EFM32
... the world's most energy friendly microcontrollers

Connect the EFM32 with a Smart Phone through the Audio Jack

AN0054 - Application Note

Introduction

This application note describes how to connect the EFM32 Tiny Gecko microcontroller with a smart phone using the audio jack interface of the phone. It demonstrates how the EFM32 can communicate and harvest power from the phone. This enables advanced smart phone accessories that does not require batteries.

This application note focuses on the Apple Iphone and the "HiJack"-concept created by researchers at the University of Michigan. The concepts and hardware however are applicable to any smart phone. Both harvesting of power and two way data communication through the audio jack of the Iphone is implemented in the included software examples for the EFM32 microcontroller.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects
  - Cadsoft Eagle schematic, layout, BOM and PCB-gerber files for a HiJack development board for the EFM32 Tiny Gecko Starter Kit, EFM32TG-STK3300.

1 Hijack Introduction

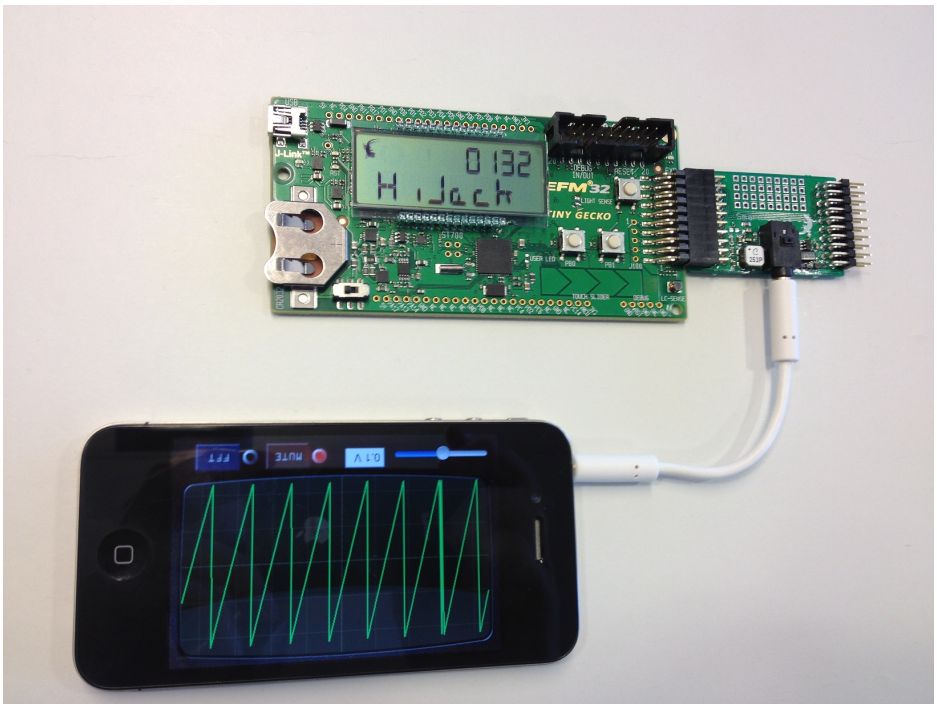
Smart phones and tablets have become powerful computing platforms the last couple of years, and with them comes the possibility of a multitude of "smart" accessories. Many of the more complex accessories are connected wirelessly to the smart phones. Simpler types of accessories are often connected through USB or other proprietary expansion ports. This application note focuses on using the very common audio jack interface which virtually all smart phones have to communicate and power smart accessories based on the EFM32 microcontroller.

A smart accessory for a phone or tablet could be anything from a medical devices to credit card readers, this application note focuses on two features which virtually all such accessories have in common; they need power and they need communication with the host device (smart phone).

1.1 Phone Connection

Many smart phones use this audio interface both for headphone output and microphone input. This gives the possibility to have two way communication between the EFM32 and the mobile phone. Often the connected accessory includes the jack plug as an integral part of itself, for other accessories it makes sense to have a cable between the phone and the accessory, see Figure 1.1 (p. 2) for examples of the connection interface.

