A universal algorithm for implementing an infrared decoder

Mrinal Ray -May 06, 2009

In today's modern world, we find a wide range of appliances managed by handheld remote controls, whose decoder function historically has often been provided by a dedicated chip. However, given the pace at which consumer electronics equipment is evolving, it often becomes necessary to upgrade a product's firmware. The new firmware sometimes provides additional functionality to the remote-control interface; as a result, designers are now looking at alternatively building the remote decoder functionality in a microcontroller for easy upgradeability.

Many IR (infrared) remote-control protocols currently exist in the industry; the NEC protocol, JVC protocol, and SIRC (Sony Infrared Remote Control) protocol are some well-known examples. Most of the IR protocols use PDM (pulse distance modulation). Multiple IR protocols means many algorithms and many different chips to decode each of them. Alternatively, a universal algorithm that can decode each of the existing PDM IR protocols will naturally save a lot of development time and resources.

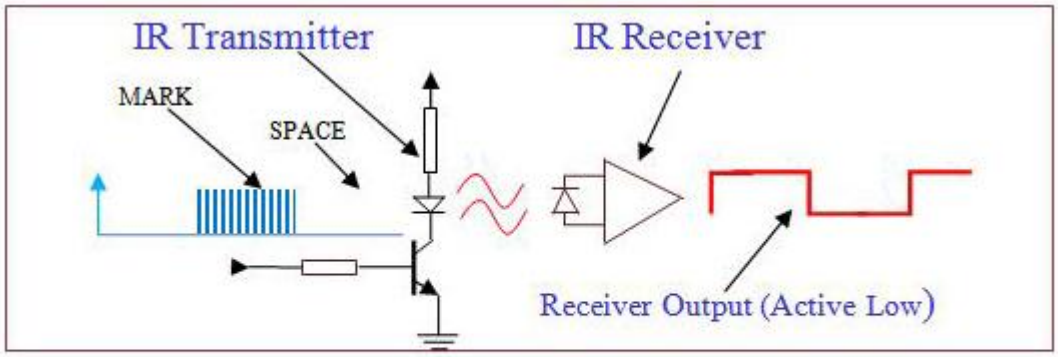

Figure 1. A typical IR transmission and reception setup can be implemented in a generic microcontroller or programmable logic device.

This article discusses a UIRDA (universal PDM IR protocol decoder algorithm). The UIRDA's implementation for multiple IR protocols shows the generic nature (i.e., the universality) of the algorithm. The robustness of the algorithm is ensured by taking care of some special cases, discussed in the ensuing sections.

Infrared communications

In serial communications, we usually speak of "marks" and "spaces" (**Figure 1**). Frequencies between 30 kHz and 60 kHz are commonly used as carrier frequency signals in remote transmitters. At the receiver side, a space is represented by a high level of the receiver's output, with a mark represented by a low level. Marks and spaces are not, however, the 1s and 0s we want to transmit. The relationship between the marks and spaces and the 1s and 0s depends on the protocol in use.

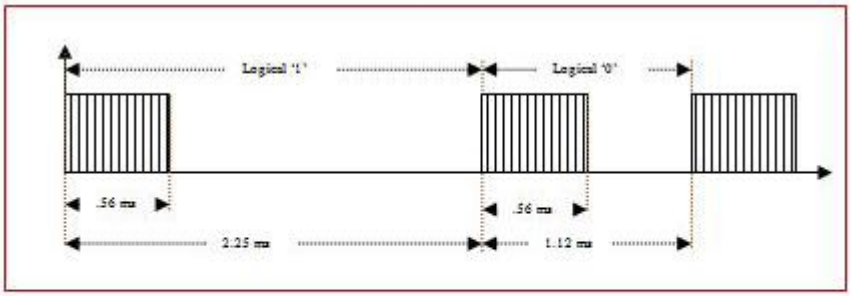

Figure 2. Typical pulse distance-modulated bits on the transmitter side of the NEC protocol differentiate between logic 0 and 1 values.

