IR - RAT
Infrared Remote Activity Transceiver
Universal Model

William Sands, Robert Thibadeau, Drew Anderson
Imaging Systems Laboratory
April, 1993
CMU-RI-TR-93-12

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Copyright © 1993 William Sands, Robert Thibadeau, Drew Anderson

This research was partially sponsored by the Defense Advanced Research Projects Agency (DOD) ARPA order 6873, under contract MDA972-92-J-1010. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Approved for public release
Distribution Unlimited

93-7 12 022
The IR-RAT was developed as the result of a need to remotely control computers. It is a microcontroller based infrared remote control interface. This document describes its design and operation so that it might be used or altered for use in the future.
TABLE OF CONTENTS

List of Figures 5
Abstract 7
1.0 Introduction 9
1.1 Conventions 9
2.0 Hardware design 10
2.1 Microcontroller 10
2.2 Memory 10
2.3 Serial Interface 10
2.4 Power 11
2.5 Reset Switch 11
2.6 Power Indicator 11
2.7 Infrared Receiver 11
2.8 Infrared LED 11
2.9 Jumpers 12
2.10 Misc. 12
3.0 Software (Firmware) design 13
3.1 Current Version 13
3.2 Infrared Signal Format 13
3.3 Misc. 14
Appendix A - Schematic Diagrams 15
Appendix B - Connector Diagrams 19
Appendix C - Single Chip Mode Board Modification 20
Appendix D - Source code for EPROM ready version 21

Imaging Systems Laboratory

IR-RAT -3
List of Figures

Figure 1. IR Representation of character A
Figure A-1. Schematic Diagram - Microcontroller
Figure A-2. Schematic Diagram - Memory
Figure A-3. Schematic Diagram - I/O
Figure A-4. Schematic Diagram - Power
Figure B-1. Modular Jack Diagram
Figure C-1. Single Chip Board Modification

Accession For

NTIS CRA&l
DTIC TAB
Unannounced
Justification

By
Distribution/

Availability Codes

Dist Avail and/or Special

A-1

DTIC QUALITY UNCLASSIFIED B
ABSTRACT

The IR-RAT was developed as the result of a need to remotely control computers. It is a microcontroller based infrared remote control interface. This document describes its design and operation so that it might be used or altered for use in the future.
IR - RAT
InfraRed Remote Activity Transceiver

1. Introduction

The IR RAT was developed and designed at Carnegie Mellon as the result of a need to use an IR remote control to operate computer software on several platforms. Therefore, the computer-RAT interface was made to be as simple as possible to allow for the most flexible use among different computers. The design and code information is publicly available. The end result of the design is a small box with the following features...

- Motorola 6811 microcontroller
- 8k x 8 EPROM for code and data storage
- Space for 8k x 8 RAM for software development
- 40 KHz carrier based IR receiver
- IR-LED capable of transmitting up to a 60 KHz carrier
- Standard RS-232 serial interface
- Powered from a 9V DC adaptor

The intended use for this box is to act as an IR remote control interface for any computer. It was designed to teach a learning remote a set of commands which, when sent by the learning remote, would cause a command to be sent to the host computer via the serial port. While this is the application for which the RAT is programmed, it could conceivably be used for a wide variety of applications. These include control by any 40 KHz carrier based IR remote control and control of virtually any IR remote capable device.

This document is intended to explain the design and implementation of the RAT so that others can understand its functionality and possibly adapt it to meet their own needs.

1.1 CONVENTIONS

Throughout this manual figures which represent hexadecimal values will be preceded by a $, values which represent binary values will be preceded by a %, and decimal values are not
2. Hardware design

2.1 MICROCONTROLLER

The IR RAT is based upon the Motorola 6811 family microcontroller. These are fairly general purpose 8-bit microcontrollers that are readily available. Most of the 6811 microcontrollers that are available in a 52 pin PLCC or ceramic quad pack are useable. For development, the MC68HC11A8FN1 was used to take advantage of the built in buffalo monitor that aided in development and debugging. For final production, a less expensive MC68HC11A0FN is used, as there is no need for internal EEPROM or masked ROM. In volume a masked ROM or EPROM version of the 6811 would be most suited as the code could easily fit in the 8 to 12 kBytes of ROM available. This eliminates the need for external support components as well as simplifying production.

