Keywords: MAXQ,remote,IR

APPLICATION NOTE 4017

How to Create a MAXQ®-Based "Learning" Remote Control

By: Ben Smith
May 19, 2008

Abstract: The clutter of infrared (IR) remote controls for consumer entertainment equipment seems to be a permanent fixture in homes around the world. But with a MAXQ2000 microcontroller and a few inexpensive parts, you can build a remote control that "learns" the codes from the other remotes and plays back the codes on demand.

Overview

The simple infrared (IR) remote control has secured an exalted place in many households. It is easy to see why. With the remote control one can command the various pieces of entertainment equipment commonly found in the modern home. You summon programming from all corners of the world, listen to music from multiple sources, play audio and video media, and even save programming for later viewing, if desired—all from the comfort of your favorite room.

Yet the ubiquity of the IR remote has evolved into a problem. Rare is the home that does not have three, four, or more of these little devices cluttering the table. There is one for the television, one for the DVR, another for the VCR, yet another for the audio system, and this list does not include the cable box or satellite receiver, distribution switch, DVD recorder, or any number of other devices that command our attention.

A solution to this mix of remotes is the *learning remote* that learns the codes from another remote control. The learning remote is placed in "learn" mode, and a second remote "teaches" the learning remote how to transmit a command, volume up, for example. From then on, the learning remote can send the learned command whenever the appropriate button is pushed.

This application note demonstrates how such a learning remote can be built around a MAXQ2000 microcontroller, one of Maxim's MAXQ RISC microcontrollers. But before the design details, some background is needed.
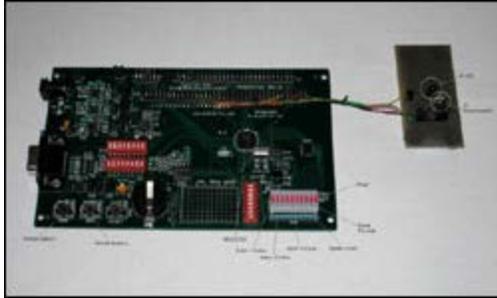
More detailed image (PDF, 388kB)

Controlling with Light

The first television remote control was the Zenith® Space Commander. It used a mechanical arrangement that generated an ultrasonic tone at a specific frequency when activated. Think of a tuning fork: when tapped, it vibrates at a fixed, predictable frequency. The sound waves emitted by the tuning fork could, in theory, be received and interpreted as a command to do something. Because that early remote was completely mechanical, it required no batteries. It had, however, only three commands: television on and off, channel up, and channel down.

As semiconductor devices became less expensive and more popular, IR light supplanted ultrasonic sound as the control mechanism of choice. At its simplest, an IR remote-control system consists of a handheld unit that transmits a modulated infrared beam, and a base unit that receives the modulated IR and interprets the modulation, most often as a command to take some action. The details of this operation, however, are not nearly so simple due to a combination of engineering feasibility, physical reality, and the needs of the market.

The hard truth is that the environment is full of IR radiation, so expecting any trivial modulation scheme to be heard above the cacophony of radiant energy is unrealistic. Everything that emits heat also emits IR radiation. Incandescent light bulbs, for example, emit more IR than visible light. Even the human body emits IR radiation. Because of this, most IR remote transmitters modulate the light with a low-frequency carrier (usually between 28kHz and 60kHz) before applying the data.

Modulating the light beam with a fixed frequency makes it easy to detect from among all the interfering IR radiation in a typical household environment. By using a simple bandpass filter, the IR signal can be isolated and interpreted. There are a variety of inexpensive integrated circuits that include both an IR photodiode and a bandpass filter just for this purpose.

