

# An ultimatic adapter for iambic keyers

Kevin E. Schmidt, W9CF  
6510 S. Roosevelt St.  
Tempe, AZ 85283  
w9cf@arrl.net

## 1 Introduction

A few years ago, my daughter Rebecca, KE7TEK, and I put together an Elecraft K2 kit. A great resource for any rig is the users' email list; I subscribed to the Elecraft email reflector<sup>1</sup> in case we needed help or advice during construction.

Recently several people on the email list commented that the built-in cw keyers in commercial rigs are iambic keyers – sometimes like the Elecraft rigs they give you the choice of Curtis mode A or Curtis mode B,<sup>2</sup> but generally not the older ultimatic mode. Since the new Elecraft K3, like other software defined radios, has downloadable firmware, the obvious, and cleanest, solution to this problem is to include a selectable ultimatic mode in the keyer firmware. This will probably be included in the future, but what if you want ultimatic mode now or want to use ultimatic mode with a rig which won't be upgraded? You have two choices: either you can use an external keyer that includes the ultimatic mode, for example the K1EL keyers,<sup>3</sup> or you can try to put together an adapter that will make the internal keyer act like an ultimatic keyer. The advantage of the adapter is that you can use the rig's front panel controls – the speed control and all of the rig's keyer memories are available. Rigs like the Elecraft K3 can transmit RTTY using the paddle input (but not the external key line) so that this feature becomes available to ultimatic users.

## 2 Ultimatic mode

John Kaye's, W6SRV, first ultimatic keyer design was published in QST in 1953.<sup>4</sup> He used a single lever paddle in his original design. His next version<sup>5</sup> used a dual paddle and it is this keyer's action that gives the ultimatic mode its name.

---

<sup>1</sup><http://www.elecraft.com/>

<sup>2</sup>When you have closed both paddles and release both of them during a dot or dash, Curtis A just completes that dot or dash, while Curtis B, completes that dot or dash and then sends the opposite element.

<sup>3</sup>Available at <http://www.k1el.com>.

<sup>4</sup>John Kay, W6SRV, *The "ultimatic" – the key with a memory*, February QST, p. 11 (1953).

<sup>5</sup>John Kay, W6SRV, *The all-electronic "ultimatic" keyer*, part I, April QST, p. 11 (1955), part II, May QST, p. 36 (1955). This design used 11 dual vacuum tubes.

While the ultimatic keyer used a dual paddle, unlike iambic keyers closing both the left and right paddles gave the same result as closing only the last one. For example, to send an x with the ultimatic mode you could hold closed the dash paddle for the first dash, then add the dot paddle for the middle two dots, releasing just the dot paddle then gives the last dash. With the ultimatic mode all letters except C can be made with one closure of the dot and/or dash paddles. Iambic keyers can make a C with a single squeeze, but need more than one closure for X and P. An ultimatic keyer can make a question mark, comma, and double dash,  $\overline{BT}$ , with one closure, while an iambic keyer can make periods, semicolons and  $\overline{AR}$ ,  $\overline{SK}$  and  $\overline{AA}$  with one closure.

After learning about the ultimatic mode, I thought it would be fun to try to come up with a circuit that would allow a standard iambic keyer to function like the ultimatic. Since you can build a whole keyer from a microcontroller, you can obviously use a microcontroller to do the job. However, this has the disadvantage that builders need to have a programmer for the microcontroller. The big advantage is that one small low power chip is all you need. An alternative is to use standard logic gates since the logic needed is rather simple. I was able to come up with a circuit that uses 6 NOR gates and an inverter, so two quad NOR gate chips will also work. In this article I will describe both routes.

A pair of quad NOR gates costs less than a dollar, so the adapter is inexpensive to build. Any microcontroller has more than enough power for this job, and the simplest ones also cost a dollar or less while the one I used in the end is around two dollars. Unless the adapter is built into the rig, connectors and the enclosure will cost more than the electronic parts.

Part of the fun of the design is working out the necessary logic. Here the design is practically a text-book example. If you just want to try out the circuit, then you can skip the logic design and just look at section 5 or section 6 for the prototype circuits. Just remember that you must have a PIC programmer to use the PIC microprocessor circuit.

### 3 Logic Design

Since closing the paddle grounds the input, I represent an open paddle as a 1 and a closed paddle as a 0 so that ground potential is a 0. I want to drive the keyer through isolating transistors which will invert the outputs. It makes sense to take the case where the output is on (i.e. the key line closed) as a 1 and the output off as a 0.

Since I use my left hand to operate paddles (I operate a bug right handed) but I have the paddles wired for right-handed use, it gets too confusing to talk about dot and dash inputs and outputs. Instead I will simply call them left and right inputs and outputs.

I write the left and right paddle inputs as  $L_{in}$  and  $R_{in}$ , and the corresponding adapter outputs as  $L_{out}$  and  $R_{out}$ . There are five possible states of the ultimatic:

1.  $L_{in} = 1, R_{in} = 1$  gives output  $L_{out} = 0, R_{out} = 0$ .
2.  $L_{in} = 1, R_{in} = 0$  gives output  $L_{out} = 0, R_{out} = 1$ .
3.  $L_{in} = 0, R_{in} = 1$  gives output  $L_{out} = 1, R_{out} = 0$ .

4.  $L_{in} = 0, R_{in} = 0$  after  $L_{in} = 1, R_{in} = 0$  gives output  $L_{out} = 1, R_{out} = 0$ .
5.  $L_{in} = 0, R_{in} = 0$  after  $L_{in} = 0, R_{in} = 1$  gives output  $L_{out} = 0, R_{out} = 1$ .

State 1 has both paddles open and neither key line keyed. States 2 and 3 have either just the left or just the right paddle closed and the corresponding key line is keyed. Finally States 4 and 5 have both paddles closed. They key the line corresponding to the last paddle closed.

