MITHIS CFXS

REVISION PT-FT2 DATASHEET 15 MAY 2007

PRODUCT OVERVIEW

The CFXS is a consumer device for interfacing analogue telephones to a computer based PBX. This device allows interfacing a complete Home or Office to VoIP technologies at extremely low cost without replacing legacy telephones.

This document contains all necessary information to develop custom code for the device and extend the software functionality.

It also includes full schematics and hardware description for further improvements of hardware.

CORE FEATURES

- Interfaces 8 telephone lines
- Suitable for Home usage, interface a whole house
- Suitable for Business usage, interface a complete office
- USB 2.0 and legacy 1.1 compatible interface
- Full Speed USB support.
- Full access to ringing signal allowing custom ringing signals
- Full access to transmission during silence periods allowing Caller ID functionality
- Full access to compounded audio signal on computer
- Support for both μ-law and A-law standards
- Operation without PC possible
- Low power consumption suitable for always on operation
- Low standby power draw of less then 10mA

This technical document describes the operation and programming interface for the CFXS board revision PT-FT2, marked “CFXS-TRY2”. This document is specific to that board. Furthur revisions of the board may have changes which require revised technical information.

Table of Contents

TABLE OF ILLUSTRATIONS

TABLE OF TABLES

Errata

Currently no errata.

Revision History

Version	Released	Changes
PT-FT2	11 May 2007	Initial Revision

Copyright

Document Key

Normal textual description

Note of importance or documenting a peculiar feature

Important warning which if ignored could cause damage or harm

```
A PIC ASM Code Sample
```

HARDWARE

Overview

The hardware for the CFXS board is built around 3 main modules. These are:

- the core processor, which is a Microchip 18F4455 PIC
- the two CODEC chips, which are National Instruments TP3094 chips
- the 8 MITHIS TIM modules, which are based on the Intersil 55185 chip

The device also uses

- Serial to Parallel 74HC595 Shift registers to increase the number of IO available
- 4.096MHz Oscillator to provide the clock for the PCM highway

Information on how to access the complete datasheets for these devices is provided in Appendix I - Component Datasheets.

As the Microchip PIC does not have a PCM interface, the SPI port has been repurposed to allow communication to the CODECs.

This makes communication with the CODEC more difficult then would otherwise be expected and careful consideration should be given to the application notes provided in both the Hardware and Software sections.

THIS DEVICE IS FOR INSIDE USAGE ONLY

This device is not suitable for connection to lines which are exposed to outdoor elements. The device does not include the protection required when operating under outdoor conditions and could be dangerous or even FATAL.

THIS DEVICE IS NOT A CARRIER GRADE TELECOMMUNICATION DEVICE

This device is for consumer applications only. It is not designed for operation in carrier grade telecommunication environment and no warranty of correct operation is given when used in these situations.

Block Diagram

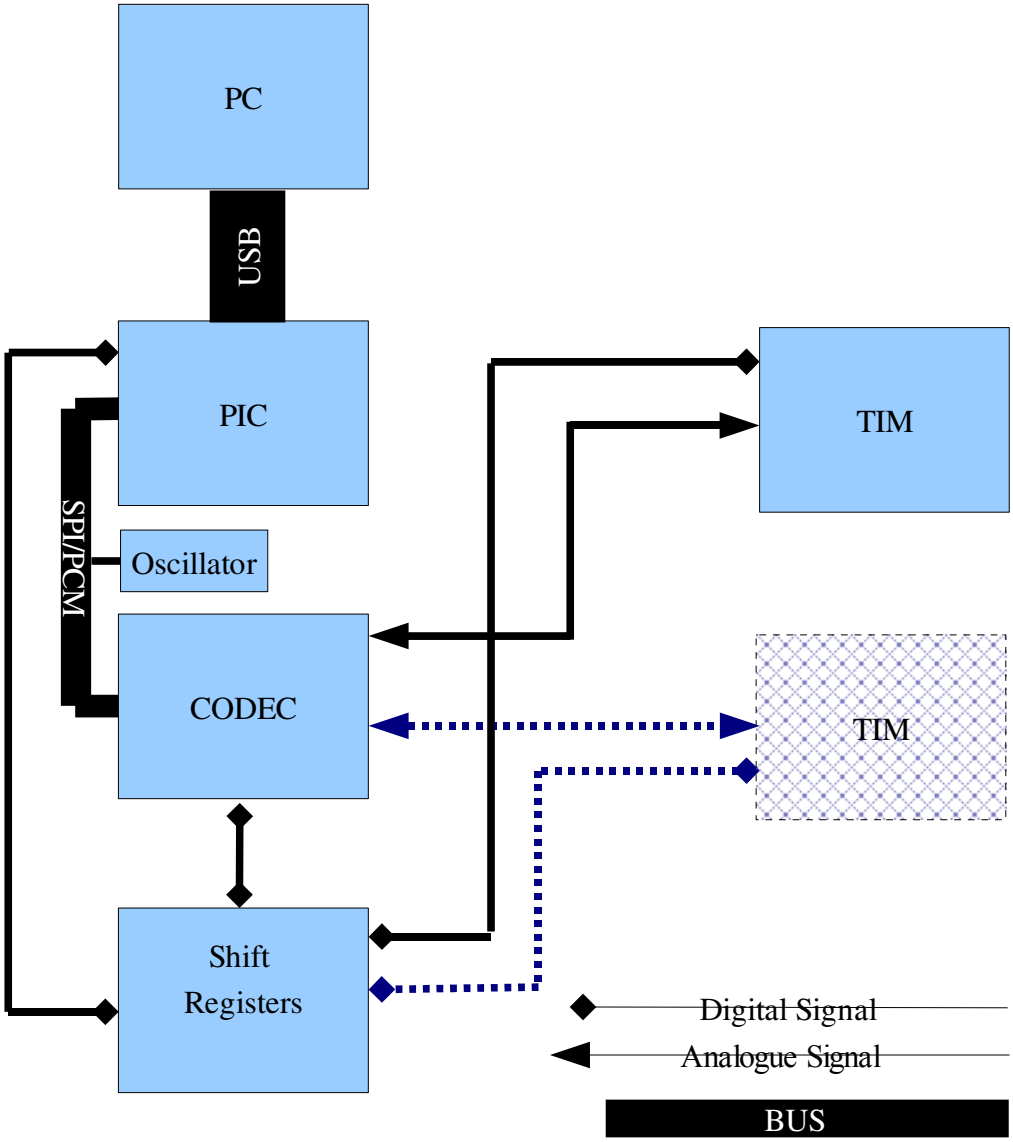


Illustration 1:System Block Diagram

WARNING

The CFXS board, revision PT-FT2 has **NO** power protection circuitry. This means that connecting the -48V incorrectly **WILL destroy the board.** ALL TIM modules which are currently plugged on the board will also be destroyed!

Power Supply

The CFXS board is powered by two different supplies: a 5V power supply for the logic circuits; and a -48V power supply for the TIM modules.

Normally the 5V power supply will be provided through the USB interface, while the -48V supply will be provided by an external "plug pack" supply.

It should be noted that these supplies share a common ground. This ground should **not** be connected to earth via the -48V supply.

Requirements for -48V Supply

The CFXS PT-FT2 board has **NO power protection circuitry.** It is therefore important to add protection on the power supply output.

Voltage Requirements

The -48V supply must provide -48V. The supply should be floating, as the whole board is connected to earth via the 5V supply.

While the board may appear to function correctly when the -48V supply is earthed, significant damage can be caused to any computer connected to the board via the USB plug.

The -48V supply **must not exceed -50V**, so it is recommended that a regulated power supply be used which will clamp any voltage spikes.

The voltage may drop as low as -40V when supplying full load without effecting operation of the device.

If a positive voltage greater then 5V is applied to the -48V input, TIM Modules will fail in a way which can cause significant damage to the board making it unusable.

Current Requirements

The power supply tables found in the Power Consumption section show the required supply amperage.

As the board will never draw more then 80mA from the -48V supply, it is recommend a 100mA fast blow fuse is added to the protection circuitry.

Requirements for 5V Supply

All the IC logic on the board is powered from the 5V power supply.

Voltage Requirements

The 5V supply must be regulated and provide very low ripple output. The voltage must not drop below a nominal level of 4.7V or rise above a level of 5.5V.

The 5V supply is normally provided by the USB interface. However, a external power supply can be used.

The board provides circuitry to prevent polarity reversal of the 5V supply.

Current Requirements

The power supply tables found in the Power Consumption section show the required current supply.

As the device requires around 150mA of supply current from the 5V supply when all lines are operating, the device must register as a *High Powered* device. Until the registration occurs, the microprocessor should keep all CODECs and TIM modules in their low power states.

Power Consumption

The device's power consumption is highly dependent on the current state of the device.

When the device is in power down state (all the TIM modules are in power down mode and the Codec PD lines are high), only a minimal 10mA is drawn from the 5V supply with less then 3mA drawn from the -48V supply.

As the device is designed to be run continuously (24 hours a day, 7 days a week), it is very important to return to this power down state when not active.

The following tables gives power consumption of the device in various states. A spreadsheet is also provided to easily calculate overall consumption values.

-48V Supply

Absolute Values	
Maximum Power Draw[1]	80mA
Minimum Power Draw[2]	1mA
For each TIM Module	
State	Consumption
Idle	0.1mA
Ringing	4mA
Forward Active	7mA

Table 1: Power Consumption from -48V Supply

5V Supply

Absolute Values	
Maximum Power Draw[3]	190mA
Minimum Power Draw[4]	12mA
For each CODEC in Mode	
Active	30mA
Power Down	5mA
For each TIM Module in Mode	
Idle	0mA
Ringing	8mA
Forward Active	20mA

Table 2: Power Consumption from 5V Supply

1 This occurs when all TIM modules are operating in *active mode*
2 This occurs when all TIM modules are operating in *standby mode*.
3 This occurs when all TIM modules are operating in *active mode* and both CODECs are in *active mode*.
4 This occurs when all TIM modules are operating in standby *mode* and both CODECs are in *power down mode*.

Processing Core

This processor is a Microchip PIC 18F4455 processor, how to obtain a full datasheet for this chip is included in the appendix. The main features of this chip include:

- Inbuilt USB module
- High 12MIPS instruction rate
- Low power consumption
- More advanced instruction set.