2.2 MEMORY

On the IR-RAT board there are provisions to support one 8k x 8 EPROM and one 8k x 8 static RAM. The EPROM used is a 2764 or compatible unit in a 600 mil wide DIP package. The EPROM is assigned to the address range $E000-$FFFF in the 6811s addressing space as this corresponds to the ROM space that is used for most internal ROM/EPROM configurations on the 6811.

Space is provided on the IR-RATs PC Board for a 300 mil wide 8k x 8 static RAM. This RAM was intended for software development only as it is physically mounted underneath the EPROM. The RAM is assigned to the $6000-$7FFF address space.

2.3 SERIAL INTERFACE

The IR RAT is designed to communicate with the host computer via the serial port. This is pivotal in allowing the device to be used with as many different systems as possible. The RS-
232 signals are generated and received through the built in serial communications interface on the 6811 with signal levels supplied by a Maxim MAX-232 driver. The modular jack on the IR-RAT conducts the RS-232 signals. See the attached diagrams for the pinout of the modular jack.

2.4 POWER

Power can reach the IR RAT in one of two ways. First, the DC coaxial jack can be used to attach a 9V DC adaptor with a positive center conductor. Second power can be supplied through the outer two conductors in the modular phone jack. As shown in the schematics, each of these power sources pass through separate barrier diodes to prohibit any harmful connections. For pinouts of the modular jack see the attached diagrams.

2.5 RESET SWITCH

There is space on the IR-RAT board adjacent to the modular connector for a reset pushbutton switch, S1. This is a normally open push-button switch that resets the microcontroller. The switch is debounced through the reset controller, U7. This switch appears only on development units, as final production units should not need a reset circuit. The software should be robust enough to avoid the need for an external reset.

2.6 POWER INDICATOR

LED D4, and resistor R5 are only necessary on development units to indicate presence of power.

2.7 INFRARED RECEIVER

The IR receiver is the Sharp GP1U52Y, which is a receiver/demodulator that detects the presence of a 40 KHz carrier. This connects to port A, bit 2, which is also the Timer Input Capture 1. This signal has a high resting state and becomes low only when a 40 KHz carrier is detected.

2.8 INFRARED LED

D1 is an IR LED that is designed to transmit IR signals. It is controlled directly by the output of I/O port A, bit 3. As a result, it can be controlled by the timer output compare 5 (TOC5).
When this value is high, the LED is on. For development, resistor R3 should be approximately 100 ohms to limit the current that the LED can draw. When it is shown that the LED will operate only at some reduced duty cycle (i.e. 50%) then a lower value can be used (approx. 50 ohms) to allow for a more powerful IR signal. This should occur only when the program is sure not to cause the LED to stay on for extended periods (as the Motorola buffalo monitor does).

2.9 JUMPERS

There are five jumpers on the IR-RAT PC board. Jumper 1 (the left most jumper) is used to place the chip in expanded multiplexed mode (jumper removed) or in special test mode (jumper present). In order to operate the microcontroller in single chip mode, hold the MOD A line low. This is accomplished by cutting the trace which connects jumper 2 to the pull up resistor network, and reattaching the non-grounded side of the jumper to the pad on the pull up resistor network that connects to MOD A (Pin 5). Then by putting a jumper across jumper 2, and leaving jumper 1 vacant the microcontroller is placed in single chip mode. See the attached diagram for the procedure.

The remaining four jumpers are attached to port E, pins 0-3. Pin 3 corresponds to jumper 5, pin 2 to jumper 4, pin 1 to jumper 3, and pin 0 to jumper 2. These are provided to allow for jumper selectable options. Note that use of the Motorola buffalo monitor requires that jumper 5 be installed. If it is not in place the buffalo monitor will attempt to start execution from the EEPROM at address $B600.

