Embedded Systems Course 0368-4166, Tel Aviv University, Fall 2013/14
Gil Freidlin (gilfreidlin at gmail dot com), Ido Livneh (idolivneh at gmail dot com)
Under the supervision of Prof. Sivan Toledo

Copy2GO: Low cost copy lab for simple remote controls using TI-Chronos platform

Abstract

We present a simple and affordable way of copying remote controls widely used for parking lot gates, garage doors and other simple systems. These simple remote controls usually use a *fixed code* (as opposed to the more secured *rolling code* used for car keys remote controls) and a simple On-Off Keying (OOK) modulation, over 433.92MHz in the ISM band. We suggest the use of the TI-Chronos wrist-watch platform for the emulation of the remote control, as this platform transmits in the same band, and can be programmed to emulate different modulations and to send user pre-defined signals.

In this report we show the complete process for copying a remote control into the Chronos platform. This process utilizes only a standard PC and low-cost hardware (less than $75 all together), alongside free software, and additional software developed by us. The process starts with recording the original remote control RF signal. It continues with automatic analysis of the recording, extracting the needed parameters of the signal. Finishing the process, we set the Chronos with those parameters. We demonstrate the copy process using a 4-channel remote control and its receiver board.

Table of contents

Introduction

Our project will show how easy and cheap it is to create not only working remote control copies, but cool ones as well! We will show how to utilize the TI eZ430-Chronos watch platform as a remote control. We will record and analyze the RF signal produced by the original remote, and then teach the watch to reproduce the same signal and open the gate with it instead. The hardware in our project costs less than $75, and that includes TI's watch itself!

During this project we:

1. Surveyed remote control systems for common features and characteristics. We then purchased a remote control and receiver kit to serve as a candidate for copying and reimplementation using the Chronos wrist watch.
2. Altered the C code for the MCU on the watch platform to support general purpose RF signaling using the platform's transceiver.
3. Upgraded a python script that is used to configure the watch with the signal parameters.
4. Recorded and analyzed the structure of the RF signal generated by the original remote control.
5. Developed a python script for robust automatic analysis of the signal recordings to extract the parameters for the watch configuration.
6. Integrated our copying system and tested our process to achieve the desired results.

This completes a full cycle of the remote control copying and embedding into the Chronos watch, to replace the original remote. Our system supports two different remote control codes in one watch system.

If you wish to build your own copy lab - jump straight to appendix A to get the simple and full instructions. The report ahead will give a thorough explanation of the elements of the copy lab, and the behavior of the remote control system.

System Flow Diagram

Phase1: Copy

Phase2: Transmit Signal

TI Chronos platform review

Texas Instruments eZ430 Chronos, is a low power MCU, sensors and RF transceiver development kit, that has an additional coolness factor as it is embedded in a sleek hand-wrist watch. The dev-kit arrives with the watch board, it's mechanical watch casing, and two USB dongles to work with it. The first dongle serves as an RF access point, and the second dongle is used to physically connect the watch board to the computer for debugging and program loading. The dev-kit's wiki is very helpful.

Figure 1: The eZ430 Chronos Kit

The board's MCU is the TI MSP430, which is an ultra-low power microcontroller SoC with the RF transceiver (based on TI's CC1101), for the <1GHz band, integrated in to it, along with many other useful interfaces. The board also introduces a 3-axis accelerometer, pressure sensor, temperature sensor, and a battery sensor - all of which we did not use in our project, but we did play with. To us, it seems that TI is using the Chronos platform and its coolness factor to introduce this MCU into the industry for companies to use for different products.

There are several versions of the watch's hardware. There is an old (black) PCB and a new (white) PCB. We worked with black PCB watches, which gave us some trouble, as you will see in a short while. Furthermore, there are 3 different models for 3 different frequency ranges. These differ by antennae matching, and firmware. The available frequency bands are 433MHz, 868MHz and 915MHz, which comply with different regulations worldwide. Gate and garage remote controls works on several sub-GHz bands. The 433.92MHz frequency is quite common and fits to the 433Mhz Chronos model. This also caused us some trouble, since the first 3 watches we got a hold of were matched to 868MHz. We identified the version of the watch, and the frequency by observation, silk prints on the PCB, and the schematics available in the Chronos user guide (Section 4.5).

The access point dongle should also be 433MHz matched (it looks like the one in the following picture, and not the last one). Through the access point dongle, we can use or build a software on the PC to communicate with the watch. We just need to connect through the serial port that the device emulates, and send packets to the watch.

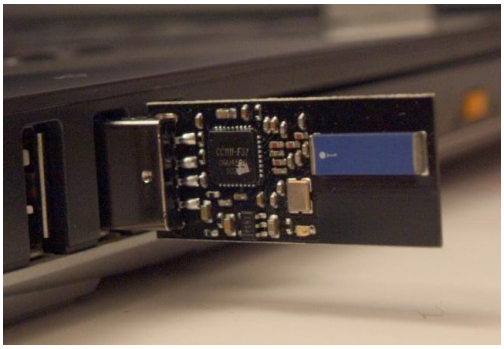


Figure 2: Wireless access point dongle

TI's platform embeds a low-power network protocol called "SimpliciTI". This protocol stack is simple and helpful, and we used it as-is to communicate between the watch and our PC.

Firmware and software

The watch's development kit user guide recommends the installation of the Code Composer Studio IDE, which is also an IT product, and is based on the open source IDE Eclipse. The development kit arrives with the full open source code for two projects - A sports watch and data logger. The project on which our software is based on is the sports watch project. Using the IDE you can change the code, compile it, and upload it to the watch platform. When the watch is connected through the debug dongle, it is also possible to debug the code, line-by-line, in the IDE while it runs on the platform itself.

ChronIC review and our adjustments

ChronIC is a software package, written by Adam Laurie, that enables the TI Chronos watch to send arbitrary RF data and signals with different modulations. As we will show later in the report, the signal we want to transmit is OOK-modulated, and the ChronIC project supports it. In our project we've used this software, implemented several changes into it, and fixed several bugs.

