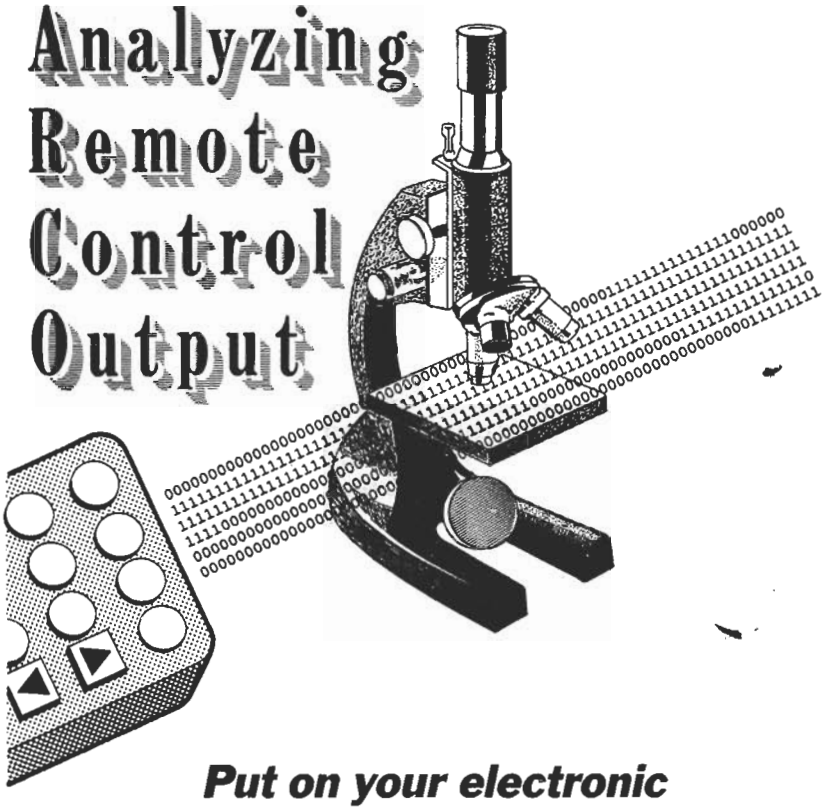
Analyzing Remote Control Output



Put on your electronic Sherlock Holmes hat and investigate the output of your infrared remote controls.

BARRY HAMILTON

THE CONSUMER ELECTRONICS MARket has become inundated with low-cost pre-programmed infrared (IR) remote controls. These can produce codes to control a variety of appliances, including TVs, stereos, VCRs, and cable boxes.

You can build an infrared receiver with less than $10.00 of parts, and use any PC with a parallel printer port as a kind of digital storage oscilloscope to examine the pulse train produced by an IR remote. The knowledge you gain will allow you to incorporate remote controls in your next circuit design project.

The receiver

Figure 1 shows a schematic of the IR receiver circuit. The heart of the circuit is MOD1, an infrared detector module that removes the IR carrier frequency and transmits only the data that is encoded in the received IR signal.

A suitable IR module is available at Radio Shack (No. 276-137) for $3.59. The IR module needs a clean 5-volt power supply that is provided by IC1, a 7805 regulator. Power is supplied to the regulator by 9-volt battery B1. The output of the module is wired to a male DB-25 multipin connector.

Most infrared remote controls encode data in the form of long and short pulses of infrared light on a 40-kilohertz carrier frequency. This method is known as pulse-width modulation, or PWM.

The infrared detector module receives a signal, filters it, and removes the 40-kilohertz carrier. The output of the module is a TTL-level signal consisting of long and short pulses. The PC records those voltage levels over time, while the signal is being sent, and stores the data in a file.

The line normally used by the PC's printer port to indicate that the printer is out of paper (pin 12) is used in this project to accept data from the IR module. The I/O port is located at address 0x379. Bit 5 corresponds to input pin 12.

Various software programs are required to let a PC store information input to its printer port. (All of the software is available on the Gernsback BBS—516-293-2283, v.32, v.42bis—contained in a file called IR-TEST.ZIP).

The source code of the first program, IRLOG.EXE, is written in C and shown in Listing 1. The program stores the value it reads from the PC's printer port into an array. When the input line is logic high, the ASCII character "1" is stored in the array. When the input line is logic low, ASCII character "0" is stored.

