

The diodes serve to make current contribution unidirectional, thus avoiding modulation of the sum-of-current by exactly how good a logic-0 is provided.

There are NO ANTIALIASING FILTERS, and no reconstruction filters. You can experiment with tones greater than one-half the sampling rate, and observe where they are placed in the sampled data spectrum. By the way, running without a 80287, my PC (12MHz AT) needs about 7 seconds to perform a 256-point FFT. On my PC, acceptable ports are: #1=0x0378, #2=0x0278, #3=0x03bc

Driver in C

```

/* This routine implements, in conjunction with an external
Digital-to-Analog converter and analog comparator, a TRACKING
ANALOG-TO-DIGITAL CONVERTER. This routine has in software an
accumulator that sums the input signal---a single bit. The bit
indicates whether the accumulator value, the current guess as
to the actual analog signal, is GREATER or LESS than the analog
signal.

Turns out that a "C" routine on a 12MHz PC-AT is more than
fast enough to sample a voice signal. The loop spins about
60,000 times per second. This allows 20 samples--of 1 bit
change per sample---for 3KHz signals, or 10 samples peak-peak
at 3KHz. The normal male voice at 500 Hz can be 60 samples
peak-peak, or 1/4 of full scale.

At one time, variable-size steps were used, but the benefit
was not observable since there are NO ANTI-ALIASING filters,
and the distortion of the voice as judged by playback quality
was not improved by dynamically varying the step size.

But the electronic parts cost is only $2.00          */

/* will bind global 'sample_rate' */
void grab_voice()
{
  unsigned char tracking_value;
  unsigned int vndx;
  char interval;
  unsigned char input_signal;
  int control_port;
  char slew_accum;          /* will vary between +5 and -5 ?      */
  unsigned char delta;      /* how much to step 'tracking_value' */
  clock_t start_t, stop_t;

  printf("\nStarting to grab voice\n");
  control_port = active_port + 2;
  outportb(control_port, 0x04); /* free up the 4 OpenCollector I/O pins */
                                /* We'll can read data from Pin 1      */

  tracking_value = 0x80;      /* Init voltage to half-scale      */
  interval = 0;
  delta = 1;
  slew_accum = 0;

  start_t = clock();

  for ( vndx = 0 ; vndx < VOICE_SIZE; ) /* until VOICE is full */
  {

```

```

input_signal = 0x01 & inportb(control_port); /* PIN 1, by chance */

if ( !input_signal ) {
/*   tracking_value = min(0xfb, tracking_value + delta);   */
   tracking_value++;
   slew_accum++;
}
else {
/*   tracking_value = max(4, tracking_value - delta);   */
   tracking_value--;
   slew_accum--;
}

/* FASTER if port is constant */

outportb(active_port, tracking_value);

if ( interval++ EQUALS 4 ) /* 10 gives about 5 seconds in 0x02ff */
{
   voice[vndx++] = tracking_value;
   interval = 0;
   if ((slew_accum > 3) || (slew_accum < -3)) /* Has this any use? */
      delta++;
   else if (delta > 2)
      delta--;
   slew_accum = 0;
}
/* end FOR ( VNDX ; VNDX ; ) */

stop_t = clock();

sample_rate = /* bind global var */
              (VOICE_SIZE * CLK_TCK) / ( stop_t - start_t ) ;

printf("Grabbed voice. Took %f seconds. Rate is %d samples/sec\n",
       ((stop_t - start_t) / CLK_TCK), sample_rate );

}
/* ----- */
/* This function supports exploring the behavior of your printer port.
You specify an input pin of that port- 1, 14, 16 17.
Pin 2 will output a signal that alternates between low and high. The
rate is determined by how fast your computer can scroll. You are
shown the state of Pin 2 as output, and the state of the input pin.
This continues until you PRESS ANY KEY to EXIT.
Needs some HELP words, to explain what is good and bad display */

void input_diddle()
{
   unsigned char diddle_out, diddle_in, bitmask;
   int pinspec, which_io_port;

   printf("\nWill alternately turn Pin 2 ON and OFF, as test output.\n");
   printf("Select an input Pin that has pullup: 1 14 16 17, as 1 4 6 7 ? ");
   printf("You must CONNECT a WIRE BETWEEN pin 2 AND your test input.\n");
   pinspec = getche();

   which_io_port = active_port + 2;
   outportb(which_io_port, 0x04); /* 0100 frees the pullups */
                                  /* of I/O bits 17, 16, 14, 1 */

   switch( pinspec )
   {
      case '1': bitmask = 0x01; break;
      case '4': bitmask = 0x02; break;

