A Homemade LD to DMTF Converter

Getting the Code onto the Microcontroller

From what I could tell the cheapest way to do this was via an Arduino. I got a Chinese Uno clone for less than three quid.

Assuming you've already got an Arduino and installed it's IDE software here's what to do.

Open the sketch `File > Examples > 11. ArduinoISP > ArduinoISP` and upload it to the Arduino.

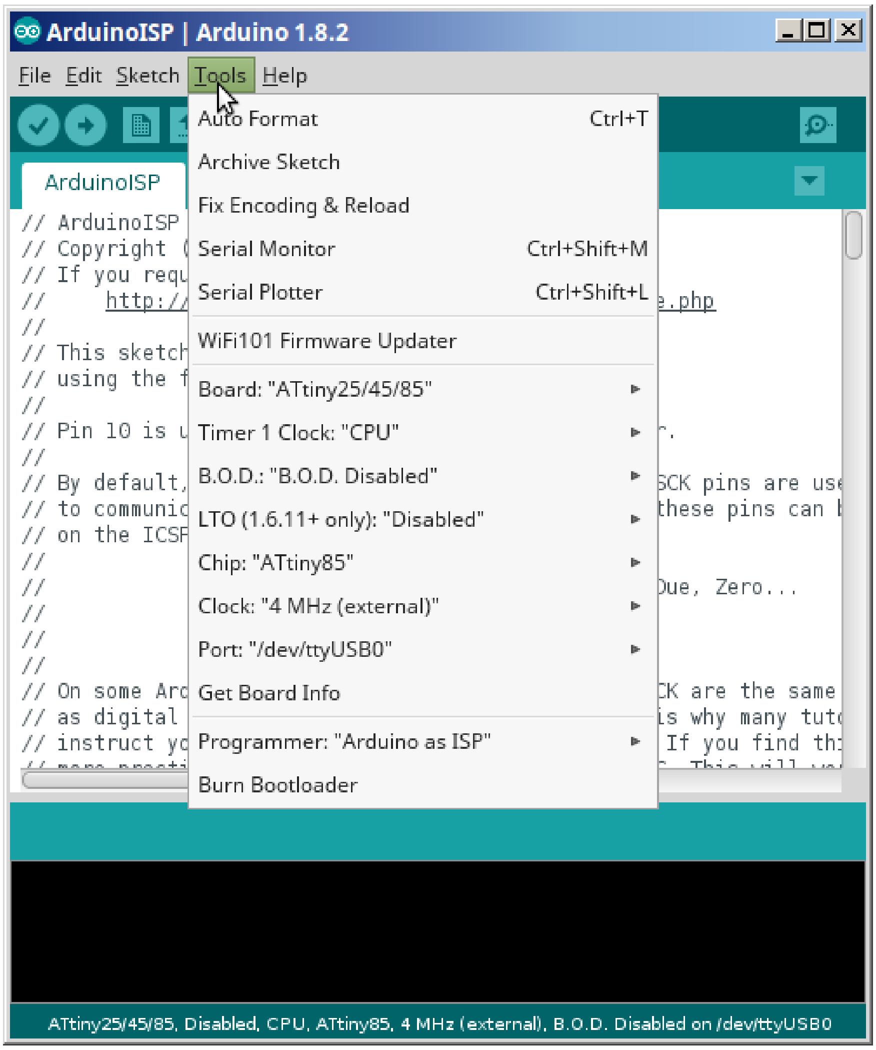
Go to `File > Preferences` and enter `http://drazzy.com/package_drazzy.com_index.json` into the `Additional Boards Manager URLs` box.

Go to `Tools > Board > Boards Manager`. Scroll down to `ATTinyCore by Spence Konde`, click it and then click `Install`.

Now go back to `Tools > Board` and choose `ATtiny 25/45/85`. Go to `Tools > Chip` and choose `ATTiny85`. Go to `Tools > Clock` and choose `4 MHz (external)`.

Finally go to `Tools > Programmer` and choose `Aurdiuno as ISP`.