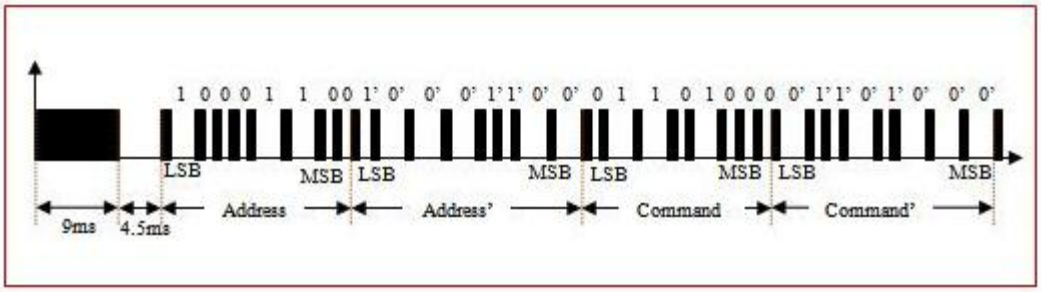

Figure 3. The bit sequence expands to create a typical pulse stream on the transmitter side of the NEC protocol

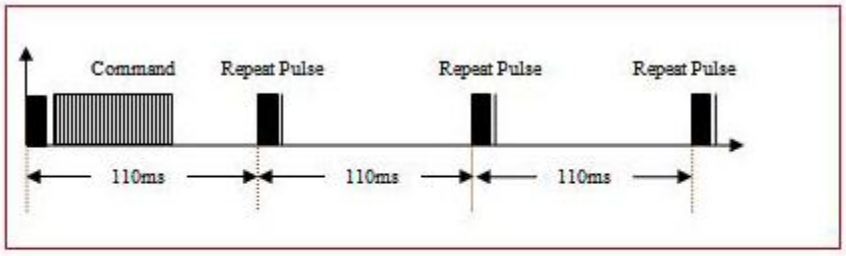

Figure 4. The NEC protocol's repeat pulse capability efficiently comprehends extended remote control button presses.

PDM is a form of serial communication in which each bit of data is signified by the distance between successive pulses. The pulse width remains constant in PDM. The NEC control algorithm is a well-known PDM example (**Figure 2**, **Figure 3**, and **Figure 4**). A typical PDM IR protocol stream consists of the following:

- A start pulse having a specific mark time, followed by a space time
- A stream of address and command bits

- Repeat pulses/commands, which get sent when a key is continually pressed for a prolonged duration; successive repetitions are transmitted with a fixed duration between them

All PDM IR protocols are similar in nature; they differ in the following parameters:

- The number of address and command bits
- The presence, in some protocols, of complemented address and command bits sent along with true values
- The mark and space time of the start pulse and repeat pulse (if the latter exists at all)
- The mark and space time of the address and command bits, thereby defining whether the bit being sent is a logic 1 or 0
- The repeat command sequence or repeat pulses (if they exist at all)
- The duration between repeat pulses (if they exist at all)

In most commercially available remote sensors, the sensor output is active low. A space is represented by a high level of the sensor output, with a mark correspondingly represented by a low level. Rising and falling edges on the transmitter side get interchanged on the receiver side.

UIRDA generalities

An algorithm capable of implementing any PDM protocol has the following characteristics:

1. When the first falling edge arrives, compare for start pulse timings (both mark and space). The start pulse timings vary with different protocols.
2. When a start pulse is detected, calculate the space times (with a high level as active low) of the address and command pulses that follow. These space times indicate if the sent pulse represents a logic 0 or logic 1, and they vary with different IR protocols.
3. If the space time is a value corresponding to logic 0, a logic 0 has been detected. Similarly, if the space time is a value corresponding to logic 1, a logic 1 has been detected.
4. Keep saving bits and shifting in RAM for a stream of X bits, where X is the number of address and command bits to be received. X varies for each PDM IR protocol.
    - If X bits are not received (i.e., if only a partial code is received), then timeout and reset the interrupt flags and counts. Return to LPM (low-power mode) and wait for a fresh start pulse.
    - If an improper bit sequence is received, reset the interrupt flags and counts. Return to LPM and wait for a fresh start pulse.
5. Error-check the received data, in two stages:
    - An XOR test is done for address and command bits.
    - You should also check if the transmitted address matches the device address.