The next step is to write what is called the transition state table. This is just a fancy name for writing the logic above in a tabular form for easier analysis. The adapter is in one of its 5 states when the input changes. The transition state table then gives the new state and outputs as shown in table 1. Each row corresponds to a current state given by the first entry. The other entries in the row correspond to the 4 possible inputs, and give the resulting state and the outputs. For example if we are in state 3, that is, the left paddle is closed and the right is open, and we close the right paddle. The input is now 0,0, and we will change to state 5 and the outputs are ( $L_{out} = 0, R_{out}=1$ ), so the entry corresponding to state 3 and inputs 0,0 is 5 (0,1). Notice that there is a “-” for state 1 with the input 1,1. That is because this corresponds to both paddles closing simultaneously and should never occur, so we don’t care what the adapter does.

We could produce a circuit to make our transition state table, but it would be more complicated than necessary. In fact, one reason to write the transition state table is to be able to see unnecessary redundancies. In table 1 rows 2 and 4 are identical and rows 3 and 5 are identical. Because the transitions from states 2 and 4 are identical, we don’t need to consider these as different states and can combine them into a single state. Similarly we can combine states 3 and 5 to make a single state. Row 1 has a dash in the last column and is identical to the other states everywhere else. It can be combined with either of the other two. (This in fact says that if we did somehow manage to close the paddles simultaneously, our choice here says it is like closing one before the other and our choice specifies which one we take to have closed first.) I choose to combine it with 2 and 4, the old states 1,2, and 4 then become new state 0, and the old states 3 and 5 become new state 1. Rewriting the transition state table using these new states gives table 2. This table is simpler and will give a simpler circuit.

Calling the state  $S$ , the truth table for the transition state table is shown in table 3. The truth table is constructed by writing for each current state value  $S_{now}$ , and input values  $L_{in}, R_{in}$ , the corresponding new state  $S_{next}$  and the output values  $L_{out}, R_{out}$ . For example from our reduced transition state table we see that if both paddles are closed  $L_{in} = R_{in} = 0$ , and the state  $S = 0$ , we keep  $S = 0$  and the outputs are  $L_{out} = 1, R_{out} = 0$ .

We now want to describe the truth table by a set of logic equations. Let’s try to figure out a logic equation for  $S$ . If we look at the rows where the  $S_{next}$  values are 1 (i.e. true), they all have  $L_{in} = 0$ . In addition, they have either  $R_{in} = 1$  or  $S = 1$  or both. The logic is  $S_{next}$  is (NOT  $L_{in}$ ) AND either  $R_{in}$  OR  $S_{now}$ . Standard notation uses a bar over  $L_{in}$  to indicate the NOT, AND is represented by the dot  $\cdot$ , and OR by the plus sign  $+$ . The logic equation becomes  $S = \bar{L}_{in} \cdot [R_{in} + S]$ . Repeating the analysis for  $L_{out}$  and  $R_{out}$  the logic equations representing the truth table can be written

$$\begin{aligned}
 S &= \bar{L}_{in} \cdot [R_{in} + S] \\
 L_{out} &= \bar{L}_{in} \cdot [\bar{S} + R_{in}] \\
 R_{out} &= \bar{R}_{in} \cdot [S + L_{in}] .
 \end{aligned} \tag{1}$$

## 4 Hardware design

Figures 1 and 2 show hardware implementations and some transforms using De Morgan's rules for the first two equations.<sup>6</sup> Using these transforms, the equations can be implemented using two NOR gates each. The third equation uses the same hardware as the second equation with the labels  $R$  and  $L$  interchanged and  $\bar{S}$  changed to  $S$ . The final simplified circuits using just NOR gates are given in Figures 1c, and 2c. An additional inverter is needed to get the inverted  $S$  input.

## 5 NOR gate circuit

Combining the logic gates for the three equations together gives the complete hardware implementation shown in figure 3.

Since I was just seeing if this would work and not trying to build a permanent adapter, I put the circuit together on a prototyping board. I had some CD4001 integrated circuits in my junk box, so I used those. My rather messy result is shown in figures 4 and 5. Ugly style construction could be used for something more permanent, or a circuit board laid out, but I have not done that.

## 6 Microcontroller implementation

I have a Microchip PICkit 2 programmer.<sup>7</sup> Since my shack computer runs linux, instead of Microchip's software I use the open source gputils<sup>8</sup> for an assembler and linker. Microchip provides compilable source code that will operate the PICkit 2 on Linux and Macintosh, and fully supported software for Windows. If you use Windows, you can simply use Microchip's software. Both Microchip's tools and the open source tools use compatible source code and produce the same hex output files used to program the microprocessors.

Just about any of the PIC microprocessors can be used. I initially tried a PIC10F200, one of the most basic PICs available. While it worked, I wanted the processor to sleep when it wasn't busy. According to its data sheet the PIC10F200 can take many milliseconds to wake up from sleep, and a wake up from sleep is like a reset and starts the program at its beginning with the processor reset to its default configuration. More powerful PICs can be programmed to wake up and continue processing the instructions after the sleep instruction. To simplify the programming and make the adapter more robust, I switched to one of these more powerful PIC microprocessors. The one I had in my junk box was a PIC16F690. It has much more power, more peripherals and more memory.

---

<sup>6</sup>You can easily verify De Morgan's rule that inverting the inputs to an AND gate gives the same truth table as a NOR gate. More information on digital basics and De Morgan's rules is given in the ARRL Handbook.

<sup>7</sup>Available from Microchip <http://www.microchip.com>. The PICkit 2 programmer costs about \$35. The PICkit 2 starter kit includes the programmer, software, demo board, and a tutorial. Current price is about \$50. If you have not programmed a PIC before, you should consider getting the starter kit. You can then test out the programming software with known good hardware on the demo board provided using the lessons that Microchip provides before building the ultimate adapter.

<sup>8</sup>Available from <http://sourceforge.net/projects/gputils/>

These are all unused and wasted for this simple application; many pins of the PIC16F690 are unused. The circuit diagram is shown in figure 6, and the source code files are given in the appendix.

The PIC16F690-I/P costs a couple of dollars directly from Microchip, with similar prices from Mouser, Digikey and other vendors. For this application you could use one of the 12F series 8-pin chips which would be a little cheaper and significantly smaller. You would need to modify the source code for the different chip or write your own from scratch – the program is very short and simple.