The main difference between the original sports watch project, and ChronIC project are:

1. ChronIC has deprecated the use of several features in the original sports watch, among which are measurement of temperature, altitude, acceleration, speed and heartrate, sending acceleration information to PC control center, controlling computer with chronos' buttons,
2. ChronIC disabled the use of TI's BlueRobin radio protocol stack, used for low power communication between devices (such as heart rate monitor).
3. ChronIC adds UP/DOWN buttons functionality to transmit the configured RF signal when time/date are showing on the screen.
4. ChronIC adds functionality to the RF Sync operation, allowing us to configure the RF signal that will be transmitted from the watch, from the computer via the access point.
5. ChronIC adds the actual transmitting functionality (most of it is in \logic\chronic.c) by configuration of SFRs.

There are two main problems with the ChronIC version that we've found online. The first problem was that it did not match the hardware version of our watch. As we mentioned earlier, we are working with the black-PCB version of the watch, and it seems as though the ChronIC code version we have is not in compliance with the new version of hardware. To fix this, we had to manually and effectively merge the ChronIC code with a newer version of the original sports watch project. We had to keep the right changes that ChronIC did to the code, and add the changes in infrastructure that TI introduced between the two versions of the original code. There weren't a lot of changes that were needed, but in a large code base, this task isn't trivial: it took us several tries until we got it right. This merge mainly includes new low level drivers to account for newer hardware, new initialization processes, as well as different RF settings between the two board types.

The second problem we've encountered was that the code contained some bugs. We've fixed some of the bugs we found, but we guess that we can find some more. It seems that the original coder had some trouble with the watchdog functionality of the watch, so he disabled it. We've turned it back on, and fixed the main bug that was in conflict with the watchdog functionality. That stopped most of the watch's freezes we've encountered. The problem with these freezes is that you have to open the watch and remove the battery in order to reset it to work again, and this is annoying. The main bug we've fixed was a problem encountered while trying to configure the code to be sent when the user presses the UP/DOWN buttons. The problematic variable is called Button_Up(Down)_Data in the code. The problem was that when resetting the watch, if you did not send a long enough code, the data would not be set, and then when trying to transmit the data, the watch would freeze. Since the watchdog was off, it would mean the watch won't work until we take out the battery, and put it back in. We fixed it by adding a missing initialization to the data just before the code tries to configure the buttons' data.

The list of changes we've made:

1. \include\project.h - Enabled the watchdog configuration.
2. \logic\rfsimpliciti.c - Added the missing initialization in case the sent code is short enough to be sent in one chunk. The change appears in both setting the UP button as well as in setting the DOWN button.
3. \logic\test.c - removed two unused variables left in the code.

Our final code for the project is in:

1. Chronos Final - The final working code after the bug fixes.

For future projects and research to be done on this code base we also give the other 3 other code bases here:

2. Sports Watch - The original sports watch project (that supports the black PCB version).
3. ChronIC - The original ChronIC project.
4. ChronIC_SportsWatch_Merge - The code merge between the above two projects.

We recommend working with Beyond Compare tool to easily find the differences between the four projects.

Remote control and gate controller overview

In order to help develop our copy lab, we needed a remote control and receiver system. We started our learning process online, and searched for general information and common parameters of these systems. The main issue we were worried about is the compatibility of the eZ430 Chronos watch for this mission. We needed to understand at what frequencies these systems work in.

There is a huge variety of systems and manufacturers for electrical gates controlled by remote controls and even more for the remote controls themselves. Typical remote control gate systems contain a receiver that controls an electrical motor through a mechanical relay that switches on and off the current to the motor. The remote control unit is a cheap low-power transmitter operating in a sub-GHz ISM frequency band and mostly uses a simple On-Off Keying modulation to transmit some repeated codes. Some known and common frequencies include the 433.92MHz, 310-315MHz and 403.55Mhz (the 868.3 MHz[1], and 916Mhz related frequencies were mentioned for rolling code remote controls).

There are two main types of encoding: *Fixed Code*, and a more modern *Rolling Code* as summarized here.[2] In this project we decided to focus on the simpler Fixed Code remotes only, in contrast to Rolling Code remotes that are much more secure and cannot simply be copied. Rolling code is used for systems such as remote car-keys or garage gates where the usage is one remote control vs. one receiver. However, for typical public gate that interacts with many remotes, rolling code scheme is not suitable and hence fixed code is still being used.

This answers the above first issue: we can use the 433MHz model of the Chronos watch to open common remote-controlled gates, since the frequencies align, and the OOK modulation is supported by the ChronIC software.

The problem remains is that because this remote control system is so simple to implement and no strict standard exists for the coding scheme, every manufacturer can have a different coding scheme. Thus, our solution must be robust.

Fixed code encoders typically use between 8 to 12 bits to set the code to be transmitted. This allows different systems to be set one next to another without affecting each other. The system usually comes with default values set in both the remote control and the receiver. Changing the fixed code in both the remote control and the receiver, involves changing the state of a dip switch, or equivalent soldering pads.

With 12 bits we have 2^{12} = 4096 different codes, which provides good chances that two near systems will not use the same code, and avoid collisions. If there is a 3-state choice for each switch, as happens to be in our remote control system, with 8 switch-code, there are 3^8 =6561 options, that is more than enough for the mentioned reasons.

As we mentioned earlier, we need our solution to be robust, but we started with one example - we purchased a remote control and receiver kit, and analyzed its structure to understand its coding

[1] http://en.wikipedia.org/wiki/Universal_remote_control_duplicator

[2] There is a 3rd type mentioned in the reference called *Learning code*, which is just a type used by encoders which can learn and duplicate another fixed code remote, and not a different encoding scheme.

scheme. We purchased[3] at the cost of 200NIS a proprietary kit that includes a remote control transmitter and a matching receiver. It is a 4-channels system, with 433.92MHz carrier frequency (17cm antenna) and fixed code operation, which includes:

1. One 4-channel remote control transmitter unit having 4 buttons, one for every channel.
2. A 4-channels board having matching receiver unit and 4 electrical relays controlled by each channel. There's a red LED for each channel to indicate when the relevant channel relay is activated – this was very helpful during our testing. This board needs a 12VDC input to work.

Although the lack of official documentation the board is easy to understand, and works well.

