

DEPARTMENTS

70 Firmware Furnace

78 From the Bench

84 Silicon Update

92 Practical Algorithms

99 ConneCTime

Ed Nisley

Infrared Home Control Gateway

You can already control your TV, VCR, and CD player from the comfort of your chair using a hand-held infrared remote. Why not control your home as well? Ed shows us the firmware involved in doing just that.



Some folks have problems in places where most of us don't even have places.

Consider the fate of the poor chap who fixed Steve's furnace last week: he didn't move around enough to keep the motion-sensing lights turned on. The house was on manual lighting control while the repairman was at work, if you can imagine that.

Now, suppose everybody in Steve's house wore a badge that broadcast a unique ID. The HCS II could then track each person and adjust the room lighting, sound system volume, security defenses, and so forth, depending on who was where. Finally he'd have a truly personalized, automated house that would work without manual adjustments.

Of course, the system can only track people (or, I suppose, dogs) wearing IR badges. But consider what the HCS II's response might be should the basement door open without a valid badge ID on either side... ..

The topic this month is the firmware needed for an IR gateway to the new Circuit Cellar Home Control System II. It receives infrared signals from remote units and passes them to the Supervisory Controller (SC) for action, so you can control the HCS II without leaving your chair, as well as transmitting IR signals that can poll

B	Set badge response delay (default 40 ticks)
CR n	Calibrate remote MC1 45030 bit clock (12.4 kHz) n=0-3 sets report detail
CT	Calibrate transmitter oscillator connect 38 kHz output to T1 input
D	Dump program status (debugging use)
E	Show and clear error flags (debugging use)
In	Set badge polling interval (default 190 ticks)
Ln	Set logging mode (bit mapped) L report current mode L0 disable (default) L1 show received IR messages L2 show transmitted IR messages L4 show generated polls
Nn	Set network/interactive mode N report current mode N0 set interactive mode N1 network mode (no error messages) (default) N2 network mode with command echo
O x=n	Set output bit x to n OA=0 set bit A OB=1 set bit B OC=0 set bit C OD=1 set bit D
P n	Set number of badges to poll (default 0, no polling)
Q	Query and reset received IDs a dump all 512 IDs in hex (130 chars in line) ID 0 is bit 0 of first byte Qn report ID n status in format 003=1 On-m report IDs n through m in hex bytes ID n is bit 0 of first byte
RESET	perform power-on reset, must be completely spelled out
S n	Send IR ID n

Figure 1—The IRGATE commands include functions to send and receive IR signals as well as control the firmware's operation. All times are in 5.12-ms units, and numeric values are decimal unless otherwise noted.

specific "people tracker" badges. An HCS II system can have up to eight of these gateways on its RS-485 network, so you can put a different unit in each room to ensure adequate coverage.

GATEWAY FUNCTIONS

The overall structure of the IR Gateway (or IR-Link) firmware resembles that of the Smart X10 controller (or PL-Link) I described in the last issue. It receives and transmits serial information over an RS-485 network, includes a simple decoder to handle ASCII commands from the HCS II SC, and is written in Micro-C.

I've taken a bit of heat for abandoning 80.51 assembly language in this column. For the record, about half of the "Micro-C" source lines for the Smart X10 and IR Gateway firmware are actually assembler code, written using Micro-C's in-line assembler. As I continue to point out, C is good for the overall program logic, but not at all suitable for high-speed bit banging and interrupt handlers.

Figure 1 presents IRGATE's command set. As with the Smart X10 controller, there are commands to send and receive IR signals as well as control how the firmware operates.

IRGATE also includes calibration routines.

Steve's description of his IR hardware explains how the various IR remote units and badges work. He has the advantage of using the MC145030 chip directly; on the firmware end of the signal, there is essentially no hardware at all! As a result, I must discuss the MC145030 data format in some detail before explaining how the firmware transmits and receives it.