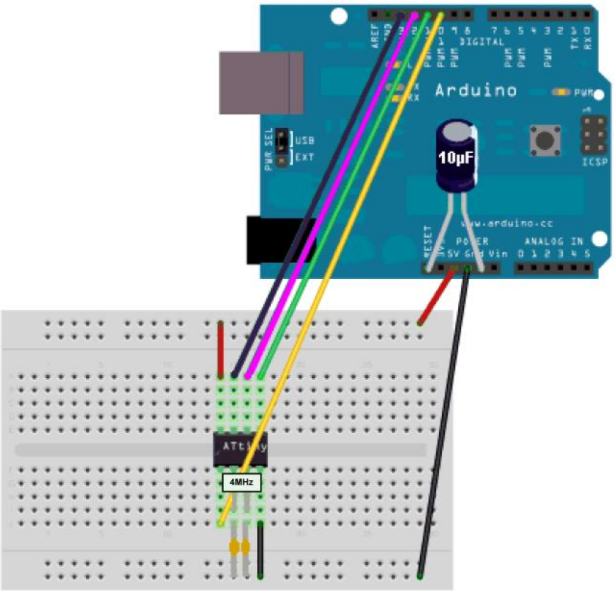
The `Tools` menu should now look like this, though the `Port` may be different:—



Board: "ATtiny25/45/85"
Timer 1 Clock: "CPU"
B.O.D.: "B.O.D. Disabled"
LTO (1.6.11+ only): "Disabled"
Chip: "ATtiny85"
Clock: "4 MHz (external)"
Port: "/dev/ttyUSB0"
Programmer: "Arduino as ISP"

ATtiny25/45/85, Disabled, CPU, ATtiny85, 4 MHz (external), B.O.D. Disabled on /dev/ttyUSB0

Next connect the ATTiny85, **plus its crystal and capacitors** to the Arduino. The circuit specifies 10pF capacitors but my crystals said to use 20pF so I did. The crystal goes between Pins 2 and 3 and a capacitor is connected between Pin 2 and Ground and another between Pin 3 and Ground. Also connect a 10µF capacitor between RESET and GND on the Arduino.



Now click `Tools > Burn Bootloader`.

Extract the file `main.hex` from the ZIP file and put it in your Home directory. On Windows that will be something like `C:\Users\Joe` or `/home/joe` on Linux.

Open a Command Prompt/Terminal and enter a command **SIMILAR** to one of these; your Home directory isn't going to `joe`, the Arduino IDE software may have been installed elsewhere and the port could be different. The commands are also in `AVRDUDE.TXT` in the ZIP for easy cutting and pasting. NB Linux commands are case-sensitive as is AVRDude on either system!

On Windows:—

```
"C:\Program Files\Arduino\hardware\tools\avr/bin/avrdude" -C"C:\Program
Files\Arduino\hardware\tools\avr/etc/avrdude.conf" -v -pattiny85 -cstk500v1 -
PCOM4 -b19200 -Uflash:w:main.hex:i
```

On Linux:—

```
/home/joe/arduino-1.8.2/hardware/tools/avr/bin/avrdude -C/home/joe/arduino-
1.8.2/hardware/tools/avr/etc/avrdude.conf -v -pattiny85 -cstk500v1 -
P/dev/ttyUSB0 -b19200 -Uflash:w:main.hex:i
```

I'll try to explain what this lot means to help making the necessary changes.

`"C:\Program Files\Arduino\hardware\tools\avr/bin/avrdude"` or `/home/joe/arduino-1.8.2/hardware/tools/avr/bin/avrdude` tells the computer to run AVRDude.

`-C"C:\Program Files\Arduino\hardware\tools\avr/etc/avrdude.conf"` or `-C/home/joe/arduino-1.8.2/hardware/tools/avr/etc/avrdude.conf` tells AVRDude to use the configuration file that the Arduino IDE software has already created and which must work or we wouldn't have got this far.

`-v` is Verbosity, ie tell it to let you know what's happening.

`-pattiny85` tells it that we're programming an ATTiny85.

`-cstk500v1` tells it that we're using a Atmel STK500 programmer with Version 1.x firmware, which is what the Arduino is emulating.

`-PCOM4` or `-P/dev/ttyUSB0` tells it which port the programmer (ie the Arduino) is connected to.