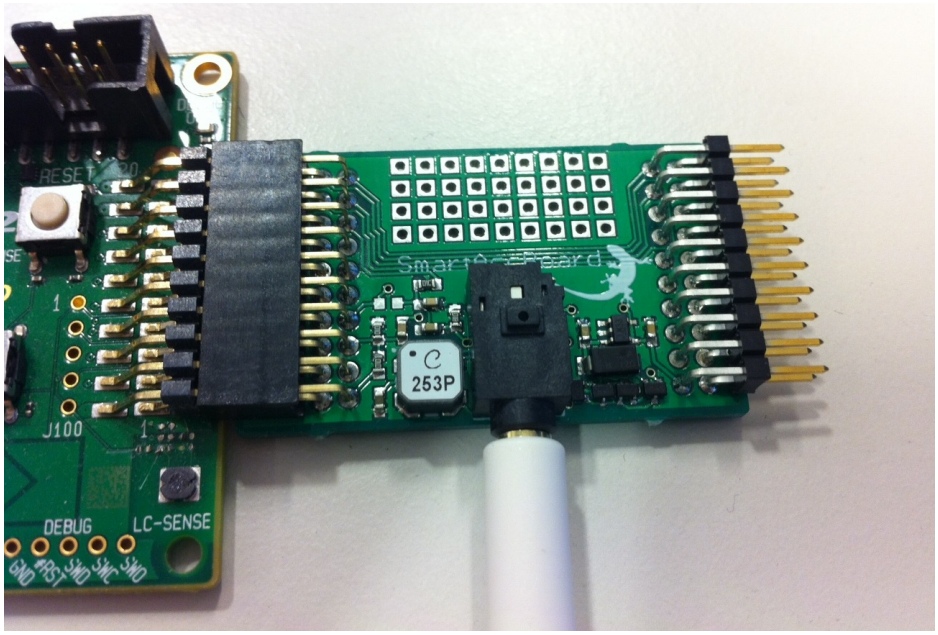
Figure 1.1. Two different interface types, one fixed jack plug and one short cable solution.



1.2 Hardware

In addition to the mechanical components like cable/plug, the hardware needed for the interface between accessory and phone consists of some passive components for the communication and a transformer, rectifier and a regulator for the power harvesting. Everything will fit within a small area on a PCB, see Figure 1.2 (p. 3). The supplied schematic, layout and PCB-files can be used to design and develop a similar solution as the one in the picture. The PCB gerber files can also be sent directly to a PCB manufacturer for the exact same development board as depicted in Figure 1.2 (p. 3) and Figure 1.1 (p. 2) .

Figure 1.2. PCB development board with harvester and communication circuitry.



1.3 Software

The software can be divided into two separate parts, one part is running on the smart phone or tablet, the other part is running on the EFM32 microcontroller. This application note focuses on the software for the EFM32 microcontroller, the hardware and schematics around the EFM32 MCU and the communication protocol. In the references chapter links to tutorials for smart phone software development are included.

2 Energy Harvesting

An audio jack plug based mobile phone accessory would not be very practical if it had to have its own power source. Harvesting power from the mobile phone through the audio plug itself is therefore of high importance. Since the audio output driver in mobile phones is designed to play audio and not deliver substantial amounts of energy, it is limited how much power is available through this interface. This makes the energy friendly EFM32 a perfect fit for audio jack connected mobile phone accessories.

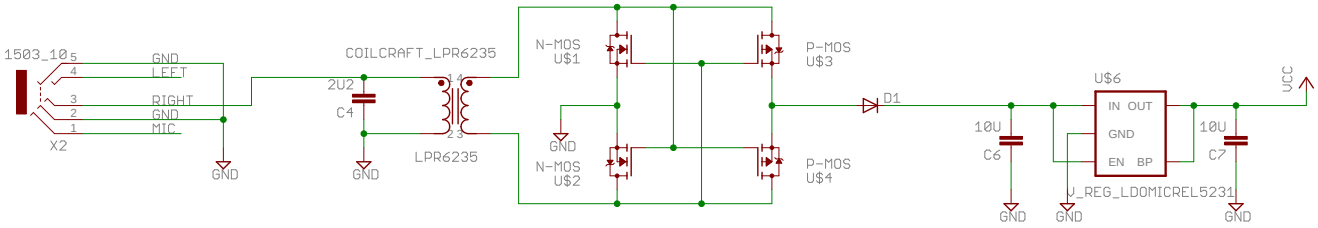
Since the audio signal available at the audio jack is designed to drive headphones, the signal is often AC-coupled. The circuit solution for converting the energy contained in the AC-coupled signal to a stable DC voltage source has been presented before in [Hijack2010] (p. 13). This section describes the circuit briefly and discusses some improvement ideas.