2.10 MISC

If an EPROM version of the 6811 (i.e. 68HC711) is available and desired for use, it is possible to reduce the part count on the board significantly. Assuming that there is no need for the RAM, the address decoding and bus latching chips, U2 and U3 can be eliminated. This is also possible if a ROM masked version of the microcontroller is manufactured.

The PC boards were fabricated by Photobeam/Brookside in Waltham, Massachusetts. Most of the other parts are commonly available from a variety of electronic vendors. The only specialized part was the Sharp IR received/demodulator.
3. Software (Firmware) Design

3.1 CURRENT VERSION

The current version of the firmware available for the IR-RAT performs two very simple functions. The first is that it gets an ASCII character from the serial port and then encodes and transmits that character as an IR signal. This is the transmitter portion of the code. The second function is it receives IR signals from a remote control and if they are in the appropriate format, decodes the ASCII character the signal represents and send that character over the serial port to the host computer. This is the receiver portion of the code. While the code is fairly self explanatory, some brief explanation is required.

3.2 INFRARED SIGNAL FORMAT

The IR signal is created by turning the 40 Khz carrier on and off for different periods of time. The periods of time are measured by the number of cycles of the 40 kHz carrier. The current encoding scheme uses two values for the periods, these are 200 cycles (long) and 20 cycles (short). By using an order of magnitude difference, determination of which value is correct becomes trivial.

IR signals from the RAT are composed of four parts, the sync, the header, the data and the inverted data. The sync is a long on pulse. This allows the receiver to "wake up" and the microcontroller time to prepare for receiving data. Once the sync pulse has been sent, all of the remaining periods of carrier presence (when the IR LED is flashing) are short, and the data is determined by the length of time that the LED is not flashing. Therefore a short is considered to be 20 cycles of no IR activity followed by 20 cycles of 40 kHz carrier and a long is 200 cycles of IR inactivity followed by 20 cycles of 40 kHz carrier.

After the sync pulse is sent, a header is sent to identify the signal as a valid RAT signal. This is a short-long-short-long-short pattern. If the decoder does not see this pattern after it receives the sync, it assumes that the IR signal is from some other source and waits for the next signal.

Next the eight bits of data representing the ASCII character are sent from least to most significant. A one is represented by a short signal and a zero is represented by a long signal. Following this the data is sent again, except this time the representation is inverted with a one
represented by a long signal and a zero represented by a short signal. By using this method, all characters sent will take exactly the same amount of time. An example of the IR signal representing the letter A (%01000001) is shown in figure 1.

![IR Representation of character A (%01000001). Note that this is not to scale. In reality long and short times vary by a factor of 10.](image)

3.3 MISC

There are a few notes that need to be made about the current firmware for the RAT. First it is interrupt based. When the program initializes it sets registers appropriately and then waits for an interrupt to occur, either from the serial communications interface or the timer input capture 1.

Second, the IR RAT depends on relocating the internal RAM to location $1000$ and the configuration registers to start at $0000$. This is done to permit the tight loops necessary to generate a $40 \text{ kHz}$ carrier using a $2 \text{ MHz}$ crystal and the internal timers on the 6811.

Finally, the program operates in batch mode. When transmitting the program computes all of the timing in advance and then starts the transmitting. While receiving the microcontroller receives all the timer data and stores it for processing after a sufficient amount of time has passed without IR activity.
Appendix B - Connector Diagrams

![Diagram](image)

Figure B-1. Front view of the modular jack on the IR-RAT.

The other end of the modular plug wire can attach to a female dB-25 connector for use with most PCs. The wiring connections are as follows.

