

INTERNAL FUNCTION RAM

By Richard Curcio

I.F.R. REVISITED V2.1

My Internal Function Ram project as it appeared in TC128 #29 has problems. This project allowed the C128 to have a Static Ram in the empty socket reserved for a function ROM or Eprom. The circuit included battery-backup so the Static Ram could retain its contents while the computer was turned off.

I offer no excuses for not detecting the problems sooner. Instead, I offer a fixed and (serendipitously) more versatile design, which will, if you already have a ROM in the empty socket, allow selection of it or the Static Ram, via a switch or software. The program for the original project has been revised as well.

I apologize for any frustrations the earlier project may have caused.

OBSTACLES

A detailed recounting of how the original design was supposed to work and why I thought that all was well - in other words, what I had overlooked - would take up too much space. The older circuit won't harm the C128 and will perform as described - with limitations. If I took the time to explain those limitations, I'm sure most readers would agree that they're unacceptable. So let's just get right to the new circuit.

The Bank 4-7 configurations of the C128 select internal function ROM. In these standard banks a ROM or Eprom in socket U36 occupies \$8000 to \$FFFF with a 4K gap at \$D000 for I/O. You can't just plug a Static Ram (Sram) into the empty socket because 1) R/W is not present on the function ROM socket and 2) when the Programmed Logic Array (PLA) detects a Write to what is supposed to be a ROM location, the function ROM enable (called /FROM in the P.R.G schematic), "goes away" and system Ram is instead enabled and written. Since the PLA won't permit writing to a ROM or anything else in the U36 socket, the new design bypasses the PLA. This is not as difficult as one might expect.

Two signals from the MMU called MS0 and MS1 tell the PLA to enable system ROM, external or internal ROM, or system Ram. When MS0 = 0 and MS1 = 1, internal ROM is selected. The PLA considers a number of other signals in determining which enable to generate. When R/W

is low, the PLA disregards the state of MS0/1. The new circuitry removes R/W from the decision and substitutes, the "1" period of the two-speed system clock which is when the processor has control of the address and data busses. The result is a nice, clean enable of the proper duration.

THE CIRCUIT

Figure 1 shows the new design in a mix of mechanical and schematic representation. The static Rams inserted in the C128's empty ROM socket with its pins 1, 20, 27 and 28 bent out. A 74HC138 decoder receives MS0, MS1 and the 2MHz clock on its inputs and its Y3 output enables the Sram. This decoder MUST be an 'HC' part if battery-backup is included. The Sram and the HC138 are powered by the computer's +5 volts OR the battery. If you choose to omit the battery and diodes, insert the Sram's pin 28 into the socket and connect pin 16 of the HC138 to +5 volts.

The Sram's Write Enable input /WE connects to FR/W, a Read/Write internal to the C128. (It's called R/WA in the SAMs schematics). It does not appear on the expansion connector, so an REU or other external device cannot pull it low and alter the Sram. (A REU-initiated Write to what is supposed to be a ROM bank will simply "fall-through" to underlying system Ram, leaving the Sram untouched). In the low-profile C128, FR/W is available at pin 1 of U57, inside the video box which is quite close to the empty ROM socket. In the C128D, this signal is at pin 3 of U61, near the left side of the main circuit board and also close to the empty socket.

The nearest point to get the 2MHz clock on the low-profile C128 is at pin 4 of U22, the 80 column controller, which is also inside the video box. On the C128D, different versions of this chip have 2MHz on different pins. For certainty, obtain this signal at pin 1 of the 8502 microprocessor, U6. MS0 and MS1 are at pins 18 and 17 respectively of the PLA, U11, which is located center front of both the low-profile C128 and the C128D. (Although they are rare, there are C128D's in existence that use a low-profile mother-board. For a positive determination, count the number of dynamic Rams at the front left of the board. Sixteen 16-pin ICs indicates a low-profile board, while four 18-pin chips means you have a true C128D).