2.1 Energy Harvesting Circuit

The harvesting circuit is based on rectification of a sine wave sent out on one of the audio output channels. Since the microphone line is intended for communication from the accessory board to the phone, it is not as easy to harvest energy from the DC-level on the microphone line. [Hijack2010] (p. 13) indicates that maximum power transfer from the audio jack occurs at 66 mA and 240 mV, this is too low to get decent efficiency if directly rectified by diodes. A transformer with coil ratio 1:20 is used to increase the voltage and decrease the current with a factor of 20.

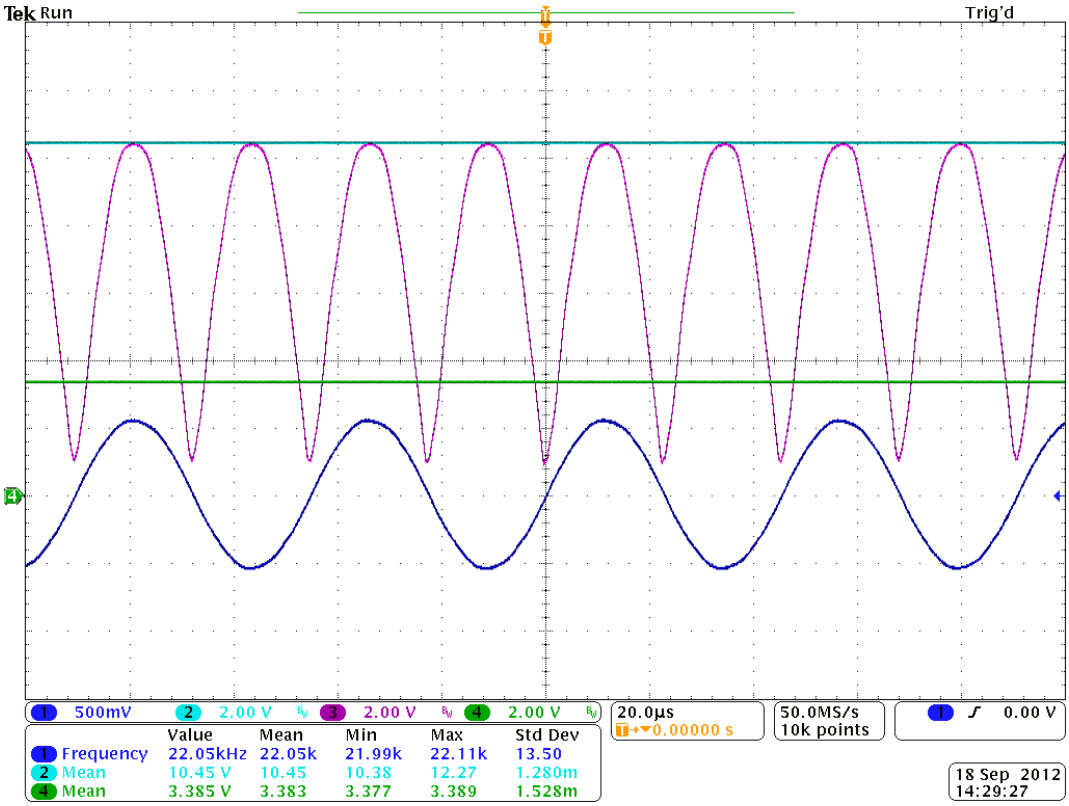
After the transformer a low voltage drop FET based rectifier is used instead of simple diodes. This reduces the voltage drop over the rectifier bridge and increases efficiency. A blocking diode and capacitor is needed after the rectifier to keep the voltage stable. To get a stable regulated voltage regardless of energy consumption a low quiescent current linear regulator is used after the blocking diode. The circuit is illustrated in Figure 2.1 (p. 4) .

Figure 2.1. Harvester circuit with transformer, rectifier and regulator



A scope plot of the different wave forms for an iPhone4 device is given in Figure 2.2 (p. 5) .