Generating such a modulated light beam is simple. Infrared-emitting LED devices are common and inexpensive; creating the modulated light beam can be as simple as driving an IR LED from an appropriate oscillator. See **Figure 1** for circuits that can modulate and receive IR-modulated data.
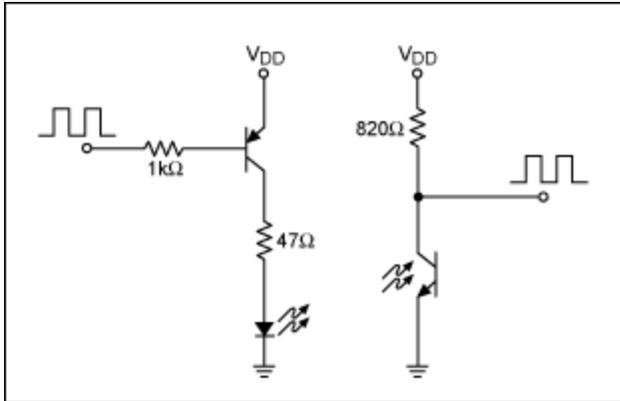
Figure 1. An electrical signal is converted into a modulated IR beam and back to an electrical signal. To extend the range of the transmitter, a PNP driver is used. Component values can be adjusted for the chosen IR LED.

With the modulated LED light source and a receiver IC operating at the same frequency as the source, one has the beginnings of a remote-control system. When the LED circuit is operating and in the receiver's range, the receiver output becomes active. If that action is all that is required (simple on-off control of an external circuit), the task would be complete.

But on-off control is not enough. Even simple remote controls can send a variety of different commands, such as volume up and down, channel selection, input source selection, and perhaps individual digits. For this reason, something else is needed, a way to further "modulate" the modulated light beam. This is where the story becomes interesting.

As IR remote controls were coming into wide acceptance, each manufacturer took their own path to modulate the light beam. While all used digital control (that is, the characteristics of the modulated beam represented a digital "1" or "0" bit), the specifics varied wildly. Some used simple nonreturn to zero (NRZ) modulation. Others used a form of pulse-width modulation (PWM) so that a long pulse represented one state and a short pulse represented the alternate state. Still others used a form of biphase modulation in which an on-to-off transition represented one state and an off-to-on transition represented the other state. This confusing situation persists to this day, and makes creating a universal remote control that will operate equipment from any manufacturer a genuine challenge.

Design Variables

There are three variables that must be accounted for when designing a universal learning remote: the carrier frequency, the bit format, and the frame format.

Carrier Frequency

Carrier frequency is the frequency at which the light is modulated. It has nothing to do with the actual bit rate. This is a constant frequency for any given system, and ranges between about 28kHz and 60kHz, but most often operates between 36kHz and 38kHz.

Bit Format

Bit format is the way the system distinguishes between a "1" and a "0" bit, and it can vary widely from one manufacturer to another. In some cases the width of the "light on" period is the determining factor. Some systems made by Sony® use a "light on" pulse of 1,100µs to indicate a "1" bit, while a "light on" pulse of 550µs indicates a "0" bit. The space between pulses is always 550µs.

Another bit format keeps the "light-on" pulses fixed, but varies the space between the constant-width pulses. Some Matsushita (Panasonic®) systems emit a constant stream of 800µs pulses, but designate a "1" bit as a 2,400µs space between pulses, and a "0" bit as an 800µs space between pulses.

One of the most commonly used code systems is the Philips RC-5 code. In this code, each bit cell consists of an 889µs burst of modulated light and an 889µs space. A bit cell is considered to represent a "1" if it consists of a "light on" period followed by a "light off" period; it is considered to represent a "0" if it consists of a "light off" period followed by a "light on" period. In the RC-5 system, bit synchronization is assured by enforcing two "1" bits at the beginning of every frame. **Figure 2** illustrates various bit formats.
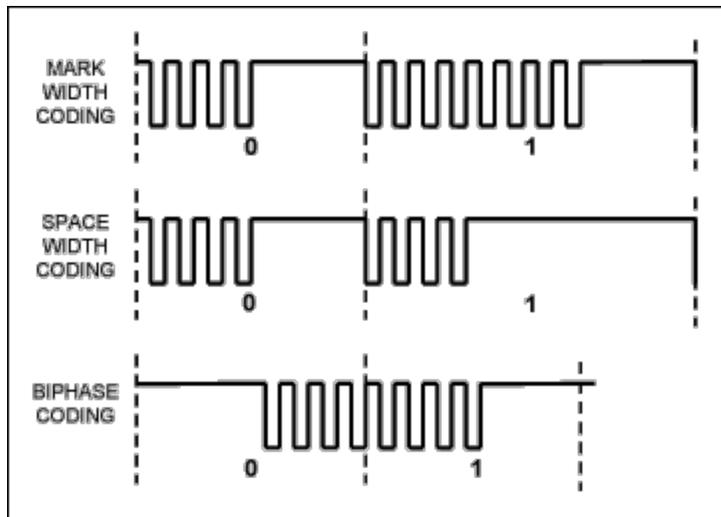