IR I/O

Figure 2 shows the MC145030 data format, adapted from the data sheet. The chip uses Manchester encoding, which defines two complementary signals for each data bit; Motorola calls the pair of signals a "bit frame" to indicate a single data bit. A complete message requires 39.5 bit frame times: 24 frames hold data or framing bits, while the remainder provide silent periods before, during, and after the "real" frames.

Motorola calls the contents of the message "address" bits because they think of the MC145030 as a widget to control a device at a specific address. I refer to them as "data" bits because our messages convey information to the HCS rather than selecting an address. Fortunately, the bits don't care what they are called and do the same thing regardless of their labels.

Steve chose the bit frame time to work correctly with the Sharp IS1U60 receiver, which specifies a 600-µs minimum On and Off time. Because a Manchester-encoded bit frame uses both states, the minimum frame time is 1200 µs. Steve picked 1290 µs, which means the MC145030 uses a 12.4-kHz oscillator.

Converting an integer (between 0 and 511, as there are only nine data

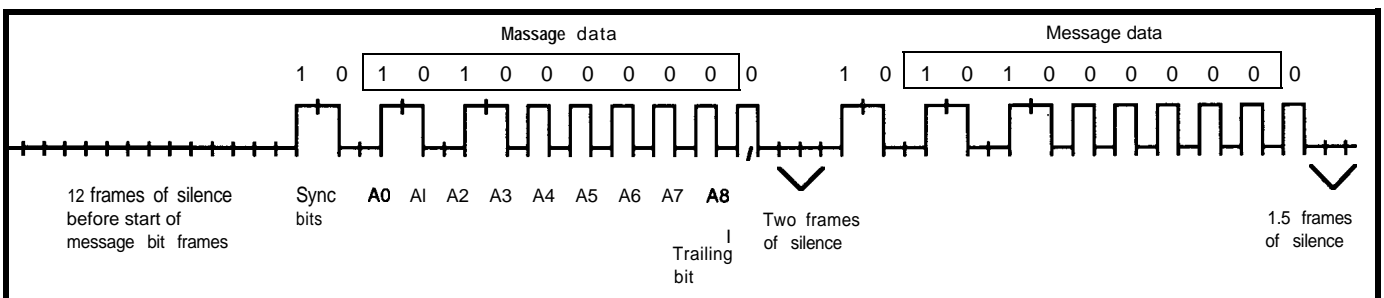


Figure 2-The MC145030 chip sends a single 9-bit number. A complete transmission includes two copies of the number amid various pauses and synchronizing Ms.

bits available) into a message is straightforward, as shown by the three routines in Listing 1. Rather than pack the output bits into bytes, I squandered 79 bytes of the 8K External RAM on a C array called ITrnBuf. Each byte holds one bit of the encoded message, so each array element represents half of a Manchester bit frame.

The final line of IRFormatMsg() sets the FlagIRPending bit, which is monitored by a timer interrupt routine that ticks every 5.16 ms. When it sees FlagIRPending set on, it cranks the timer interrupt to 645 μ s and shovels bits from the array to the output pin on each interrupt. After finishing the message, the interrupt handler resets the timer interrupt period to 5.16 ms.

The interrupt handler updates several software timers that count off intervals of a few seconds. The motivation for the peculiar 5.16-ms "normal" rate is that it is exactly eight times the 645 μ s dictated by a 1.290-ms Manchester bit frame. After sending all 79 bits, the code simply adds 10 counts to the software timers. This addition keeps the timers reasonably accurate even if the firmware sends many IR messages.

Because the interrupt handler knows nothing about the message format, the higher-level code can send any bits it wants. In this application, I used the MC145030 data format, but similar code could mimic nearly any other chip. Such mimicking is especially valuable when the chip may change at any time, or when the requirement becomes "either of these two remote control chips" after the first one becomes obsolete.

Sending a message is as simple as issuing the "S" command to IRGATE over the serial link. The number after the "S" must be in the range 0 to 511. The firmware holds outgoing messages in a ring buffer, so it can accommodate bursts of commands from the SC or your program.