6. Store the decoded data, or send it to the main processor through the connecting link (which may be I^2C, SPI, or another popular low-data-rate interface).
7. Wait for a start or repeat pulse.
8. If a repeat pulse is detected, increment the repeat pulse count and re-send the decoded information.

The timeout is an effective tool for determining if it's appropriate to take the microcontroller out of its decoding state, in cases such as:

- Only the mark time of the start pulse is received.
- Only the start pulse is received, without a further stream of additional pulses.
- Only a partial stream of pulses is received.

These situations can arise due to partial or spurious reception of IR signals, caused by invalid signal sources such as fluorescent lamps and other infrared radiators.

In general, the timeout should be chosen to be greater than the mark time of a start or repeat pulse. However, the following constraints must be kept in mind while deciding the timeout:

- **Timeout for protocols with repeat commands:** The timeout should be made equal to the duration between repeat commands so that the microcontroller does not come out of the decoding phase until the last set of repeat commands is received. As successive sets of command bits are received, the decoded bits are sent to the main processor.
- **Timeout for protocols with repeat pulses:** The timeout should be kept less than the duration between repeat pulses so that the microcontroller comes out of decoding phase before a repeat pulse is received. When a repeat pulse is received, then the previously decoded data is sent again to the main processor.

Implementing the NEC IR protocol

The NEC IR protocol has the following key characteristics:

- 8 address bits and 8 command bits
- Both the address and command bits are complemented and sent after the true values are first transmitted in order to enhance the reliability of communication. Thus, the total number of bits received is 32.
- The start pulse has a mark time of 9 ms and a space time of 4.5 ms.
- Address and command bits have a mark time of 0.56 ms and a space time of 0.56 ms (for a logic 0) or 1.7 ms (for a logic 1).

- The repeat pulse has a mark time of 9 ms and a space time of 2.25 ms.
- The duration between successive repeat pulses is 110 ms.