Because the pin-out of the Sram is slightly different from that of a ROM or Eprom, pin 1 must be bent out and connected to address bit A14. On my flat C128 I found A14 at the feed-through immediately to the right of the "2" of the identifier "R32" near U32 and U33. On the C128D, A14 is at pin 27 of ROMs U32 and U34 or pin 18 of U42 (74LS244).

The 32K x8 static Ram, could be a 62256, 43256 or 58256. For maximum battery life it should be a low-power device (-L or -LP suffix), having a "standby" current consumption of 100 microAmps. Regular power Srams have a standby current of 2 milliAmps, which is still pretty miniscule.

I installed the HC138 in my low-profile C128 by sticking it to the main board upside-down using double-stick foam tape. Connections were made by wire wrapping directly to the pins. Lithium "coin" batteries are available with solder-tabs, and these can have wires soldered to them. A holder for tab-less coin batteries can be constructed with two rectangular pieces of un-etched printed circuit board; solder wires to the bare copper, put the battery in between the two pieces and hold together with rubber bands and wrap the sandwich with paper and more rubber bands. (Of course the polarity of the battery is important, but the diodes will prevent reversed polarity from damaging the Sram, HC138 or the computer itself). This can then be tucked behind the video box or someplace where it won't flop around. In the C128D, there's enough room for a holder for two AA cells. These won't last as long as lithium, but they should last their shelf life - at least a few years.

Note that the Sram in the U36 socket still receives the PLA generated /FROM on its Output Enable (/OE) pin. Normally the PLA enables each ROM via /OE. A write will therefore disable the Sram output and that's fine; for the Sram, /OE is "don't care" when /WE=0. (Certain other memory ICs require that /OE=1 when /WE=0).

OPTIONS

When the optional Write Protect switch within the dotted lines in figure 1 is open, FR/W can't reach the Sram. Use this to prevent accidental writes to the Sram or to fully emulate an Eprom.

The 10k resistor on the HC138 B input insures that the proper combination of inputs causes output Y3 to go low, selecting the Sram. If B is made low by grounding it with the optional

Select switch, then another Sram or an Eprom could be selected by connecting its /CS to Y1 (pin 12). One would think that, since the C128 has only one empty socket, this idea is mere theory. There are ways to obtain the room for a second 28-pin IC, but different methods are needed for the low low-profile, and C128D. These will be covered later on.

TESTING/SOFTWARE

After installing the circuit, use the Machine Language Monitor "m" command to display the first page of the IFR: M 48000 <return>. You should see random values. Cursor up to the M command and hit return. If any of the previously displayed values change, bank 4 is empty - the Sram isn't getting enabled. Turn off your computer and find your mistake. If the random bytes remain constant, fill that first page with some value, say \$55 or \$AA:

```
F 48000 480FF 55
```

Now use the M command to confirm that the fill was successful. If it was, turn off the computer, wait a few seconds, then power up and confirm that the battery-backed Sram retained the fill value.

Before your Internal Function Ram can be used from Basic, it must be initialized. Use the mlm Transfer command to copy the routines in the last page of system ROM to the same locations in Bank 4:

```
T FFF05 FFF44 4FF05
```

Cursor up and change the "T" to a "C" (Compare) and hit return. The mlm should print nothing, indicating that all the locations match. Now transfer the system vectors to the last six bytes of Bank 4:

```
T FFFFA FFFFF 4FFFA
```

Ignore the "?" that appears upon completion. You can now safely access your IFR with PEEK, POKE, BLOAD, BSAVE and SYS. (The mlm safely accessed bank 4 before the initialization because it disables interrupts during "M", "F", "T" and "C"). Bear in mind, however, POKE and BLOAD will also alter underlying system Ram (Ram 0 when Bank 4). Note also that the standard IFR Banks 4-7 and 12 include I/O in the \$D000-DFFF range.