BITS FROM THE ETHER

Receiving bits from a remote MC145030 transmitter is somewhat more difficult. A Sharp IS1U60 IR receiver translates the IR signal into a TTL voltage for the 8031's INTO input

Listing 1—These three functions translate an integer between 0 and 511 into the format required by the MC145030 chip. Each entry in the output array represents the state of the output pin for half of a 1290- μ s bit frame. The entries are transferred to the output by an interrupt handler routine driven by the on-chip timer.

```

/*-----*/
/* Convert argument into Manchester data frame (two      */
/* outputs per call)                                     */
/* Uses only low bit of argument to simplify the calls   */
/* Output is 1 for "IR On" and 0 for 'IR Off'           */
/*-----*/

IRFormatFrame(MsgBit, pMsgFrame)
unsigned int MsgBit;
BYTE *pMsgFrame;
{
    if (0 == (MsgBit & 0x0001)){
        *pMsgFrame = 1;
        *(pMsgFrame+1) = 0;
    }
    else {
        *pMsgFrame = 0;
        *(pMsgFrame+1) = 1;
    }
}

/*-----*/
/* Convert argument into Manchester-encoded burst at ptr */

IRFormatBurst(MsgNum, pMsgBase)
unsigned int MsgNum;
BYTE *pMsgBase;
{
    BYTE Index;

    IRFormatFrame(1, pMsgBase+0);           /* start bits */
    IRFormatFrame(0, pMsgBase+2);
    pMsgBase += 4;

    for (Index=0; Index<NUMDATABITS; ++Index) { /* encode msg */
        IRFormatFrame(MsgNum, pMsgBase);
        MsgNum >>= 1;
        pMsgBase += 2;
    }
    IRFormatFrame(0, pMsgBase);           /* trailing bit */
}

/*-----*/
/* Convert integer into bit pattern in the transmitter buffer */

IRFormatMsg(MsgNum)
WORD MsgNum
{
    memset(ITrnBuf, 0, sizeof(ITrnBuf)); /* flush previous bits */

    IRFormatBurst(MsgNum, ITrnBuf+24); /* first transmission */
    IRFormatBurst(MsgNum, ITrnBuf+52); /* repeat one time... */

    INumSend = 79;                       /* how many half-frames in buffer */
    SETBIT(FlagIRPending);                /* indicate ready to go */
}

```

pin. The firmware sets up INTO to produce an interrupt for each down-going edge, but a little study of Figure 2 will show that the Manchester data format does not produce an interrupt for each bit frame.

The firmware can take advantage of the bit frame timing to anticipate

when each bit should arrive. The first sync bit produces an interrupt in the middle of the bit frame that defines when all of the other bits should occur. The INTO interrupt handler sets the hardware timer to interrupt one quarter of a bit frame later, in the middle of the last half of the first bit

frame, and sets a variable to indicate that reception is in progress.

The timer interrupt handler then begins sampling the input and verifying that the data matches the MC145030 data format. Each sample must be followed by its complement (01 or 10) to form a valid bit frame, and the silent periods (00) must occur at the right times. During the two groups of frames holding data bits, the interrupt handler shifts the second sample of each frame into a pair of 16-bit variables.

After the final silent period, the firmware compares the two variables to ensure that both data values are equal. If so, and if all the silent periods were truly silent, the firmware adds the data value to the receiver's ring buffer for later analysis by the higher-level C code.

The INTO input continues to generate interrupts whenever the IR signal goes on, so the firmware puts that information to good use. The timer should be midway in its count to the next sample at each negative transition, so the INTO handler reloads it with exactly one-quarter of the bit frame time (think about it), which allows the firmware to track remote units with slightly off-frequency oscillators. Timing the interrupts this way ensures the data samples occur in the "middle" of each half of the bit frame, where they are least likely to be corrupted by noise or timing errors.

Because the receiver and transmitter use the same hardware timer in different ways, IRGATE cannot receive while it is transmitting, so, unlike Smart X10, IRGATB cannot monitor its own output. The INTO handler checks the timer handler's state variable before clobbering the timer; the valid state transitions are Idle→Receive and Idle→Transmit, but not Receive→Transmit or vice versa.