Figure 3: The receiver/transmiter Kit we used

By looking at the silk print on the boards and understanding the electronic design, we found that the transmitter unit uses a PT-2260 encoder and the receiver uses the matching PT-2272 decoder. Note that as we later analyzed the code from the RF signal recording, we didn't really need to reverse engineer the boards. However, as a preliminary stage we did look into it in order to understand what we should expect to see.

The encoder/decoder pair should be set to the same code for the system to work. As it has 4-channels it probably transmits 4 different codes. In this fixed code encoder/decoder pair a tri-state 8-switch code can be set by soldering (see images), having 3 possible states (as in the datasheets) for each of the legs: 0 (to GND), 1 (to V) and a middle state. As will be explained in the Signal Analysis part, we changed the configuration of these switches, to understand how they affect the signal.

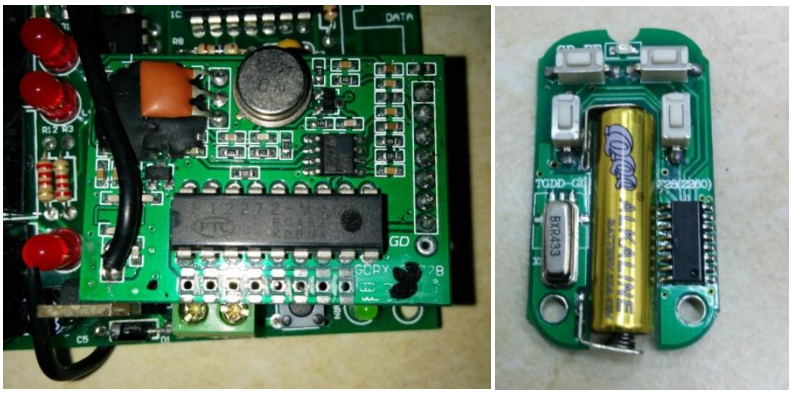


Figure 4: Detailed View on the recievers' decoder and remote controls' encoder

[3] We bought it at "beltzer electronic" shop in 17 har-tzion st. TA. We originally meant to buy this 1-channel receiver/transmitter kit, but didn't because it was verified in the store as using a rolling code (opposed to earlier checks). Instead, we decided to buy the above describe 4-channels system which works great although came with very poor (one page) documentation. We had some luck with that, since we finally utilized more than one code to use in the watch.

Signal Analysis

Signal description

In order to allow the copying of the remote control signal, we need to analyze it's parameters, and how it works. This analysis would allow us to understand how these signals generally works, and build a tool for robust automatic analysis of these signals.

In order to record the signal we used a professional Software Radio RF transceiver called *USRP N200 Ettus Research* located in Prof. Sivan Toledo's lab. The carrier frequency was set to 433.92MHz with sample rate of 1MSamples/s. the samples are saved with the I and Q data interleaved where each sample is a 16 bit integer. For convenience, we chose the record length to be 10 sec. (which was more than enough).

In order to analyze the signals and view the differences between the channels codes, we have recorded each of the 4-channels. Recording was done by starting the USRP operation, then continuously pressing the button of a specific channel for a about a second or two (to avoid bouncing errors), then release it before the USRP recording ends. To validate that the signal was transmitted, the receiver board was on, and the LEDs were our indication.

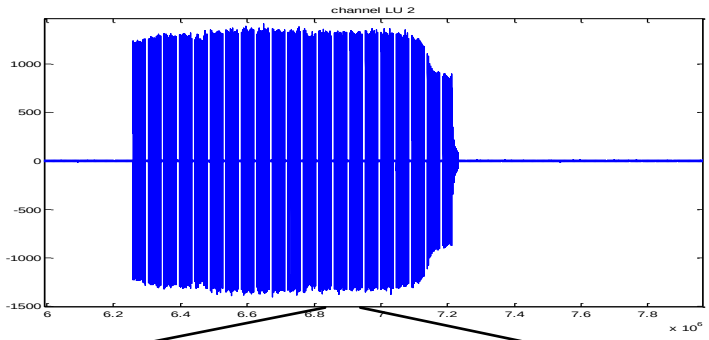
For convenience we named the different channels as they appear on the remote control unit, Left Up (LU), Left Down (LD), Right Up (RU) and Right Down (RD). From this point on, we refer them as with their abbreviation.

At this stage the saved recorded signals were manually observed and analyzed through MATLAB environment using simple scripts and by some manual measurements.

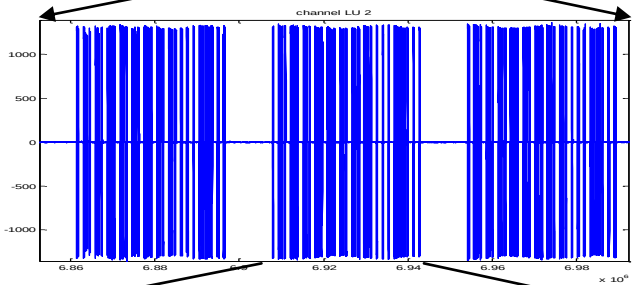
Our recordings showed that the signal generated by the remote control was sometimes noisy and not so stable, and its amplitude varied. Even though, the receiver decoded the signal correctly, and this is due to the simple modulation scheme – which in turn allows very simple and low-cost implementation of the electronics in the system.

Detailed description

When the transmitter is active (button is pressed) the signal shows repeated bursts or packets. Each packet is a sequence shaped according to the current code and channel. Examining finer time resolution revealed the code structure of one duration from the transmitted signal, that as expected consists from a sequence of OOK signals representing the code content. This is demonstrated using the LU signal:

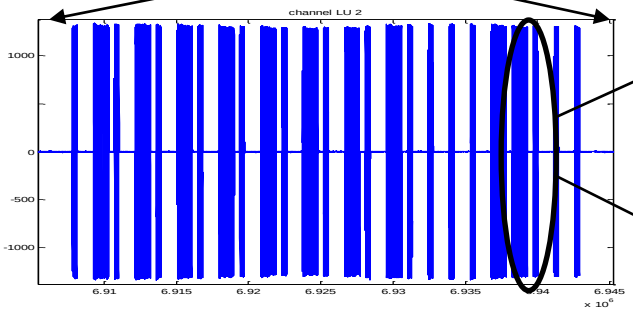


The entire transmission. 21 packets are observable, and its length is about 1MSamples, which means its of the length of 1 second.