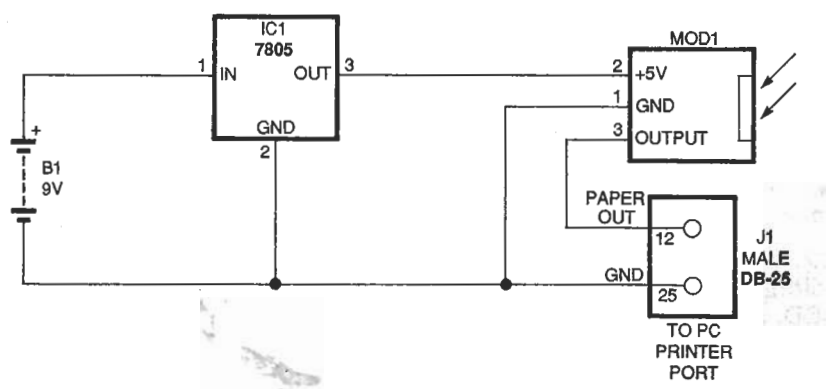


FIG. 1—SCHEMATIC OF THE IR RECEIVER CIRCUIT. The heart of the circuit is MOD1, an infrared detector module that removes the IR carrier frequency and sends only the data encoded in an IR signal.

IRLOG.EXE is a simple loop. The program reads the value of the line, stores it in the array store[], increments the array index X, and then waits a user-defined time delay before repeating itself.

When the array is full (30,000 points), the program dumps the array to a file and then waits for a keypress to take another 30,000 points. Pressing the Escape key terminates the program.

The program does not try to write values to disk while it is sampling the infrared input be-

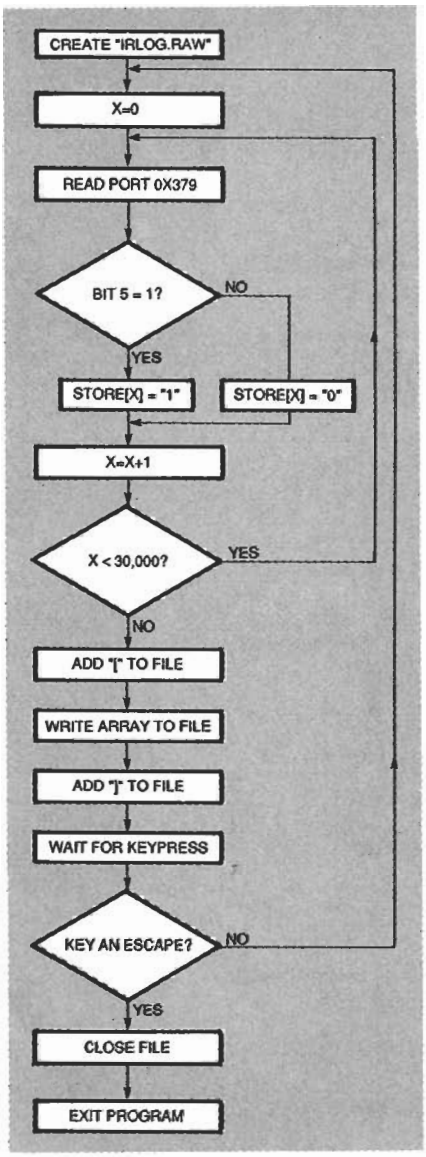


FIG. 2—FLOWCHART OF IRLOG.EXE shows how the program stores raw data in a file filled with 1's when no signal was received and stretches of 1's and 0's during times where the infrared was received.