IRGATE records all received codes in a table that it displays when you issue the "Q" [query] command over the serial link. There are several different formats so you can get the values you need in the fewest characters. After sending the table's contents, IRGATB clears it so each "Q" reports only new codes.

BADGE TRACKING

Although IRGATE's main purpose is receiving inputs from push-button remotes to control Steve's new HCS, I included all the functions you need to build a system that can track people wearing IR badges. Ken's SC has code built into it to support active badges (more in a moment), and allows you to initiate actions based on what room a particular badge is in, so the ground-work is done.

As far as IRGATE is concerned, there are two types of badges: active and passive. Active badges are essentially identical to the remote units described above, except they transmit their ID code (0 through 5 11) periodically. Passive badges transmit their ID only when they "hear" that ID from another source.

Active badges are quite simple, because they do not need any receiver hardware. However, because each badge transmits without regard for the others, a room full of badges might produce no recognizable messages due to all the collisions. You must pick the

repeat rate to minimize the number of collisions while not introducing too much delay in determining when a new badge enters or leaves the room.

Passive badges, on the other hand, must include a receiver that is active continuously, so they will drain their batteries faster. In addition, IRGATE must know which badge IDs to send, because a passive badge responds only to its own ID code. While there are no collisions, IRGATE must poll all possible badge IDs to find out which ones are within range.

I'll just point out that designing the badge hardware is not easy and leave it at that. The receiver and transmitter hardware used by IRGATB need the changes Steve describes in his article to ensure it can cover an entire room.

If your system uses active badges, IRGATB needs no changes because it cannot tell the difference between a remote control unit and a badge. After all, there are only 5 12 possible codes and the bits mean whatever you want them to. Your controller must poll

Want low power?

Is 10 microwatts low enough? (That's not for a CPU chip, but for a complete *nanoLINK* Controller board in a typical application. The automatic power-cycling technology that makes that possible is standard and built-in.)

Fast development?

What if we write the device drivers for you? And the timing-critical, low-level code? And throw in a complete BIOS? And a full monitor with source code? And what if the hardware includes all the real-world interfaces you'll probably need, like A/Ds, buffered digital I/O, and high power switching?

And low cost?

What if the standard board package *includes* a macroassembler? An integrated editor? A communications interface? Complete, illustrated technical references
What if the hardware includes a switching supply and even the serial cable?
What if you don't even need an EPROM programmer?

There is a better way.

For information, call:
1-800-GET-DATA (4383282)
or
602-996-0255 (FAX)

nano | **Power**
Devices

A division of AirDigital Corporation

IMAGENATION...

VIDEO FRAME GRABBERS

- ◆ Simplicity
- ◆ Functionality
- ◆ Affordability
- ◆ Accuracy

Cortex-I

\$495⁰⁰

- ◆ Real-Time Capture
- ◆ Half Slot XT/AT
- ◆ 512 x 484 x 8 Bit
- ◆ RS-170/CCIR
- ◆ External Trigger

Cortex-STD

\$595⁰⁰

- ◆ Dual Video input
- ◆ Opt. XMS Mapped
- Low Power Options
- ◆ STD-80 or 32 Bus
- ◆ External Trigger

Vidmux-4

\$99⁰⁰

- ◆ 4 to 1 MUX
- ◆ Half Slot XT/AT

Includes Software...

- ◆ C Library & Source
- ◆ Image Capture Utility
- ◆ Tiff Utilities
- ◆ "Image" Drive Ram Disk Emulation

STARTER PAK...

Cortex-Pak

\$1395⁰⁰

- ◆ NEC/TI-23EX Camera With Lens
- ◆ 9" Video Monitor
- ◆ Frame Grabber
- ◆ Software & Cables

OEM PRICING AVAILABLE

IMAGENATION CORP.