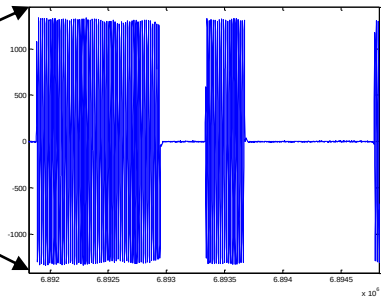


A zoom-in on 3 packets

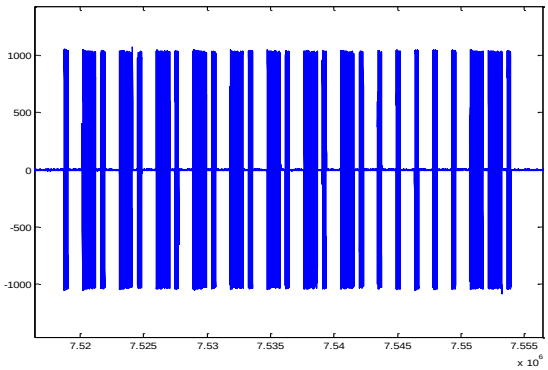
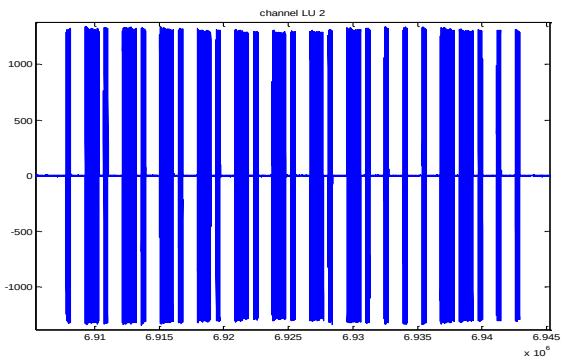
A single packet

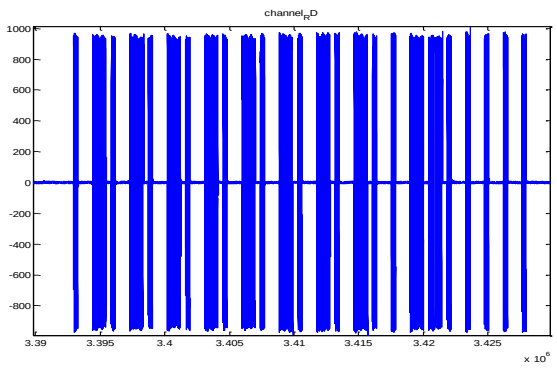
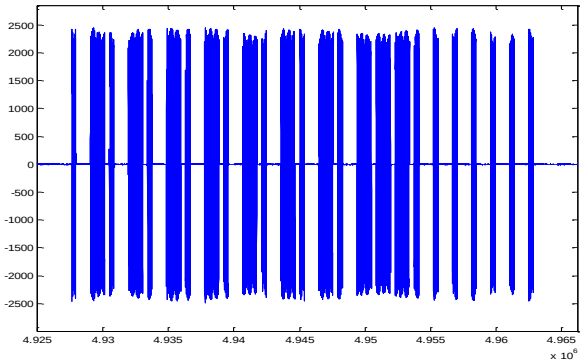


2 pulses



Here are all the other signals for the default code, zoomed-in on a single packet:

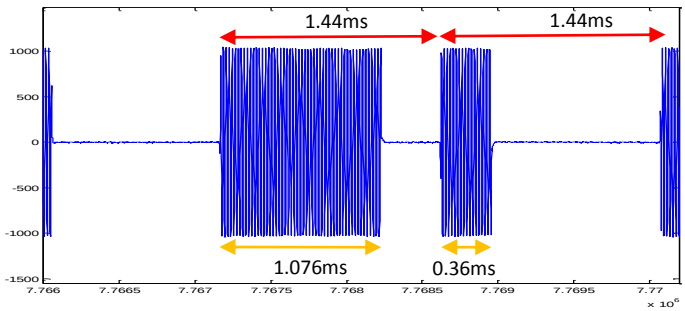
RU	
LU	

RD	
LD	

We also measured the lengths of the different pulses. The parameters are summarized in the table below:

parameter	value
Short 'on' duration	0.36ms
Long 'on' duration	1.076ms
Full OOK symbol suration	1.44ms

The timing values measured are based on averaging few (manual) measurements. In graphic view:



According to the measured signals so far, we can conclude that:

1. The signal is a repetition of a constant packet, which in turn consists of a sequence of OOK symbols.
2. There are two types of pulses in the signal: one with short "on" period, and one with a long "on" period. Both have the same total length, but differ in the length of the "on" and "off" parts. The short pulse lasting 1/4 of the total period, while the long pulse lasts for 3/4 of the total period.

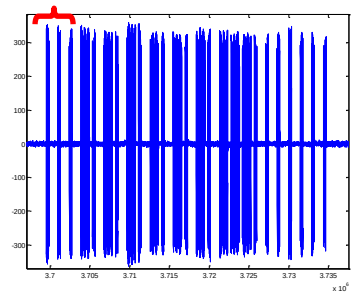
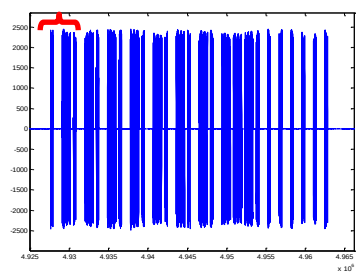
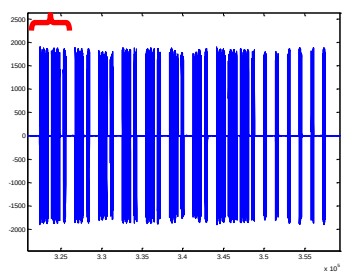
3. We can code using a 0' symbol to represent the short pulse, and a 1' symbol to represent the long pulse. If we do that, then we can translate the 4 channels to:

	CODE
RU	01 01 01 01 01 01 01 01 00 00 00 **11** 0
LU	01 01 01 01 01 01 01 01 00 00 **11** 00 0
RD	01 01 01 01 01 01 01 01 00 **11** 00 00 0
LD	01 01 01 01 01 01 01 01 **11** 00 00 00 0

4. The channels are separated one from another by the 8 symbols that are close to the end of the signal. The last symbol is, as mentioned before, always 0'. That means we can interpret the above table to be:

	CODE	Channel	F (always constant.)
RU	01 01 01 01 01 01 01 01	00 00 00 **11**	0
LU	01 01 01 01 01 01 01 01	00 00 **11** 00	0
RD	01 01 01 01 01 01 01 01	00 **11** 00 00	0
LD	01 01 01 01 01 01 01 01	**11** 00 00 00	0

5. To understand how the code part works, we manually connected one of the encoder legs to different, optional contacts, causing the corresponding bit (2 symbols combination) to change e.g. from 01' to 11' (in OOK symbol):

bit0 connected to GND --> 00'	bit0 not connected (default) --> 01'	bit0 connected to Vcc --> 11'
		

Now we can conclude that every 2 symbols correlate to the state of one tri-state "bit". The first 16 symbols are the code from the encoder. They are set by the input from the encoder legs.
Each code bit is in fact a tri-state bit that can take one of the values, based on where the encoder leg is connected to:

a. 01 – Default.
b. 00 – connect to GND line.
c. 11 – Encoder's Vcc.

6. The full frame length of the code is 12 "bits" or 24 a sequence of symbols + one last symbol that is always 0' which we referred to as a "Fin" sign.

A different representation of the signal we will use later on, is an explicit representation of the "on" and "off" periods. That is – a short 1/4 length pulse would be coded as 1000, while a long 3/4 length

pulse will be coded as 1110. This representation will allow us to build our robust automatic analysis tool. This tool only assumes that there is a defined pulse length, or baud rate, and that in each time interval, the transmitter is either on, or off.

Now we can conclude that what need to be analyzed are the lengths of the pulses, and that the amplitude is in fact much less important. Moreover the timing itself does not need to be exact as well due to low-cost transmitter and receiver in the system. As we found later, minor changes from the exact parameters still managed to open the gate.

We also measured the time between two bursts and the full length of the transmission caught in the recording:

Delay between two bursts (The end of burst to the start of the next one)	~11ms
Full packet duration	~36ms

which correlates with our observation that about 21 packets are transmitted in one second.

This concludes the structural analysis of the signal. We now understand how our remote control system works, and we can reproduce this signal using the Chronos watch. The problem is still that expensive lab equipment was used, and that our solutions is not robust enough for other possible remote control coding schemes. In the following chapters we describe how we made our process more robust, and less expensive.

Recording the RF signal With SDR Software and low-cost Hardware

In this chapter we show how we can eliminate the usage of expensive and stationary lab equipment to complete the copying cycle. We were guided by Prof. Toledo to do this with the popular HDSDR and a low-cost RTLSDR device. specifically we used this model which costs about 10-15 on eBay. Many other models can do the same job.

Software Defined Radio (SDR) is an ongoing and emerging trend. With it, we are able to implement in software on a PC or embedded systems what would need to be implemented otherwise by hardware. The way this usually works is by utilizing the PC's sound card as an ADC, with an antenna attached to a front-end unit to receive/transmit the RF signal, while the software in the CPU does the rest of the signal processing.

RTL-SDR[4] is the name for low-cost devices originally meant for DVB-T. They are based on RTL2832 demodulator that typically arrives as a USB dongle, attached to an antenna. It was discovered that they can be used as a cheap SDR since they transfer the raw I/Q samples. They output 8-bit I/Q samples with max (theoretically) sample-rate of 3.2MSamples/s. The frequency range depends on the tuner and the sampling rate.

We used a model with RTL2832+R820 Tuner which is sufficient for our needs, enabling frequency range of 24MHz to 1766MHZ with up to 2.7MHz bandwidth.

HDSDR software (current: V2.70): is a free, easy-to-use, impressive and very popular SDR software enabling software radio listening, visualizations, Rx/Tx, signal measurements, recording and many other functions for manipulating the received\transmitted signal. Moreover, this program can be used with a variety of SDR devices through the EexIO interface.

The HDSDR software and other necessary tools, as well as a practical "how to" guide can be found here. A good and comprehensive installation guide can be found here. Higher permissions are required for the installation process. In appendix B we bring in detail the settings and workflow with HDSDR.

Figure 5: Left - HDSDR main window , Right - the RTLSDR we used

To conclude this section, we need to show that the low-cost substitute we suggest reproduce the same results as the USRP equipment gave. The records from the HDSDR as we configured it are raw RF samples, 8 bit I/Q in 800kHz BW around center frequency.

[4] based on definition from http://www.rtlsdr.com/about-rtl-sdr/

The signals we recorded using RTLRSDR+HDSDR show excellent match to those we recorded with the USRP lab equipment. The SNR ration that is produced by the RTLSDR setup is worse than the SNR of the USRP setup, but it is still good enough to analyze our OOK signal. Hence, the parameters from analyzing those signals are reliable enough, and so we can use the RTLSDR with the HDSDR software for recording the remote control RF signals which we wish to duplicate by emulating it with the Chronos watch.