The PIC processing core was chosen because of the numerous free programming tools available for the device.

Some boards may have the compatible PIC 18F4550 core. This device is identical to the PIC 18F4455 but has more program memory.

PCM SPI Interface

Normally in telecommunication system designs, data would be directly clocked from one CODEC into another CODEC. This however is very inflexible and does not allow any processing of the data. It would also be impossible to send the data to the computer with this design. Instead in this board all sample data flows through the core processor.

The CODEC's would normally interface to a PCM highway. However, the PIC processor however does not have a PCM interface.

A PCM highway is a serial based interface, SPI is also a serial interface. Thus, the SPI port of the PIC was connected to the PCM highway.

Although it would appear natural to have the processor generate the master PCM clock and run the SPI device as a master, this is not possible for two reasons:

- The PCM highway requires a clock which can not be generated from the PIC clock
- The CODECs only operate when supplied a continual PCM clock

Instead an external Crystal Oscillator is used to provide the clock and the SPI device is operated in slave mode.

As well, the transmit data must appear on data bus **before** the appropriate Fsync lines go low.

This offers unique programming challenges which are discussed in the software section.

The current version of the board uses a 4.096MHz crystal. As this value is not optimal, it will probably be replaced with a different clock in future revisions.

USB Interface

All technical information provided by the datasheet for the Microchip PIC 18F4455 “UNIVERSAL SERIAL BUS (USB)” is directly applicable to the board. Information on how to access this datasheet is provided in Appendix I - Component Datasheets.

The board is using, the internal pull-up resistors and the internal transceiver configuration as described on page 165 of the PIC datasheet.

The board is also powered directly from the USB bus as described in section 17.6.1 on page 183 of the PIC datasheet.

For example code for programming the USB interface please see the Software section of this datasheet.

Output LEDs

The board has 3 LED indicators on the top of the board. They are in a traffic light configuration on the top board, with the top one being Red, the middle Orange and the bottom Green. Table 3 shows which pins these LEDs are connected too.

LED Pins	
Red Led	Port A, Pin 2
Orange Led	Port A, Pin 3
Green Led	Port C, Pin 2

Table 3: Shift Register Pin Mappings

It is recommended these lights be used for status indication.

Shift Registers

All the shift registers on the board are 74HC595. This IC is a Serial to Parallel Shift Register with Output Latch. These registers are extremely common in digital circuits and have been around since the late 1980s. The logic diagram for these shift registers is shown in Illustration 2[5].

The actual IC used is the Texas Instruments SN54HC595. How to obtain the full datasheet can be found in Appendix I - Component Datasheets.

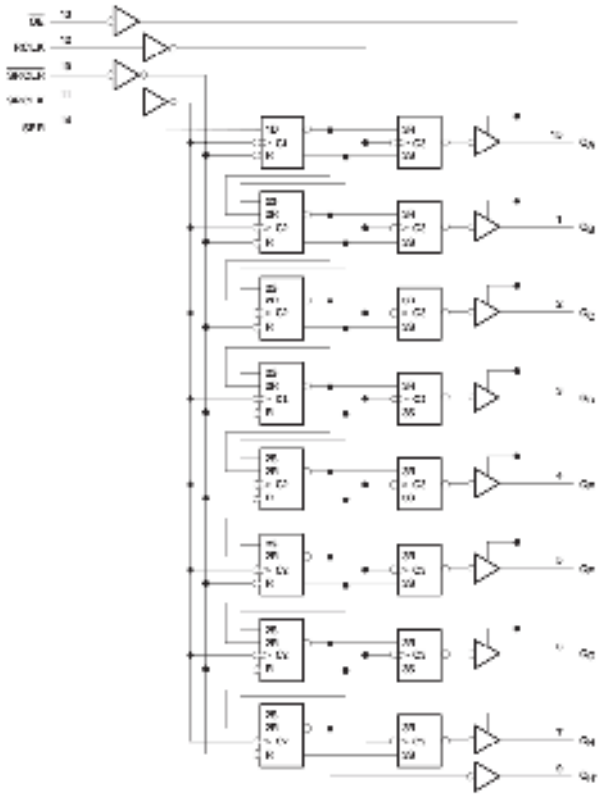


Illustration 2: 74HC595 Logic Diagram

For ease of reference, Table 4 provides which pins each shift register is connected to on the PIC processing core.

5 Logic Diagram Sourced from http://focus.ti.com/docs/prod/folders/print/sn54hc595.html

STROBE Register	
RCLK	PORT A, PIN 1
SCLK	
DATA	PORT A, PIN 0

TST Register	
RCLK	PORT E, PIN 0
SCLK	PORT E, PIN 1
DATA	PORT E, PIN 2

F0 Register	
SCLK	PORT B, PIN 6
RCLK	PORT B, PIN 7
DATA	PORT B, PIN 5

F1 Register	
SCLK	PORT B, PIN 3
RCLK	PORT B, PIN 4
DATA	PORT B, PIN 2

Table 4: Shift Register Pin Mappings

Strobe Register

The strobe register provides the FSync lines needed by CODEC. Illustration 3 shows a block diagram of how this register is connected.

Unlike the other shift registers this register does not need the output latch. Instead the Serial and Register Clock are connected together. This produces a 1 cycle delay in the output of the register that must be corrected for.

The CODEC specifies that the Fsync lines must occur at 8kHz synchronous with the PCM bus clock. To provide the correct signal, the Strobe Register should be provided with a 64kHz clock with every 8th data bit being High.

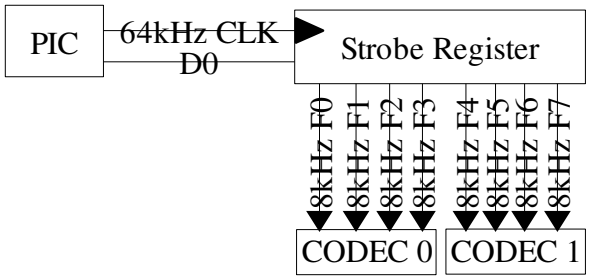


Illustration 3: Strobe Block Diagram

The strobe register is the most important register on the board. If the signals produce from this register are not correct, then nothing else will function correctly.

More information about the timing requirements of the output of the Strobe register can be found in the CODEC section.

TST Register

The TST shift register connects to the PDx lines of the CODEC. This register allows both, putting the CODEC into power down modes and into Test modes.

The RCLK pin, as connected to the CODEC's TST pin. However, the CODECs will only go into Test mode when the TST pin stays high for longer then 16 MCLK cycles (With the 4.096MHz clock this is 47 instructions). This gives plenty of time to prevent the device from entering the test mode.

The Test modes that the CODEC can be put into is taken directly from the TP3094 datasheet, but have been included as Table 5 for easy reference.

Fx-SR Registers

The F0-SR and F1-SR registers connect to the Fx control lines of the TIM modules.

Only the F2 and F0 lines on the TIM modules are connected to the registers, both the E0 line and F1 lines are tied high by the pull up resistors.

For more discussion on the various modes that are possible with this configuration see the TIM module section.

The F0 register connects to the TIM modules on Line 4-7, while F1 register connects to the TIM modules on Line 0-3. A block diagram of the F0 register is shown in Illustration 4.

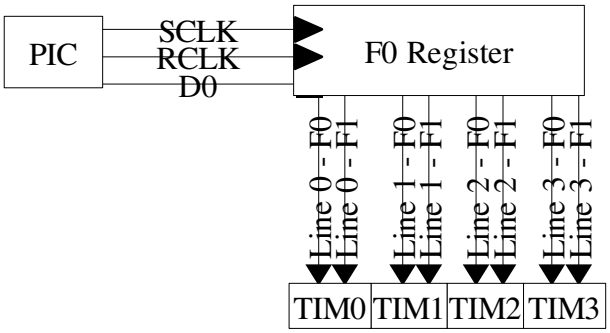


Illustration 4: F0-SR Block Diagram

The F1 register has a similar block diagram, except it is connected to TIM modules 4-7.

CODECs

The two CODECs used in this device are National Semiconductor's TP3094 Quad PCM CODECs.

FSync and Data Lines

Data is transmitted and received from the CODEC via the PCM bus. The first bit of the data is placed onto the PCM bus and then the appropriate FSTx or FSRx line is taken high.

Unlike operation in normal telecommunication systems, the CODEC's FSTx and FSRx lines are tied together. This means that the CODEC will both transmit and receive a byte on the same FSync pulse. This operation works because the data is going into the processor rather then another CODEC.

There are 8 FSync lines (4 for each CODEC) which directly relate to individual lines. Each pulse on a FSync line must be 8kHz apart but there does not need to be any relationship between the different FSync lines. (Generally, the FSync pulses will be generated from a 64kHz clock and hence be evenly spaced – but this is not a requirement.)

The FSync pulses are generated by the Strobe Register, for more information about how this should be done, please see the Strobe Register section.

The PCM interface of the CODEC has

Test Modes	TST	PDN0	PDN1	PDN2	PDN3	Description
Normal Operation	0	x	x	x	x	
Single Channel Digital Loopback	1	1	1	A0	A1	Ch. select with PDN2, PDN3
Single Channel Analog Loopback	1	0	1	A0	A1	Ch. select with PDN2, PDN3
Single Channel DC Conversion	1	1	0	A0	A1	Ch. select with PDN2, PDN3
4 Channels Digital Loopback	1	0	0	0	0	
4 Channels Analog Loopback	1	0	0	0	1	
4 Channels DC Conversion	1	0	0	1	0	
Invalid States	1	0	0	1	1	

Where A0, A1 select the channel under test, according to the following table

A0	A1	Channel Selected
0	0	Channel 0
1	0	Channel 1
0	1	Channel 2
1	1	Channel 3

Table 5: CODEC Test Modes

very strong timing requirements. The timing diagrams can be found in the TP3094 datasheet, the relavent diagram can also be seen in Illustration 5. For instructions on how to satisfy these requirements using the PIC's SPI interface, please see the Software Section.

> It should be noted that the DR data must be on the bus by the first falling clock edge after the FSRx line goes high.
>
> This timing requirement can be very hard to satisfy can a discussion of how to do this is available in the "CODEC Module" software section.

PDN Lines

The CODEC's PDN lines can be access through the TST Register. They should stay high most of the time and only be lowered when the associated line is actually in use. This will reduce power consumption.