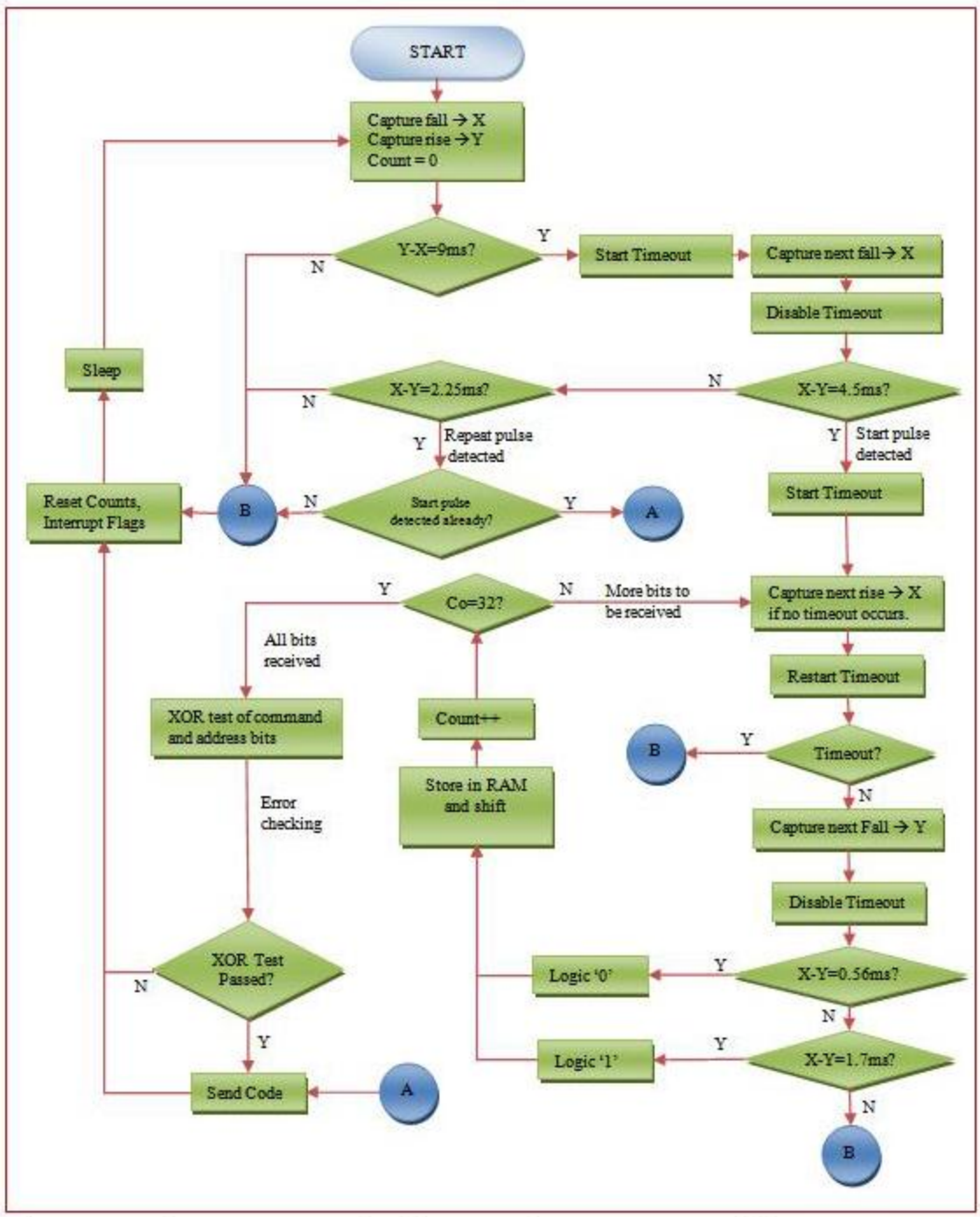

Figure 5. The UIRDA can implement the NEC protocol by comprehending particular parameters and potential error conditions.

A UIRDA implementation of the NEC IR protocol therefore comprises the following steps (**Figure 5**):

1. When the first falling edge arrives, compare both mark and space times for start pulse timings.
2. When a start pulse is detected, calculate the space times (with a high level as active low) of the following address and command pulses.
3. If the space time=0.56 ms, a logic 0 has been detected (a space is logic high on the receiver side). Otherwise, if the space time=1.7 ms, a logic 1 has been detected.
4. Keep saving bits and shifting them into RAM for a total stream length of 32 bits.
    - If 32 bits are not received (i.e., if a partial code is received), then timeout, reset interrupt flags and counts, and return to LPM mode.
    - If an improper bit sequence is received, reset interrupt flags and counts and return to LPM mode.
5. Error checking occurs in two stages:
    - An XOR test is done for address and command bits
    - Also check to see if the transmitted address matches the device address.
6. Send decoded information to the main processor through the connecting link.
7. Wait for a start or repeat pulse.
8. If a repeat pulse is detected, increment the repeat pulse count and resend the previously decoded information.

Generalization of the concept

The UIRDA can be easily modified to decode any of the existing PDM IR protocols by re-defining the various parameters corresponding to the protocol to be decoded. The timeout value, for example, should be suitably chosen for different protocols. For example, here are the UIRDA parameters specific to JVC's IR protocol (**Figure 6**, **Figure 7**, and **Figure 8**):

- 8 address bits and 8 command bits
- The total number of received bits is 16.
- The start pulse has a mark time of 8.4 ms and a space time of 4.2 ms.
- Address and command bits have a mark time of 0.526 ms and a space time of 0.526 ms (logic 0) or 1.574 ms (logic 1).
- Repeat commands are sent after every 50 ms.