```

```

    case '6': bitmask = 0x04; break;
    case '7': bitmask = 0x08; break;
    default: ;
}

diddle_out = 1;

for ( ; ; )
{
    if ( kbhit() )
    {
        getch();          /* eat char so won't exit the outer loop */
        break;
    }
    outportb(active_port, diddle_out);
    diddle_in = inportb(which_io_port);
    diddle_in = diddle_in & bitmask;
    printf("Output > %x <, Input value > %x < Phase-%s Default-%s\n",
        diddle_out, diddle_in,
        ((pinspec=='6') ? "NonInvert" : "Invert"),
        ((pinspec=='6') ? "High" : "Low")
    );

    diddle_out = ((diddle_out == 1) ? 0 : 1);
}

}
/* ----- end function INPUT_DIDDLE ----- */
/* This routine allows keyboard/cursor-key definition of the relative
output analog voltage. Thus you can test the threshold characteristics
of the comparator. Early on, the comparator had a non-voice-in
threshold of level 174; this was moved to about 135.
The size of the transition is 3 bits, so a continuous dither occurs.
*/

void define_level()
{
    unsigned char level;
    int action;
    int which_io_port;
    unsigned char diddle_in;

    level = 128;

    which_io_port = active_port + 2;
    outportb(which_io_port, 0x04);          /* 0100 frees the pullups */
                                           /* of I/O bits 17, 16, 14, 1 */

    printf("\nStarting at level of 128/midscale, move (u) or (d)own.\n");
    printf("The comparator digital output is reported. SPACE to abort.\n");
    printf("\nHardware MUST be in DIGITIZE mode, not PLAYBACK.\n");

    for ( ; ; )
    {
        action = getch();
        if ( action EQUALS ' ' )
            break;

        switch(action)
        {
            case 'u': level = level + 1; break;
            case 'd': level = level - 1; break;
            default: ;
        }
    }
}