<table>
<thead>
<tr>
<th>Modular Cable</th>
<th>dB-25</th>
</tr>
</thead>
<tbody>
<tr>
<td>YELLOW</td>
<td>Pin 7</td>
</tr>
<tr>
<td>RED</td>
<td>Pin 3</td>
</tr>
<tr>
<td>GREEN</td>
<td>Pin 2</td>
</tr>
<tr>
<td>BLACK</td>
<td>Not Connected</td>
</tr>
</tbody>
</table>

Additionally, for prototyping with Motorola's Buffalo monitor pins 5, 6, 8 and 20 should be shorted together.
Figure C-1. This shows the necessary board modifications to allow jumpers 1 and 2 to control MOD B and A respectively.
Appendix D - Source Code for EPROM Ready Version

* IR-RAT SOURCE CODE - EPROM VERSION of ratv2.s
* BY: William H. Sands IV whs@j.gp.cs.cmu.edu
* Created February 26, 1992
* Copyright (c) 1993 - Carnegie Mellon University
*
* Following is the source code for the IR-RAT. This version
* is a complete version that can be compiled and loaded directly
* into EPROM. It is designed for EPROM at $E000 - $FFFF.

* Constants
BDRATE EQU $30 $31 for 4800, $30 for 9600
TIMEDELAY EQU 1 mS TO DELAY
LONG EQU 200
SHORT EQU 20
BREAK_PT EQU 100

* Memory map alias
RAMBS EQU $1000 start of internal ram
EXRAMBS EQU $6000 start of external ram
REGBS EQU $0000 start of registers
INTVECBS EQU $FFD6 where interesting int vectors start

* Registers
PORTA EQU REGBS+$00 I/O port A
OC1M EQU REGBS+$0C
OC1D EQU REGBS+$0D
TCNT EQU REGBS+$0E timer count
TIC1 EQU REGBS+$10 timer input capture 1
TOC1 EQU REGBS+$16
TOC5 EQU REGBS+$1E
TCTL1 EQU REGBS+$20 timer control 1
TCTL2 EQU REGBS+$21 timer control 2
TMSK1 EQU REGBS+$22 timer mask 1
TFLG1 EQU REGBS+$23 timer flag 1
TMSK2 EQU REGBS+$24 timer mask 2
BAUD EQU REGBS+$2B
SCCR1 EQU REGBS+$2C
SCCR2 EQU REGBS+$2D
SCSR EQU REGBS+$2E
SCDR EQU REGBS+$2F
HPRIO EQU REGBS+$3C highest priority interrupt
INIT EQU $103D
OPTION EQU REGBS+$39

*Internal memory allocations
STACK EQU RAMBS+$40

Imaging Systems Laboratory

IR-RAT .21
LOWMEM EQU RAMBS+$00
HIMEM EQU RAMBS+$FF

ORG $E000

RESET VECTOR - This is where eprom execution begins on any reset
COP/CLOCK/EXT RESET/PWR UP etc.

RESET
LDAA #$13 Set up sys configuration options (OPTION)
STAA OPTION A/D power off - LEVEL IRQs - is COP TIMEOUT
LDAA #$10 Set RAM to $1000 and Registers to $0000
STAA INIT
LDAA #$0 Mask all timer interrupts
STAA TMSK2
LDS #STACK Set the stack pointer
JSR ONSCI Set up the SCI Port
JMP main Start execution
ERRI JMP ERRI

UNK_INT
LDAA #$50 This routine is the unknown interrupt
TAP handler. Simply hangs the machine.
STOP (necessary)
JMP UNK_INT

PORTIONS OF OUTA, OUTSCI and ONSCI ARE FROM THE 6811 BUFFALO MONITOR V2.5

OUTA
PSHA
PSHB
PSHX
JSR OUTSCI
PULX
PULB
PULA
RTS

OUTSCI
BSR OUTSCI2 Puts a character out on the SCI
CMPA #$10D
BNE OUTSCI1
LDAA #$10A
BRA OUTSCI2