P.O. Box 84568
Vancouver, WA 98684

PH/FX (206) 944-9 13 1

Listing 2—Measuring the remote MC145030's oscillator frequency requires determining the bit frame time based on the received data. This firmware routine analyzes an array holding the time of each negative transition in the IR signal. The first element of the array is the length of the pulse formed by the first two sync bit frames. All the debugging output statements controlled by ShowDetail are removed to save space in this listing.

```
IRCallRemote(ShowDetail)
int ShowDetail;
{

BYTE FrameNum,FrameCount,Index;
unsigned int FrameTime,FrameTime1,FrameTime2,AvgFrTime;

do {
memset(IRTimes,0,sizeof(IRTimes));
SerWaitOutDone();

IRSample(); /* get the IR transition times */

/*-- get differential time between each sample */
/* [0] is start bit frame time; estimates true frame time */
/* [1] is time since the start. so it is already a delta time */

for (FrameNum = MAXIRFRAMES-1; FrameNum > 1; --FrameNum) {
IRTimes[FrameNum] = IRTimes[FrameNum] - IRTimes[FrameNum-1];
}

/*-- convert times from timer ticks to microseconds */
/* magic number is 1.085 microsecs per timer tick. but we */
/* must do it in two tiny chunks to avoid overflowing 16 bits */

for (FrameNum = 0; FrameNum < MAXIRFRAMES; ++FrameNum){
FrameTime1 = IRTimes[FrameNum] * 2; /* 0.08 */
FrameTime1 = (FrameTime1 + 12) / 25;
FrameTime = (IRTimes[FrameNum] + 100)/200; /* 0.005 */
IRTimes[FrameNum] += FrameTime1 + FrameTime2; /* 1.085 */
}

/*-- set up initial frame sorting thresholds */
/* this is done in chunks to avoid overflows. too */
/* first frame is special-cased sample. needs 80% derating to */
/* real time because the trailing edge is delayed by IR */
/* receiver */

FrameTime1 = IRTimes[0] * 8;
FrameTime1 = FrameTime1 / 10;
IRTimes[0] = FrameTime1;
for (Index = 0; Index < NUMTHRESH; ++Index){
IRThresh[Index] = IRTimes[0] * (2*(Index+1) + 1);
IRThresh[Index] >>= 2;
}

/*-- now find the average frame time */
/* determine each interrupt duration */
/* (1.0 1.5 2.0... bit frame times)

AvgFrTime = 0;
FrameCount = 0;
for (FrameNum = 1;
(FrameNum < MAXIRFRAMES) && (IRTimes[FrameNum]);
++FrameNum) {
if (IRTimes[FrameNum] <= IRThresh[0]) {
continue;
}
else {
if (IRTimes[FrameNum] > IRThresh[NUMTHRESH-1]) {
continue;
}
}
}
}
```

(continued)

Listing P-continued

```

for (Index = 1; Index < (NUMTHRESH-1); ++Index){
  if (IRTimes[FrameNum] <= IRTresh[Index]) {
    FrameTime = IRTimes[FrameNum] * 2;
    FrameTime /= Index + 1;
    AvgFrTime += FrameTime;
    ++FrameCount;
    break;
  }
}
}
}
/*--- figure average frame time and display the result */

AvgFrTime /= FrameCount;
sprintf(CmdRespBuff, "Bit frame time is %u us\n", AvgFrTime);
Nputstr(CmdRespBuff);

} while (!chkch()); /* assume interrupts */

SerFlushIn(); /* discard inputs */
}

```

IRGATE to determine which ID codes have been "seen" since the last poll, and it must also determine how to handle new and missing IDs.

The "P" command tells IRGATE how many passive badges require polling. The default is "PO," or no

polling. Polling starts with ID 0 and goes up to (# badges) - 1, so "P10" will poll badges 0 through 9 inclusive. Your badges must respond to those ID codes for this feature to work!

The "I" command sets the polling interval, with the default being two

seconds when polling is enabled. You should adjust this value to suit your purposes, taking into consideration battery life and response time. The interval is the time between successive polls, so polling badges 0 through 9 with a 2-second interval would take 20 seconds. The minimum value is 20 ticks, which is about 100 ms.

