

I'm not a robot



[illegible]

ways they were coded, so, for matching receiver and transmitter, repeat this connection for receiver. Now it's obvious why it called fixed-code; so if we want to use another remote, it's simple: just copy the transmitter's code pattern; but it is not a secure method because if anyone who find your code can make a compatible remote with your transmitter. But learning-code remotes have more security. It means every remote has a 20 bits written random code in its memory. This code is unique for every remote, so for syncing, receiver has to save every remote's code and for every request check if it's a valid remote or not. Receiver learns the code! Fortunately, both HS1527 and EV1527 (or similar families) use the same protocol and OTP encoder. In this method we have 24 bits of data consist of 20 bits that is remote's code and 4 bits for the remote's buttons statuses. First we have a Preamble situation then wait for 24 bits of data; so we need 3 unique situations: Preamble: in this situation if we have 1 for one μ s, must have 0 for 30 μ s. Logical 1: in this situation if we have 1 for 3 μ s, must have 0 for one μ s. Logical 0: in this situation if we have 1 for one μ s, must have 0 for 3 μ s. Note that the times mentioned are for example and depends on inner oscillator; but ratios are true. Remote program topology I have been seen plenty of related codes both free or not; none of them uses principled method, some of them need a specific oscillator's resistor (because only the value of zeros was checked), some didn't use a good board, or remote didn't work properly because of low battery level and made lots of problem. After that you became familiar with remote function, now it's time to write a program that recognizes receiving data and reacts to it depends on available features and options. The best solution for decoding data: at a clock's positive edge, a counter start to count and at clock's negative edge read counter's value and set it zero, at the next positive edge compare previous and current counter's value, that should be one of the above situations; pay attention that you follow the remote pattern of receiving data (first Preamble then 24 bits of valid data) because, otherwise, noise can affect the remote at random. The important question is about counting rate(frequency) for accuracy of measuring pulse width; to answer this question take a look at datasheet of IC; we face this table: This is necessary time for transferring one frame (24 bits plus Preamble) of data depends on voltages and oscillator's resistor. The shortest time is 1.34 milliseconds for transferring a frame. Every frame contains 126 pulses, so, every pulse will be 10 microseconds; if counter counts with 1 MHz frequency (1 microsecond), we are able to measure a pulse width with a good accuracy. Explain the existing source There's two versions of source code (for ATmega8 microcontroller): Code Vision and GCC. Receiver's data output connected to IC's pin number 4 that interrupt's input pin is zero (for detecting edges) and for measuring every pulse width using timer 0. This program is portable that you can use any microcontroller and using external oscillator with 8 MHz frequency. This code was tested with 3 type of remote controller, both 315 and 433 frequencies. Testing with remote-433 worked about 8 meters without receiving antenna! Download Source [download id="23812"] [download id="23815"] Author: Zeus Translate: Golnoosh