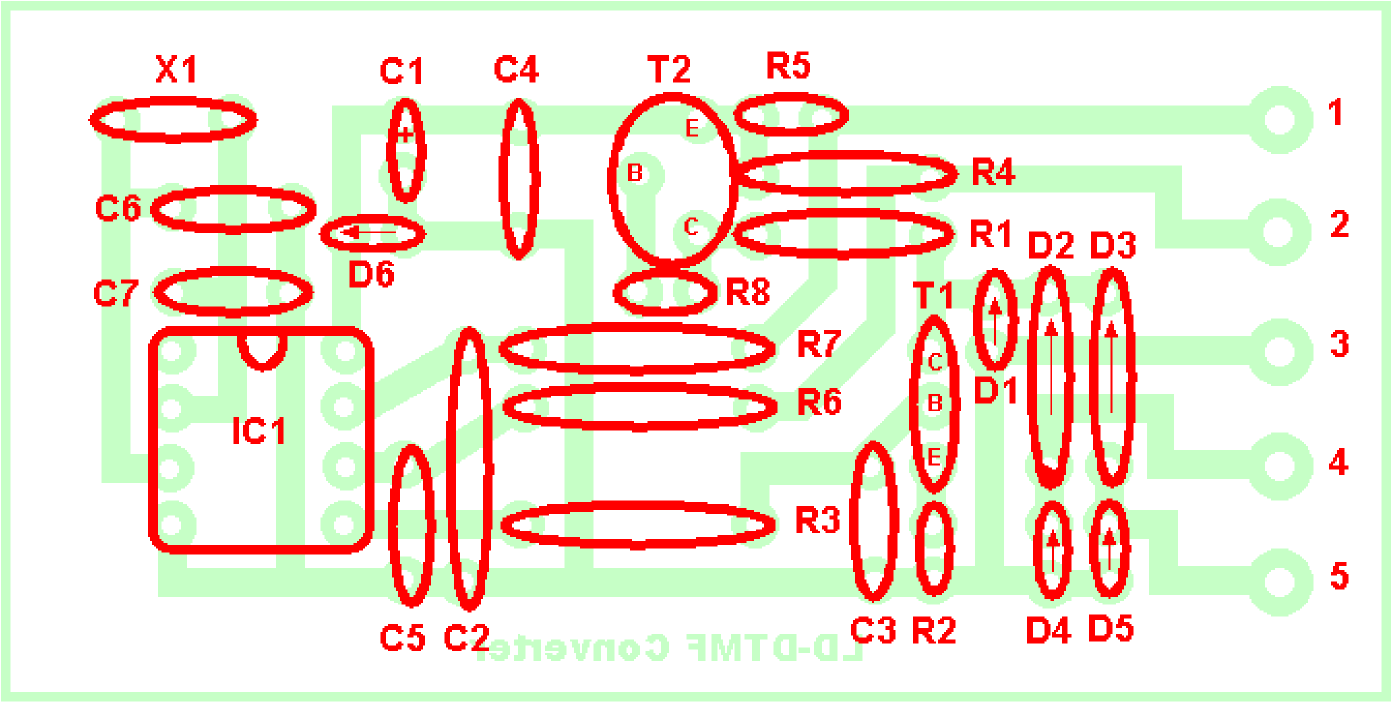
`-b19200` sets the baud rate.

`-Uflash:w:main.hex:i` tells it to write `main.hex` to the chip's flash ROM.

Hopefully the Tiny85 is now ready for use. If something has gone wrong it may go into a sulk and refuse to talk to you anymore. If this happens there are details in the ZIP on how to reset it so you can start again.