The "B" command sets the amount of time IRGATE waits for a badge response after sending an ID code. The default is 40 clock ticks [about 200 ms]. When polling is not active, this delay determines the minimum time between IRGATE transmissions.

You can use remote control units as well as badges in the same system if you choose the ID codes carefully. I suggest you put all your control codes in the range 256-511 so the high bit distinguishes badges from remote controls and the low byte holds the entire ID code. If you are using passive badges, you must allow enough "dead air" between polls to accommodate remote control inputs; set the polling

Affordable 8031 Development

Single Board Computers, Assemblers, Compilers, Simulators, and EPROM Emulators

Control-R Series, Single Board Computers

Two models of Control-R series computers make prototyping, one of a kind products, or small production runs easy and economical. Both feature RS232 compatible serial ports, single 5 volt supply operation, and direct access to Ports 1 and 3 of the 8031. Additional features are as follows:

Control-R Model 1 \$49.95

Fully populated board with I/O header for Ports 1 and 3, serial port, and 8K EPROM socket. 3.0" x 4.0"

Control-R Model 2 \$79.95

Same features as the Control-R 1 plus 8K of SRAM and expansion bus with data, address, RST, INT1, WR, RD, PSEN, ALE and T1. 3.5" x 4.5"

Software and Hardware Development Tools

Control-C 8031 Cross-Compiler \$200.00

The Control-C 8031 cross-compiler is a full featured K&R style C development system available at an affordable price. Optimized for embedded system use, it will produce ROMable code for any 8051 based system including designs using only the 128 bytes of internal RAM. Package includes compiler, pre-processor, assembler, simulator, printed documentation and complete library source code. Requires IBM PC or compatible. 5.25", 360K disk.

PROMulator 256 \$189.95

An EPROM emulator lets you avoid "Bum and Test" development cycles. In circuit emulation of 2K-32K 27xx series EPROMs. ABS Plastic case. Assembled or compiled code is downloaded directly to the target hardware.

Cottage Resources Corporation

Suite 151, 10271 South 1300 East
Sandy, Utah 84094

VISA/MC, COD. Call to Order: (801) 268 - 2875

interval to 5 or 10 seconds to keep from clobbering yourself.

CALIBRATION CONSTANTS

Although I describe the IR signal as being either "on" or "off" when transmitting or receiving a message, that is not quite accurate. The IS1U60 receiver expects the IR signal to have a 38-kHz modulation, so an "on" signal is actually a burst of 38-kHz IR pulses.

Producing the modulating frequency with firmware isn't feasible (at least not while monitoring a serial network connection!), so Steve added some hardware to chop the output signal. But he insisted on a way to calibrate the oscillator against the 8031's 11.0592-MHz crystal rather than use a frequency counter or scope.

So IRGATE includes a transmitter calibration routine that displays the frequency of that modulating signal. To use it, just connect the oscillator to the 8031 T1 input pin, enter the "CT" command, and tune the frequency. You even get a simple tuning graph... try it and see!