OUTSCI1
CMPA #$10A
BNE OUTSCI3
LDAA #$10D

OUTSCI2
LDAB SCSR
BITB #$80
BEQ OUTSCI2

Imaging Systems Laboratory
ANDA #$7F  
STAA SCDR  
RTS  

* 
ONSCI  
LDAA #$BDRATE  
STAA BAUD  
LDAA #$00  
STAA SCCR1  
LDAA #$0C  
STAA SCCR2  
RTS  

*** START OF ACTUAL IR RAT CODE ***

main  
SEI  
LDS #STACK  
Reset stack pointer  
LDAA SCCR2  
Turn on serial RDRF IRQ  
ORAA #$20  
STAA SCCR2  
Make sure level sensitive IRQ  
LDAA #$13  
STAA OPTION  
Discon. output pins from timer  
CLR TCTL1  
Make sure IR LED is off  
CLR PORTA  
Set input capture 1 to trigger  
LDAA #$30  
STAA TCTL2  
Set highest priority input to be  
LDAA HPRIO  
on rising edge of signal  
ANDA #$F0  
STAA HPRIO  
Un-mask timer input capture 1  
ORAA #$08  
STAA TMSK1  
LDX #stoend  
Point to where to store data  
loop_temp  
CLI  
WAI  
JMP main  
Go into waiting until an interrupt  

**** RECEIVER ****  
* This module receives the IR signal.  
* When the IR receiver is activated, it causes an interrupt which  
* forces execution to begin at timer_irq. It will continue to look  
* for data until it has filled memory, or it has lapped the timer.  

timer_irq  
LDX #stoend  
Point to where to store data  
LDD TIC1  
Get Timer Event Data  
STD 0,X  
STD TOC1  
Reset the timeout timer  
LDAA #$84  
STAA TFLG1  
Reset the interrupt flag  
LDAB #$80  
STAB TMSK1  
Leave only the timeout IRQ  
Imaging Systems Laboratory  

IR-RAT -23
CLI

timer_irq1

    DEX
    DEX
    CPX #stor1
    BLE timeout

Test for out of memory

irq_loop

    BITA TFLG1
    BEQ irqloop
    LDD TIC1
    STD 0,X
    STD TOC1
    LDAA #$04
    STAA TFLG1
    BRA timer_irq1

Tight loop to check for IR activity

timeout

    BRA timeout

Endless loop for timing out

* This interrupt routine is called when the timeout timer has lapped with no IR activity. This is the post processing routine

timeout_irq

    LDAA #$00
    STAA TMSK1
    LDAA #$80
    STAA TFLG1
    STX point
    JSR find_freq
    JSR on_off
    JSR recog
    PULX
    PULX
    PULX
    PULX
    PULX
    PULX
    PULA
    ORA #$10
    PSHA
    LDX #main
    TSY
    STX 7,Y
    RTI

Mask all timer events from IRQ
Reset Timeout flag
Store the end of data
Determine the freq of signal
Calculate the on/off cycles
Perform recognition on signal
Pull this interrupt off the stack
Get CCR from stack
Set Interrupt bit
Put CCR back on stack
Hardcode a return to main
Go back to main

* Since we are dealing with a fixed frequency, this simply says that the frequency is 50 cycles, which on a 2 MHz micro is 40 kHz.

find_freq

    LDX #$0032
    STX freq
    RTS

* This routine handles computing the values that represent the series of on/off cycles. Like the actual timer data, the on/off frequency values are stored from high to low memory. They are 16 bit values that terminate with a zero value. They represent the number of cycles that the signal is alternatively on and off.
on_off  
  LDY #stoend  Point to start of data
  STY count
  DEY
  DEY
on_off_loop  
  CPY point  Check to see if we are at the end
  BNE finis
  LDD 0,Y  Find the difference between the current value and the next.
  SUBD 2,Y  Goto the next timer event
  DEY
  DEY
  LDX freq  Determine the number of cycles that the difference represents
  IDIV
  LSLD
  CPD freq  Perform rounding of the number of cycles
  BLT no_round
  INX
no_round  
  PSHY
  LDY count  Store the results
  STX 0,Y
  DEY
  DEY
  STY count
  PULY
  JMP on_off_loop
finis  
  LDX #$0000  When we are done, store a 0000 as a tail
  LDY count
  STX 0,Y
  RTS