LISTING 1

```
//This is IRLOG.C - Monitors OUT_OF_PAPER Input, writes to file.
// (C) 1994, Barry Hamilton, M.S.E.E.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

int main (int argc, char *argv[])
{
 int timeDelay;          //User Variable to adjust Port Sampling Rate.
 int loop = 1;           //Control For Sampling Loop, Set to 0 by ESCAPE key.
 int key;                //To store Keypress.
 int x;                  //Array Index.
 int z;                  //Time Delay Counter.
 int clnData;            //Store Byte from Port.
 int limit = 30000;      //Array Limit.
 int store[30000];       //Array to store Samples.
 FILE *fp;               //File Pointer For IRLOG.RAW Output.

if (argc != 2) {
   printf("IRLOG - Samples Pin 12 of printer port monitoring IR Detector\n");
   printf("    USAGE:  IRLOG TIMEDELAY\n");
   printf("    Like: IRLOG 200\n");
   printf("The output file will be called IRLOG.RAW\n");
   exit(1);
}
   clrscr();
   timeDelay = atoi(argv[1]); //Note No Checking is done...

   if ( (fp=fopen("IRLOG.RAW","wb"))==NULL) {
      printf("cannot create IRLOG.RAW\n");
      exit(2);
   }
   //=============================
   while (loop == 1) {
      //Record Input Samples into Array....
      for (x=0;x<limit;x++){

         clnData = inportb(0x379);

         if( (clnData & 0x20) != 0 )
               store[x] = 0x31;   //Store an ASCII "1"
         else
               store[x] = 0x30;   //Store an ASCII "0"

         //User Selectable Time Delay between Samples...
         for(z=1;z<timeDelay;z++);
      }
      //=============================
      //Sampling is over, now time to save array...
      fputc('[',fp);

      for (x=0;x<limit;x++){
          fputc(store[x],fp);
      }

      fputc(']',fp);
      putch('.');   //Visual Progress For user...
      sound(440); //Beep To denote End of Sampling...
      delay(20);
      nosound();

      key = getch(); //Hit ESC to Exit Program or any other Key to repeat.
      if (key == 0x1B) loop = 0; //ESC exits...
   } //End of While loop == 1
   //=============================
   return(0);
} //<EOF> IRLOG.C
```

LISTING 1

```
//This is IRLOG.C - Monitors OUT_OF_PAPER Input, writes to file.
// (C) 1994, Barry Hamilton, M.S.E.E.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

int main (int argc, char *argv[])
{
 int timeDelay;          //User Variable to adjust Port Sampling Rate.
 int loop = 1;           //Control For Sampling Loop, Set to 0 by ESCAPE key.
 int key;                //To store Keypress.
 int x;                  //Array Index.
 int z;                  //Time Delay Counter.
 int clnData;            //Store Byte from Port.
 int limit = 30000;      //Array Limit.
 int store[30000];       //Array to store Samples.
 FILE *fp;               //File Pointer For IRLOG.RAW Output.

if (argc != 2) {
   printf("IRLOG - Samples Pin 12 of printer port monitoring IR Detector\n");
   printf("    USAGE:  IRLOG TIMEDELAY\n");
   printf("    Like: IRLOG 200\n");
   printf("The output file will be called IRLOG.RAW\n");
   exit(1);
}
   clrscr();
   timeDelay = atoi(argv[1]); //Note No Checking is done...

   if ( (fp=fopen("IRLOG.RAW","wb"))==NULL) {
      printf("cannot create IRLOG.RAW\n");
      exit(2);
   }
   //=========================================
   while (loop == 1) {
      //Record Input Samples into Array....
      for (x=0;x<limit;x++){

         clnData = inportb(0x379);

         if( (clnData & 0x20) != 0 )
               store[x] = 0x31;   //Store an ASCII "1"
         else
               store[x] = 0x30;   //Store an ASCII "0"

         //User Selectable Time Delay between Samples...
         for(z=1;z<timeDelay;z++);
      }
      //=========================================
      //Sampling is over, now time to save array...
      fputc('[',fp);

      for (x=0;x<limit;x++){
          fputc(store[x],fp);
      }

      fputc(']',fp);
      putch('.');   //Visual Progress For user...
      sound(440); //Beep To denote End of Sampling...
      delay(20);
      nosound();

      key = getch(); //Hit ESC to Exit Program or any other Key to repeat.
      if (key == 0x1B) loop = 0; //ESC exits...
   } //End of While loop == 1
   //=========================================
   return(0);
} //<EOF> IRLOG.C
```

cause the time it would take to write to the disk would slow down the sampling process. Therefore the array is filled, sampling is stopped, and then the data is appended to file IRLOG.RAW.

The program places brackets around each array's worth of samples to delineate the beginning and ending of each subsequent sample. The flowchart of IRLOG.EXE, shown in Fig. 2, illustrates this process. The program IRLOG.EXE stores raw data in a file filled with 1's when no signal is received and stretches of 1's and 0's during times when the infrared is received. A sample interval of the output file IRLOG.RAW is shown in Fig. 3.