As noted in the TST Register section, the CODEC's TST pin connected to PORT E, PIN 0 which is also the RCLK pin for the test register. To get the CODEC to enter a test mode, this pin must be held high for more then 16 MCLK cycles. The CODEC will then enter the TST mode shown by Table 5.

Law Selection

The LAW pin of the CODEC is connected to PORT C, PIN 1 setting this line high will cause both CODECs to use Mu-Law companding. Setting it low will cause the CODECs to use A-Law.

For information on Mu-Law and A-Law please see Appendix VIII - Law Formula

All lines must use the same Law format, it is recommended that Mu-Law format is used when given a choice.

TIM Modules

The board uses 8 MITHIS TIM modules to interface the CODEC's to the telephone lines. Each module is connected in identical configuration.

The full functionality of the TIM modules

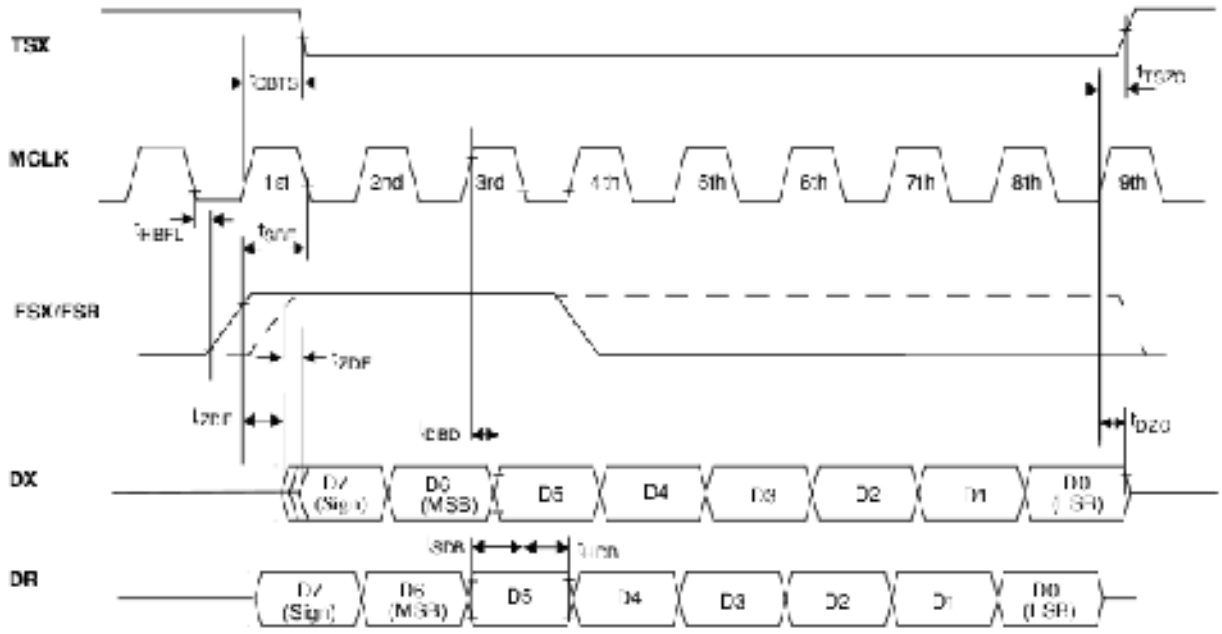


Illustration 5: CODEC Timing Requirements

are not exploited. Instead to reduce costs, only a limited number of modes are available.

To reduce the required number of IO pins, E0 is tied high via pull up resistors. There is no way to change this configuration. This means that only Switch Hook Detect (SHD) modes are available. Ground Key Detect (GKD) modes are not available and should be ignored.

All the F1 line are tied together and pull low via a external resistor. The F1 lines are also connected to PORT C, PIN 6 on the processor, this allows putting all the modules into a power denial mode by setting the pin as an output and driving it high.

The ALM pin of the TIM modules are tied together and also connected to PORT C, PIN 6. By configuring the pin as an input the current alarm state can be read.

PORT C, PIN 6 under normal operation should be configured as an input. When a thermal alarm occurs, the ALM pin will go high automatically causing F1 to go high allowing the processor to put the TIM modules into a powered denial state.

The modes each TIM module can be in (under normal operations) can be seen in Table 6.

F0	F1	Mode
0	0	Low Power Standby
0	1	Forward Active
1	0	Ringing
1	1	Forward Loopback

Table 6: TIM Modes

These modes are sufficient for developing even an advanced telephone system.

SOFTWARE

Overview

The example software has been split into a number of modules which can all be reused when developing new software. These modules abstract away all the complicated parts of the board, making the programmers life much easier.

All code has been designed to be compiled with the MPLAB IDE suite. An example MPLAB project file is included in the code package.

Shift Register Software Module

The shift register software module is divided into two sections, a module which is generic for writing to a 74HC595 register (74hc595.inc), and a module for dealing with the specific registers found on the board (shift.inc).

As the Strobe Register functions quite differently from the other registers on the board it is not covered by this code. The CODEC Software Module covers the software for dealing with this register.

74hc595.inc

The 74hc595.inc file provides a macro which will write the byte currently in WREG to the an external 74HC595 register and then cause the byte to be copied to the output latch.

The macro requires that all the pins be on the PIC output port. The pins which are given must be set up to be output pins before using the macro for correct operation.

As it is a macro, no call stack is needed. The macro requires no RAM and the WREG will also be preserved. The macro expands to 50 PIC instructions.

The calling convention for this macro follows,

```
_74hc595_send port, D0, SCLK, RCLK
```

shift.inc

The shift module is for dealing with specific shift registers. It defines 6 macros, 3 for shifting WREG into the shift registers, and 3 for setting them up. It is dependent on the macro from 74hc595.inc file.

This file is the only place where the pin mappings for these registers appear.

These macros require no PIC RAM and preserves WREG. The macros take no arguments.

It also includes two alias, shift_30 which maps to shift_f0 and shift_47 which maps to shift_f1.

Example usage for these macros follows,

```
; Set up the PIC pins to output
setup_tst
movlw       0xff
; Send 0xff to TST Register
;This will take 50 instructions
shift_tst
```

```
; Set up the PIC pins to output
setup_30
movlw       0x7A
; Send 0xff to f0 Register
; IE TIM Lines 3 to 0
;This will take 50 instructions
shift_30
```

```
; Set up the PIC pins to output
setup_74
movlw       0xff
; Send 0xff to f1 Register
; IE TIM Lines 7 to 4
;This will take 50 instructions
shift_74
```

LED Software Module

The Led software module consists of one file *led.inc*.

led.inc

This file contains 3 macros for making code more readable. Instead of doing a bit set in the middle of code, a useful name is instead used. As these are macros which expand to a single instruction, there is no reason not to use them.

Each led macro can be called with either "ON", "OFF" or "TOGGLE". Which will do exactly what they sound like.

Example usage follows below,

```
; Turn the Orange LED on
led_orange  ON
; Turn the Red LED off
led_red     OFF
; Toggle the Green LED
led_green   TOGGLE
```

CODEC Software Module

The CODEC module is one of the most advanced modules. It is made up of one file, *codec.inc* but depends on the Shift Register Software Module.

codec.inc

As the CODEC has very tight timing requirements, it is recommended that extreme care be taken when modifying this file. Comments in the file give more information about how to make sure the requirements are met.

The CODEC module requires 38 bytes of RAM for correct functioning. **It also reserves the FDR0 set of indirect registers**. **The SPI interface will also be used solely for the CODEC communication and cannot be multiplex. As well, it must be the only module to have a high priority interrupt.**

The CODEC ISR takes 35 instructions to execute and will occur at a rate of 64kHz. This effectively reduces the processing power available to the rest of the application by $1/8^{th}$.

The CODECs are set up by calling the codec_setup macro. This macro takes no arguments and will set all the required interrupts, TRIS bits and memory values. Once the set up is called, the CODEC ISR routine will start occurring straight away.

The CODECs are abstracted to a line interface. The actual CODEC which a line appears on does not need to be worried about.

Each line has two bits, the first is if the line is power on, the second is where the CODEC is getting data for that line.

To change the power state of a line, the codec_linex macro must be called. This macro takes one argument, which can be either ON or OFF. After any change to codec_linex values, the codec_commit macro must be called. This will send the values to the CODECs, this allows multiple changes to be committed at once. A code example of this follows,

```
; Turn on Line 1
codec_line1       ON
; Turn off Line 2
codec_line2       OFF
; The CODEC's power state has
; not change yet.

; Send the values to the codec
codec_commit
```

Each line also has a source of data. The possible sources are;

- CODEC_ZERO, continually output zero
- CODEC_INCF, a continually incrementing number
- CODEC_RING, output data from the ring table
- CODEC_TALK, take the data from another line
- CODEC_EXTERNAL, take the data from a memory location

To change the status of each line, call the *codec_set* macro. The following code samples show how this is done. Unlike the power down status, this change occurs straight away and the next sample will come from the appropriate source.

As an interrupt could occur at any time, it is only possible to go from CODEC_ZERO into CODEC_TALK or CODEC_EXTERNAL.

The following code shows how to change between various modes.

```
; Set Line 0 to INCF mode
codec_set 0, CODEC_INCF

; Do some other stuff

; Now we want to go to
; CODEC_TALK with data
; from Line 4
; ---------------------------
; We must return to CODEC_ZERO
; before entering the new mode
codec_set 0, CODEC_ZERO

; Set the source to Line 4
codec_source 0, 4

; Set the mode
codec_set 0, CODEC_TALK
```

As a register can only address a byte of RAM the EXTERNAL source must be in the first 255 bytes of memory.

TIM Software Module

The TIM module hides the fact that the TIM modules are connected to shift registers. Like the CODEC Module it abstracts the interface so that it is line based. This makes code much simpler and easier to read.

tim.inc

The TIM module takes 2 bytes of RAM and consists only of macros.

Each TIM module can be in one of the following modes,

- TIM_STANDBY, the TIM module is in a lower power standby mode, waiting for the phone to be picked up

- TIM_RINGING, the TIM module is in a mode to generate the high voltage ringing signal.

- TIM_ACTIVE, the TIM module is in transmission mode. When on hook this is used for Caller ID transmission. When off hook this is the talk sound.