Construction

The components should be installed on the PCB as below:—



R1 1kΩ
R2 100Ω
R3 4.7kΩ
R4 10kΩ
R5 10kΩ
R6 10kΩ
R7 10kΩ
R8 10kΩ

C1 100μF
C2 0.1μF
C3 0.1μF
C4 0.1μF
C5 0.1μF
C6 10pF*
C7 10pF*
C8 0.33μF

* or as recommended by datasheet for X1

D1 P6KE22A
D2-D5 1N4007
D6 3.9V Zener
T1 BC337
T2 BC337
IC1 ATTiny85

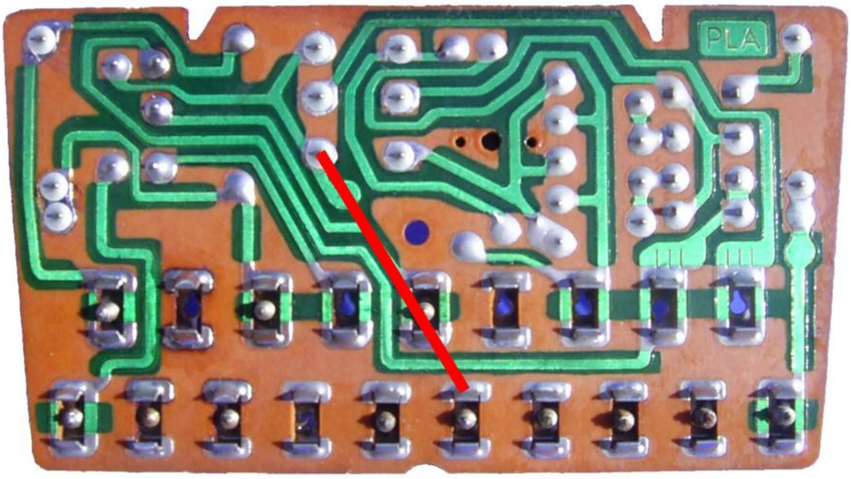
X1 4MHz Crystal

Installation

After building the circuit it needs installing in a telephone.

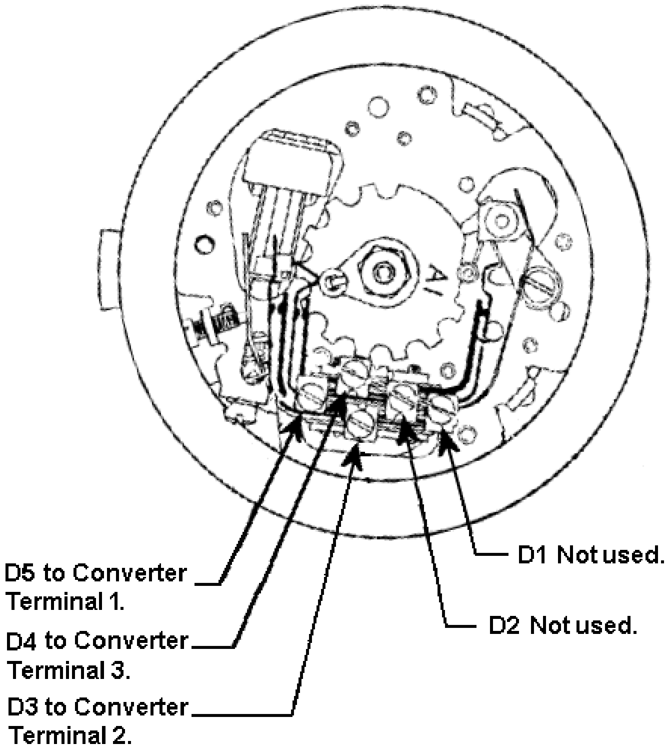
Firstly remove the dial and its associated wiring. Connect T8 and T10 with two 4.7V. zener diodes connected nose-to-nose. These old 'phones will happily gobble all the power they can get and these put it on a diet meaning it will get all it needs but leave some for the converter.

Later 746s had extra terminals labelled T19A and T19B. Early ones don't. The easiest fix is to use one of the spare terminals, say T14. Solder a bit of wire between the underside of that and Pin 4 of the Induction Coil.

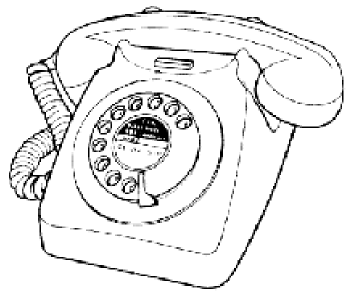


Connect terminal 1 on the converter to terminal 5 of the dial and terminal 2 to terminal 3. Connect dial terminal 4 to terminal 3 of the converter. Finally connect converter terminal 4 to T8 and terminal 5 to T19A or T14. See diagrams below.