Figure 2.2. Scope plot of the different signals related to the energy harvesting. Ch1: Raw sine wave from iphone, Ch2: Unregulated voltage, Ch3: Rectified sine wave, Ch4: Regulated 3.3 V



The scope plot in Figure 2.2 (p. 5) can be related to the schematic in Figure 2.1 (p. 4) in the following way:

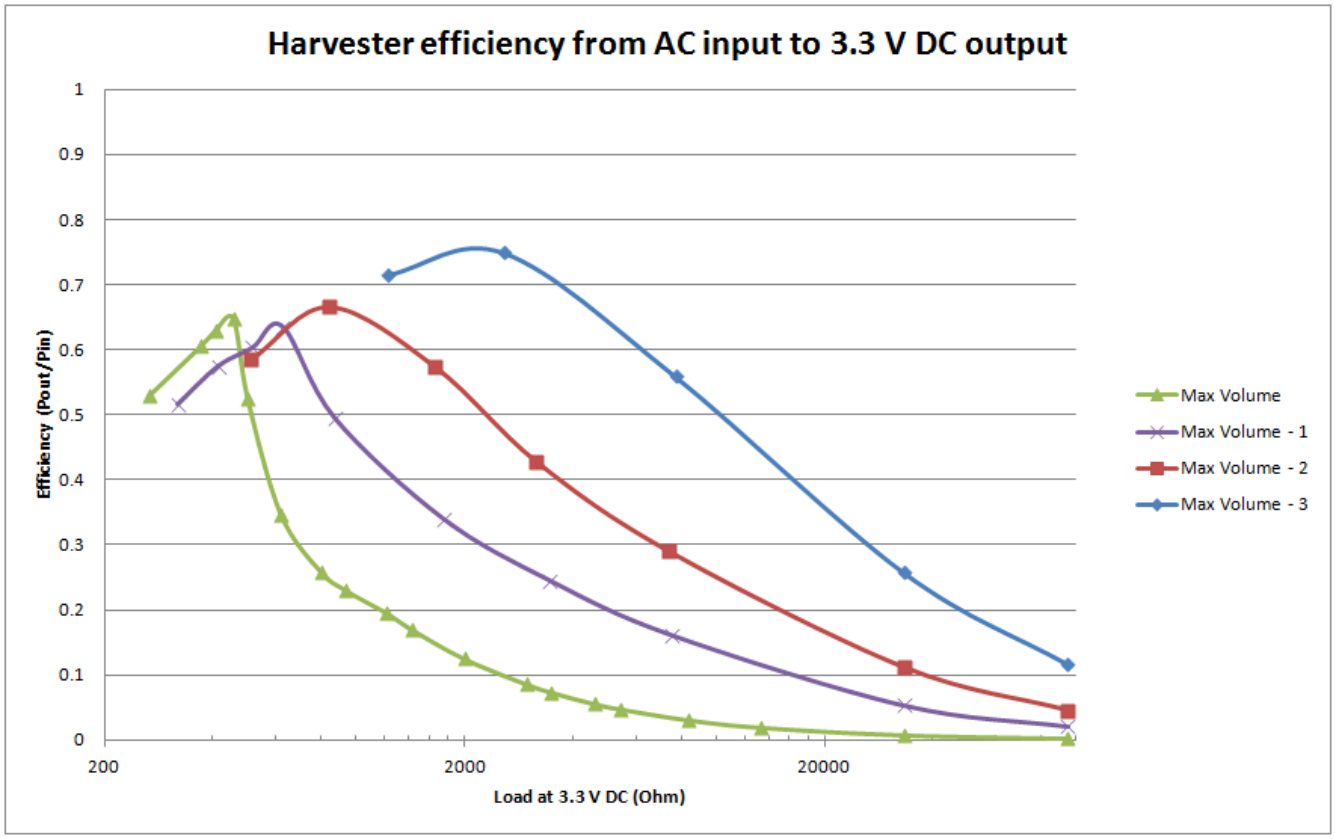
- Channel 1: Signal measured over capacitor C4, right before transformer.
- Channel 2: Signal measured directly after diode D1.
- Channel 3: Signal measured directly before diode D1.
- Channel 4: Signal measured over capacitor C7, right after 3.3 V regulator.