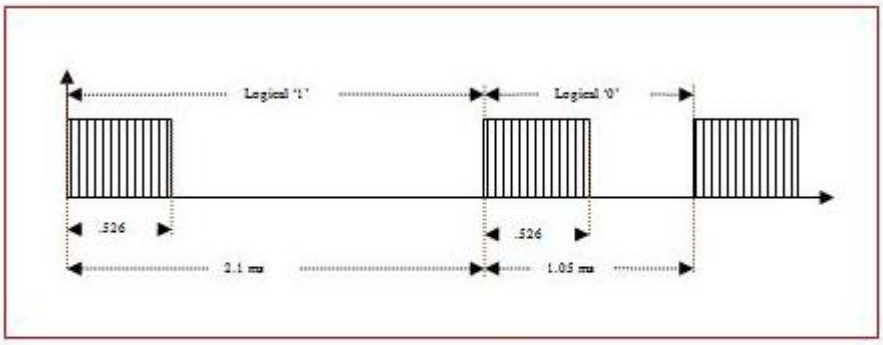

Figure 6. The JVC protocol's modulation parameters differ from those of the NEC algorithm.

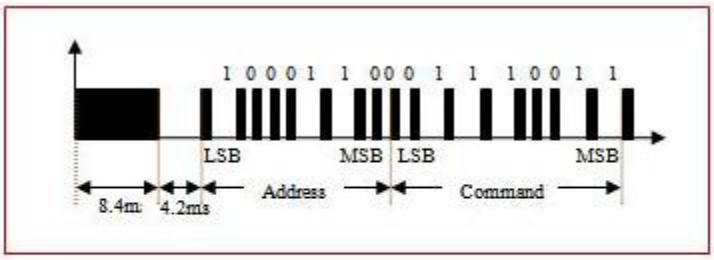

Figure 7. Differentiation between the JVC and earlier NEC approach extends to the JVC pulse train.

The UIRDA can be implemented on virtually any microcontroller, FPGA, or CPLD. Two timers are required for implementing UIRDA in a microcontroller. One timer tracks and captures rising and falling edges. The other implements the timeout feature. The microcontroller is consequently interrupted on every capture edge and at the end of timeout.

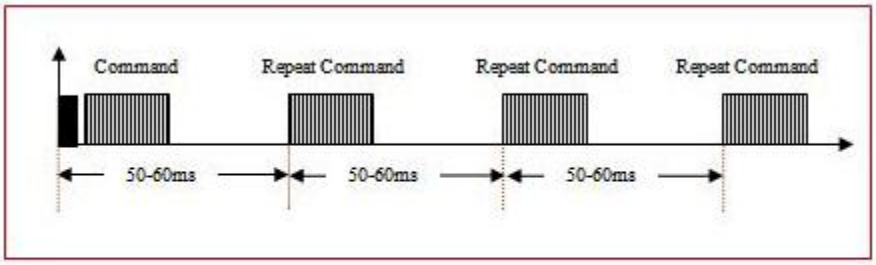

Figure 8. JVC's protocol also comprehends repeat-command capabilities.

My company has to date implemented UIRDA for the NEC protocol, testing it on Texas Instruments' MSP430 microcontroller. In the process of developing support, we have taken care of the following spurious cases:

- *Partial reception:* Timeout takes place as the microcontroller waits for the expected number of bits. The algorithm exits of the decoding phase and again waits for a proper start pulse.
- *Improper codes:* No decoding takes place, since improper bits have been received. The microcontroller comes out of the decoding phase and waits for a proper start pulse.
- *A repeat pulse or repeat command:* As explained earlier in this article, the timeout feature takes care of this case.