Figure 2. Several bit formats can be used in IR remote-control systems, but all involve modulating the light beam with a carrier, then modulating the carrier using one of several techniques.

Frame Format

Once the bit format has been decided, the designer must determine the frame format. In many cases, this will consist of a synchronization pulse (typically a pulse somewhat longer than ordinary data pulses), followed by data bits in a particular format. Usually the data will consist of two parts: a "data" part that conveys the desired function, and a "custom" part that corresponds to the piece of equipment to be controlled. Thus, a certain data item for one device can correspond to something different for another device.

Some codes transmit information twice per frame: once in normal mode and once with the bit sense reversed. In this way, a level of rudimentary error checking is provided. If the two copies do not match, the command is assumed to be invalid.

After a frame is transmitted, it is typically repeated over and over. Common frame-repetition rates range from about 10 to 20 frames per second. A few protocols transmit the codes only once, followed by a repeating "key down" code. These protocols will not be handled by the system described in this article, since it is assumed that every repeating frame contains the data and custom codes.

Finally, some protocols, including the RC-5 protocol, invert a bit at each keypress. This helps recognize whether a gap in reception is due to a signal loss, for example, someone walking between the remote control and the base unit, or whether it actually represents a second keypress. This feature is not implemented in this project.

Seeking Universality

Based on the above discussion, it would appear that a universal learning remote needs to know each of these bit formats to accomplish its task. If we were concerned about the size of the resulting data set, this would be true: a typical IR remote control message is only a few dozen bits long. But given that memory is relatively inexpensive, we can simply sample the incoming bit stream and record the samples.

Thus for this project, *we really do not care about the bit format or frame format*. This is because this system simply records and plays back whatever it sees. By remaining agnostic to the system in use, true universality is guaranteed.

Receiving and Recording

The receiver circuit is simplicity itself. A phototransistor pulled up to V_{DD} forms the input circuit, and can be connected directly to an input pin on the MAXQ2000 microcontroller. No special receiver IC is needed or is, in fact, desirable. We are not concerned about range, but rather wish to record the actual modulation envelope regardless of the carrier frequency.

Operating the phototransistor in a saturating mode presents a small problem. Phototransistors are not particularly fast devices; they have a recovery time from the completely-on to the completely-off state that is greater than the bit time of most systems. Thus, if there is too much optical signal, the phototransistor will saturate. It will miss the carrier frequency completely and only follow the outline of the modulating waveform. But if there is too little signal, there will be no recognizable waveform at all. **Figure 3** illustrates these conditions.
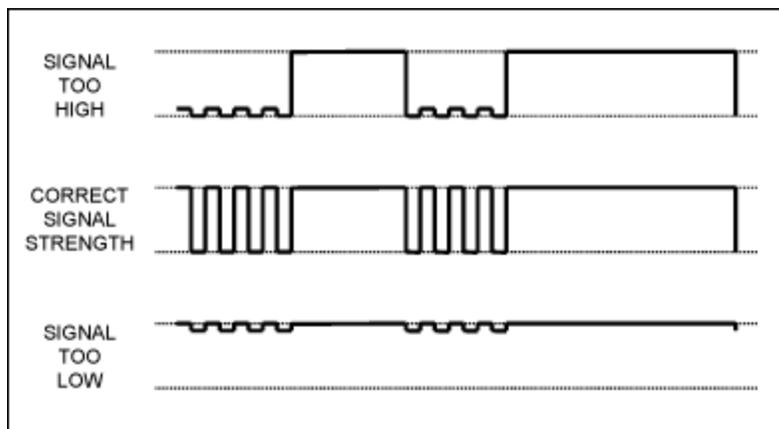