To change the TIM module into various different states the *tim_set* macro is used. Like the CODEC module, these changes don't take effect until the *tim_commit* macro is called.

The following code sample shows how this is used.

```
; Set Line 0 to Active mode
tim_set 0, TIM_ACTIVE

; Set Line 6 to Standby mode
tim_set 7, TIM_STANDBY

; Send the modes to the modules
tim_commit
```

Appendix I - Component Datasheets

Microchip18F4455, *High-Performance, Enhanced Flash, USB Microcontrollers*

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010293

http://ww1.microchip.com/downloads/en/DeviceDoc/39632c.pdf

Intersil HC55185, *VoIP Ringing Subscriber Line Interface Chip*

http://www.intersil.com/cda/deviceinfo/0,0,HC55185,0.html

http://www.intersil.com/data/fn/fn4831.pdf

National TP3094, *COMBO Quad PCM Codec/Filter*

http://www.national.com/pf/TP/TP3094.html

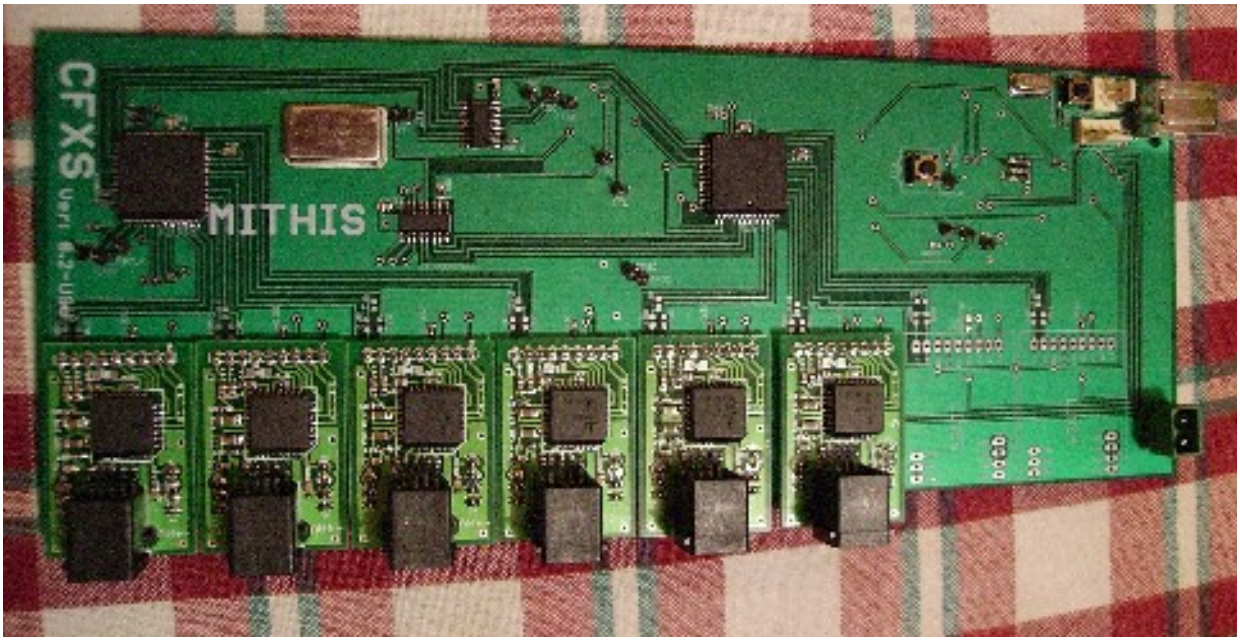
Generic 74HC595, *8 Bit Serial-In, Parallel-Out Shift Register with 3 State Output Register*

http://focus.ti.com/docs/prod/folders/print/sn54hc595.html

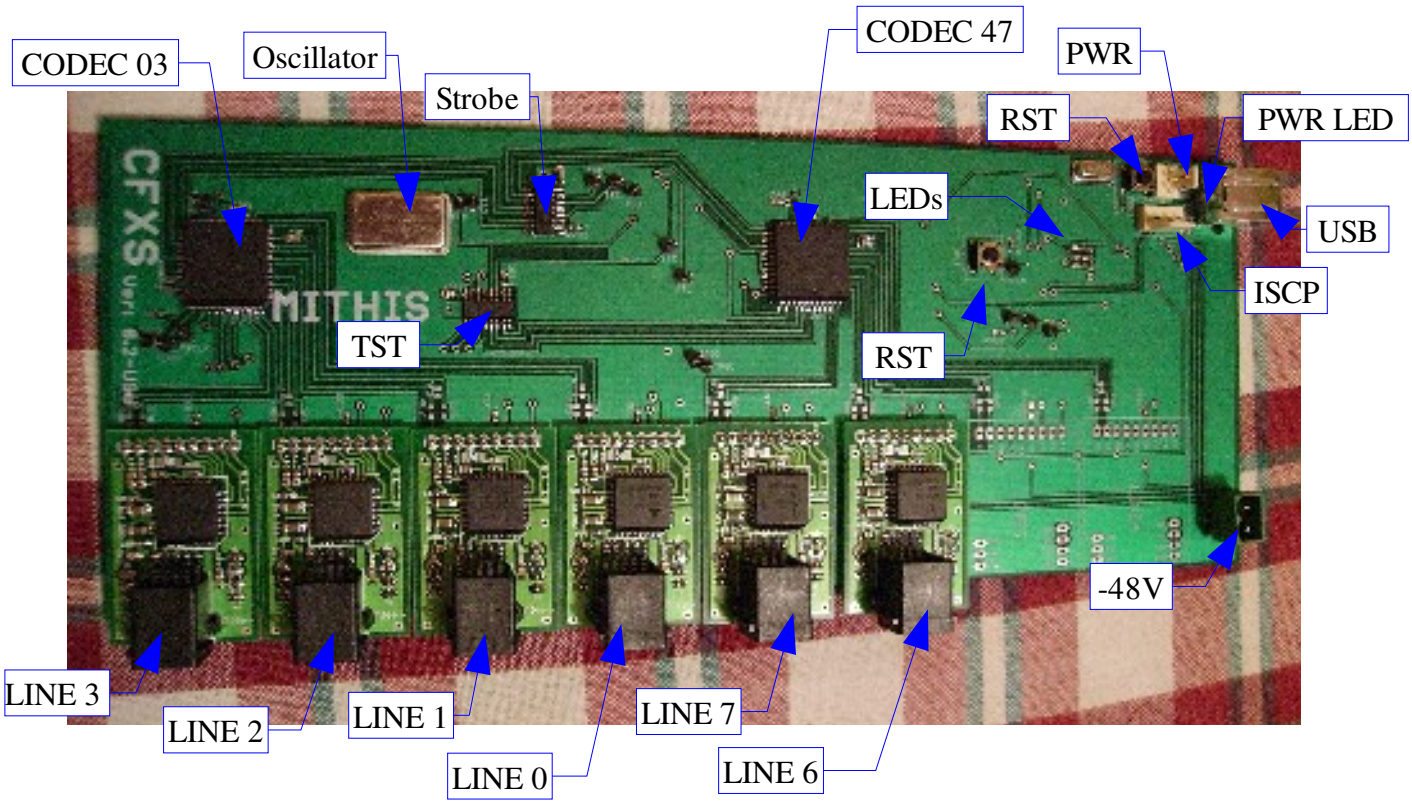
http://www.ti.com/lit/gpn/sn54hc595

APPENDIX II - PICTURES OF PROTOTYPE BOARD

Unannotated Picture



Annotated Picture



Appendix III - Test and Sample Code

Miscellaneous Tests

LED Flash Test

This test is used to exercise all LEDs on the prototype board. This test should only be used when no TIM modules are connected and is only useful on prototype boards which have all LEDs installed. It is also a good test to make sure all LEDs are placed around the right way. For a video of the output please see the data package.

USB Interface Test

This test is used to exercise the USB module and connections.

The device will appear as a USB HID device which will continually type “foobar”. If toggle switches are installed then pressing the switch will cause the HID device to turn caps-lock on.

Ringing Outputs

There are two important parts of producing a ringing signal. The following traces show output of the device when the system is correctly operating in ringing mode. In all these traces the following key applies:

Trace Colour	Signal
Green Trace	Tip
Purple Trace	Ring
Red Trace	Tip + Ring
Blue Trace	CODEC Output

Bell Ringing

When causing the phone's bell to ring, a signal of at least $40V_{rms}$ between 20Hz and 70Hz needs to be produced[6]. The system produces a trapezoid signal to maximise the output voltage change. Illustration 6 shows the the output of the device when ringing a TIM module. The RMS value and frequency of the output (as calculated by the oscilloscope) can be seen on the side.