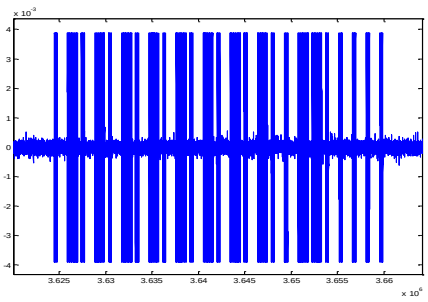
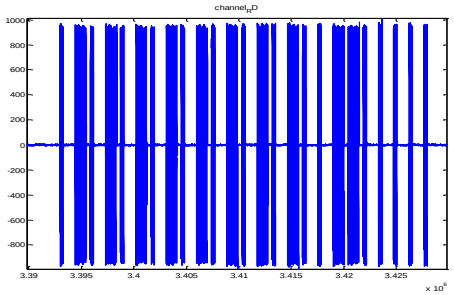
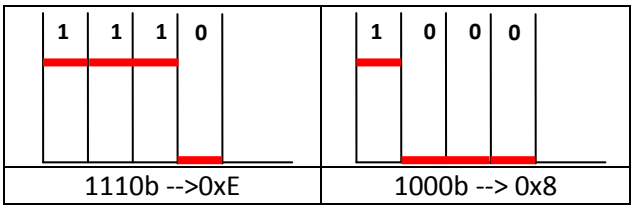
RD channel using RTLSDR+HDSDR	RD channel from USRP
	

Table 1: Comparison between the 2 recording methods. RD channel from using USRP (Right) and Using RTLSDR+HDSDR (Left). Both share the same timing parameters.

Automatic analysis and parameters extraction

After manually recording the signal for the first time, and analyzing its structure and parts, we need to extract some parameters to load into the watch. As we saw, the signal is an OOK modulated signal, with specific positions of the on and off sections. We can tell what is the implemented code out of the signal, and on which channel are we broadcasting, but it doesn't really matter - what matters is just the specific remote control transmitted pulse shape.

The parameters we are looking for are the symbol rate, or the smallest pulse length, the delay time between two packets, how many times to repeat the code, and the pulse shape itself. We represent the pulse shape using a binary code in the following way:



Note that we do not assume those are the only possible pulse shapes, we can also support for example 1010, 0101 etc.

We have the wave file from the recording as an input. We developed a python script that robustly extract the parameters out of the recording. It can be found here. It's called analyze_signal.py. Following is an explanation of the script's mechanics.

The first thing to do is to turn the wave sample into a string representing the OOK modulation of the pulse. The sample itself is very slow (1MSamples/s) in comparison with the carrier frequency of 433.92MHz, but it doesn't really matter, because we are looking for energy when the transmitter is on, and for quiet otherwise. In the script we read the data in the file, and choose one of the channels. The two channels are the I and Q of the RF signal. They are simply phase shifted from one

another, so choosing any one of them will do the work. We notice that the data in the wave file is stored as a 2's complement value, and our sample was 16bit (2Bytes) for each one.

First, we calculate the DC value of the sample, since it is not mandatory that the sampling system would be calibrated to zero. Next, we divide the sample into time windows in which we will calculate the energy of the signal. The window size will determine the precision of our results. We chose to work with windows of 10us. Every window has 10 samples, and we know from our manual analysis, that the pulse widths are in the order of hundreds of microseconds, so we feel good with this precision. Calculating the energy is simple: we square each sample, and sum the results in each window. Now we choose the average of the energy in the sample as a threshold value, where windows above this value would be considered as ON in the OOK modulation scheme, and windows below the energy threshold would be considered as OFF. We create a string with 1's for ON, and 0's for OFF, and return it.

The next stage is to extract the packet's parameters from the OOK string we've created. This part is trickier and involves more heuristics - in order to achieve a robust calculation of the parameters. This challenge is hard since noise during the recording tampers with our calculations and the pulse shape in our OOK string has errors in it.

The first thing we do, is fix single points of error - if one window in series of 0's had a burst of noise-energy and became a 1, we change it to 0, and vice versa. Next, we change the description of the recording from the OOK string, to 2 vectors of lengths - the lengths of 0's in the OOK string, and the lengths of 1's. Next, we cut these vectors from one long recording into a series of packets - each packet is represented by its own vectors of 1's and 0's lengths. The cut is implemented by cutting out the long 0's sections which represent the time before and after the button-press during the recording, and the short stops (delays) between packets.

This process also gives us the DELAY parameter needed for the watch's configuration, and the amount of pulses we saw in the recording. We suggest the REPEAT parameter to be so, that the entire RF pulse the watch will transmit would be 1 second long. We achieve this by calculating how long a single packet is, along with the DELAY between two packets, and divide one second in that time.

Now, the main trick that helps us make the entire process so robust - we choose the "best" packet out of all the packets in the recording. Because the recording usually has about 50 packets, or more, there is a very good chance that at least one of the packets will be error-free. We chose the best packet using the assumption that if no errors occur, we will have the minimal amount of different lengths in the 1's and 0's lengths vector of the packet - because if an error occurs, it splits a long pulse to two smaller ones, probably with different lengths, thus both the interference, and the two smaller pulses add new lengths to this packet, rendering it worse. In all of our recordings, with different parameters of gain and distance, we always managed to chose a perfectly clean packet.

From the best packet we calculate the smallest pulse width using some averaging on both the smallest ON pulse, and the smallest OFF pulse. This gives us the PULSE parameter that is used to set the baud-rate on the watch. Using the length of the smallest pulse, we translate the 1's and 0's lengths into binary code. For example, if the length of the smallest pulse is 36us, and the lengths are: 36 1's, 108 0's, 35 1's, 109 0's... (the numbers are not exact), it is easily translated into 10001000

code. We pad the code with 0's at its end to be a complete byte-code length (multiplication of 8), and reduce the DELAY parameter by subtracting the padding length, multiplied with the PULSE parameter. That is it - we have everything.

Our script outputs the parameters to the screen, and also to an output .txt file that we use to automatically configure the Chronos watch. We've added the output file we got from some of our records ("gilido_params.txt"). Pay attention that it has two codes for both the UP button and the DOWN button. Each wave file analysis will output one code, with an UP prefix. You can edit the parameters file and add another code - as we did - to set the DOWN button as well.