To run IRLOG, enter the following form on the command line:

IRLOG <TIMEDELAY

Where <TIMEDELAY is the value that will be used in the delay loop between samples. The au-

LISTING 2

```
//This is IRGRAPH.C - Produces IRLOG.GPH From IRLOG.RAW
// (C) 1994, Barry Hamilton, M.S.E.E.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>

void main(void)
{
 FILE *fpIn;              //File Pointer for IRLOG.RAW Input File.
 FILE *fpOut;             //File Pointer for IRLOG.GPH Output File.

 int inChar;              //Present Character retrieved from IRLOG.RAW.
 int lastChar;            //Used to compare the Character before with present.
 int totalCnt = 0;        //Character Counter for runs of the same character.
 int nLimit = 80;         //Limit of Graph String.
 char s2[81];             //String of "111" or "000" to be printed to IRLOG.GPH

  char s11[] = "11111111111+11111111111+11111111111+11111111111+\
11111111111+11111111111+11111111111+11111111111+";

  char s10[] = "00000000000+00000000000+00000000000+00000000000+\
00000000000+00000000000+00000000000+00000000000+";

//=================================
  if ( (fpIn=fopen("IRLOG.RAW","rb"))==NULL) {
    printf("cannot open IRLOG.RAW\n");
    return;
  }
  if ( (fpOut=fopen("IRLOG.GPH","wb"))==NULL) {
    printf("cannot open IRLOG.GPH\n");
    return;
  }
//=================================
  while ( (inChar = fgetc(fpIn)) != EOF ) { //Main Loop...

    if (inChar == '0' && lastChar == '0') {
      totalCnt++;
    }

    if (inChar == '1' && lastChar == '1') {
      totalCnt++;
    }
  //=================================
    if (lastChar == '[') { //Resets on Beginning of Sample..
      totalCnt = 1;
    }

    if (lastChar == ']') { //Resets on End of Sample...
      totalCnt = 0;
    }
//

  //=================================
    if (inChar == '0' && lastChar == '1') {
      if (totalCnt < nLimit) {
        strcpy(s2,"");
        strncat(s2,s11,totalCnt);
        fprintf(fpOut,"%s\n",s2); //Print String of "1"s.
      }
      else {
        fprintf(fpOut,"                | %04d 1 \n",totalCnt);
      }
      totalCnt = 1;
    }
  //=================================

    if (inChar == '1' && lastChar == '0') {
      if (totalCnt < nLimit) {
        strcpy(s2,"");
        strncat(s2,s10,totalCnt);
        fprintf(fpOut,"%s\n",s2); //Print String of "0"s.
      }
      else {
        fprintf(fpOut,"                | %04d 0 \n",totalCnt);
      }
      totalCnt = 1;
    }
  //=================================
    if (inChar == ']' && lastChar == '0') {
      if (totalCnt < nLimit) {
        strcpy(s2,"");
        strncat(s2,s10,totalCnt);
        fprintf(fpOut,"%s]\n",s2); //Print String of "0"s and ].
      }
      else {
        fprintf(fpOut,"                | %04d 0] \n",totalCnt);
      }
      totalCnt = 0;
    }

  //=================================
    if (inChar == '[' && lastChar == '1') {
      if (totalCnt < nLimit) {
        strcpy(s2,"");
        strncat(s2,s11,totalCnt);
        fprintf(fpOut,"%s]\n",s2); //Print String of "1"s and ].
      }
      else {
        fprintf(fpOut,"                | %04d 1] \n",totalCnt);
      }
      totalCnt = 0;
    }

    lastChar = inChar; //Always update lastChar...
  } //End of Main Loop...

  fclose(fpOut);
  fclose(fpIn);
} //<EOF> IRGRAPH.C
```