The main advantages of the PIC solution is having only one integrated circuit and on chip pull up resistors for the paddle inputs. This can help if space at a premium, for example when building the adapter inside a rig. A disadvantage is that the PIC has a clock oscillator, and this could produce rf interference, especially if mounted inside the rig near the receiving circuitry. I built my circuits so that they would drive transistors that would interface to the keyer input. My prototypes did not include rf bypass capacitors on the input and output lines. Adding these would be a good idea. I would try bypassing with 0.001  $\mu$ F monolithic capacitors, and of course the power supply line should be bypassed to keep from adding noise.

Since I had already done the logic design, I programmed the PICs to implement the truth table, table 3. The program has a variable for the state  $S$  which is initially set to zero and stored. The microprocessor then loops continuously reading its inputs and forming an index running from 0 to 7 corresponding to the binary number of the first three columns of table 3. The table entry corresponding to the binary number from the last three columns of table 3 is returned and used to set the outputs and the new state. The process repeats.

The PIC16F690 internal oscillator starts rapidly, so when the paddles inputs have not changed, the microprocessor can be put to sleep, and set to awaken when the paddle inputs change. Since the microprocessor clock does not run when it is sleeping, power consumption and rf interference should not be a major problem.

The PIC only needs to have isolating transistors on its output lines. The input lines are programmed to have internal pull-up resistors. Again, I used a couple of 2N3904 transistors with base current limiting resistors that I had on hand. The resistors could be eliminated and current consumption reduced by changing these to 2N7000 FETs. If you know that the keyer input voltages and currents are within specifications, it would be easy to modify the source code to use the PIC's tri-state I/O pins to change so a high impedance gives the keyer input line open case, and then pull the keyer input lines low for the closed case. This would eliminate the interface transistors.

Two photos of the prototyped circuit are shown in figures 7 and 8. The microprocessor is being powered by the PICkit 2 programmer through the computers USB port. Either a 3 to 5.5 Volt source could be used or 5 Volts obtained from a 12 Volt supply using a 78L05 three-terminal regulator.

## 7 Tests and conclusion

I have three iambic keyers available. An Accukeyer,<sup>9</sup> a CMOS super keyer III,<sup>10</sup> and the internal Keyer in the Elecraft K2. Both breadboarded circuits worked as expected on all three keyers –

---

<sup>9</sup>James Garrett, WB4VVF, *The WB4VVF accu-keyer*, August QST, p. 19 (1973).

<sup>10</sup>Jeff Russell, KC0Q and Bud Southard, N0II, *The CMOS super keyer 3*, August QST, p. 26 (1995).

turning them into ultimatic mode keyers.

Both the NOR gate and PIC microprocessor solutions work well. If your present keyer only does iambic modes, you want to try the ultimatic mode with your dual paddle, and don't want to buy a new keyer, you can breadboard either of these circuits in less than an hour. If you want to get started with microprocessor programming, putting together and programming the PIC16F690 is an easy first project.

## 8 Addendum

Shortly after I posted this article, John Crabtree, KC0G, sent me an email with several very useful comments. He pointed out that the discrete ultimatic adapter can be realized with 6 nor gates rather than 7 by including an additional state in the analysis.

His analysis exploits the left-right symmetry of the adapter. He keeps two of the equations I derived in Eq. 1

$$\begin{aligned} S &= \bar{L}_{in} \cdot [R_{in} + S] \\ R_{out} &= \bar{R}_{in} \cdot [S + L_{in}] \end{aligned} \tag{2}$$

and realizing the left-right symmetry, adds a state  $T$  to give

$$\begin{aligned} T &= \bar{R}_{in} \cdot [L_{in} + T] \\ L_{out} &= \bar{L}_{in} \cdot [T + R_{in}] \end{aligned} \tag{3}$$

The neat thing here is that the hardware implementation of  $S$  also gives  $T$ ; it is the output of  $U_{1b}$  in figure 3. His simplified circuit is shown in figure 9. As John said,

It is ironic that the introduction of an extra state leads to a simpler hardware design.

While John's improved circuit still requires two CD4001 or equivalent, it is esthetically much more pleasing since the circuit is now completely symmetric between the left and right paddle inputs and outputs.

John also points out that the unused inputs on the CD4001 chips should be grounded, and I have added this statement to the caption of figure 3.

# Tables

Table 1: The transition state table. In the entries, the first number is the new state, and the numbers in parentheses are outputs,  $(L_{out}, R_{out})$ . The dash indicates a state that should not occur so we do not care what the result is. The inputs are the values of  $L_{in}, R_{in}$

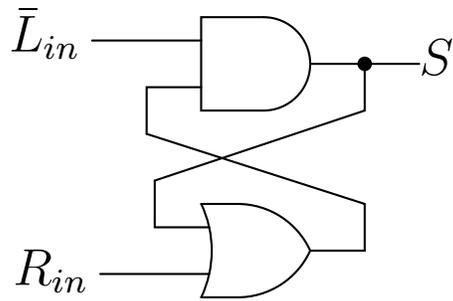
Current State \ inputs	1,1	1,0	0,1	0,0
1	1 (0,0)	2 (0,1)	3 (1,0)	-
2	1 (0,0)	2 (0,1)	3 (1,0)	4 (1,0)
3	1 (0,0)	2 (0,1)	3 (1,0)	5 (0,1)
4	1 (0,0)	2 (0,1)	3 (1,0)	4 (1,0)
5	1 (0,0)	2 (0,1)	3 (1,0)	5 (0,1)

Table 2: The reduced transition state table after combining equivalent states in table 1.

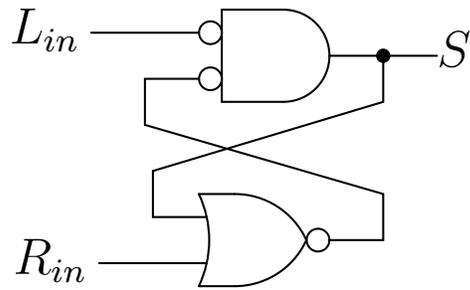
Current State \ inputs	1,1	1,0	0,1	0,0
0	0 (0,0)	0 (0,1)	1 (1,0)	0 (1,0)
1	0 (0,0)	0 (0,1)	1 (1,0)	1 (0,1)

Table 3: The truth table for the reduced transition state table.