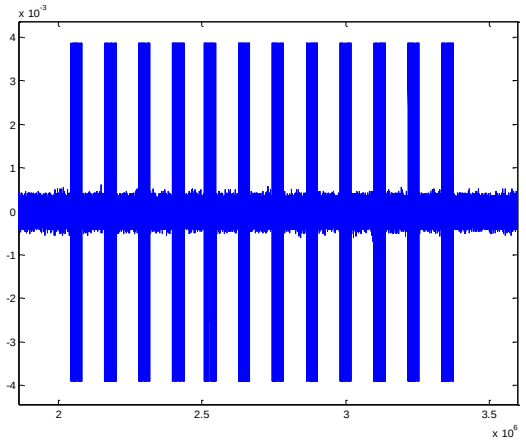
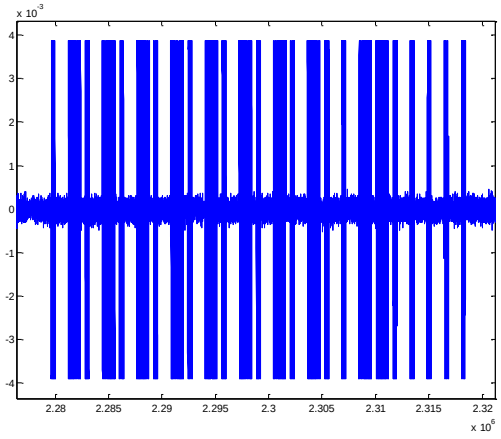
Take note that we assume 3 parameters to be known: The frequency is set to 433.92MHz, the modulation of the signal is OOK, and there is no Manchester coding of the signal. We still output them to the file, so the configuration tool will use them.

Parameter loading and system testing

The ChronIC package that we used for the watch's firmware, comes with a python helper script, that connects to the serial port to which the RF SimpliciTI access point is connected, and sends data to the watch when it is in Sync mode, containing the configuration parameters we need to set for the RF pulse to be as we want it. The main script is chronic.py. There's also a command line tool script which is chronic-cli.py. We used it during our testing, but we wanted to help automate the process, so we wrote a script that takes the configuration file, uses chronic.py, and configures the watch with the parameters from the file. It's called configure_chronos.py, and it is quite self explanatory.

The usage is: "python configure_chronos.py <parameters_filename>".

After loading the parameters of our analyzed signal to the watch, we tested it with the receiver board, and it worked. We can compare the recording of the RF signal transmitted by the watch to the original signal recorded from the remote control. This is demonstrated by the RD channel's signal transmitted by the chronos watch:

Chronos signal: The full recording	One packet of the RD channel
	

It is clear that the signal is practically identical to the original signal, thus the emulation of the recorded signal with the Chronos watch was done successfully.

Finally, Here is a video that shows the watch operating the receiver relays.

Summary, and Possible Future Work

In this project we presented a simple and complete flow for copying the signal of a remote controlled gate system, using the TI Chronos platform designed as a watch. We showed that the needed tools are low-cost and easy-to-use. We explored and analyzed remote control signals, measured them with different equipment, developed automatic analysis tools to extract the needed parameters, and utilized code for configuration of the chronos watch with the signal to transmit. By the end of the process, the watch could transmit a requested code by demand. Finally, we demonstrated the use of this procedure against a live transmitter-receiver system that emulates a remote controlled gate.

We believe that our project can be continued in several ways:

- Adding capability for storing more codes on the watch.
- Adding a GUI to simplify the whole copying process.
- Under some legal consideration, our system might be used for receive analyze and store codes of gate remote controls from a distance, although it may require a more advanced RF front end, and additional work on the robustness of the analysis script.
- Developing the recording and analysis parts of the project to be embedded into the watch platform itself.

Acknowledgements

We would like to thank Prof. Sivan Toledo for his guidance during this project.

We would also want to thank Oren Kishon for his assistance.

Appendix A: Complete instructions for building up a copy lab

Hardware needed:

1. A standard PC running windows.
2. TI ez430 Chronos watch kit, along with its RF access point and, and reprogramming dongle. It costs 58$ (as of May 2014) directly from the TI e-store, or 99$ from amazon.
3. RTL-SDR USB Radio kit. We specifically used RTL2832+R820. It can be bought for about $10-$15 at eBay.
4. The remote control that you wish to copy.

Software needed:

1. TI Chronos Control Center. It installs the drivers for the access point as well. Installing the 3rd party drivers on windows8 might be problematic. If so, follow these instructions to proceed.
2. TI Code Composer Studio V5. Use this IDE in order to compile and upload the code to the watch platform. Watch this for good initial tips for working with the IDE. You can otherwise use the control center to update the code on the watch without connecting it directly to the computer. This is a slower process and much more costly as per the watch's battery life.
3. The code of our Chronos project. You can find it here.
4. Aiding python scripts. You can find them here. In there you will find:
    a. configure_chronos.py – our configuration tool that connects to the watch through its RF access point.
    b. chronic.py – library needed for communication with the Chronos watch.
    c. analyze_signal.py - our analysis tool that will analyze the original RF signal, and output the needed configuration for the watch.
    d. gilido_params.txt - this is the parameters file output for our remote control. You don't need it, but it can be used as reference.
5. Be sure to be running python2.7. You also need to download the pyserial python library. We recommend using pip tool that finds, downloads and installs the correct version of python libraries.
6. The HDSDR software and other necessary tools, as well as a practical "how to" guide can be found here. A good and comprehensive installation guide can be found here. Higher permissions are required for the installation process.
7. You can visually analyze the wave file outputs using the open-source audacity platform which we recommend. It's not needed in order to complete the task, but it's nice and it helped us in debug. If you have access to MATLAB environment, it can be easily done through it as well.

Instructions:

1. Record the original remote control signal:
    a. Plug the RTLSDR device to a USB port on your computer (Note: do not use USB3 ports).
    b. Start HDSDR. Set and verify the settings according to the instructions in Appendix B: General Settings.

c. Record the remote control's signal. follow the instructions in Appendix B: Recording with HDSDR.
d. Make sure that the recording holds the entire transmission of the signal: start the recording before the signal starts, and end it after you finish transmitting. For reliable results it's recommended to press the remote continuously for 2-4 seconds, with only a single press per recording.