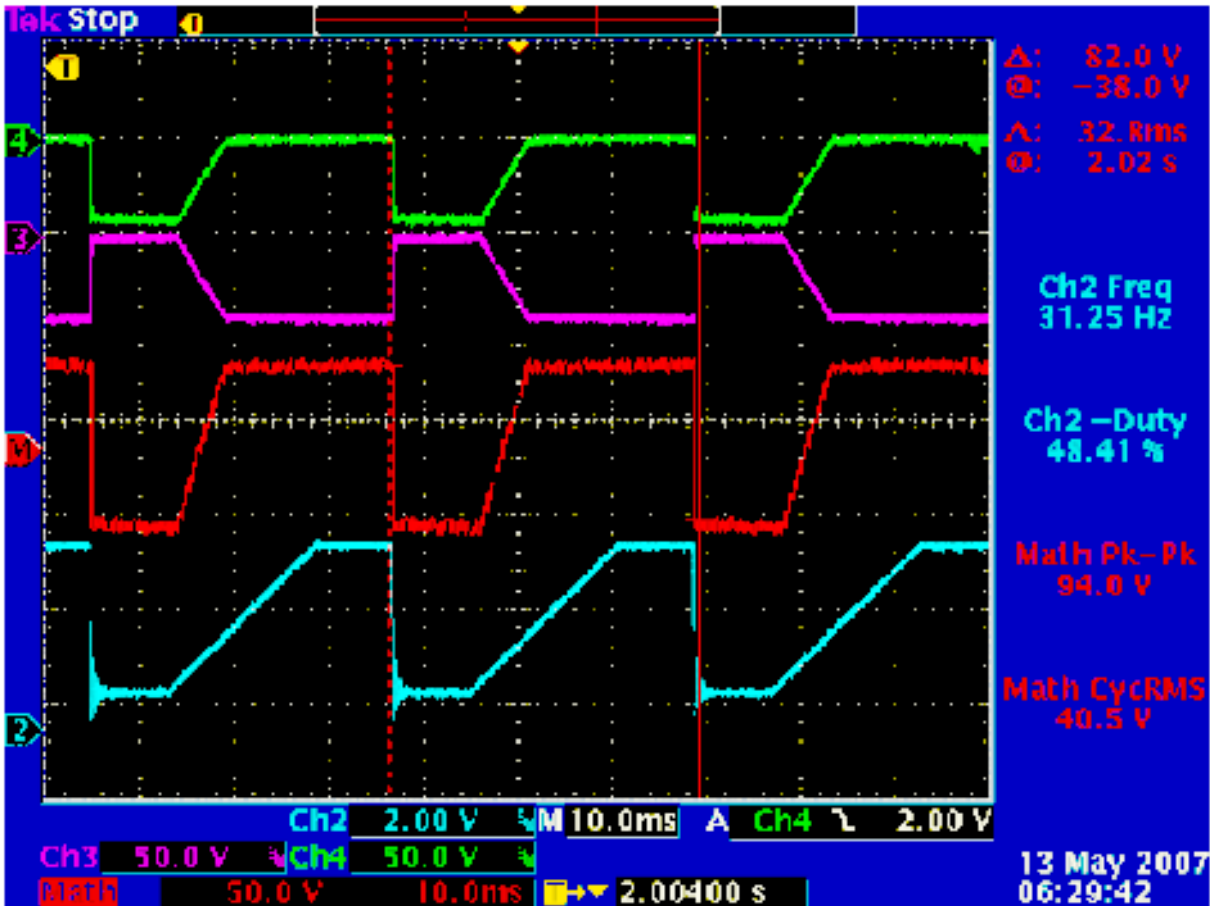


Illustration 6: Ringing Signal

6 ACIF, (Updated 2005) *"AS/ACIF S002:2005 - Analogue interworking and non-interference requirements for Customer Equipment for connection to the Public Switched Telephone Network"*, http://www.acif.org.au/documents_and_lists/standards/S002_2005

Ringing Tone

To indicated that a remote ringing phone is ringing, a tone must be produced of around 400Hz with slight modulation via a signal under 100Hz needs to be used[7]. This values to be outputted where calculated using a MATLAB script as described in the "MATLAB Code to Generate Tables of Constants" section. Illustration 7 shows the output of the CODEC and TIM modules.

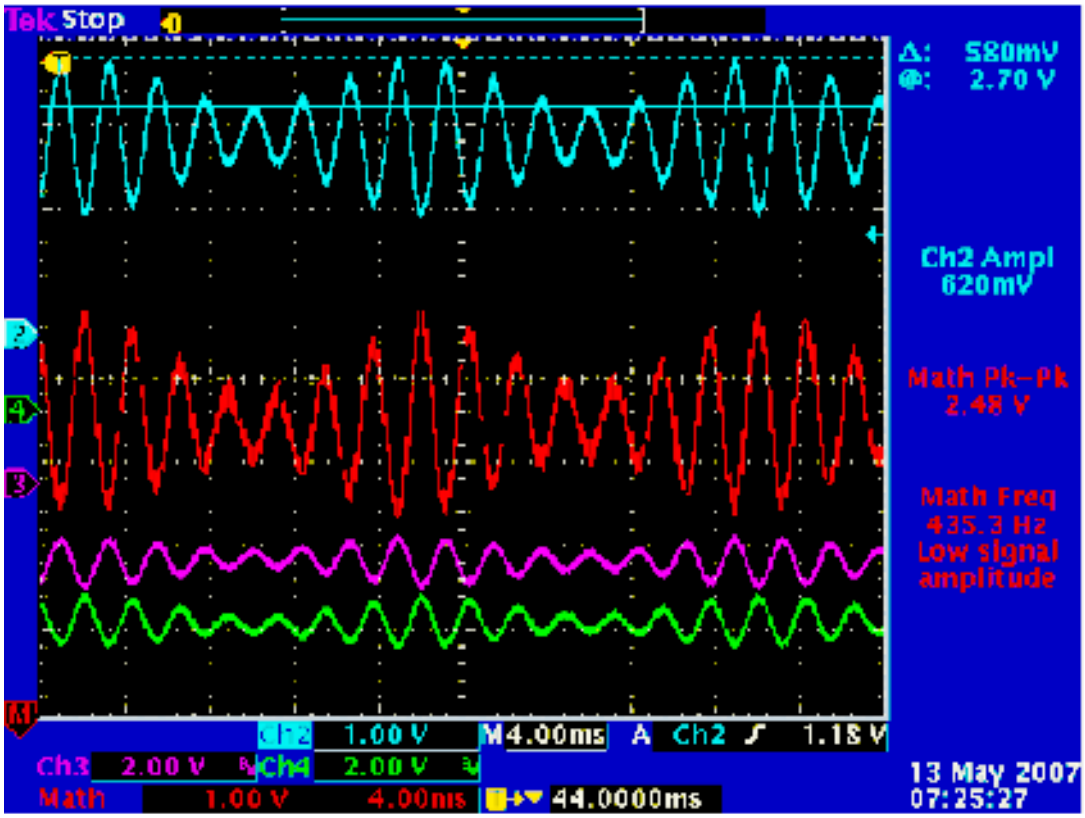


Illustration 7: Ring Tone Signal

7 ACIF, (Updated 2005) *"AS/ACIF S002:2005 - Analogue interworking and non-interference requirements for Customer Equipment for connection to the Public Switched Telephone Network"*, http://www.acif.org.au/documents_and_lists/standards/S002_2005

Candecense Patterns

The ringing output is modulated, consisting of on and silence periods. This is called the candecence, and a variety of patterns exist. The “PSTN Standard S002-2005” dictates that patterns should repeat every 3 seconds and consist of 200ms steps, this gives 15 bits per cycle. As this is not a power of two, 16 bits where used instead (giving a cycle of 3.2 seconds).

The candecense patterns are produce via a MATLAB script “ringpatterns.m” (this can be seen in Code 1), this script also produces a graph of the candecense patterns which can be seen in Illustration 8. Illustration 9 shows part of the candecense DR0 pattern as produced by the device.