** This is the ending routine for recog. Needs to be here so that branches can access it.

end_delay  
  LDX #TIMEDELAY  DELAY FOR X mS
delay_bit  
  LDD O,X  First check from a long on pulse
  CPD #BREAKPT  BGT no_end_here
  JMP end_delay
no_end_here  
  DEX

* This is the recognition routine that determines if this is an IR-RAT signal and if it is sends the ASCII character over the serial port.

recog  
  LDX #stoend
  LDD 0,X
  CPD #BREAK_PT  First check from a long on pulse
  BGT no_end_here
  JMP end_delay
no_end_here  
  DEX

Imaging Systems Laboratory  IR-RAT -25
DEX
JSR  get_next
BGT  end_delay
JSR  get_next
BGT  end_delay
JSR  get_next
BGT  end_delay
JSR  get_next
BGT  end_delay
JSR  get_next
BGT  end_delay
CLRA
CLRB
CLR  smal
LDY  #$0009
first_char LSRA  Get first copy of the character
ABA
PSHA
JSR  get_next_2
PULA
BLT  first_short
CLRB
BRA  first_long
first_short LDAB  #$80
first_long TST  smal
BNE  end_delay
DEY
BNE  first_char
STD  point
CLR  count
JSR  comp_parity
CMPS  point+1
BEQ  no_par_error
LDAA  #$01
STAA  count
no_par_error CLRA
CLRB
LDY  #$0009
second_char LSRA
ABA
PSHA
JSR  get_next_2
PULA
BGT  second_long
CLRB
BRA  second_short
second_long LDAB  #$80
second_short TST  smal
BNE  par_2_err
DEY
BNE  second_char
STD  freq

Imaging Systems Laboratory

IR-RAT -26
```
JSR comp_parity
CMPB freq+1
BEQ no_par_2_err
par_2_err
TST count
BNE end_delay_j
LDD point
BRA print_it
no_par_2_err
LDD freq
TST count
BNE print_it
CMPA point
BNE end_delay_j
print_it
JSR OUTA
end_delay_j
JMP end_delay

get_next
LDD 0,X
BEQ end_found
DEX
DEX
DEX
DEX
DEX
CPX #stor1
BLT end_found
CPD #BREAK_PT
RTS
end_found
PULX
JMP end_delay

get_next_2
LDD 0,X
BEQ end_found_2
DEX
DEX
DEX
DEX
DEX
CPX #stor1
BLT end_found_2
CPD #BREAK_PT
RTS
end_found_2
LDAA #$FF
STAA smal
RTS
comp_parity
LDY #$0008
CLRB
parity_loop
LSLA
BCC no_parity
INCB
no_parity
DEY
BNE parity_loop
ANDB #$01
LDY #$0007
par_shift
LSLB
```

*Imaging Systems Laboratory*
DEY
BNE par_shift
RTS

***********************************************************************
* TRANSMITTER
***********************************************************************

serial_irq LDAB SCSR
LDAA SCDR
STAA point Temp holder for character
BITB #$20
BEQ end_serial
JSR comp_parity
STAB freq Temp holder for parity
JSR compute_sig
JSR find_in_freq
JSR flash
end_serial PULA Get CCR from stack
ORA #$10 Set Interrrupt bit
PSHA
LDX #main Hardcode a return to main
TSY
STX 7,Y
RTI

compute_sig LDX #stoend
LDY #LONG Store long on pulse
STY 0,X
DEX
DEX
LDY #SHORT
JSR store_pulse
LDY #LONG
JSR store_pulse
LDY #SHORT
JSR store_pulse
LDY #LONG
JSR store_pulse
LDY #SHORT
JSR store_pulse
LDAA point
LDAB #$08 Store first copy character
first_send LDY #SHORT
LSRA
BCS first_l_shrt It's a one (short)
first_l_ing LDY #LONG It's a zero (long)
first_l_shrt JSR store_pulse
DECB
BNE first_send
LDY #SHORT