Program 1 is the loader for a Mover which will transfer data to the IFR without altering the

system Ram under it. It is designed to reside and execute in the IFR. The ml can be relocated to a different start address by changing the variable SA in line 200. SA must be between 32768 and 48924. Access the routine with

BANK 12: SYS SA, HOST BANK,
DIRECTION,,, HOST START, HOST END,
IFR START

You MUST use Bank 12, which includes the Kernal and I/O. The host bank is system memory in the standard Bank configurations; 0-3, 14 and 15. Banks 4-13 are not allowed. This restriction can be bypassed. Direction is zero to move data TO internal function RAM and greater than zero to recall data FROM that Ram. The three commas must be present. Host start and end are self explanatory, while IFR start must be at least 32768. No address can be greater than 65279, as that would affect MMU registers at \$FF00-\$FF04 and the important routines and vectors in page \$FF. As the routine moves data, it checks its pointers and will halt the move and set the Carry bit if either the source or destination reaches page \$FF. Carry is also set if either pointer "wraps" to zero page - which theoretically can't happen, but one never knows. Use the RREG function to test the Carry from Basic. The routine does not detect if an IFR destination will write over the Mover itself.

The Mover calls the system INDSTA and INDFET routines at their Rom 0 locations instead of through the Kernal jump table. Calling these routines via the jump table is time consuming, because the bank number is converted to a configuration value for each byte. To speed things up, the Mover performs the host bank to configuration conversion just once. Also to access \$D000-\$DFFF of the Sram a modified Bank 4 must be used because the standard Banks include I/O in that range.

Because writing to the Sram also writes system Ram, the Mover performs two loads and stores for each byte moved to the IFR. First the byte in Ram 0 "under" the IFR destination is read and stored on the stack. Then the host source byte is written to the IFR, changing Ram 0 in the process. The byte on the stack is then restored to Ram 0. In this way, we do not lose the use of the underlying system Ram. This is why Banks 4-13 are not allowed as "host". If the "from" portion of the routine were to move data to the IFR as host, the byte under the IFR destination would be lost. The "to" operation would work properly, but this is a "dumb" mover; if the source and

destination are in the same bank, and they overlap, the move becomes a fill. If you really need to move data around inside the IFR, and don't care about the under-bytes, you may use Banks 4-13 as host by calling the routine at sa+27.

The "to" portion of the Mover assumes that the MMU Pre-Configuration Register at \$D502 (PCRA) contains its default value, selecting the Bank 0 configuration when any value is stored in the corresponding Load Configuration Register at \$FF01 (LCRA).

I must emphasize that writing to Bank 4 or the other internal function ROM configurations enables the Static Ram and system Ram simultaneously, a very different situation than what normally occurs when attempting to write to ROM. Any routines you place in your IFR should not use the short-cut method of writing to an internal function ROM region to accomplish a write of system Ram that is not at the moment visible. Use INDSTA. However, if the optional Write Protect switch is installed and opened, then the Sram is as unwritable as a ROM or Eprom.

C128D PLUG-IN BOARD

In a C128D, to have a choice of Sram or Eprom in Bank 4, a board can be constructed to plug into the empty ROM socket. As shown in figure 2, socket 1 is intended to hold an Eprom. This should be a wire-wrap socket with "2-level" length pins. These are long enough to allow the plug-in board to clear the main board components that will be under it; 3-level length pins are acceptable, but longer than necessary and you may have trouble keeping them properly aligned. The perforated board should have "pad-per-hole" copper plating so that the socket can be firmly soldered to it. Pin 20 of socket 1 is cut close to the board so it does not make contact with the corresponding receptacle of the main board U36 socket. The two other sockets could be solder "tail" or wire-wrap with the pins cut as short as possible.