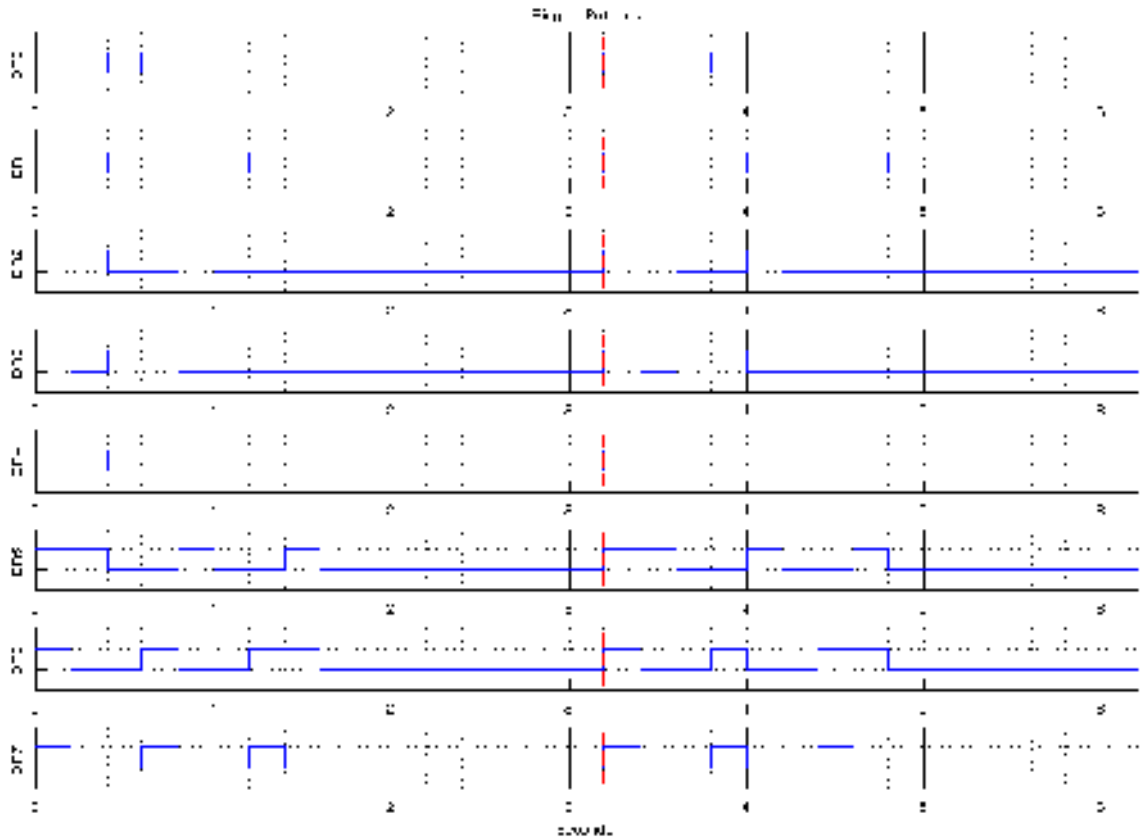


Illustration 8: Candecense Ringing Patterns

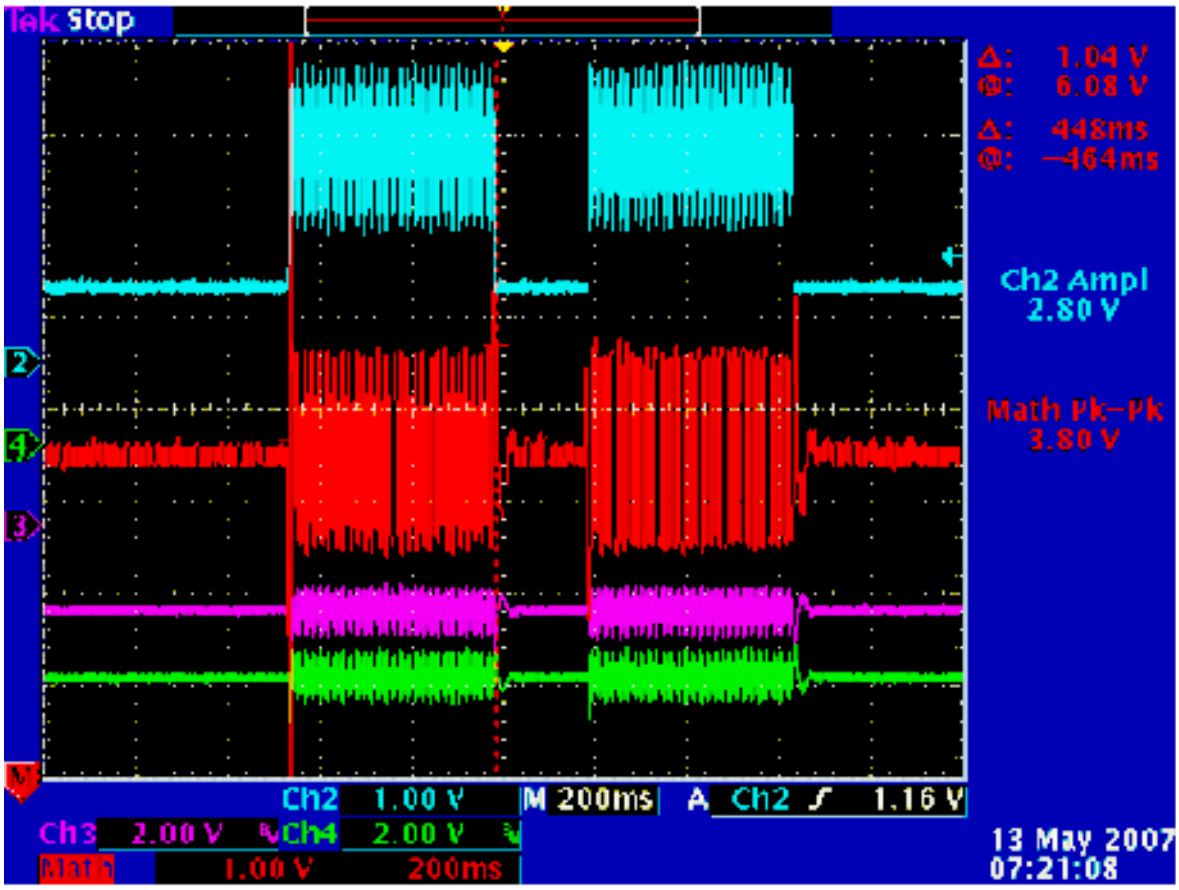


Illustration 9: Ringing Candecense

```
close all
clear all

% Related to as defined in PSTN Standard S002-2005
%
% Ringing Patterns
% - Cycles are 3.2 seconds long which gives 16 samples at 200ms
%   Technically it should only be 3 seconds long - but thats useless number

% Normal
% 400ms On, 200ms Off, 400ms On, ..
DR0 = [1 1 0 1 1 0]

% DR1
% 400ms On, 400ms Off, 200ms On, 200ms Off, 400ms On, ..
DR1 = [1 1 0 0 1 0 1 1]
```

```
% DR2
% 400ms On, 400ms Off, 200ms On, ...
DR2 = [1 1 0 0 1]

% DR3
% 200ms On, 200ms Off, 400ms On, ...
DR3 = [1 0 1 1]

% DR4
% 400ms On, 2600ms Off, ...
DR4 = [1 1]

% DR5
% 400ms On, 400ms Off, 200ms On, 400ms Off, 200ms On, ...
DR5 = [1 1 0 0 1 0 0 1]

% DR6
% 200ms On, 400ms Off, 200ms On, 200ms Off, 400ms On, ...
DR6 = [1 0 0 1 0 0 1 1]

% DR7
% 200ms On, 400ms Off, 200ms On, 400ms Off, 200ms On, ...
DR7 = [1 0 0 1 0 0 1]

patterns = {DR0, DR1, DR2, DR3, DR4, DR5, DR6, DR7}

% How many cycles to do
cycles = 2

for i = 1:length(patterns)

    % Pad the expression out to 16 samples
    l = length(patterns{i})
    patterns{i} = [patterns{i} zeros(1, 16-l)]
    subplot(length(patterns), 1, i)

    if i == 1
```

```
        title('Ringing Patterns');
    end

    % Samples
    samples = cycles*16-1

    hold on
    stairs(0:samples, [patterns{i} patterns{i}])

    set(gca,'XTick', 0:5:samples)
    set(gca,'XTickLabel', 0:(samples/5))
    set(gca,'XMinorGrid','on')

    set(gca,'YTick',[0 1])
    set(gca,'YTickLabel', {'', ''})

    for j = 0:16:samples
        plot([j j],   [-2 2], 'r--')
    end
    grid on
    axis([0 samples -1 2])
    ylabel(sprintf('DR%i', i-1))
    hold off
end
xlabel('Seconds');

fid = fopen('ringpatterns.inc','w');
fprintf(fid,  ';This  file  is  generated  from  MATLAB  -  please  don''t
modify\n\n');
for i = 1:length(patterns)
    fprintf(fid, '#define RING_DR%i 0x%02x\n', i-1, bi2de(patterns{i}));
end
fclose(fid);
```

Code 1: tone_400Hz.m – Ring tone generation

MATLAB Code to Generate Tables of Constants

MATLAB code was generated which converts a MATLAB data array to a table of constants suitable for usage in the PIC. This function is called “pictable.m” and the full code listing can be found in sample Code 3.

```
% pictable(file, data)
%
% Write the data out
%
function pictable(name, file, data)

    width = length(data)-1;

    if abs(round(log2(width)) - log2(width)) > 0
        error('Data must be a power of two!');
    end
    if length(width) > 2^8
       error('You have to much data! Must be less then 256 samples!');
    end

    fid = fopen(file,'w');
    fprintf(fid, '; This code is generated from MATLAB\n');
    fprintf(fid, '; DO NOT MODIFY!\n');
    fprintf(fid, ';\n');
    fprintf(fid, '; getsample_%s position\n', name);
    fprintf(fid, ';\n');
    fprintf(fid, '; Gets a sample from the table with position pointed via WREG\n');
    fprintf(fid, '\n');
    fprintf(fid, '.sample_%s      code                                    \n', name);
    fprintf(fid, 'getsample_%s:\n', name);
    fprintf(fid, '        local sampledata, sampledata_end                \n');
    fprintf(fid, '                ; Clear the table pointer               \n');
    fprintf(fid, '                clrf            TBLPTRU                 \n');
    fprintf(fid, '                clrf            TBLPTRH                 \n');
    fprintf(fid, '                clrf            TBLPTRL                 \n');
    fprintf(fid, '\n');
    fprintf(fid, '                ; Mask the incoming value               \n');
    fprintf(fid, '                andlw           0x%02x                  \n', width-1);
    fprintf(fid, '                ; Setup the Low byte                    \n');
    fprintf(fid, '                addwf           TBLPTRL, F              \n');
```

```
fprintf(fid, '\n');
fprintf(fid, ';        if high(sampledata_end) != 0                    \n');
fprintf(fid, ';          if high(sampledata) != high(sampledata_end)   \n');
fprintf(fid, '                ; Watch out for overflows               \n');
fprintf(fid, '                btfsc           STATUS, C               \n');
fprintf(fid, '                incf            TBLPTRH, F              \n');
fprintf(fid, ';          endif                                         \n');
fprintf(fid, '\n');
fprintf(fid, '                ; Setup the High byte                   \n');
fprintf(fid, '                movlw           upper sampledata        \n');
fprintf(fid, '                addwf           TBLPTRH, F              \n');
fprintf(fid, ';        endif                                           \n');
fprintf(fid, '\n');
fprintf(fid, ';        if upper(sampledata_end) != 0                   \n');
fprintf(fid, '                ; Setup the High byte                   \n');
fprintf(fid, '                movlw           high  sampledata        \n');
fprintf(fid, '                movwf           TBLPTRH                 \n');
fprintf(fid, ';        endif                                           \n');
fprintf(fid, '\n');
fprintf(fid, '                ; Get this sample                       \n');
fprintf(fid, '                tblrd*                                  \n');
fprintf(fid, '                movf            TABLAT, W               \n');
fprintf(fid, '\n');
fprintf(fid, '                return\n');
fprintf(fid, '\n');
fprintf(fid, '\n');
fprintf(fid, '.sampledata_%s  code           0x2000                   \n', name);
fprintf(fid, 'sampledata\n');
for i = [1:length(data)/2]
     fprintf(fid, '               db 0x%02x, 0x%02x\n', data(i*2-1), data(i*2));
end
fprintf(fid, 'sampledata_end\n');
fprintf(fid, '\n');
fprintf(fid, ';        if upper(sampledata) != upper(sampledata_end)\n');
fprintf(fid, ';             error ''The upper bytes of the table must match!''\n');
fprintf(fid, ';        endif\n');
fclose(fid);
```

Code 2: pictable.m – Code for outputting PIC code from MATLAB data

MATLAB allows easy generation of various wave forms in mu-law format. An example, which is used to create the table used by the ring tone, can be seen in example Code 3.

```
%
% Generate a 400Hz tone for various tones on the phone
%

fs = 8000

width  = 128;
cycles = 7;

% We are actually going to generate
% =~ 437Hz signal as that fits into 64 samples
samples_per_cycle = width/cycles;

freq = 1/(samples_per_cycle * (1/fs))
freq2 = 1/(width * (1/fs))

x = [0:width];
y = 0.75*(0.25*sin(2*pi*freq2*(x*(1/fs)))+0.5).*sin(2*pi*freq*(x*(1/fs)));

plot(x, y);
axis([0 width -1 1]);

% Convert to 2's comp Mulaw
y = lin2mu(y);

pictable('400Hz', 'tone-400Hz.inc', y);
```