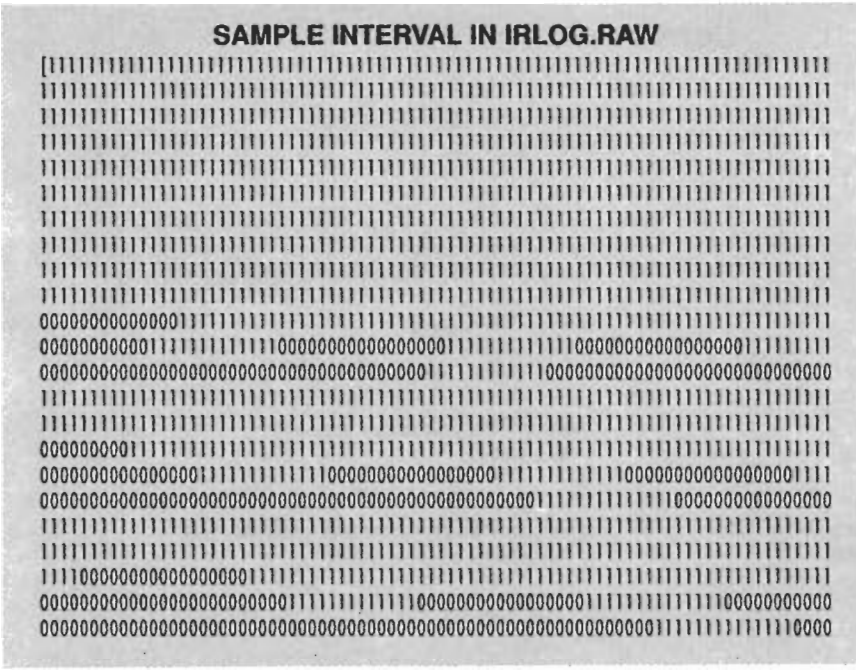


FIG. 3—SAMPLE INTERVAL of the output file IRLOG.RAW.

thor found that a delay value of 200 works well for a computer based on a 50-MHz 486 processor and that a delay of 20 was required for an IBM AT-class computer. Those values will vary with the computer used and the particular pulse train being analyzed. You can observe the output generated by IRGRAPH to adjust the delay value.

The program IRGRAPH.EXE reduces the data stored by IRLOG.EXE to a more convenient form. It counts how many 1's occur in a row and reduces them to an output of "nnn 1's." The same is true for the number of 0's that occur.

To simplify the visual analysis of the output data, IRGRAPH takes those sequences of 1's and 0's that fall between 1 and 80 and prints a string of 1's or 0's to create a bargraph. The source code for IRGRAPH.EXE is shown in Listing 2, its flowchart is shown in Fig. 4, and its output (IRLOG.GPH) is shown in Fig. 5.

The output in Fig. 5 was obtained from a One For All Universal TV remote set up for manufacturer's code of 1111. The example sampling recorded the remote's power button being pressed.

The output starts with a trigger pulse that is longer than all other pulses. A series of long and short pulses interspersed with constant-width sync pulses is sent after that. The trigger has an interval of 65, the long pulse corresponding to a "1" has an interval of 34, the short pulse corresponding to a "0" has an interval of 19, and the sync pulses have an interval of 12. There is a repeat time of 649 intervals between the bursts of infrared.

Note that in this example, the sync pulses are sent as intervals when the signal is high. This means the sync pulses use the "mark" state versus the "space" state. The mark state is considered the idle state, i.e., the value output when no infrared signals are received.

The sync pulses can be considered as the rest times between sending data. Knowing whether these rests occur during the period that the signal is logic high or low will allow you to identify the data that would be of the opposite state.

One last program, IRFINAL.EXE, shown in Listing 3, reduces the data to a long pulse corresponding to a 1 and a short pulse corresponding to a 0. IRFINAL.EXE examines the raw data generated by the program IRLOG.EXE.

The flowchart for IRFINAL.EXE is shown in Fig. 6. The program counts the numbers of 1's and 0's. If the count falls within the parameters defined as a long pulse (MAXZERO AND MAXONE) it will output a 1. If the count is less than MAXZERO, it is considered to be a short pulse and it will output a 0. These characters will be sent to the file called IRLOG.FNL, as shown in Fig. 7.

To run the program IRFINAL, enter the following instruction

ANNOTATED OUTPUT
FROM IRLOG.FNL

```
Trigger/Address & Data
| 0065 0=101010010000
| 0063 0=101010010000
| 0063 0=101010010000
| 0063 0=101010010000
| 0062 0=101010010000
| 0063 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0063 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0062 0=101010010000
| 0063 0=101010010000
| 0062 0=101010010000
| 2660 1]
```

FIG. 7—IRFINAL.EXE outputs characters to the file called IRLOG.FNL, as shown here.

TABLE 1—OUTPUT DATA

BINARY (MSB FIRST) Address	Data	BUTTON	HEX Address	Data
0000	1001 0000	Channel Up	09	0
0000	1001 0001	Channel Down	09	1
0000	1001 0010	Volume Up	09	2
0000	1001 0011	Volume Down	09	3
0000	1001 0100	Mute	09	4
0000	1001 0101	Power	09	5