Dial Connexions — All Instruments

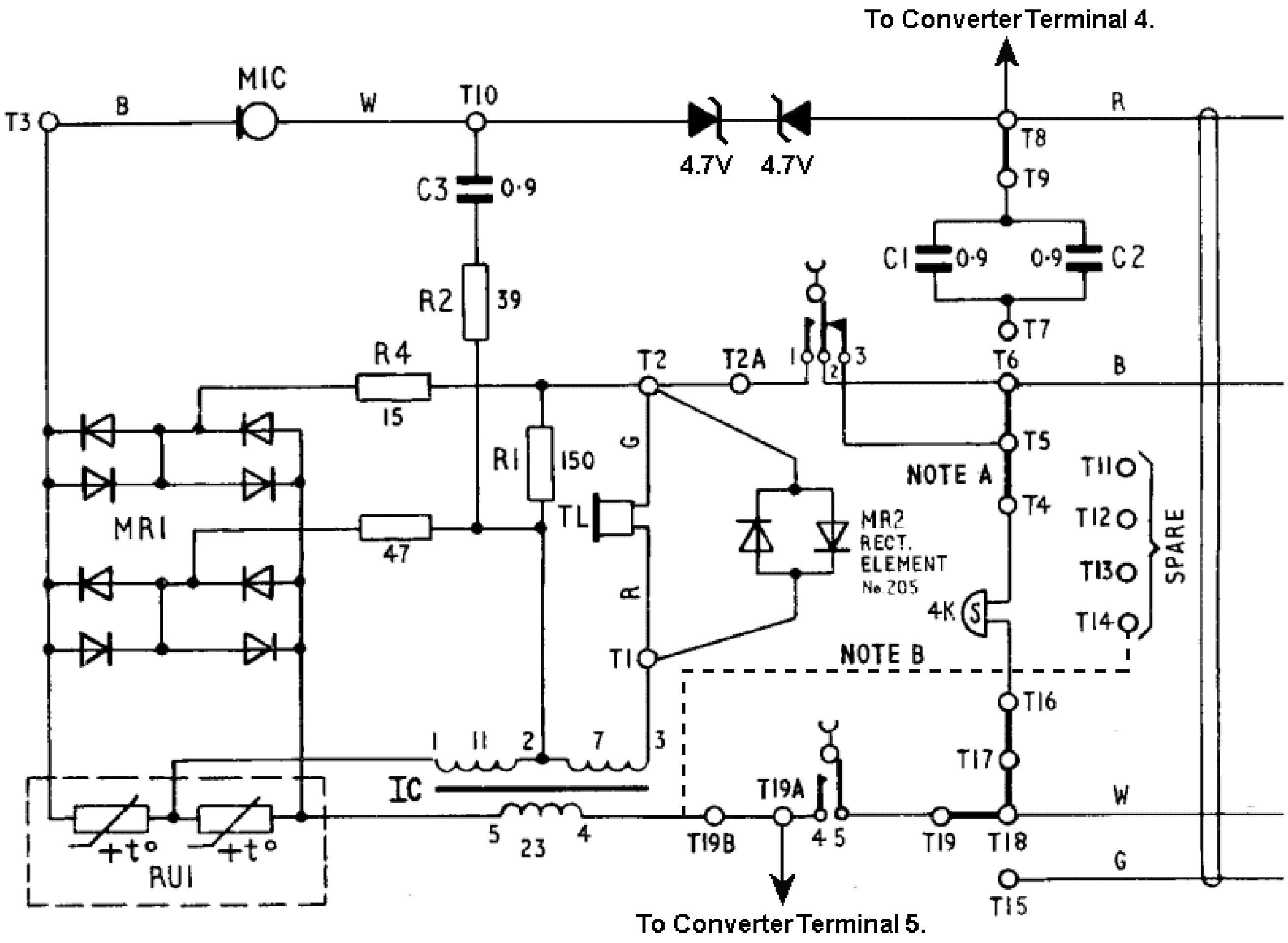

A1
D5 to Converter Terminal 1.
D4 to Converter Terminal 3.
D3 to Converter Terminal 2.
D1 Not used.
D2 Not used.