Imaging Systems Laboratory
TST freq
BNE con_p_first
LDY #LONG
It’s a one (short)
con_p_first JSR store_pulse
LDAA point
LDAB #$08
It’s a zero (long)
second_send LDY #LONG
LSRA
BCS con_second
It’s a one (long)
second_l_lng LDY #SHORT
It’s a zero (short)
con_second JSR store_pulse
DECB
BNE second_send
It’s a one (long)
TST freq
BNE con_p_second
It’s a zero (long)
con_p_second JSR store_pulse
LDY #$0000
LDY #LONG
TST freq
BNE con_p_second
It’s a one (long)
LDY #SHORT
It’s a zero (long)
con_p_second JSR store_pulse
LDY #$0032
LDX #$0000
JSR store_pulse

find_in_freq LDX #$0032
STX freq
RTS

flash LDAA #$00
STAA TMSK1
turn off interrupts/etc
STAA TCTL1
turn off control to port
init_flash LDD TCNT
Give some time to start
ADDD #$100
STD TOC5
CLR PORTA
turn off LED
CLR OC1M
turn off OC1
CLR OC1D
LDAA #$01
STAA TCTL1
LDAAA #$0F8
STAA TFLG1
LDX #stoend
STX point
LDD freq
LSRD
STD small
carrier_on LDY 0,X
get first on val
BEQ done_flash
DEX
DEX
STX point
reset_on LDX TOC5
LDAB small+1
LDAA #$08
on_flash BRCLR TFLG1 #$08
on_flash

Imaging Systems Laboratory
IR-RAT -29
off_flash

BRCLR TFLG1 #$08
STX TOC5
STAA TFLG1
DEY
BNE on_flash
CLR TCTL1
LDX point
LDD 0,X
BEQ done_flash
LDA freq+1
MUL
STD count
LDD 0,X
LDB freq+1
MUL
TBA
CLR B
ADDD count
SUBD small
ADDD TOC5
STD TOC5
LDAA #$08
STAA TFLG1
LDAA #$01
STAA TCTL1
DEX
DEX
STX point
BRA carrier_on
done_flash
CLR TCTL1
CLR PORTA
CLR TMSK1
RTS

store_pulse

STY 0,X
DEX
DEX
LDY #SHORT
STY 0,X
DEX
DEX
RTS

*** Interrupt Vectors ***

ORG INTVECS

FDB serial_irq
FDB UNK_INT

serial_irq SCI Serial System
SPI Transfer Complete

Imaging Systems Laboratory
<table>
<thead>
<tr>
<th>FDB UNK_INT</th>
<th>Pulse Accumulator Input Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDB UNK_INT</td>
<td>Pulse Accumulator Overflow</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Overflow</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Output Compare 5</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Output Compare 4</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Output Compare 3</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Output Compare 2</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Output Compare 1</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Input Capture 3</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Input Capture 2</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Timer Input Capture 1</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Real Time Interrupt</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>¬-IRQ</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>¬-XIRQ</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>SWI</td>
</tr>
<tr>
<td>FDB UNK_INT</td>
<td>Illegal Opcode Trap</td>
</tr>
<tr>
<td>FDB RESET</td>
<td>COP Failure</td>
</tr>
<tr>
<td>FDB RESET</td>
<td>COP Clock Monitor Fail</td>
</tr>
<tr>
<td>FDB RESET</td>
<td>Reset Vector</td>
</tr>
</tbody>
</table>

*** Program RAM allocations ***

```plaintext
ORG STACK+2
smal FCB 0
    FCB 0
freq FCB 0
    FCB 0
point FCB 0
    FCB 0
count FCB 0
    FCB 0
storl FCC
    ORG HIMEM-1
stoend FCC
END
```

*Imaging Systems Laboratory*