Pins 28 and 1 of socket 1 bring +5 volts to the plug-in board, while pin 14 supplies ground. Each pin of socket 1 is wired to the same pin of socket 2 EXCEPT for pins 1, 20, 27 and 28. Pin 27 of socket 1 connects to pin 1 of socket 2 (address bit A14). Pin 27 of socket 2 is the Sram Write

Enable /WE and it gets wired to FR/W, either directly or through the optional switch. Both 28-pin sockets continue to receive /FROM at their /OE pins 22. Pin 20 of each socket gets wired to the specified HC138 outputs. All other signals from the main board can connect directly to the plug-in or better yet, through connectors and pins so that the whole assembly can be removed without unsoldering. There's a large hole, in the front of the C128D chassis which will permit wires from the plug-in to reach any switches you mount on the front panel. Other, smaller holes can be used to secure a double AA battery holder using twist ties.

To have a second Sram instead of an Eprom, the simplest method would be to plug it into socket 1 with its pins 1, 22, 27 and 28 bent out and wired as shown in figure 1. As illustrated in figure 2, the plug-in is somewhat roomier than absolutely necessary, but about as roomy as it ought to get.

LOW-PROFILE STRATEGY

To gain another 28-pin socket in the low-profile C128 you need an Eprom programmer so that the two 16K ROMs holding Basic and the Machine Language Monitor can be combined into one 27256 32K Eprom. One ROM, U33, holds the Basic interpreter from \$4000 to \$7FFF (call it baslo). The other ROM, U34, holds the rest of Basic and the MLM. (bashi, \$8000 to \$BFFF). The original ROMs, can be copied without removing them from the system board by saving their contents to disk:

```
BSAVE "BASLO", B15, P16384 TO P32768 (end + 1)
BSAVE "BASHI", B15, P32768 TO P49152
```

Burn baslo into the lower 16K (0-\$3FFF) of a 200 nanoSecond 27256 and bashi into the upper 16K (\$4000-\$7FFF). It's a good idea to label each original ROM with the socket number it came from, so they can be re-installed correctly, if needed. (If your Eprommer software works only in C64 mode, use the C128 mlm to transfer bashi to \$4000-\$7FFF in Ram 0, then BSAVE "BASHI", B0, P16384 TO P32768. In C64 mode load and burn each file separately).

Figure 3 shows how I installed this Basic Eprom in my flat C128. Diode logic enables the Eprom's /OE when baslo (labeled /ROM2 in the P.R.G. schematic) OR bashi (/ROM3) go low. These signals are obtained at feed-through holes near the sockets. (Note that the bashi feed-through is partially covered by the U34 socket.)

Additionally, baslo pulls the Eprom's A14 low, selecting the lower 16K. The Sram can then be installed in the U34 socket, and a function ROM or ready programmed Eprom (if I ever get one), plugged into U36 with its pin 20 lifted and wired to pin 14 of the HC138. I could instead install a second Sram.

The Eprom could also be burned with baslo in the UPPER 16K, and bashi in the lower, duplicating the arrangement in C128D Roms. In this case, the feed-through connections would be reversed, so that bashi pulls pin 27 low. Incidentally, all C128 system ROMs have large areas containing \$FF; that is, unused. Once a ROM has been copied to Eprom, the ambitious might consider burning their own routines into these unused areas. Note that the C128 Kernel ROM must be removed from the computer to copy it because 4K of Z80 start-up code is "hidden" while the C128 is in 8502 native mode. You'll need to use another C128 or C64.

SOFT SELECT

Since a logic 0 or 1 on input B of the HC138 allows a choice of pins 12 or 14 as the device enable, this can be accomplished via software, instead of a switch. One possibility is to use a Cassette control line. CASS SENSE, which detects when play/record on the Datasette is pressed, is normally an input. It defaults to logic 1 on reset. Connect this to pin 2 of the HC138 and, when low, it will select whatever is connected to pin 14 (Y1). The first time you want to change the state of CASS SENSE you'll have to change bit 4 of the data direction register at location 0: Poke 0, Peek(0) OR 16 (or the ml equivalent) does this without changing the other ddr bits. Thereafter, a 0 on bit 4 of location 1 will select pin 14 of the HC138 as the active output, while a 1 will select pin 12. Use Poke 1, Peek(1) AND 239 for a zero bit 4, and Poke 1, Peek(1) OR 16 for a 1. (Note that some software, after manipulating location 1, may not return it to the state in which it was found. Be wary of programs that use custom characters in 40 columns or tinker with the VIC's Color Memory blocks. All the system routines that alter location 1 do so in a "considerate" manner.)