TELEPHONE No.746F

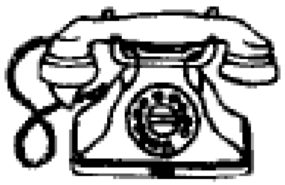


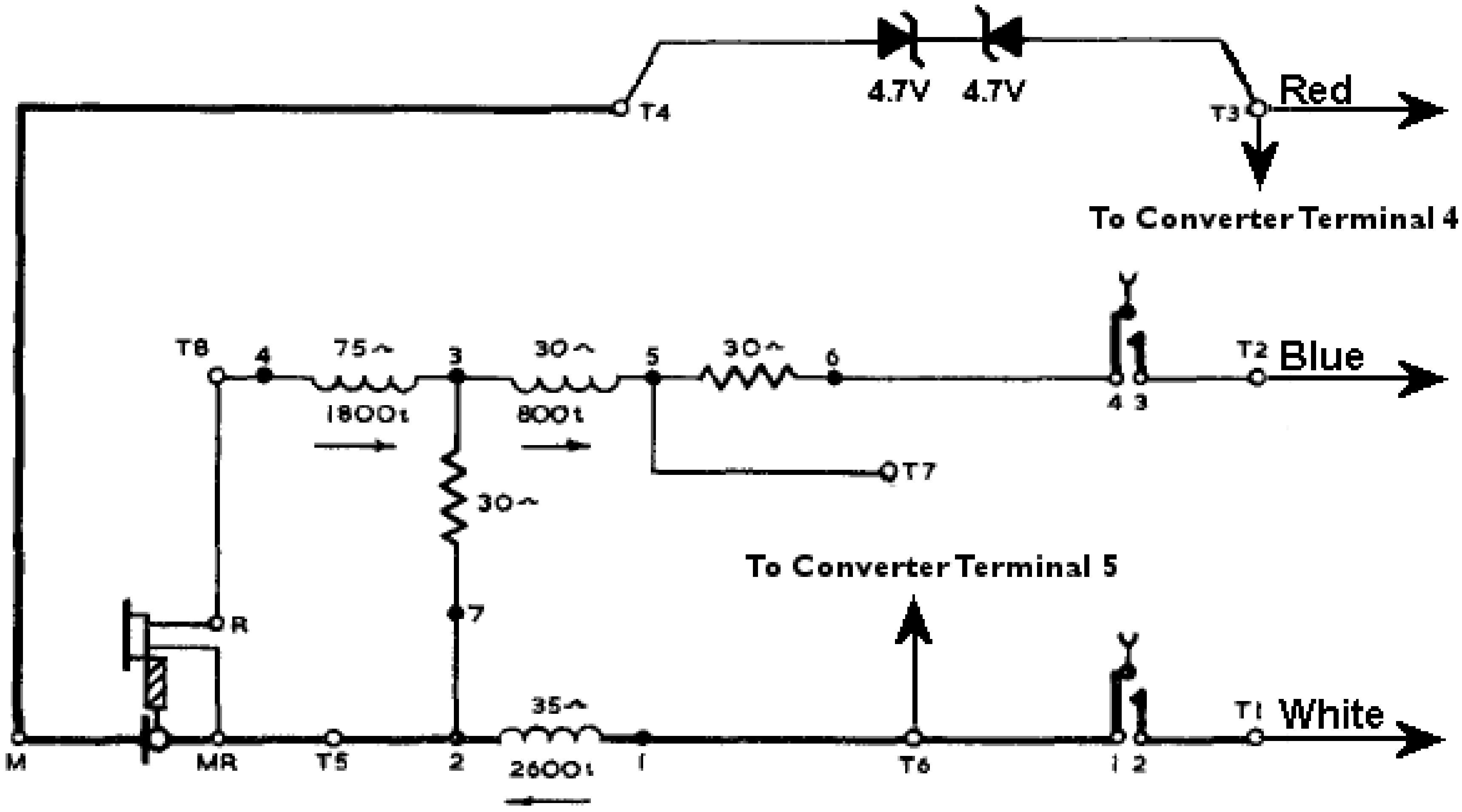
NOTE A
If instrument is fitted with 500 Ohm bell coils replace link T4—T5 with 3.3K resistor.

NOTE B
Connect wire as shown if instrument not fitted with terminals T19A and T19B.
Connect Converter terminal 5 to T14.