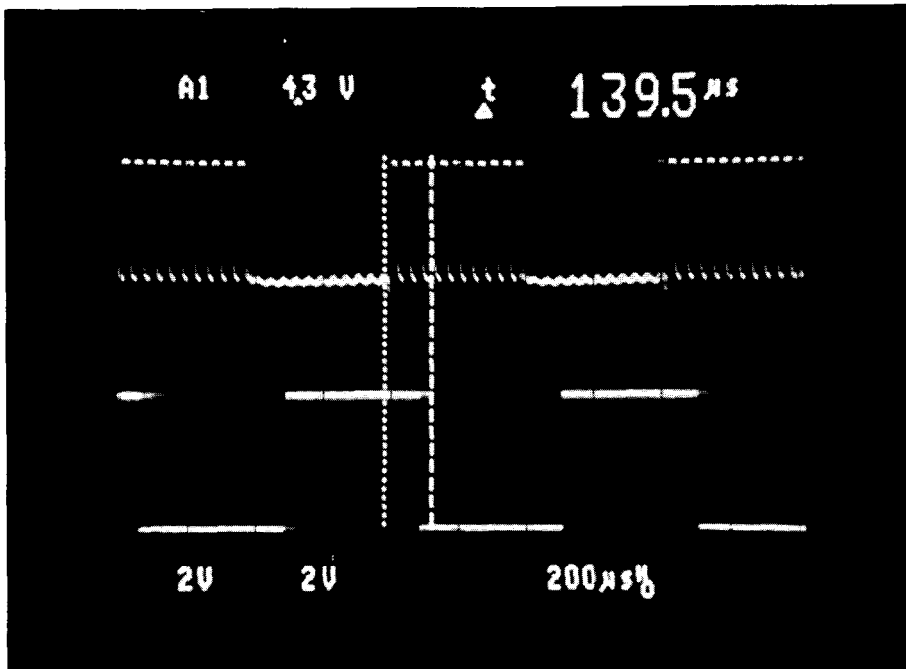
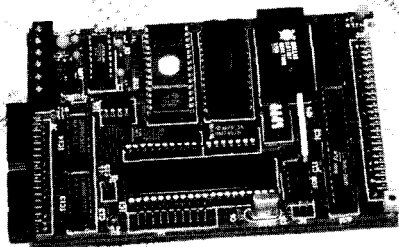


Photo 1-The IS1U60 needs some time to decide that it is seeing an IR signal and more time to decide that it is not. The top trace shows the raw IR signal, while the bottom trace shows the IS1U60's output.

A second calibration routine measures and displays the remote unit's bit frame time. The nominal value is 1290 us, but if you study Figure 2

again, you'll see how difficult it is to determine this value; remember you don't get an interrupt for each frame and you don't know what the data is!

ProControl!



Introducing the Most Expandable 3 1/2 Board Available

MCU-31/2 from \$149.95

On Board Options Include:

- * 16 Channels 10 bit A/D -14µs w/S/H
- * Dallas 1287 RTC
- * Flexible mem config's: RAM -8/32KB
ROM -8/16/32/64 KB
- * RS-232 or 485 -jumper selectable
- * Watchdog timer w/jumper sel. reset source
- * Pre decoded external bus for very easy user interface -
directly compatible with other ADS boards
(see back issues of Circuit Cellar INK)

More I/O modules are available.
Call for our FREE catalogue today!

(404) 352-4788

ADS Advanced
Design
Solutions
1920 Moores Mill Road
Atlanta, GA 303 18

#14

FAST CODE RELIEF!

Safe and effective for the whole 8051 family.

BCI51 Basic Compiler.

- Soothes assembly and C irritation
- * Long-lasting relief of time and budget headaches
- * Maximum strength error handling
- The affordable cure for long code winters!

Fast integer math-supplement...
BASIC-52 or stand-alone
outstanding technical support.

Only \$299 (Assembler & utilities included)

Dallas DS5000 extensions \$149
Assembly Language Toolkit \$149

Call 801-487-7412

FAX: 801-487-3130

SYSTRONIX Inc.

754 East Roosevelt Ave.
Salt Lake City, Ut. 84105

#150

My first inclination was to measure the pulse produced by the first two sync bits, which looks like it should be exactly one bit frame long. Unfortunately, the IS1U60 cannot switch instantly, because it requires some time to decide that it is seeing an IR signal and more time to decide that it is not. Photo 1 is a magnified view of a few IR bursts to show the IS1U60 response time.

I finally realized that the time between successive negative transitions of the IR signal must be 1.0, 1.5, or 2.0 bit frame times, so the code could extract the frame time without knowing the data values. The trick is to use the length of the initial sync pulse to estimate the frame time, then classify each transition time and compute the average frame time. The code in Listing 2 does just that.

Because Micro-C does not yet implement long integer (32-bit) variables, some of the computations are more laborious than you might expect. As an exercise, compute the frequency (near 12400 Hz) from the

average frame time (near 1290 us) without either overflowing 16 bits or discarding most of the significant bits.