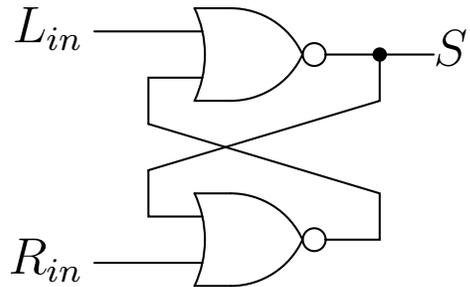
$S_{now}$	$L_{in}$	$R_{in}$	$S_{next}$	$L_{out}$	$R_{out}$
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	0	0	0



a. •

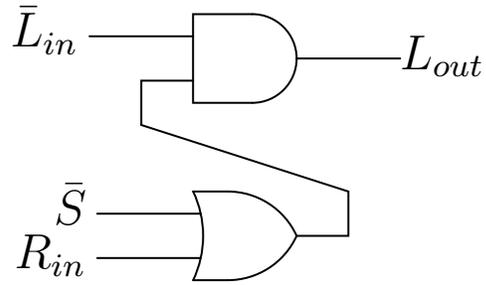


b. •

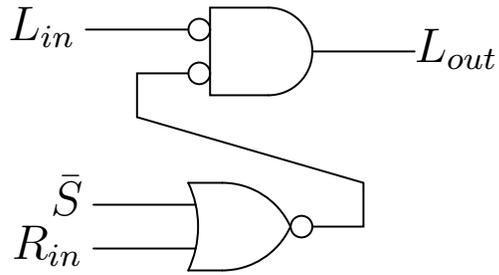


c. •

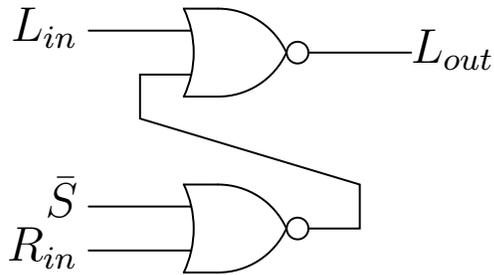
Figure 1: S equation realization. At a. the expression for  $S$  in Eq. 1 is used directly. At b. a negation at the input of the AND gate allows us to use  $L_{in}$ , while negating the output of the OR gate and the other input of the AND gate does not change the logic. At c. the negated input AND gate is changed to the equivalent NOR gate.



a. •



b. •



c. •

Figure 2: The  $L_{out}$  equation realization. At a. the expression for  $L_{out}$  in Eq. 1 is used directly. At b. and c. the same manipulations are used as in figure 1.

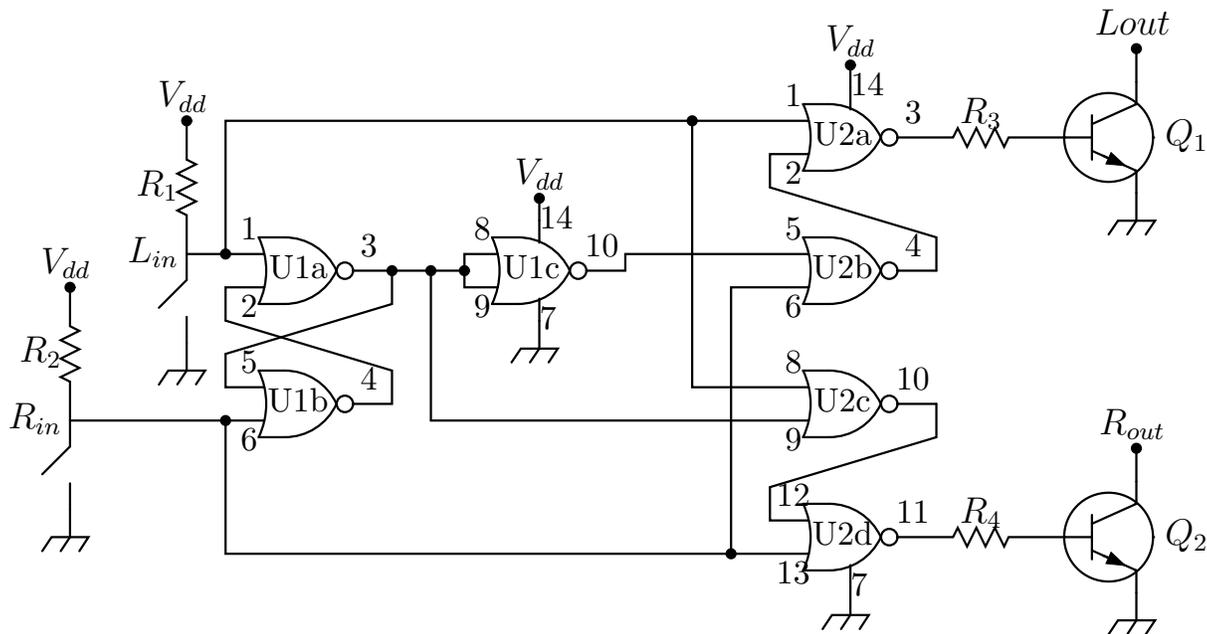


Figure 3: The complete NOR gate circuit. Values are what I had available.

$Q_1, Q_2$  2N3904 general purpose NPN transistors

$R_1, R_2$  2.2K Ohm 1/4 watt

$R_3, R_4$  2.0K Ohm 1/4 watt

$U_1, U_2$  CD4001, quad NOR gates. (Mouser 595-CD4001BE)

I used 2N3904 transistors with base current limiting resistors. The base resistors can be any value that sufficiently limits the current and saturates the transistors; use larger values than I did if you have them available. As mentioned in the text, the resistors could be eliminated if 2N7000 field effect transistors were used. I used a pair of CD4001 integrated circuits, but 74LS02 quad two-input NOR gates (or any other integrated circuit family) could also be used. The CD4001 can use a wide range of supply voltage; I powered this circuit from an adjustable power supply usually set at  $V_{dd} = 9$  Volts. The input pull-up resistors can be any reasonably convenient value that holds the input gates high. Using FET output transistors and larger valued pull-up resistors would reduce the current consumption. Rf bypass capacitors should be included on all the external connections in a permanent installation. The unused inputs on the CD4001 should be grounded.