2.2 Efficiency and Output Power

Efficiency of the harvesting circuit is dependent on how much power is needed by the application. Since the efficiency from mobile phone battery to AC-output of headphone jack varies between different models of mobile phones, this section will only cover the efficiency from AC-input of harvester to regulated DC-output of harvester. The [Hijack2010] (p. 13) paper also discusses efficiency from mobile phone battery to harvester output, but without a voltage regulator, it concludes with about 50% efficiency from iPod battery to an ideal resistive load at unregulated DC-output.

After doing efficiency measurements from AC-output of an iphone 4 to regulated 3.3 V output at different volume settings, the following results were obtained:

Figure 2.3. Harvester efficiency for different volume settings



The drop in efficiency after the peak is caused by a drop in output voltage when the output has reached its maximum driving capabilities at that volume setting. Notice that at lower volume settings the efficiency is higher through out the load range. This is mainly caused by larger voltage drop over the linear regulator for the higher volume settings. These results would look different if a switching DC/DC regulator was used instead of a linear voltage regulator. For a linear regulator at the output, reducing the phone volume at low loads will increase the efficiency.

In addition to efficiency of the harvester, the maximum available power is also an important parameter. This is highly dependent on the output driver of the mobile phone, therefore it can vary a lot between different manufacturers and even between models from the same manufacturer. The maximum available power after regulation to 3.3 V for a couple of different devices is given in Table 2.1 (p. 6). Note that the current was measured when reducing the load resistance at the regulated output until the voltage dropped to 3.2 V. The current (mA) at 3.2 V is therefore a measurement of maximum available power.

Table 2.1. Maximum current (mA) at 3.2 V for a few different devices

Mobile Phone	Maximum current (mA)	Comment
Iphone 4	7.2 mA	
Samsung Galaxy S2	0.24 mA	
Samsung Galaxy S3	1.5 mA	Maximum current increased from Galaxy S2 level during a period of a several minutes, possibly caused by output impedance adaptation.
HTC Sensation XE	0.56 mA	Maximum current increased from Galaxy S2 level during a period of a several minutes, possibly caused by output impedance adaptation.

The [Hijack2010] (p. 13) paper indicates maximum power available from an iphone 3GS device to be 15.8 mW which is below our measurements for the iphone 4. From the table above it is also clear that the amount of power varies from device to device. All results were obtained with volume set at maximum.

2.3 Improvement Ideas

Improving the available power at the expense of a more complex and expensive circuit is possible. Purpose built harvesting circuits with MPPT (Maximum Power Point Tracking), low quiescent current and built in output voltage regulation could harvest energy at higher efficiencies when the output power is varying.

To increase the battery life of the mobile phone it is feasible to make the phone aware of the power needed by the accessory. The phone could then adjust the volume/frequency or duty cycle the output to adapt to different power levels. For example if the EFM32 stays in a deep sleep mode, the harvesting circuit only needs to supply a couple of uW. Duty cycling the output in this case would increase efficiency because there will always be a static consumption by the mobile phone driving the audio output, even if no energy is consumed by the accessory.

More information about this circuit, bill of materials cost and a more thorough analysis can be found in the [Hijack2010] (p. 13) paper.

3 Data Communication

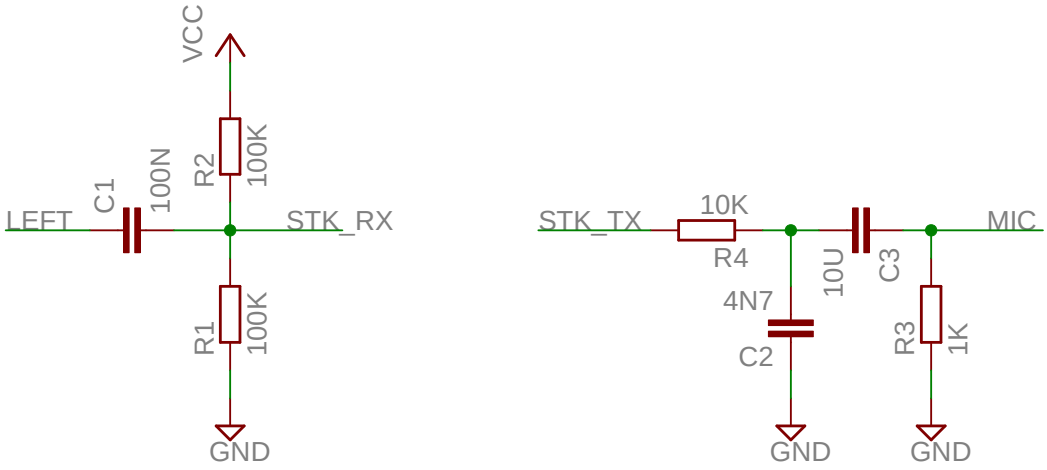
Communicating with the mobile phone through the audio jack can be achieved in multiple ways. Two way communication is also possible in many cases since the analog audio interface already includes a microphone connection for many mobile phones. The simplest form of communication with an accessory is the passive approach which operates by passing an analog AC-coupled voltage directly from a sensor in the accessory to the phone through the microphone connection. Similarly information can be passed in the other direction through the analog audio channel not used for energy harvesting.

