constructed from an infrared LED and two 1/2-inch sections of heat-shrink tubing. See How-To #6 on [LearnCSE.com](http://LearnCSE.com) for instructions.
**Procedure:**

1. Using the Arduino™ Uno and bread-board, construct the circuit shown in Figures 13-5 and 13-6.

![Figure 13-5. IR transmitter / receiver schematic](image)

When used together, as in this lesson, the headlight is placed in front or alongside of the receiver, as shown in Figure 13-6. This is to prevent the receiver from responding to the light source itself.

![Figure 13-6. Proximity of headlight to receiver](image)
1. Connect the Arduino™ to the computer and bring up the Arduino™ IDE.
2. Create a new Arduino™ sketch to be named Lesson13IRTransmit.
3. Enter the heading and the defined values as shown in Snippet 13-1.

Snippet 13-1.

```cpp
/* Lesson13IRTransmit
 <author>
 <date>
 */

// define the pins to be used
#define pinLED 2       // for LED
#define pinIRED 3      // for IRED
#define pinIRDetect 8  // for IR Sensor
...
```
4. Initialize the pins in the `setup()` method.

**Snippet 13-2.**

```c
... void setup(){
    pinMode(pinLED, OUTPUT);
    pinMode(pinIRED, OUTPUT);
    pinMode(pinIRDetect, INPUT);
}
```

5. Add the `loop()` method to light the LED while an obstacle is detected.

**Snippet 13-3.**

```c
... void loop() {
    boolean obstacle;
    obstacle = lookForReflection();

    if(obstacle){
        digitalWrite(pinLED, HIGH);
    } else {
        digitalWrite(pinLED, LOW);
    }
    delay(10);
}
```

6. Finally, add the helper method that lights the IRED then looks at the IR sensor for evidence of a reflection.

**Snippet 13-4.**

```c
... boolean lookForReflection(){
    // Send IR for 1 ms
    int halfPulse = 13;
    for(int counter = 0; counter < 32; counter++){
        digitalWrite(pinIRED, HIGH);
        delayMicroseconds(halfPulse);
        digitalWrite(pinIRED, LOW);
        delayMicroseconds(halfPulse - 4);
    }
}
```
if( digitalRead(pinIRDetect) == LOW){
    return true;
}
return false;

1. Save the sketch. Upload it and test by moving a hand or other obstacle in front of the IRED and removing it. When an obstacle is directly in front of the IRED the LED should light. Otherwise, the LED should be dark.

**Exercises:**
The range of the detector is influenced by three factors. These exercises explore each factor.

*Exercise 13-1. Reflectivity of the obstacle*
Begin with a flat, white surface for the obstacle. What is the maximum distance the surface can be from the sensor and still light the LED? How does the LED respond to other colors? Complete Table 13-4.

<table>
<thead>
<tr>
<th>Color</th>
<th>Maximum distance (inches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td></td>
</tr>
<tr>
<td>Warm color</td>
<td></td>
</tr>
<tr>
<td>Cool color</td>
<td></td>
</tr>
</tbody>
</table>

*Exercise 13-2. Brightness of IRED*
The IRED’s current-limiting resistor influences how much current passes through the diode. Increasing this resistance will reduce the brightness. Experiment with different resistors and record the maximum distance from which a white surface can be detected. Complete Table 13-5.

<table>
<thead>
<tr>
<th>Resistor</th>
<th>Maximum distance to white surface (inches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>150 ohm brown-green-brown</td>
<td></td>
</tr>
<tr>
<td>220 ohm red-red-brown</td>
<td></td>
</tr>
<tr>
<td>1000 ohm brown-black-red</td>
<td></td>
</tr>
<tr>
<td>2000 ohm red-black-red</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 13-3. Frequency of modulation
Replace the IRED's current-limiting resistor with the original 220 ohm (red-red-brown). Change the modulation frequency by modifying the value assigned to the integer variable `halfPulse` in the `lookForReflection()` method. Complete Table 13-6.

Table 13-6. Observed distance as function of pulse width

<table>
<thead>
<tr>
<th>Value of <code>halfPulse</code></th>
<th>Maximum distance (inches)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Complete listing 13-1. Lesson13IRTransmit

/* Lesson13IRTransmit
   by W. P. Osborne
   6/30/15
*/
// define the pins to be used
#define pinLED 2       // for LED
#define pinIRED 3      // for IRED
#define pinIRDetect 8  // for IR Sensor

void setup(){
    pinMode(pinLED, OUTPUT);
    pinMode(pinIRED, OUTPUT);
    pinMode(pinIRDetect, INPUT);
}

void loop() {
    boolean obstacle;
    obstacle = lookForReflection();

    if(obstacle){
        digitalWrite(pinLED, HIGH);
    } else {
        digitalWrite(pinLED, LOW);
    }
}

boolean lookForReflection(){
    // Send IR for 1 ms
    int halfPulse = 13;
    for(int counter = 0; counter < 32; counter++){
        digitalWrite(pinIRED, HIGH);
        delayMicroseconds(halfPulse);
        digitalWrite(pinIRED, LOW);
        delayMicroseconds(halfPulse - 4);
    }

    if( digitalRead(pinIRDetect) == LOW){
        return true;
    }
    return false;
}