RELEASE NOTES

The BBS files this time include **I RGAT E . HEX**, which is the EPROM data for the full-function IR gateway firmware. The source code for **I RGAT E . HEX** is available for licensing from Circuit Cellar Inc.

Also included are the Micro-C source files needed to create **I RMDN . HEX**, which is a receive-only, nonnetworked version of IRGATE. It will report all the MC145030 codes it "sees" and includes the calibration function needed to adjust the remote unit's oscillator to 12.4 kHz. You can use this code as the basis for receiving and decoding other remote control codes, if you are inclined to use a different chip.

Next time, a networked Home Control System LCD panel and keypad, so the HCS II Supervisory Controller can show you what it's up to and you can give it suggestions.

Ed Nisley is a Registered Professional Engineer and a member of the Computer Applications journal's engineering staff. He specializes in finding innovative solutions to demanding and unusual technical problems.

SOURCE

Please see page 31 for more information about the availability of HCS II components.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

IRS

- 419 Very Useful
- 420 Moderately Useful
- 421 Not Useful

The
only
8051/52
BASIC
compiler
that is
100 %
BASIC 52
Compatible
and
has full
floating
point,
integer,
byte & bit
variables.

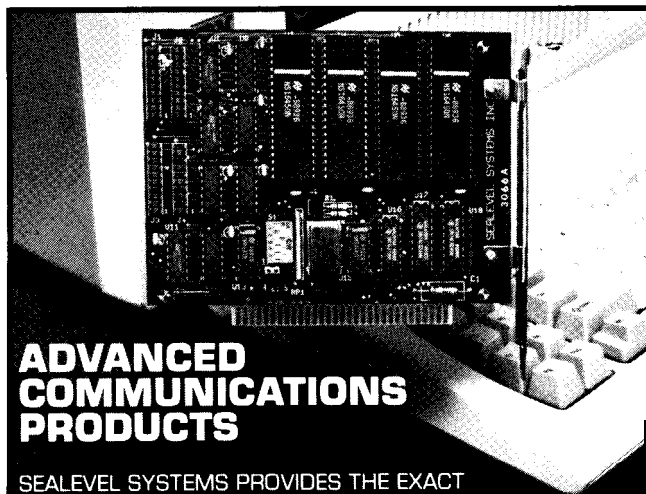
- Memory mapped variables
- In-line assembly language option
- Compile time switch to select 8051/8031 or 8052/8032 CPUs
- Compatible with any RAM or ROM memory mapping
- Runs up to 50 times faster than the MCS BASIC-52 interpreter.
- Includes Binary Technology's SXA5.1 cross-assembler & hex file manip.util.
- Extensive documentation
- Tutorial included
- Runs on IBM-PC/XT or compatible
- Compatible with all 8051 variants
- **BXC51 \$ 295.**

603-469-3232
FAX: 603-469-3530



Binary Technology, Inc.

Main Street • P.O. Box 67 • Meriden, NH 03770



ADVANCED COMMUNICATIONS PRODUCTS

SEALEVEL SYSTEMS PROVIDES THE EXACT COMMUNICATION CARDS YOU NEED.

PRODUCTS:

- 1, 2 OR 4 PORT RS-232 AND RS-422/485 BOARDS
- CURRENT LOOP SERIAL INTERFACES
- HIGH SPEED SYNC (HDLC, SDLC) AND ASYNC WITH DMA
- RS-530 AND V.35 INTERFACE BOARDS
- DIGITAL AND RELAY I/O BOARDS
- DISKLESS EPROM BOARD WITH PROMKIT SOFTWARE BY ANNABOOKS.
- NEW LAP-TOP ADD ONS!
- DELIVERY FROM STOCK
- MADE IN USA
- SATISFACTION GUARANTEED
- EXCELLENT TECHNICAL SUPPORT

 **SEALEVEL**
COMMUNICATIONS & I/O

SEALEVEL SYSTEMS INC.
PO BOX 830
LIBERTY, SC 29657
(803) 843-4343