The analog communication method works well for some applications like microphones, piezo electric sensors or magnetic card readers. But as soon as a microcontroller is involved, often with many sensors that needs to be integrated into one communication channel, the simplest analog approach will not work. One solution is to use the microcontroller to encode data and pass it digitally to the mobile phone. This application note discusses one way of achieving digital communication in both directions between an EFM32 and the mobile phone. Specifically, to solve the AC-coupling issue, data is transmitted using manchester encoding.

3.1 Signal Interface Circuit

The audio output from the mobile phone and the microphone input expects a signal referenced to ground with zero offset. Since the EFM32 cannot operate on signals below ground, the offset needs to be shifted to Vdd/2. This is achieved in both directions by a simple voltage divider and AC-coupling of the signal. The circuit for both directions of communication is illustrated in Figure 3.1 (p. 8) .

Figure 3.1. Communication circuit to AC-couple the signal in both directions



In Figure 3.1 (p. 8), STK_TX and STK_RX are connected directly to GPIO pins on the EFM32. LEFT and MIC are connected to the corresponding signals in the jack connector of the smart phone.

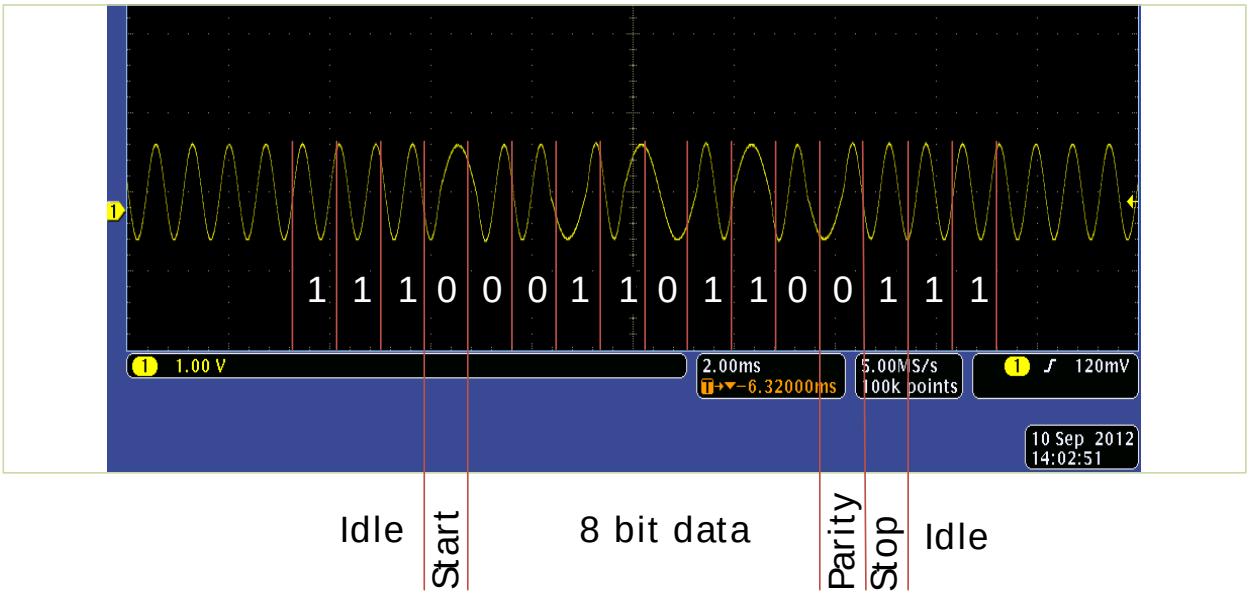
Notice in Figure 3.1 (p. 8) that the microphone output is both low pass filtered, ac-coupled and includes a 1K resistor so that an iphone and other i-device detects that a microphone is connected. This resistor might not be needed for other manufacturer's phones.