LISTING 3

```
//This is IRFINAL.C - Converts Raw Data to actual address/data code.
// (C) 1994, Barry Hamilton, M.S.E.E.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <string.h>

#define MARK  1
#define SPACE 0

int main (int argc, char *argv[])
{
 FILE *fpIn;                //File Pointer for IRLOG.RAW Input File.
 FILE *fpOut;               //File Pointer for IRLOG.FNL Output File.

 int inChar;                //Present Character retrieved from IRLOG.RAW.
 int lastChar;              //Used to compare the Character before with present.
 int totalCnt = 0;          //Character Counter for runs of the same character.

//========= User Adjustable Variables... =========
 int MaxZero;  //Greatest relative Time Interval Of Pulse Considered a "0".
 int MaxOne;   //Greatest relative Time Interval Of Pulse Considered a "1".

 int synchState;//MOSTLY SPACE. Check By looking at IRLOG.GPH.
//================================================

if (argc != 4) {
  printf("IRFINAL - Produces Address and Data Codes from IRLOG.RAW\n");
  printf("    USAGE:  IRFINAL M(ark synch)/S(pace synch) MAXZERO MAXONE\n");
  printf("    Like: IRFINAL M 25 40\n");
  printf("The output file will be called IRLOG.FNL\n");
  exit(1);
}
  if (strcmp(argv[1],"M") == 0) //Note Little Error Checking...
    synchState = MARK;
  else
    synchState = SPACE;

  MaxZero = atoi(argv[2]);
  MaxOne  = atoi(argv[3]);

  if ( (fpIn=fopen("IRLOG.RAW","rb"))==NULL) {
    printf("cannot open IRLOG.RAW\n");
    exit(2);
  }
  if ( (fpOut=fopen("IRLOG.FNL","wb"))==NULL) {
    printf("cannot open IRLOG.FNL\n");
    exit(3);
  }

//================================================
  while ( (inChar = fgetc(fpIn)) != EOF ) {

    if (inChar == '0' && lastChar == '0') {
      totalCnt++;
    }

    if (inChar == '1' && lastChar == '1') {
      totalCnt++;
    }


//================================================
    if (lastChar == '[') {  //Resets on Beginning of Sample...
      totalCnt = 1;
    }

    if (lastChar == ']') {  //Resets on End of Sample...
      totalCnt = 0;
    }

//================================================
    if (inChar == '0' && lastChar == '1') {
      if (synchState == SPACE){//Count MARK (1) As 1 (Long) and 0 (Short).

        if (totalCnt <= MaxZero) {
          fprintf(fpOut,"0");
        }

        if (totalCnt > MaxZero && totalCnt< MaxOne) {
          fprintf(fpOut,"1");
        }

        if (totalCnt >= MaxOne) { //NOTE: MARK is ALWAYS Idle State!
          fprintf(fpOut,"\n| %04d 1=",totalCnt);
        }
      }
      totalCnt = 1;
    }

//================================================
    if (inChar == '1' && lastChar == '0') {
      if (synchState == MARK) {//Count SPACE (0) As 1 (Long) and 0 (Short).

        if (totalCnt <= MaxZero) {
          fprintf(fpOut,"0");
        }

        if (totalCnt > MaxZero && totalCnt< MaxOne) {
          fprintf(fpOut,"1");
        }

        if (totalCnt >= MaxOne) {
          fprintf(fpOut,"\n| %04d 0=",totalCnt);
        }
      }
      totalCnt = 1;
    }


//================================================
    //Shows When Sampling Period Ended...

    if (inChar == ']' && lastChar == '0') {
      fprintf(fpOut,"\n| %04d 0]");
      totalCnt = 0;
    }

    if (inChar == ']' && lastChar == '1') {
      fprintf(fpOut,"\n| %04d 1]",totalCnt);
      totalCnt = 0;
    }

    lastChar = inChar;
  }

  fclose(fpOut);
  fclose(fpIn);
  return(0);
} //<EOF> IRFINAL.C
```

ANNOTATED OUTPUT OF IRLOG.GPH

```
                | 8628 1  <-No Signal Received Time
000000000+000000000+000000000+000000000+000000000+000000000+00000 <-Trigger
111111111+1 <-Synch Pulse
000000000+000000000+000000000+0000 <-Long Pulse = "1"
111111111+1 <-Synch Pulse
000000000+000000000 <-Short Pulse = "0"
111111111+1
000000000+000000000+000000000+0000
111111111+1
000000000+000000000
111111111+1
000000000+000000000+000000000+0000
111111111+1
000000000+000000000
111111111+11
000000000+00000000
111111111+11
000000000+000000000+000000000+000
111111111+11
000000000+00000000
111111111+11
000000000+00000000
111111111+11
000000000+00000000
111111111+11
000000000+00000000
                | 0649 1 <- Time Between Repeats
000000000+000000000+000000000+000000000+000000000+000000000+000
111111111+11
000000000+000000000+000000000+000
111111111+11
000000000+00000000
111111111+11
000000000+000000000+000000000+000
111111111+111
(Continues)...
```

FIG. 5—IRLOG.GPH OUTPUT that results from running the program IRGRAPH.EXE.