2. Analyze the recorded signal
    a. In the command shell, run the analysis python script on the wave file:
    "python analyze_signal.py <input wave filename>"
    b. The results will be printed to the screen, and also - an output file will be created with the needed data for the configuration script: "params.txt".
3. Prepare the Chronos watch:
    a. Connect the Chronos watch module using the debug dongle to your PC.
    b. Open our Chronos project with Code Composer Studio. Compile the project and upload it to your watch.
4. Configure the Chronos watch with the parameters of the signal:
    a. Plug in the RF access point dongle. In the computer's device manager check what COM-Port the USB access point was assigned, and verify that this is the same port that is configured in the configuration script "configure_chronos.py".
    b. Put your Chronos watch in "Sync" mode. While the watch is waiting for sync, run the configuration script with the configuration file:
    "python configure_chronos.py params.txt"
    c. Pay attention to the computer screen and watch screen while the configuration is underway. For every parameter successfully loaded to the watch, it will appear on the watch's screen. You can re-run the configuration script several times if you're unsure - it does no harm.
5. That's it. You will see a blinking triangle pointing upwards next to the time on the watch.
6. Go to the watch main screen. To transmit the remote control signal press the UP (or DOWN) button.

Remarks:

1. This entire scheme assumes several basic facts about the original transceiver system: the carrier frequency is 433.92MHz, the signal modulation is On-Off-Keying (OOK), and the signal is not Manchester coded. These are pretty standard for common remote control gate openers.
2. If you want to set 2 codes in your watch, analyze another signal, and in the configuration file, copy the "UP" line, change the word "UP" to "DOWN", and copy the other code instead. Note that both signals should have the same parameters, except for the code.
3. Except for code memory, the watch's memory is completely volatile. Thus, if you must reset your watch, you will have to reconfigure it.

Appendix B: HDSDR Operating Instructions

General Settings:

1. First, we should set the device control parameters through the ExtIO interface. press the ExtIO button. then on the opened form:
    a. Verify the *device hint* is set to 'RTL'. if not, change it to 'RTL'.
    b. set *sample rate* to '1Mbps' and the *offset* is '0'.
    c. set the *'Gain/Attenuation'* to maximum level by pressing *'Max'* or by scrolling the arrow to the right end.
    Note: According to our tests we recommend to set the gain to maximum, in order to get the signal as strong as it can be compared to the existing noise. The signal measured with *Mid* gain option could be used but the noise is clearly detectable. With *Min* gain option the signal suffers from the noise blend with the true signal.
2. In HDSDR main window we should set:
    a. LO A (center freq.) to 433.92Mhz. Pay attention that there are 2 optional LO. make sure the other one is set far enough from our region (as thumb role: more than +/-2Mhz around our center frequency). You could choose to set LO B instead.
    b. Tune enables in-depth inspection of part of the whole collected spectra. we use the time signal , and with 1Mbps we collect a region of 1MHz around our center frequency, guaranties we collected the signal we need.
    c. As sanity check, verify you see a large peak signal rising when pressing your RC, and that the frequency is not so far from the center. to do so:
        i. press *'Start'* button. Now the measured signal will be presented.
        ii. check *LO* and *Tune* value did not accidently changed and it is as you set it. is not – correct its value to the center.
        iii. press the remote to transmit the signal.
        iv. Press *Stop* (mark in light green) to end the test.

3. Note it is usually needed to reject the power spike caused by the internal VFO (at center frequency) of some soundcards (including ours). Look here for more details. This can damage the true signal we want to capture. In order to minimize this effect we use the built in solution as recommended on the online instructions:
    a. Press *Options -> Input Channel Calibration* and on the top of the window set the *RX DC Removal* to "Auto".
    It can be seen that the constant power appeared in the spectrum over time in the place of the LO was significantly decreased (and the time signal is, consequently, much reliable).

Recording with HDSDR

The recording process is a very simple procedure. We use the default setting as in the instructions. First (needed once), verify and set the settings:

1. On main window, press *options* -> *recording setting/scheduler*
2. Set the folder where you wish to save the files. The file names are automatically set by the program. The files are saved as wave format (.wav).
3. check that the following boxes are checked:
    a. *Recording Mode*: RF *; Recording Format:* Winrad *; Sample Type:* Auto (means: PCM16)
4. Press *OK*

In order to start record:

1. Press *'Start'* button. Now the measured signal will be presented.
2. To start recording: press the button marked with red circle.
3. Stop recording by pressing the stop button (marked with rectangle).
4. To stop Signal measured and presented by HDSDR, Press *Stop* (mark in light green).

Appendix C: The remote control and receiver kit

בלצר אלקטרוניקה בע"מ

שד' הר-ציון 17 ת"א 66057 טל:03-5377022 פקס: 03-6390195
www.belshop.biz Email: beltzer@zahav.net.il

משדר/ מקלט 4 ערוצים

להזמנה: #170981

מודול מקלט זה פועל במתח 12V DC מיוצב.

למקלט 4 ממסרי מיתוג מגע יבש מסומנים ביציאות C1–C4

חיווי מצב ערוץ בעזרת נורות לד.

טווח קליטה כ- 50 מ' בשטח פתוח.

ניתן לקודד את המשדר והמקלט בקוד אחר בהתאם להלחמות גישור מותאמות במקום המסומן במשדר ובמקלט, בהתאם לתמונות המצורפות.
הלחמות אלו יבוצעו על ידי טכנאי בלבד במלחם שהספקו אינו עולה על 30W . לא תינתן כל אחריות בגין תקלה עקב קידוד או הלחמות לא טובות / לא נכונות.

ניתן לכוון את ערוצי המקלט בנפרד לשני סוגי הפעלות בעזרת המגשרים על המעגל.
(1 ON - OFF
(2 MOMENTARY

במצב הפעלה ON ניתן לעבור חזרה למצב OFF ללא השלט , בעזרת מפסק רגעי (N.O) בהתאם לחיבורים במקלט המסומנים באותיות R1–R4 ולנקודת מתח משותפת המסומנת כ- +5V

נקודות הלחמה במשדר לשינוי קידוד.

נקודות הלחמה במקלט לשינוי קידוד.

BELLINE®

הערה: לפני הפעלת מקור המתח יש לוודא חיבורים בקוטביות הנכונה, בהתאם לרשום על המעגל.
מפסק לחיצה המסומן LEARN במעגל והלד הירוק, אינם בשימוש בגרסה זו.

ניתן לזוד את מעגל המקלט בקופסא מספר #040754