```

```

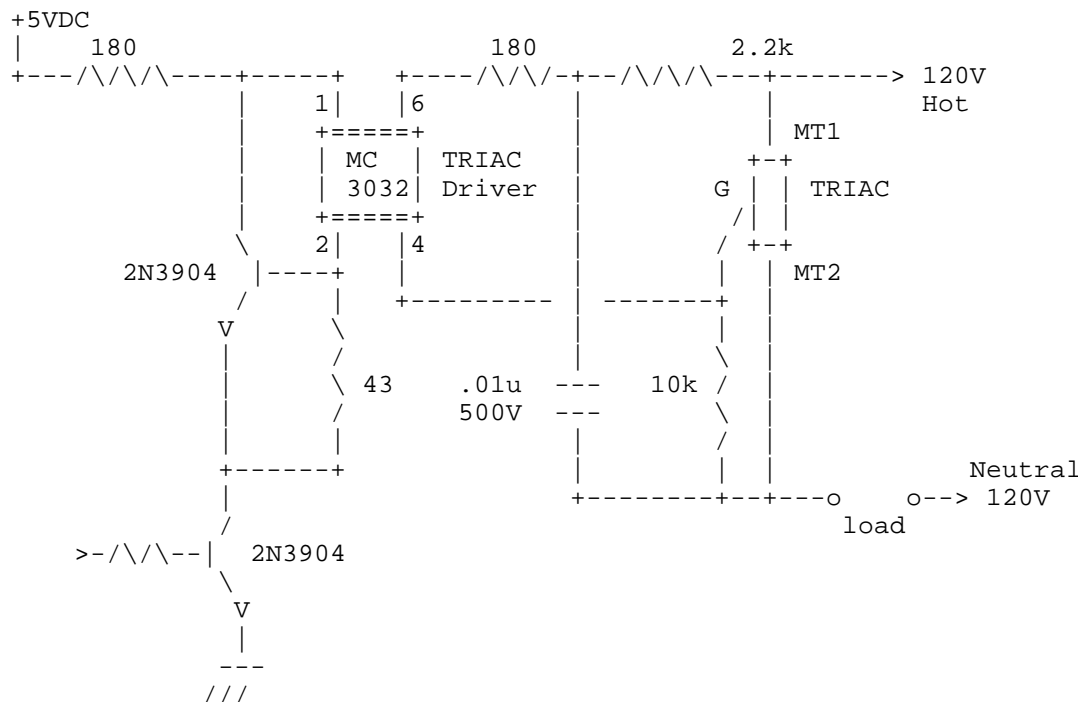
    outportb(active_port, level);
    diddle_in = inportb(which_io_port);
    diddle_in = diddle_in & 0x01;
    printf("Level %d Comparator %d\n", level, diddle_in);
}      /* end FOR */
}

```

2. AC switching with TRIAC from TTL

From: weissj@psd.gs.com (Jeffrey Weiss)

Here's a TRIAC-based solid-state relay circuit:



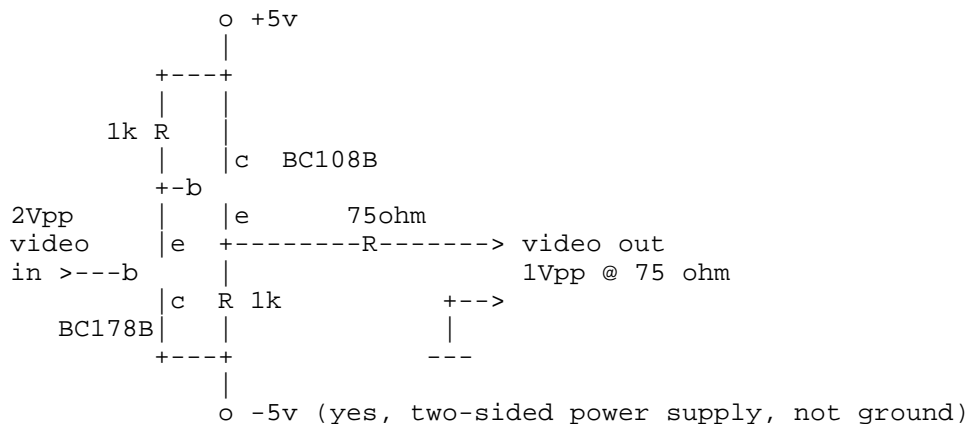
Circuit Description

The MC3032 is an optoisolator TRIAC driver. The 180-ohm resistor sets the current for the LED emitter in the optoisolator. Change the value of this resistor - if necessary - to get reasonable current (e.g., 15 mA).