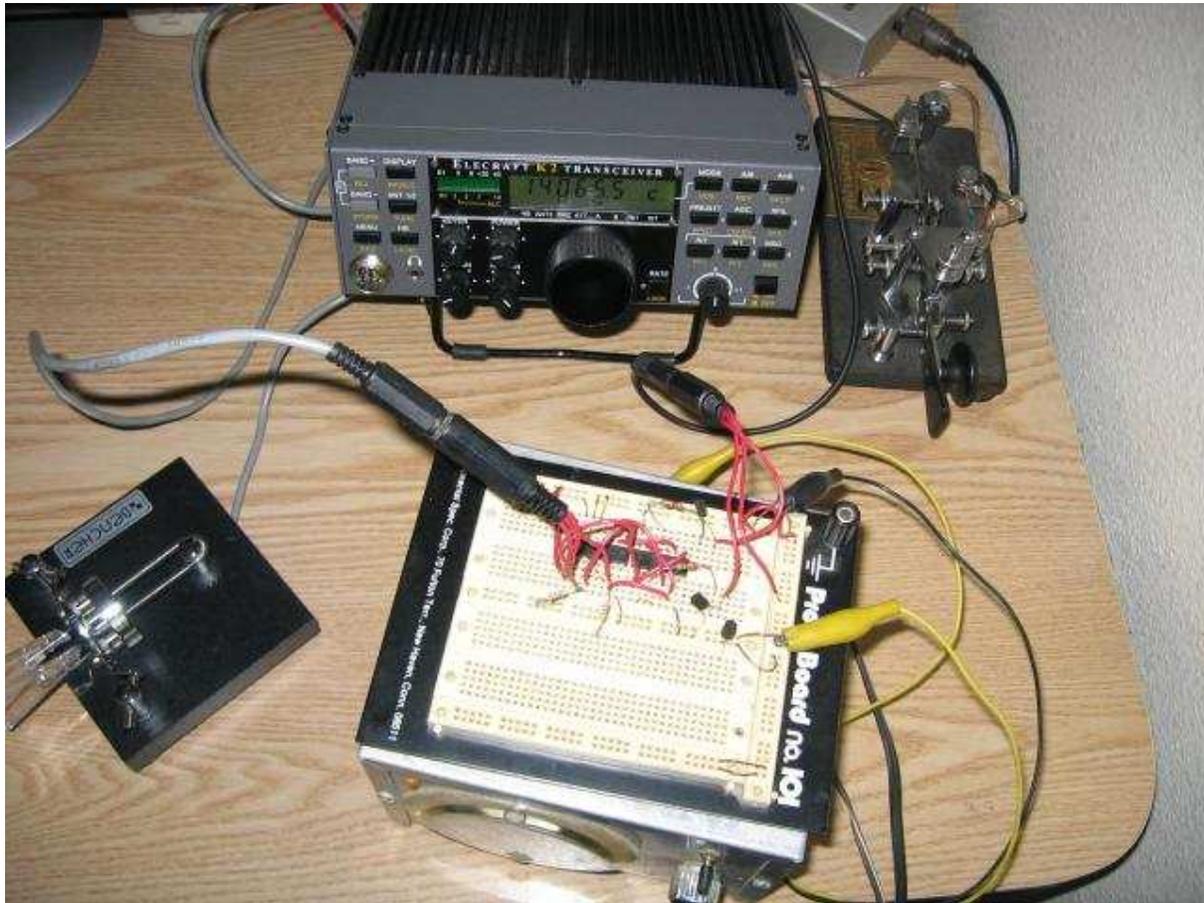


Figure 4: A quick prototype of the NOR gate circuit using parts on hand. The prototype board is sitting on a homebrew adjustable power supply. The circuit is connected between the Bencher BY-1 paddle and the Elecraft K2's keyer input.