Figure 3. When receiving the IR signal, the signal strength must be correct. If the signal strength is too high, the phototransistor will saturate and only the low-frequency portion of the signal will be detected. If the signal strength is too low, the carrier frequency will never cross the detection threshold.

It is thus crucial to place the master remote and the learning remote at exactly the right distance apart. But what is that distance? To determine this, the software in the learning remote performs some presampling to determine if the distance is correct. Before it begins recording, the learning remote (i.e., the MAXQ2000 microcontroller) samples the signal for transitions on the input channel. If there are none, the remote assumes that the signal level is too low and lights an appropriate LED. If, however, the microcontroller remote sees transitions on the input channel but no pulses of 100µs or less (carrier frequencies can be assumed to be greater than 10kHz), it assumes that the signal level is too high and lights a different LED. Finally, if the microcontroller observes bursts of fast pulses separated by "off"

periods, it assumes that the signal level is ideal, in the "sweet spot." The code for the learning remote then transitions to the recording state.

Several things must happen in the recording state. The microcontroller remote must determine the carrier's incoming frequency. Since the microcontroller is operating at 16MHz and the frequency of the carrier is (at most) 60kHz, the carrier frequency can be measured precisely. Four samples, from trailing edge to trailing edge, are accumulated. Then the result is divided by eight to determine the high- and low-period times.

Next the receiver begins searching for a transmission gap greater than 10ms. Every protocol places a gap between repeated transmissions of the same code, and no protocol allows a gap within a single code of greater than about 1ms. When the gap is found, the receiver knows that this is the beginning of a code sequence. Recording can begin.

To record the code, the microcontroller remote accumulates the time during which the carrier is on. When it observes loss of carrier, the remote accumulates the time in which the carrier remains off. This results in a vector of on-and-off times that can be used to recreate the signal when needed.

Since this is a demonstration project and not a finished product, these vectors of on-and-off times are stored in volatile RAM. In an actual product, a software subroutine would likely copy these vectors into a nonvolatile store (such as an EEPROM).

Playback

After the buttons have been programmed, the CPU enters sleep mode. In this mode, registers and RAM are maintained, but the CPU clock is stopped. Only an interrupt (or reset) can wake the CPU.

When a button is pressed, the CPU is awakened and scans its input pins to determine which button was pressed. It then points to the vector in RAM that contains instructions on how to play the code associated with the button.

The RAM vector consists of: a header that includes the count of the number of on-off cycles; a value that represents the carrier frequency; and a sequence of value pairs that represent the on-time and off-time for each on-off cycle. The first header value, the number of on-off cycles, is stored in a loop-counter register (LC1). Keeping this value in a counter register makes it easy to loop through all cycle values.

The second value in the header, the carrier period, is scaled and stored away. During the IR on times, this value is loaded into another loop-counter register (LC0). Since the MAXQ2000 is a single-cycle core, timings through program loops are completely dependable. Thus, the carrier can be generated by executing through a four-instruction cycle loop for the high period, then executing through a four-instruction cycle loop for the low period. Execution continues this way, executing the on-period loop and then the off-period loop.

Here is where the execution path would stay, generating the carrier by turning the IR LED on for a period and then off for a period, were it not for a timer. For each half-bit time, the vector contains a value to write to one of the MAXQ2000 timer channels. The timer is operated in divide-by-32 mode, so that the timer resolution is about 2µs.

At the beginning of each half-bit period, the timer is loaded with the duration of that period. Then, while program code is turning the IR LED on and off (during an on period) or simply leaving the IR LED off (during an off period), the timer is frequently tested to determine if the bit time has expired.