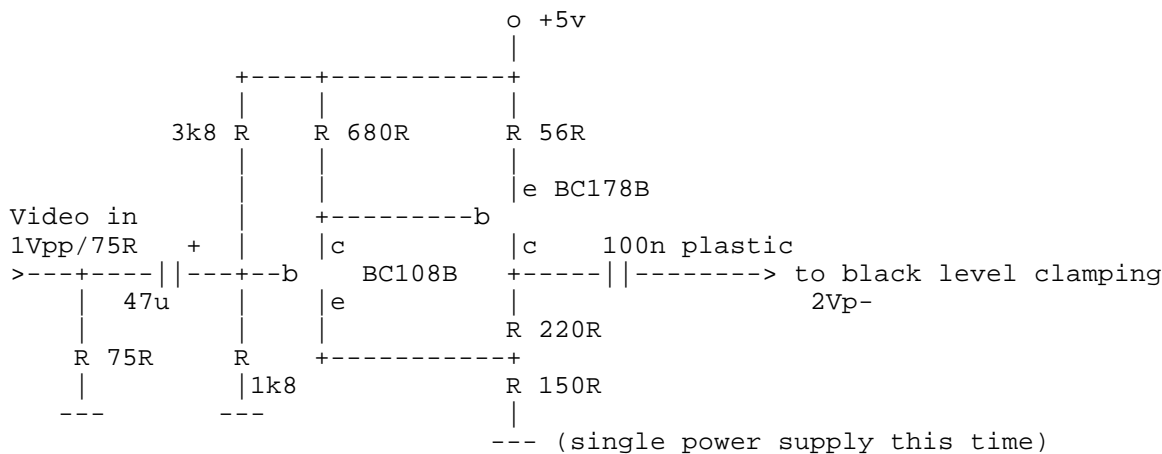
Note that you cannot test this circuit without a load. The TRIAC will not switch unless connected to an AC voltage source, so you can't test it for simple switching w/o applying AC and a load. Note the 500V rating on the .01 cap.

resistor with standard video...). A typical video input has a 75 ohm terminating resistor to ground and then the signal is fed to the input buffer via a ~50uF electrolytic cap.

Anyway, here's a simple discrete video output stage. Can't get much simpler than this. Note that there's a serial matching resistor on the output, so you'll have to feed 2Vp-p video into the buffer to get the usual 1Vp-p into the equipment you're driving. This is the way it's usually done. Sorry for the crude transistors, but I hate doing ASCII graphics.



And while I'm at it, here's the input stage to go with it. It provides the 2x voltage gain you need to feed the output buffer above.



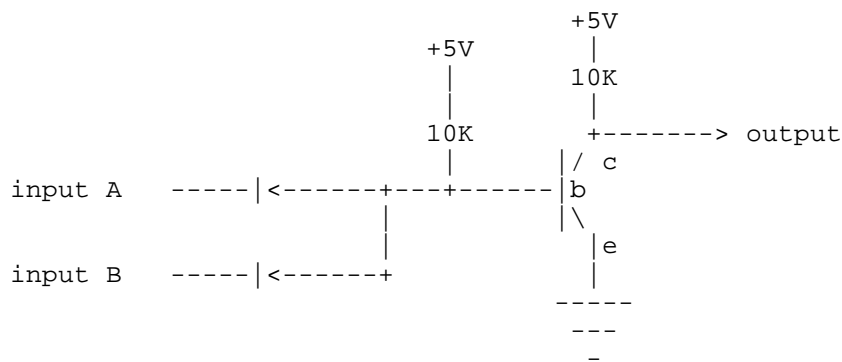
The simplest black level clamp consists of a signal diode (1N4148) reverse-biased to ground from the output line of the input buffer above and a 4k7 resistor in parallel with it. That forces the sync tips to be at (gnd - threshold voltage of the diode), which shifts the black level of a 2x amplified video reasonably close to ground. Add that and you can connect the two circuits above together and see how they work. They should be very good as far as the signal quality goes (maybe not broadcast quality, but no visible signal degradation). Don't forget good power supply bypassing, use at least 220u of electrolytics and 100n ceramic caps near the transistors on both circuits (the output stage needs them on both supplies).

7. Build logic gates using discrete parts

(From Richard Steven Walz)

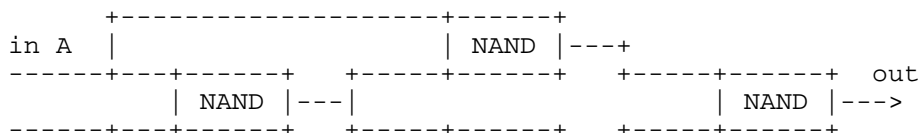
Here is the design for some gates I made for a kid's lab show and tell which did all the different gates and the culminating two projects were a JK flip-flop and a four bit binary adder.