3.2 Data Modulation

Since the audio interface for mobile phones in most cases does not support DC-voltages, data must be encoded in such a way to eliminate DC-voltages. A couple of different approaches for achieving communication over the audio interface is discussed in [Hijack2010] (p. 13). The discussion ends up with recommending manchester encoded data transmitted in a UART fashion with start-, parity- and stop-bit. Manchester encoding works by replacing every 1 in the data stream with 01 and every 0 with 10. This encoding doubles the amount of bit transferred, but it ensures on average a DC-level of zero. The code is also self-synchronizing since it guarantees a transition for every bit.

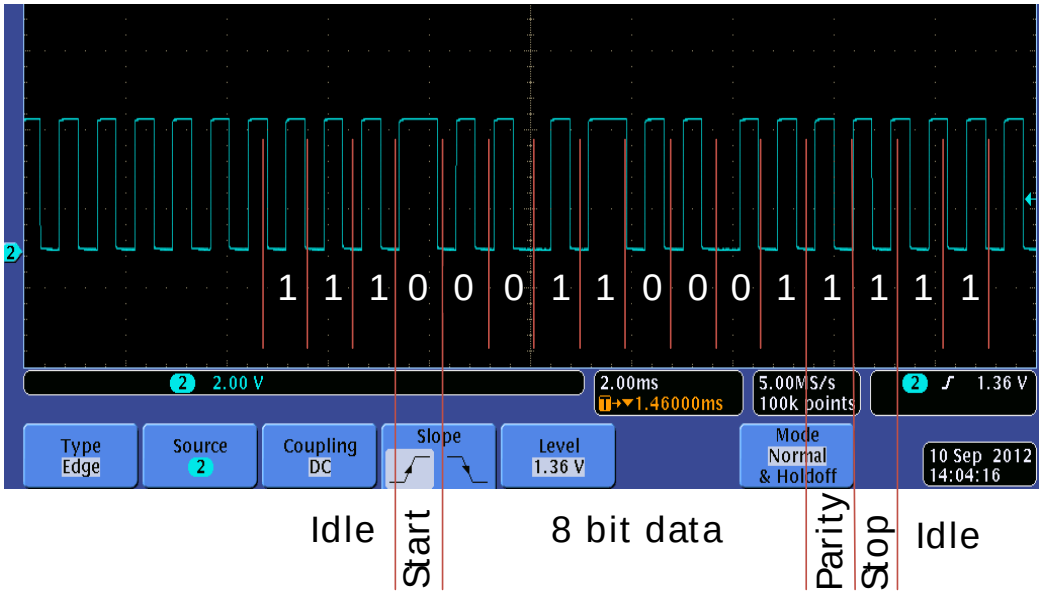
Figure 3.2 (p. 9) illustrates one byte of transmitted data from an iphone to the EFM32. Notice that the rising edge represents a 1, while a falling edge represents a 0. The transmission includes start, stop and parity bit, just like a UART serial data transmission.

Figure 3.2. One manchester encoded byte transmitted from iphone to EFM32



Similarly the data transmitted in the other direction is also manchester encoded, but now as a square wave signal since this is easier to achieve with a microcontroller, see Figure 3.3 (p. 9). The square wave is low pass filtered and AC coupled before it is passed on to the mobile phone microphone input.

Figure 3.3. One manchester encoded byte transmitted from EFM32 to iphone



In the included software example, the manchester encoding and decoding is achieved by using an EFM32 timer module. The input and output signals are interfaced directly by using general purpose pins.

Because of interference between the transmitted and received signal the interface is half duplex, data is only flowing in one direction at a time.