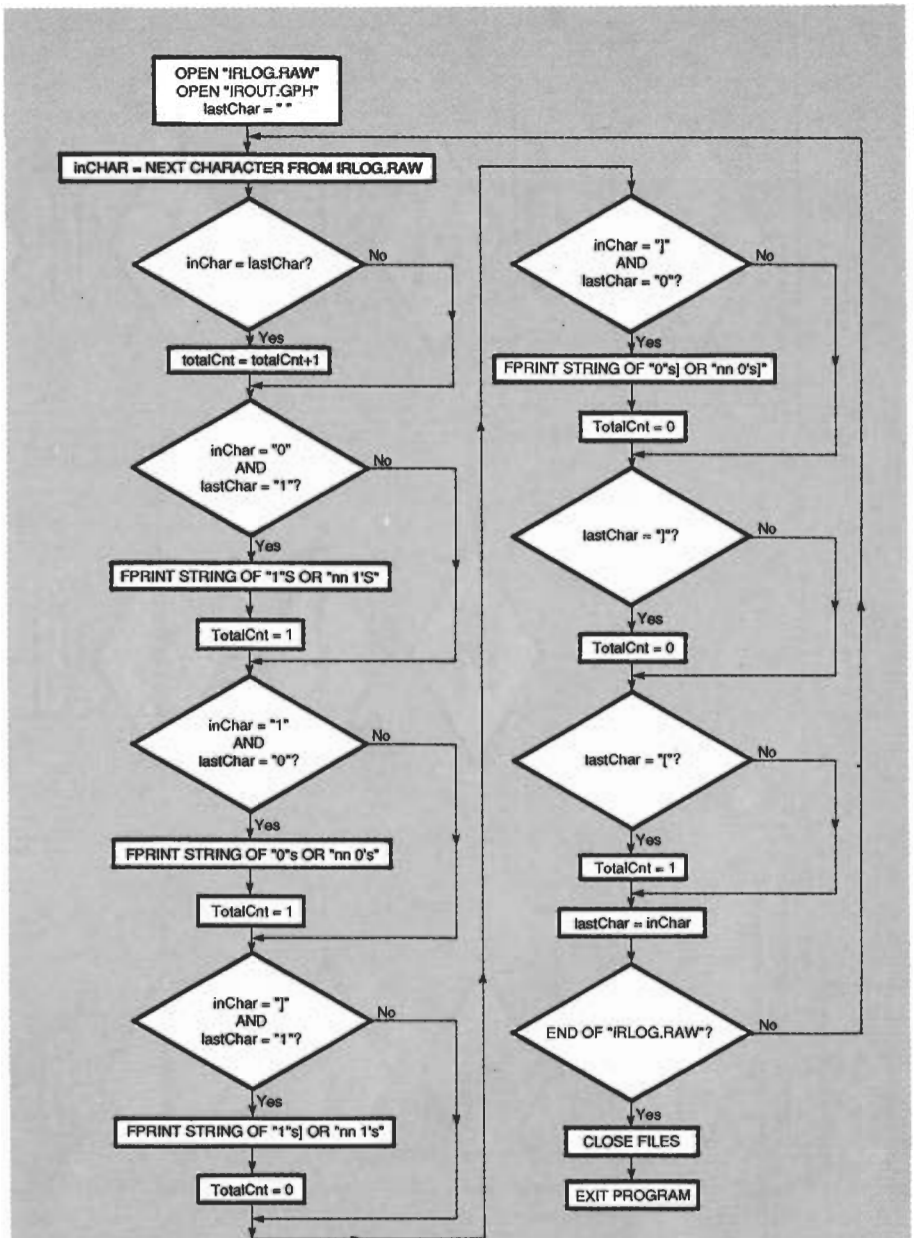


FIG. 4—THE PROGRAM IRGRAPH.EXE counts how many 1's occur in a row and reduces them to an output of "nnn 1's."