Code 3: tone_400Hz.m – Ring tone generation

APPENDIX IV - PARTS LIST

Supplier	Part Number	Package	Part Description	No
Microchip	PIC18F4455-I/PT	TQFP 44	PIC18F4455	1
TI	SN74HC595D	SOIC 16	74HC595 – Shift Registers	4
National	TP3094	PLCC 32	TP3096 – CODEC	2
		603	Resistors – Various	45
		805	Capacitors – Various	13
		SOD83	Diode – Various	3
		603	LED – Various	3
		DIP 14	4.096MHz Oscillator	1
		603	LED	3
		HC18U	4MHz Crystal	1
			TIM Modules	8
		PN61729	USB Connector	1
			Power Connector	1

Table 7: Prototype Board Parts List

	PIN Header			1
	RJ12 Connector			1
HC55185ECR	SLIC, Ringing, VoIP, 75/85/100V, Gnd Start, LB = 53/58dB			1
	1206 Capacitor	4.7uF	CDC, CFB	2
	0805 Capacitor	0.1uF (Decoupling) – 50V	CPC, CPSH	2
		470nF (Filtering)	CRT, CRX, CTX	3
	0603 Resistor	47 (50 preferred)	RPR, RPT	2
		18k	RTL	2
		22k	RRT	1
		47k	RSH	1
		68k	RIL, RS	1
	SOT80 Diode		DALM	1