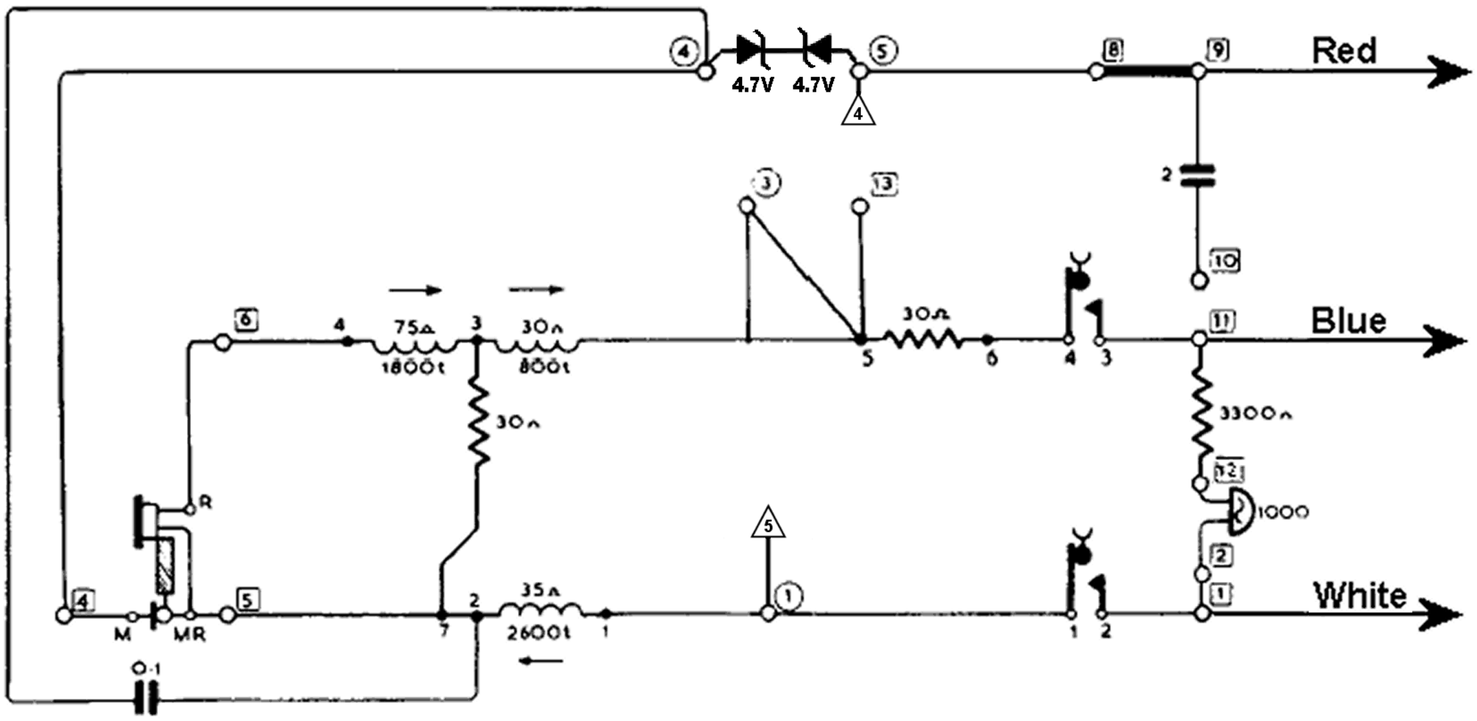


TELEPHONE No.232 & 1/232...