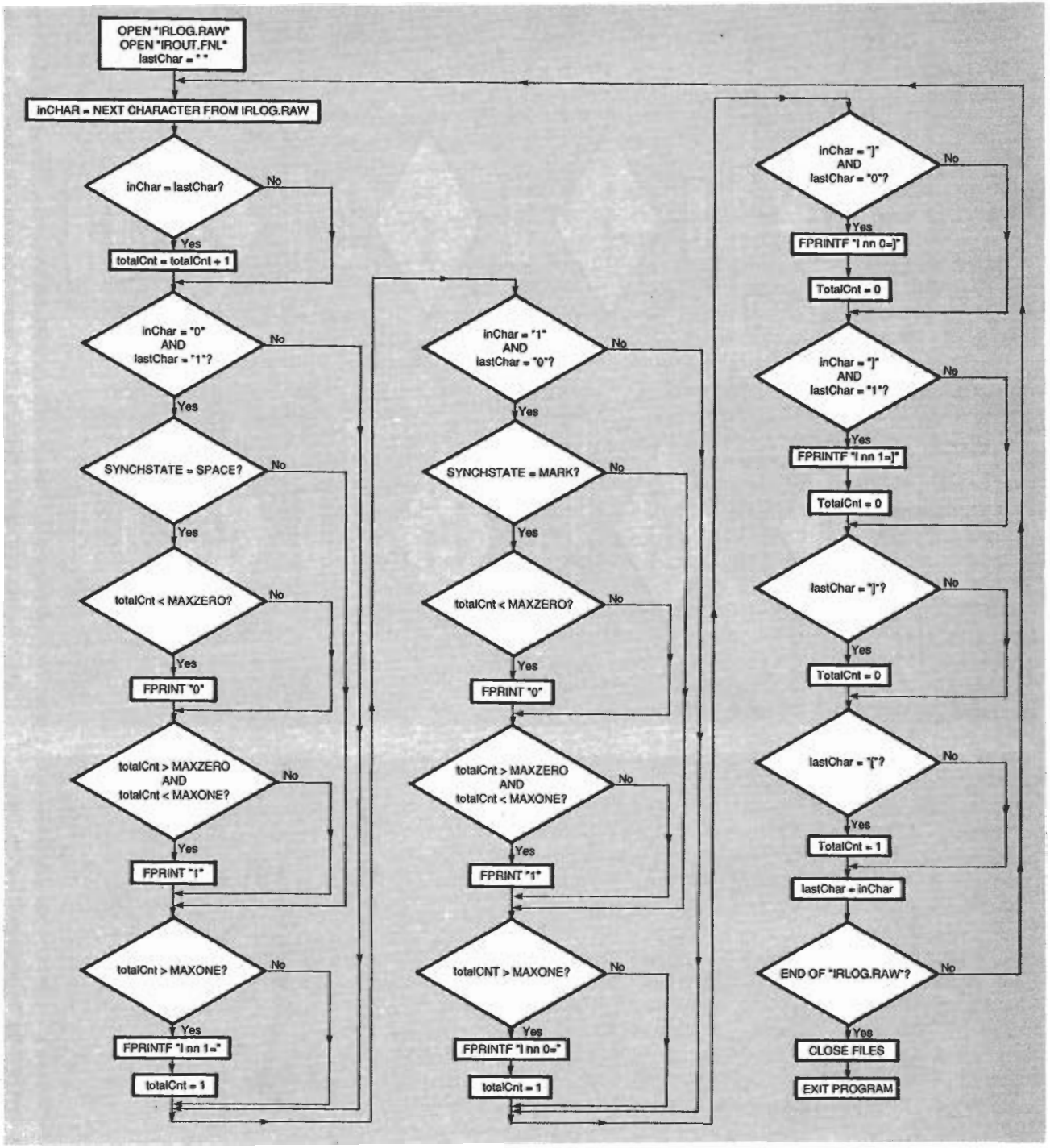


FIG. 6—IRFINAL.EXE FLOWCHART. This program reduces the data to a long pulse corresponding to a 1 and a short pulse corresponding to a 0.

on the command line:

IRFINAL M/S MAXZERO MAXONE

where M or S is mark or space sync state, MAXZERO is the maximum data length you want to be a Data 0 value, and MAXONE is the maximum data length you want to be a Data 1 value.

Note in Fig. 7 that since only one key was pressed, the code repeats itself. The example remote control sends 12 bits at a time.

Other remotes typically send 12 to 15 bits. Many manufacturers precede the data code with an address code, and most send the least significant bit first.

Some remotes send the information twice with each keypress, sometimes inverting the repeated data. Some use a checksum for error checking. Many send the pulse train once, and then wait and send a special code meaning "repeat last command." The data in Table 1 was generated for the one for all universal TV remote. Ω