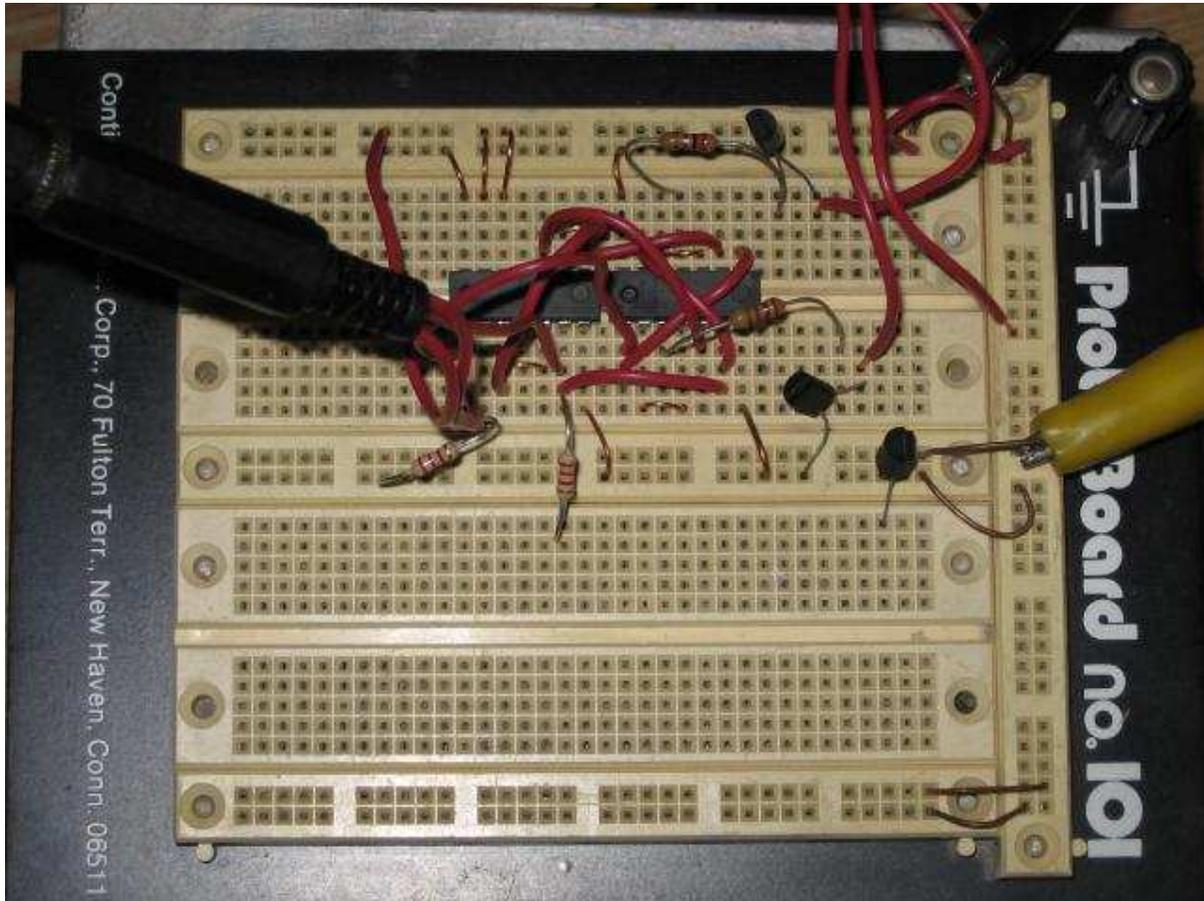


Figure 5: A close up photograph of the NOR gate prototype.

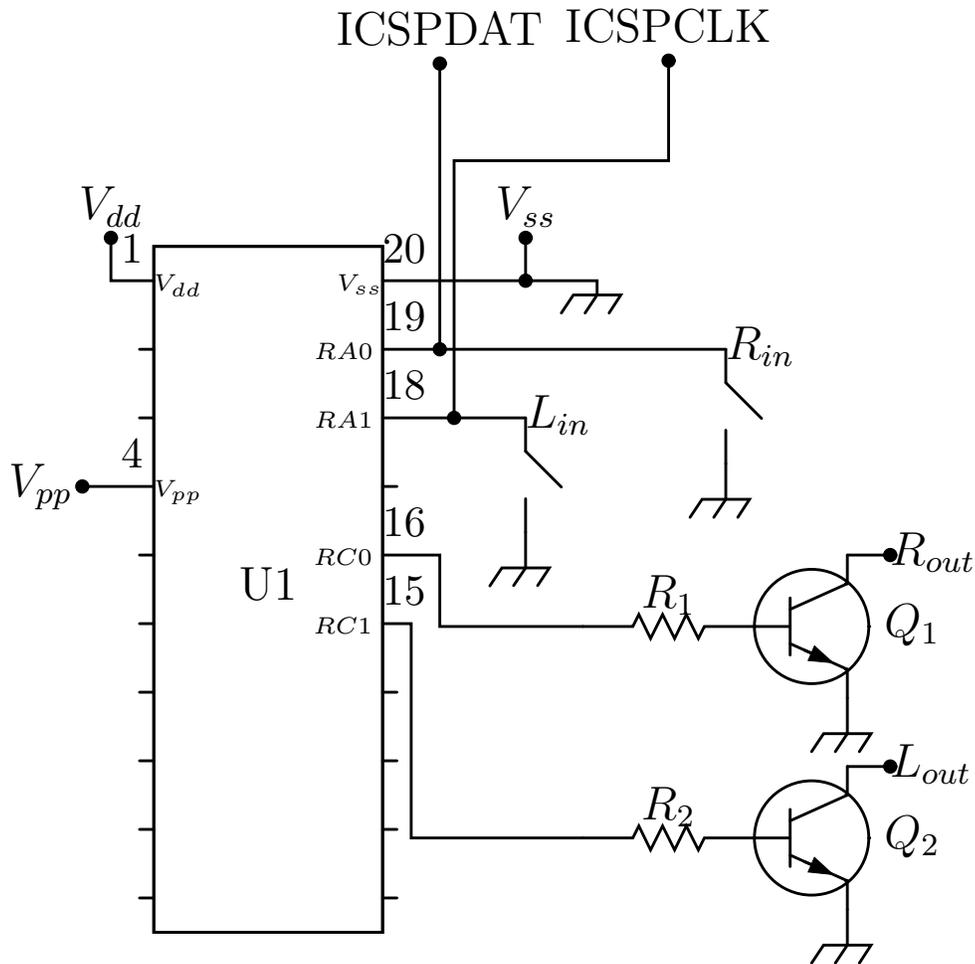


Figure 6: The PIC16F690-I/P circuit. Most of the pins are not connected.

$Q_1, Q_2$  2N3904 general purpose NPN transistors

$R_1, R_2$  2.0K Ohm

$U_1$  Microchip PIC16F690-I/P (Mouser 579-PIC16F690-I/P)

The same discussion as in figure 3 about changing the output transistors and the pull-up resistors applies here. The five labeled lines  $V_{pp}$ ,  $V_{dd}$ ,  $V_{ss}$ , ICSPDAT and ICSPCLK going to the microprocessor pins can be connected to a standard header connector for use with the PICKIT 2 for in circuit serial programming (ICSP).  $V_{dd}$  should be connected to a supply voltage of 3.0 to 5.5 volts for operation.



Figure 7: The prototyped PIC16F690-I/P circuit connected between the paddles and the K2. The CMOS Superkeyer III is the aluminum box with 6 pushbuttons to the right. The microprocessor is being powered by the PICkit 2 programmer shown connected to the in-circuit serial programming header.