Table 8: TIM Module Parts List

Appendix V - Schematic

Prototype Board

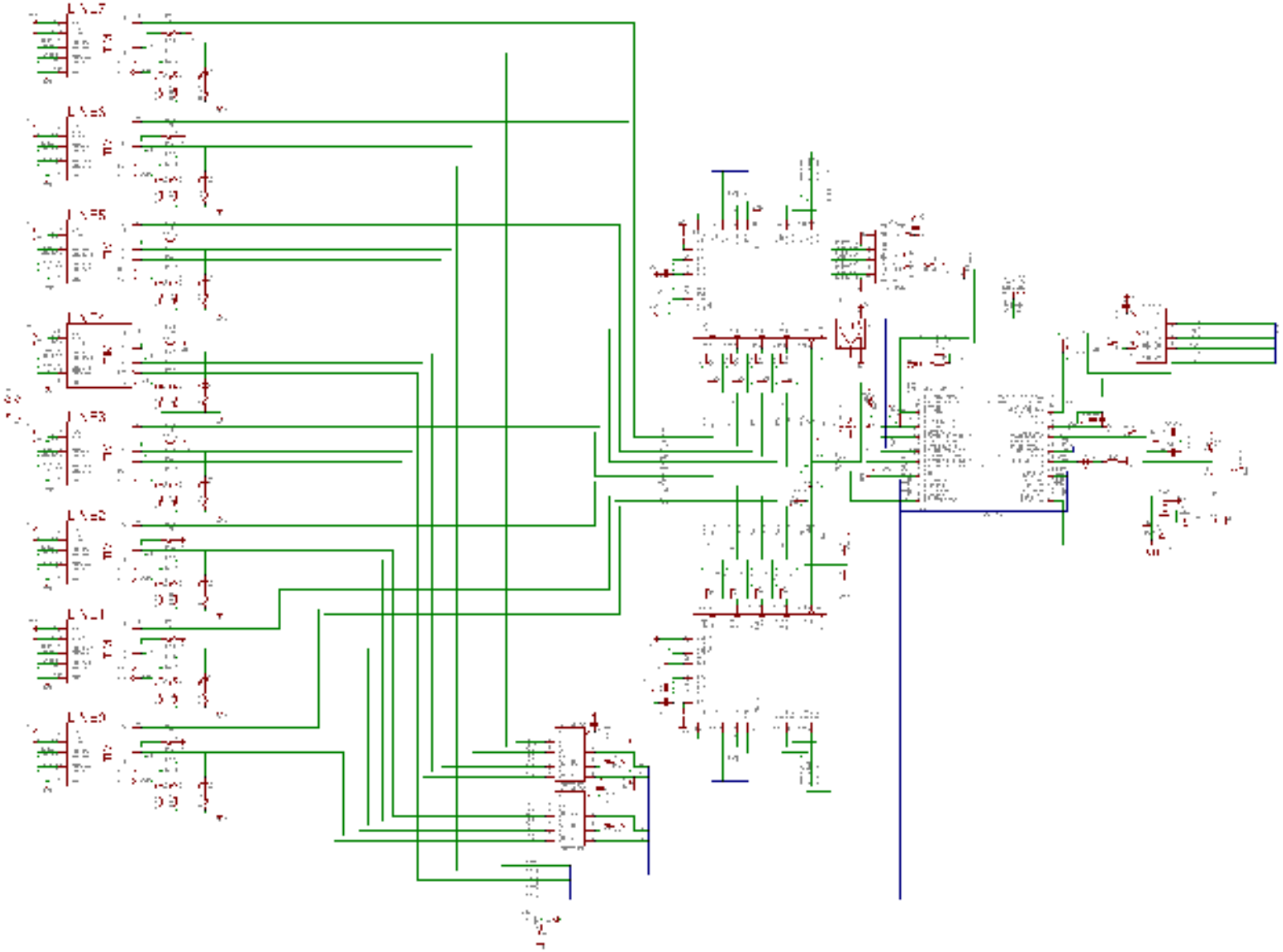


Illustration 10: Prototype Board Schematic

TIM Module

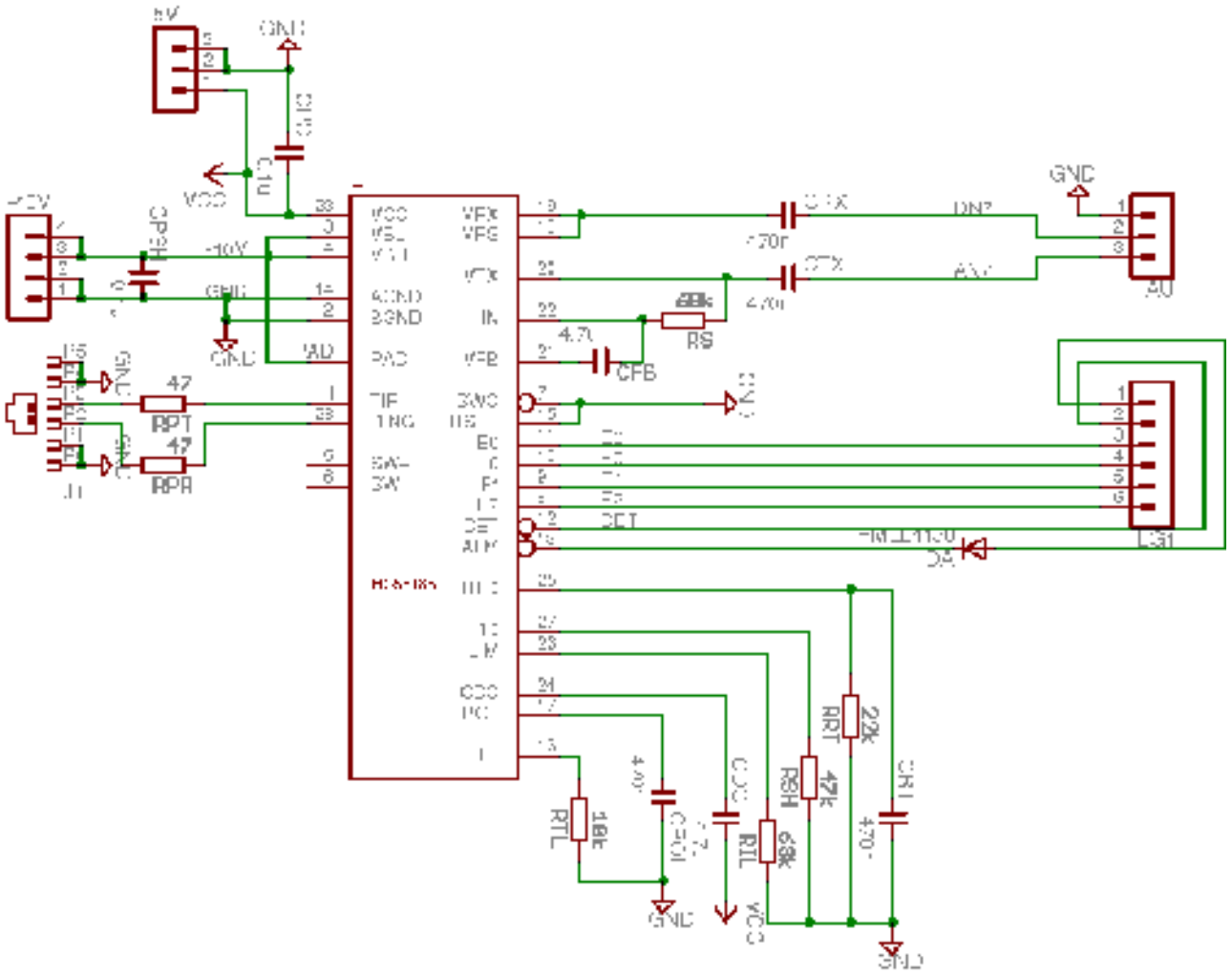


Illustration 11: TIM Module Schematic

Appendix VI - Annotated PCB Gerber Layout

Prototype Board Gerber

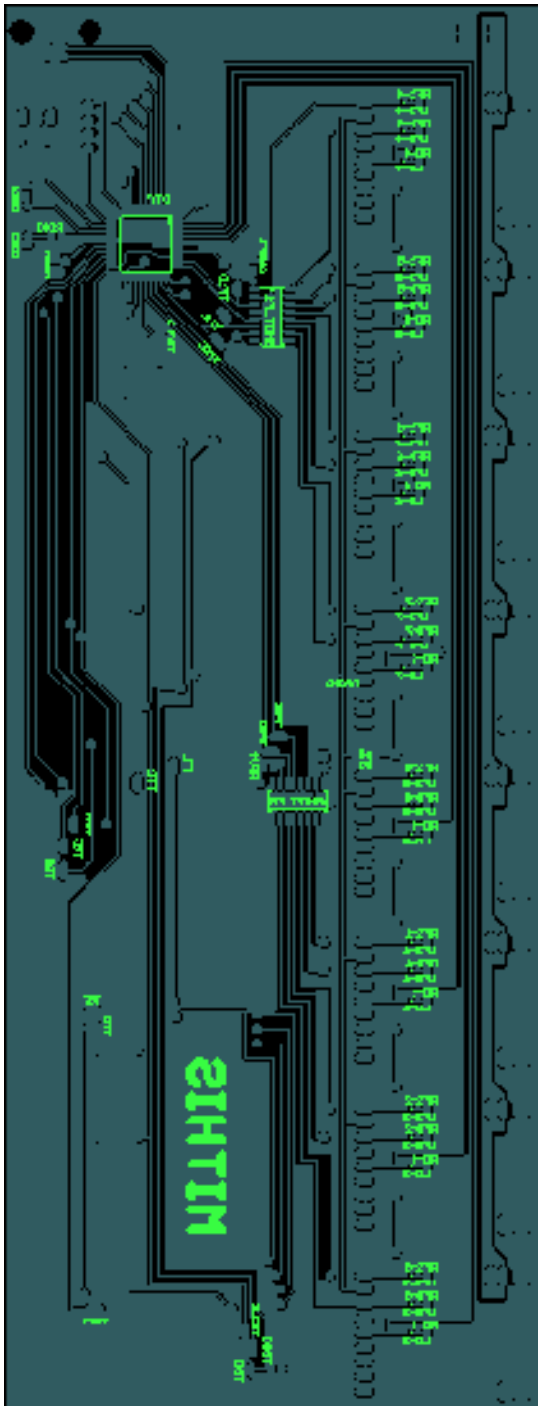


Illustration 12: Prototype Board Solder Side Gerber

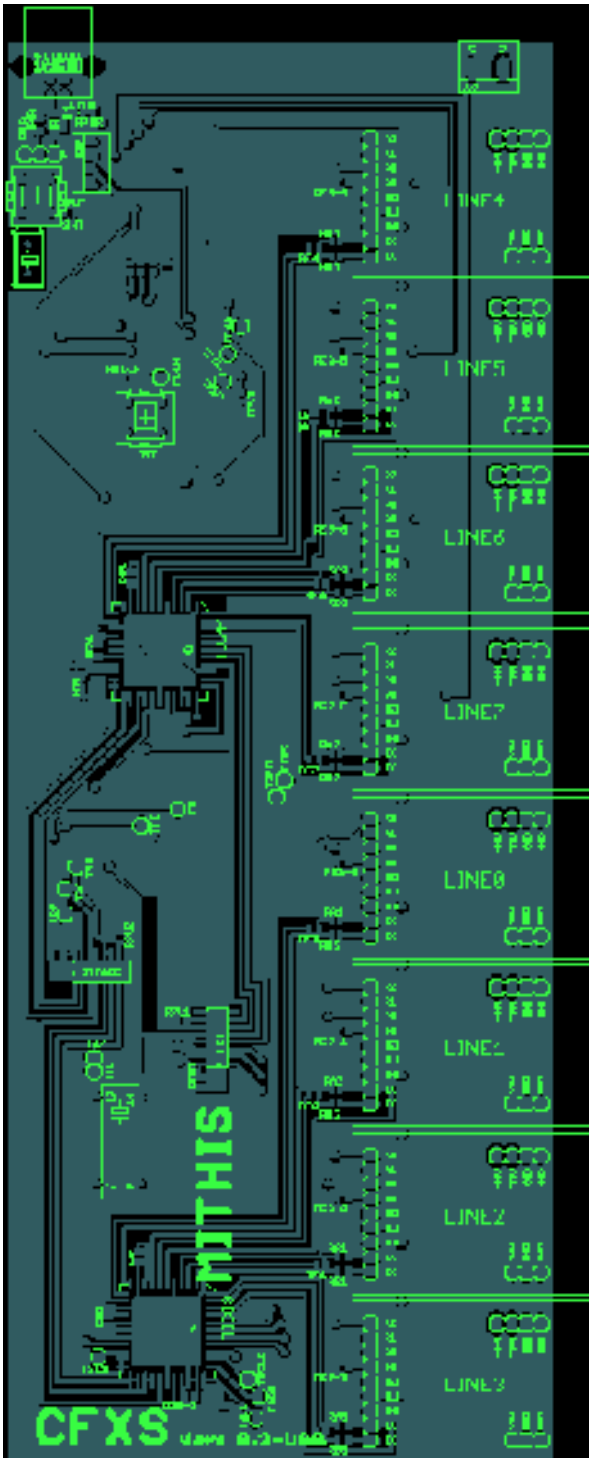


Illustration 13: Prototype Board Component Side Gerber

TIM Module

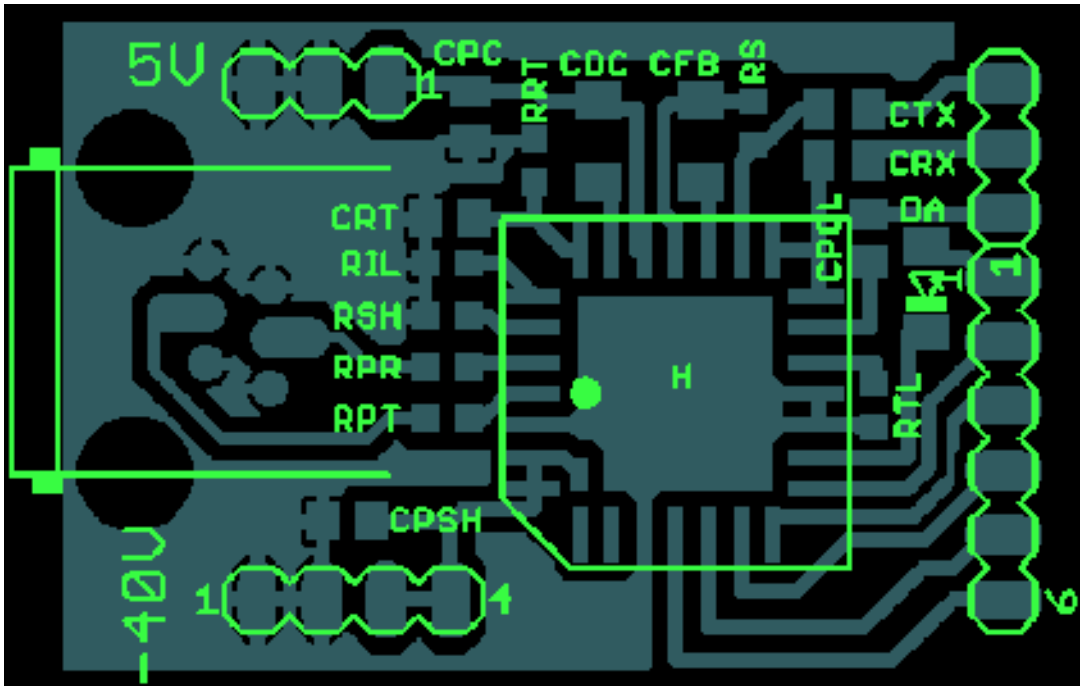


Illustration 14: TIM Module Component Side Gerber

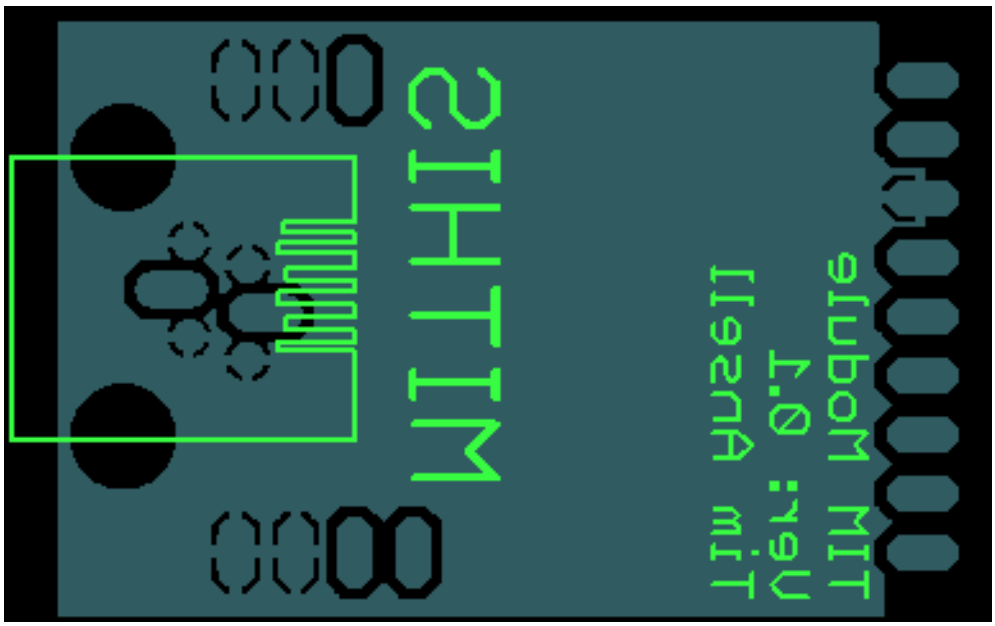


Illustration 15: TIM Module Solder Side Gerber

APPENDIX VII - LAW FORMULA

The Mu-Law and A-Law Formula are included below for your references.

Mu-Law

For the Mu-Law used in the Codec, μ = 255. The values are quantised as described in the G.711 ITU-T recommendation.

$$F^{-1}(y) = \mathrm{sgn}(y)(1/\mu)[(1+\mu)^{|y|} - 1] \qquad -1 \le y \le 1$$

$$F(x) = \mathrm{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)} \qquad -1 < x < 1$$

A-Law

For A Law used in the Codec, A = 87.6. The values are quantised as described in the G.711 ITU-T recommendation.

$$F(x) = \mathrm{sgn}(x)\begin{cases}\frac{A|x|}{1+\ln(A)}, & |x| < \frac{1}{A} \\ \frac{1+\ln(A|x|)}{1+\ln(A)}, & \frac{1}{A} \le |x| \le 1\end{cases}$$

$$F^{-1}(y) = \mathrm{sgn}(y)\begin{cases}\frac{|y|(1+\ln(A))}{A}, & |y| < \frac{1}{1+\ln(A)} \\ \frac{\exp(|y|(1+\ln(A))-1)}{A}, & \frac{1}{1+\ln(A)} \le |y| < 1\end{cases}$$

APPENDIX VIII - CONTACT INFORMATION

For all information on the device please contact,

TIM ANSELL
MITHIS

38 ARLINGTON TCE
WELLAND, 5007
SOUTH AUSTRALIA
AUSTRALIA

PH: +61 8 8346 773