You use open-collector logic and thus the output is simply the collector of the transistor for each simple gate. An inverter and a NAND can be made quite easily, and then from those, any other gate you wish! You pull up the base with a 10K ohm resistor or even larger, and you also hang diodes to that connection as well which are the inputs. They isolate each input from the others, (if any), and allow the current to the normally ON transistor to pull the collector low except when one of those inputs with a diode is grounded by a previous collector or switched input. It might be a good idea to pull up the output (collectors) weakly as well with a 10K to the 5V source. The emitters always go to ground. Thus a typical inverter/NAND gate looks like this:



This is usually sufficient to have some fun with kids to relate transistors to chips. Note that this is an inverter or a NAND gate, depending on whether both inputs are used! Note how to create the other gates from this kind, as it is easiest to make with 2N2222's. The NOR would be two inverted inputs to a NAND and then another inverter on the output: Total, 4 transistors. And XOR is either-or but not both, so you use an OR, (3 transistors, the NOR above without the output inverter) and then use two AND inputting into it, and then put appropriate inverters on one input for one input line and then the other, with inverted and non-inverted signals appropriately. Get Forrest Mims, III 's book from RShack for all the ways to make one gate from another.

Then set about reducing gates which are double inversions, and look up DeMorgan's Laws to manipulate the sense of AND and OR with inverters for your optimal use of transistors. If some book doesn't show you the "magic castle" means of making an XOR with 4 NANDS, 4 transistors, then here it is:





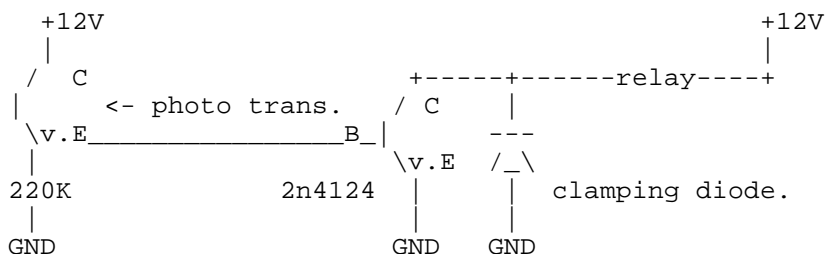
And of course, the X-NOR, or comparator, is just one more inverter on the output of that mess. If you draw it with proper NAND gate symbols and then look at it with the output at the top, you can see why it's called the "magic castle", it looks like one! A memorable circuit, and a good Boolean Algebra final exam question!!!

If you really want to be true to form, you can replicate the standard old original TTL NAND gate with the original components except with discrete packages, using 3 of this transistor and the oft stated values for the components in an actual 7400 TTL QUAD 2-INPUT NAND gate, even as the plain 7400's are today! The voltage thresholds come out rather accurately, actually! You can find that design in many textbooks and old databooks on TTL logic. The logic I had you use is called open collector DTL logic, and it was used in the early 70's both on chips and out of discrete components. Experiment with fan-out and the power dissipation and compare it to the TTL emulation. Oh, and that funny multi-emitter thing used in the input end of the TTL can be accurately replicated with diodes, as its transistor nature is really not being exploited. It would be more correct to draw it as diodes, as it uses the bc junction for a diode as well!

8. Driving a relay with a CPU

(From Kip J. Mussatt)

Try using a simple transistor like a 2N4124 just feed the base from the emitter of the opto-isolator. Don't forget to put a clamping diode across the relay (reverse biased) and maybe a 1uF cap. Drive the photo transistor with an IR LED and couple it with some heat shrink tubing.



Please see document for document author. [\[Feedback Form\]](#) | [\[mailto\]](#). The most recent version is available on the WWW server <http://www.repairfaq.org/> [\[Copyright\]](#) [\[Disclaimer\]](#)