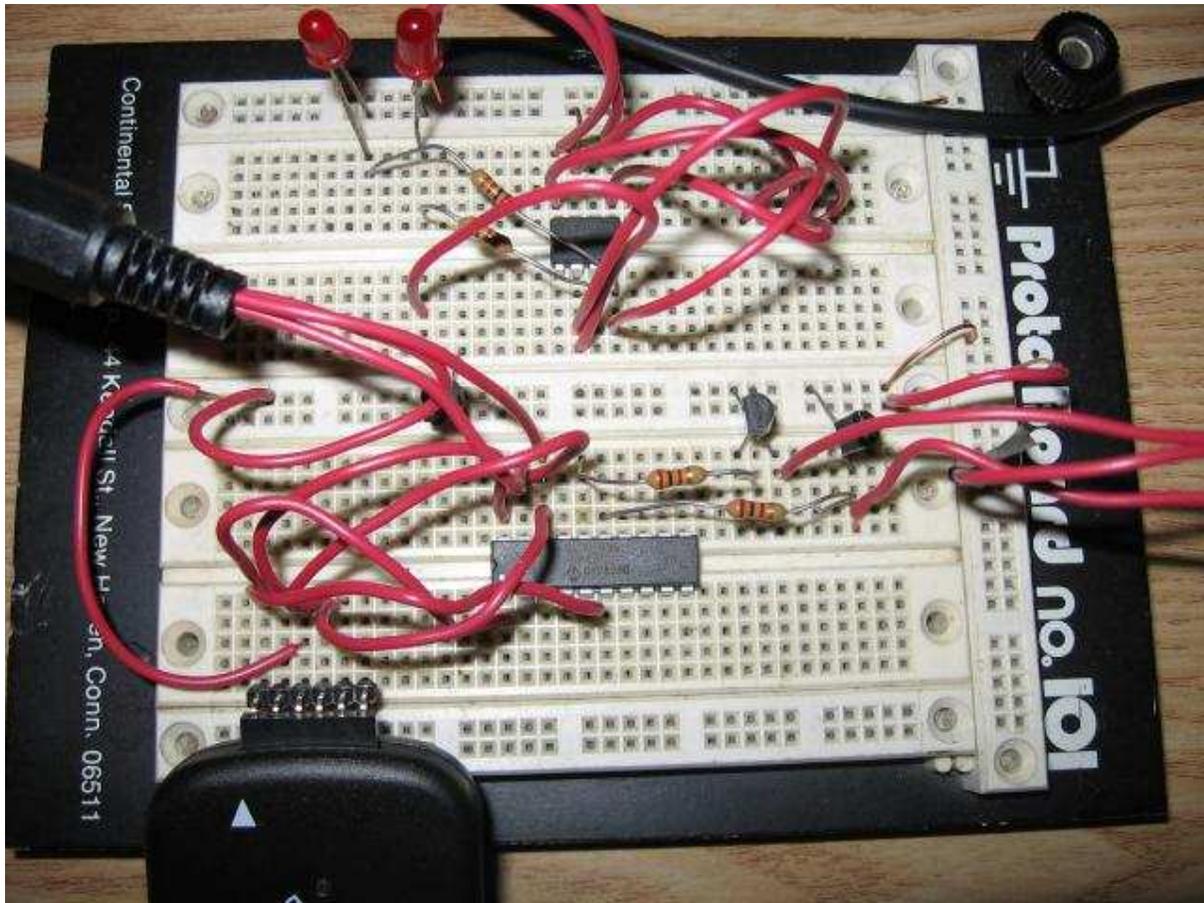


Figure 8: A close-up photograph of the PIC16F690-I/P circuit. The 16F690 is the larger chip on the bottom half of the prototype board. The parts on the upper half are for the prototyped PIC10F200-I/P circuit that was abandoned.

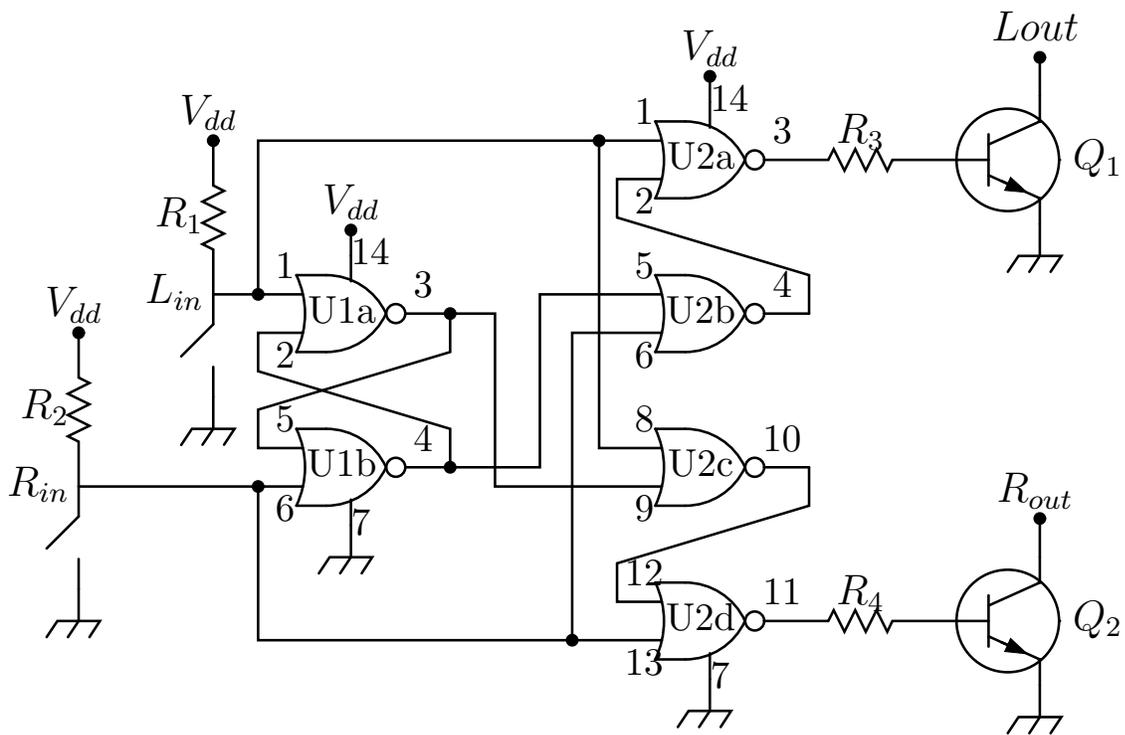


Figure 9: The improved nor gate circuit suggested by John Crabtree, KC0G. See caption to figure 3 for component details.

# Appendix

The assembler code and linker script for the 16F690 are given here.