After the half-bit time has expired, the loop-counter register (LC1) is decremented and tested for zero. If it is not zero, then there are more bits to transmit and a branch is taken to the top of the loop. Otherwise, the button is tested to see if it is still pressed. If the button is still active, the entire cycle (reading initialization values from the vector and reinitializing counters) begins again. Otherwise, the CPU is put back to sleep until the next button is pushed.

Enhancing the Basic Remote Operation

To this point we have a working learning remote, but one with only two buttons. The most obvious enhancement to this design would be more buttons. Adding those buttons is a straightforward process that involves only a small amount of additional hardware. When the CPU is idle, all the row drivers are set to output a "1" state. Soft (that is, high resistance) pulldowns on each of the columns keep those inputs low in the idle state. When the user presses any button, the corresponding column is driven high and the CPU is awakened (**Figure 4**). The CPU can then set each row high, one row at a time, and determine the row and column in which the button resides.
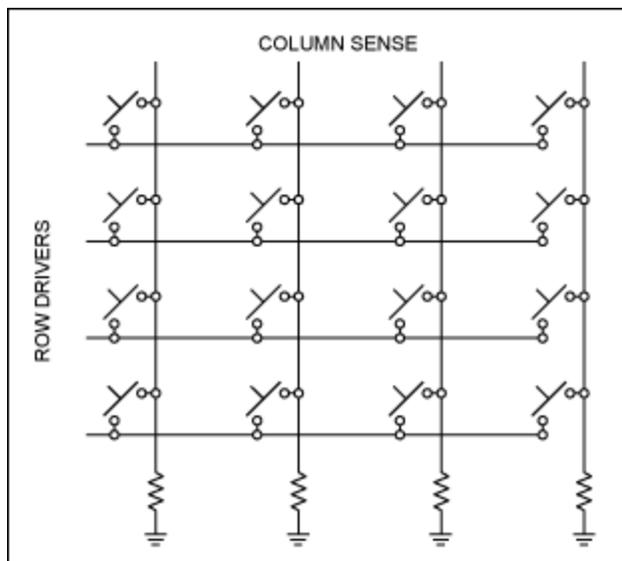

Figure 4. To augment the project with additional keys, activation of any key must interrupt the processor. During sleep mode, all column lines are kept low by soft pulldown resistors, and all row lines are driven high. When any key is pressed, the column line is pulled high, thus waking the processor and initiating the row scan process.

A second enhancement would use the time-of-day clock peripheral in the MAXQ2000 to awaken the CPU to perform a programmed sequence of IR commands at a specific time. The time-of-day clock is a low-power peripheral designed to operate for a long time on battery power. While the MAXQ2000 is in sleep mode with its high-frequency clock stopped, the time-of-day clock continues to operate. The clock can generate an interrupt to wake the CPU based on time of day or time interval. The time-of-day clock could, for example, wake the CPU so that the remote could send a command to a cable or satellite box, then a command to a VCR or PVR to begin recording a program. At the end of the program, the CPU could once again be awakened to end recording.

A third possibility is to connect the MAXQ2000 to a personal computer. In this way, the PC could be used as a programming station, potentially pulling programming information from the web and automatically loading it into the universal remote.

So, with only a few external components and a little software, the sophisticated MAXQ2000 microcontroller becomes the heart of a universal learning remote.

A similar article appeared in the December 2007 issue of *Circuit Cellar*.

MAXQ is a registered trademark of Maxim Integrated Products, Inc.
Panasonic is a registered trademark and registered service mark of Panasonic Corporation.
Sony is a registered trademark and registered service mark of Kabushiki Kaisha TA Sony Corporation.
Zenith is a registered trademark of Zenith Electronics Corporation.

Related Parts		
MAXQ2000	Low-Power LCD Microcontroller	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 4017: http://www.maximintegrated.com/an4017
APPLICATION NOTE 4017, AN4017, AN 4017, APP4017, Appnote4017, Appnote 4017
Copyright © by Maxim Integrated Products
Additional Legal Notices: http://www.maximintegrated.com/legal