4 Software Example for the EFM32

The software example project with the current application note uses the EFM32 CMSIS including STK BSP and emlib to demonstrate Hijack communication. The firmware has been intended to be used on the EFM32TG-STK3300 Starter Kit and initial target devices are the EFM32 Tiny Gecko family. It is required to replicate the hardware setup shown in the hardware chapter in order to test the application. The HiJack Oscilloscope application from the itunes app store can receive the transmitted data.

http://itunes.apple.com/us/app/hijack-oscilloscope/id434388871?mt=8

The software consist of a main demo application and a port of the hijack driver for EFM32.

4.1 main.c

The main() function will initialize the kit, LCD display and hijack driver. The main HF clock is set to 8 MHZ in order to keep power consumption low. After this it will continuously send a sawtooth wave to the iphone. If button 1 or 2 is pushed 0x55 or 0xAA is sent.

4.2 Hijack driver

The Hijack used a manchester encoded UART formatted byte and the Hijack driver uses Timer 0 to capture the pulse widths of the received signal and a manchester decoder function is used to decode the bits into a byte. Timer 1 is used to output a byte encoded correctly. Note the compiler optimizer should be set to "speed" in order to prevent communication errors.

The Hijack driver consist of 3 main functions:

```
void HIJACK_Init(HIJACK_TxDoneFuncPtr_t pTxDone)
```

Initialize the Timer0 and 1 to be used for sending and receiving data. pTxDone is a callback function pointer that called when a byte has been sent

`bool HIJACK_ByteTx(uint8_t byte)` Used to send data to the iphone.

```
void HIJACK_ByteRx(uint8_t *pByte)
```

Called when a byte is received from the iphone, today this function prints the received byte to the display.

Depending on hardware and clock speed it might be required to "trim" the Hijack driver. this is done through four defines:

```
#define HIJACK_RX_SHORTINTERVAL (56)
```

sets the number of timer ticks for a short interval.

```
#define HIJACK_RX_LONGINTERVAL (150)
```

sets the number of timer ticks for a long interval.

```
#define HIJACK_TX_INTERVAL (45)
```

sets the number of timer ticks used by the TX timer.

```
#define HIJACK_TIMER_RESOLUTION timerPrescale256
```

timer prescaler used for TX and RX timer.

5 HiJack Smart Phone Development

This application note does not cover how to develop iPhone or Android applications for Hijack, however the original authors have provided all their iPhone source code on **http://code.google.com/p/hijack-main/** A tutorial is available as well.

Below is an example of how the iphone software initialize and send and receive data. It is taken from the iPhone code by the original authors, [Hijack2010] (p. 13) :

Initialize

```
hiJackMgr = [[HiJackMgr alloc] init];
[hiJackMgr setDelegate:self];
```

Send:

```
(IBAction)SendByte{
[self.viewController.textLabel setText: @"send test"];
[hiJackMgr send: 0x10];
}
```

Receive:

```
-(int) receive:(UInt8)data
{
// do something with ”data”...
return 0;
}
```

References

[Hijack2010] .

Ye-Sheng Kuo, Sonal Verma, Thomas Schmid, and Prabal Dutta, "Hijacking Power and Bandwidth from the Mobile Phone's Audio Interface", First Annual Symposium on Computing for Development (DEV'10), Dec. 2010.
http://www.eecs.umich.edu/~prabal/pubs/papers/kuo10hijack.pdf

[LibHiJack] .

Tutorial for iphone development (LibHiJack).
http://rawrwarbear.com/2011/08/21/hijack-in-the-box-part-1/

6 Revision History

6.1 Revision 1.05

2014-05-07

Updated example code to CMSIS 3.20.5

Changed source code license to Silicon Labs license

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

6.2 Revision 1.04

2013-10-14

New cover layout

6.3 Revision 1.03

2013-05-24

Fixed erroneous license text in source files.

6.4 Revision 1.02

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

6.5 Revision 1.01

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

6.6 Revision 1.00

2012-10-04

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

A.2 Trademark Information

B Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

List of Tables

silabs.com


ZERO
ARM Cortex-M0+
TINY
ARM Cortex-M3
GECKO
ARM Cortex-M3
LEOPARD
ARM Cortex-M3
GIANT
ARM Cortex-M3
WONDER
ARM Cortex-M4