Main code listing is:

```
; $Id: ultimatic.asm,v 1.4 2008/10/05 18:57:44 schmidt Exp $

#include "processor.h"

#define CF1 _FCMEN_OFF & _IESO_OFF & _BOR_OFF & _CPD_OFF & _CP_OFF
#define CF2 _MCLRE_OFF & _PWRTE_OFF & _WDT_OFF & _INTRC_OSC_NOCLKOUT
__config (CF1 & CF2)

;turn off crossing page boundary message
ERRORLEVEL -306,-302

    udata
state res 1          ;state
index res 1         ;index into table

TABLES code        ;truthtable
newstate:
    data 0x02,0x06,0x01,0x00,0x05,0x06,0x01,0x00

STARTUP code
    goto main

PROGO    code
main:
    call setup          ;set up ports etc.
loop:
    movf PORTA,w        ;read paddles
    bcf INTCON,RABIF    ;clear change interrupt
    movf PORTA,w        ;and read again to be current
    andlw 0x03          ;mask off inputs
    iorwf state,w       ;or in state
    movwf index         ;index into truthtable
    banksel EEADRH      ;begin standard code to read program memory
    movlw low(newstate) ;get low byte of address
    banksel index       ;add in index
    addwf index,w       ;
    banksel EEADR       ;move to EEADR
    movwf EEADR
    movlw high(newstate);get high byte
    movwf EEADRH        ;
```

```

btfsc STATUS,C           ;increment if there was a carry
incf EEADRH,f           ;should not happen here since table is small
banksel EECON1          ;do read reads
bsf EECON1,EEPGD
bsf EECON1,RD
nop                      ;and wait for result
nop
banksel EEDATA
movf EEDATA,w           ;move result to w
banksel state
movwf state             ;result stored in state and w
andlw 0x03              ;mask off output port
movwf PORTC             ;set output
movlw 0x04              ;form mask
andwf state,f           ;mask state bit
sleep                   ;sleep until next state change
nop
goto loop

```

setup:

```

banksel state           ;bank 0
clrf state              ;initialize state to 0
banksel PORTA           ;bank 0
clrf PORTA              ;clear ports
clrf PORTB
clrf PORTC
banksel OSCCON          ;bank 1
bsf OSCCON,IRCF2        ;8MHz internal oscillator
bsf OSCCON,IRCF1
bsf OSCCON,IRCFO
bsf OSCCON,SCS
clrf TRISB              ;set PORTB to outputs
clrf TRISC              ;set PORTC to outputs
movlw 0x03
movwf TRISA             ;set PORTA ra0 and ra1 to inputs
bcf OPTION_REG,NOT_RABPU ;allow weak pull-ups
movlw 0x03
movwf WPUA              ;set weak pull-ups on ra0 and ra1
banksel ANSEL           ;bank 2
clrf ANSEL              ;make PORTS digital
clrf ANSELH             ;make PORTS digital
banksel IOCA
clrf IOCA               ;clear interrupt on change for porta
                       ;this should cause a wake up from sleep,
                       ;but not call the interrupt routine

```

```

bsf IOCA,IOCA0      ;set interrupt on change for ra0
bsf IOCA,IOCA1      ;set interrupt on change for ra1
banksel PORTA       ;bank 1
clrf INTCON         ;disable all interrupts
bsf INTCON,RABIE    ;enable PORTA/B change interrupt
movf PORTA,w
bcf INTCON,RABIF    ;clear PORT change interrupt flag
return

end

```

The include file processor.h is:

```

;$Id: processor.h,v 1.1 2008/10/03 03:20:45 schmidt Exp $
list p=16f690
#include <p16f690.inc>

```

The linker script is:

```
//Linker file for 16f690
```

```
LIBPATH .
```

```

CODEPAGE  NAME=vectors      START=0x0      END=0x4      PROTECTED
CODEPAGE  NAME=page0       START=0x5      END=0x6FF
CODEPAGE  NAME=tablecode   START=0x700    END=0x7FF
CODEPAGE  NAME=page1      START=0x800    END=0xFFF
CODEPAGE  NAME=.idlocs    START=0x2000   END=0x2003   PROTECTED
CODEPAGE  NAME=icd_inst   START=0x2004   END=0x2004   PROTECTED
CODEPAGE  NAME=mfg_code   START=0x2005   END=0x2005   PROTECTED
CODEPAGE  NAME=.device_id START=0x2006   END=0x2006   PROTECTED
CODEPAGE  NAME=.config    START=0x2007   END=0x2007   PROTECTED
CODEPAGE  NAME=.calib     START=0x2008   END=0x2008   PROTECTED
CODEPAGE  NAME=eedata     START=0x2100   END=0x21FF   PROTECTED

DATABANK  NAME=sfr0       START=0x0      END=0x1F     PROTECTED
DATABANK  NAME=sfr1       START=0x80     END=0x9F     PROTECTED
DATABANK  NAME=sfr2       START=0x100    END=0x11F    PROTECTED
DATABANK  NAME=sfr3       START=0x180    END=0x19F    PROTECTED

DATABANK  NAME=gpr0       START=0x20     END=0x6F
DATABANK  NAME=gpr1       START=0xA0     END=0xEF
DATABANK  NAME=gpr2       START=0x120    END=0x16F

SHAREBANK NAME=gprnobnk   START=0x70     END=0x7F

```

```

SHAREBANK NAME=gprnobnk START=0xF0      END=0xFF
SHAREBANK NAME=gprnobnk START=0x170     END=0x17F
SHAREBANK NAME=gprnobnk START=0x1F0     END=0x1FF

SECTION   NAME=TABLES   ROM=tablecode // tables
SECTION   NAME=STARTUP ROM=vectors   // Reset and interrupt vectors
SECTION   NAME=PROG0   ROM=page0     // ROM code space
SECTION   NAME=PROG1   ROM=page1     // ROM code space
SECTION   NAME=IDLOCS  ROM=.idlocs   // ID locations
SECTION   NAME=ICD_INST ROM=icd_inst  // ICD instruction
SECTION   NAME=MFG_CODE ROM=mfg_code   // Manufacturing code
SECTION   NAME=DEVICEID ROM=.device_id // Device ID
SECTION   NAME=CONFIG  ROM=.config   // Configuration bits location
SECTION   NAME=CALIBR  ROM=.calib    // Calibration bits location
SECTION   NAME=DEEPROM ROM=eedata   // Data EEPROM

```