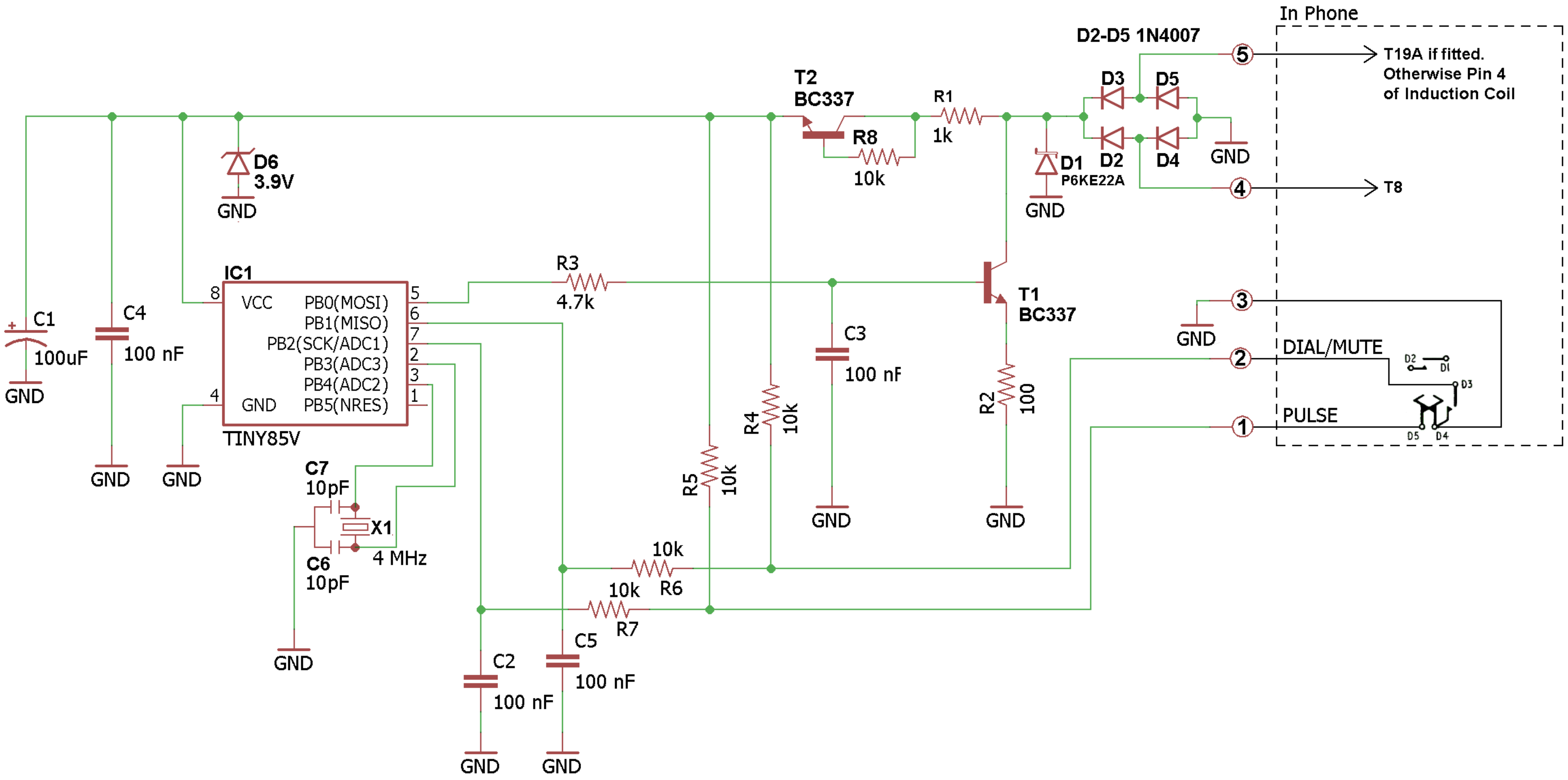
TELEPHONE No. 332



Direction of windings indicated by arrow thus →

TERMINAL POINTS	
□	Telephone Terminal Strip
○	Dial Strip
△	Converter

LD-DTMF CONVERTOR WITH POLARITY PROTECTION BRIDGE



Notes by Arnie Weber:

- Based on work by Boris Cherkaskiy (http://boris0.blogspot.ca/2013/09/rotary-dial-for-digital-age.html)
- Reduced R1 from 2k to 220R to minimize voltage droop when dial/pulse switches are closed.
- Added filtering to debounce switch inputs.
- Swapped Pins 6 and 7 on ATTiny85 to allow pulse counting to trigger on pin change interrupt (INTO).

Notes by Joe Freeman:

- Changed diagram to illustrate connecting to GPO 746 telephone.
- Added Bridge (D2-D5) to allow functionality independent of line polarity.
- Added Transient Voltage Suppression diode (D1).
- Added T2 and R8. Changed zener diode (D6) from 5.1V to 3.9V.
- Changed R1 to 1k
- Reduced R2 from 330R to 100R as ATA didn't always detect tones.
- Reduced C1 from 220μF to 100μF.

Operation

To use this devise you simply dial as normal and when dial returns to rest it will send the appropriate DTMF tone. It will however do more.

To dial a *, dial 1 and hold the dial against the stop until you hear a beep. Let go and when the dial has returned it will send a *.

To dial a #, dial 2 and hold the dial against the stop until you hear a beep. Let go and when the dial has returned it will send a #.

Numbers 3 to 9 can be used for speed dialling. To set these up dial 0 and hold it until you hear a beep. When the dial has returned it will play a tune. Now dial the figure you wish to program; it will then play a shorter tune. Now dial the number you wish to store. When you've finished dial and hold 0 again.

To call a stored number dial and hold the figure you stored it under.

As I said at the beginning, I wanted this to use with an ATA; a Linksys PAP2 to be exact. As entering numbers via a dial is slower than by buttons I found I had to lengthen the Inter-Digit Timer valve to five seconds to be able to use it comfortably. This MAY mean that after dialling the final figure there's a noticeable pause before the call is connected, though if your Dial Plan is properly set-up there won't be.