CASS SENSE is available at pin 26 of the 8502 (U6) or finger 6 of the cassette connector. If CASS SENSE = 1 selects Sram as the default internal Function device, you could write an auto-starting program that looks for a certain keypress on start up to keep the Sram or select function ROM, if any. All of this assumes you will not be using

cassette storage.

Note that ground on input C of the HC138 selects pins 12 or 14 as active outputs. Connecting C instead to another control line (CASS WRT?) would allow pins 10 and 7 to select two more Internal Function devices for a total of four! This, I think, might be taking things a bit too far, especially in the cramped quarters for the flat C128.

PROGRAM NAME: IFR.MOVER.BAS

```
100 rem written by richard curcio
110 rem copyright (c) 1992
120 rem parsec inc po box 111
130 rem salem ma 01970-0111 usa
140 :
150 rem program name "ifr.mover.bas"
160 :
170 rem *** initialize bank 4 ***
180 bank15:poke53274,0:rem irqs off
190 fori=65285to65348:rem ff05-ff44
200 bank15:x=peek(i):bank4:pokei,x:next
210 fori=65530to65535:rem fffa-ffff
220 bank15:x=peek(i):bank4:pokei,x:next
230 bank15:poke53274,241:rem irqs on
240 :
250 rem *** install ifr mover ***
260 rem !!! destroys ram 0 bytes !!!
270 sa=34000:rem relocating
280 ifsa<48923andsa>32768then300
290 print"bad address!":end
300 ck=0:bank12
310 fori=0to226:readd:pokesa+i,d:ck=ck+d
320 next
330 ifck=29213then350
340 print"error in data!":end
350 x=sa+192:gosub420
360 pokesa+38,1:pokesa+39,h
370 x=sa+203:gosub420
380 pokesa+165,1:pokesa+167,h
390 print"ifr mover installed in"
400 print"bank 12,"sa"to"sa+226
410 end
420 h=int(x/256):l=x-h*256:return
430 data 201, 16,144, 15,169, 15,162,125
440 data 160, 40,133, 2,134, 3,132, 4
450 data 76,227, 2,201, 14,176, 4,201
460 data 4,176,233,134,207,170, 32,107
470 data 255,133,206,162, 11,189,192, 19
480 data 157, 16, 1,202, 16,247, 32, 16
490 data 1,132,172,133,173, 32, 16, 1
500 data 201,255,240,200,132,174,133,175
510 data 32,183,238,176,191, 32, 16, 1
520 data 201,255,240,184,201,128,144,180
530 data 132,195,133,196,162,195,160,172
540 data 165,207,240, 65,142,170, 2,140
550 data 185, 2,160, 0,169,255,197,173
560 data 240, 41,197,196,240, 37,162, 23
570 data 32,162, 2,166,206, 32,175, 2
580 data 56,165,172,229,174,165,173,229
590 data 175,240, 18,230,172,208, 4,230
600 data 173,240, 8,230,195,208, 8,230
610 data 196,208, 4, 56, 96, 24, 96,165
620 data 207,208,201,240, 19,140,170, 2
630 data 142,185, 2,162, 23,189,203, 19
640 data 157, 16, 1,202, 16,247,160, 0
650 data 169,255,197,173,240,221,197,196
660 data 240,217, 32, 16, 1, 56,176,185
670 data 32,221, 2, 32, 15,136,162, 6
680 data 76,201, 2,141, 1,255,177,195
690 data 72,166,206, 32,162, 2,162, 23
700 data 32,175, 2,104,145,195,162, 6
710 data 76,201, 2
```

PROGRAM NAME: IFR.SRC

```
1000 sys4000
1010 :
1020 ;written by richard curcio
1030 ;copyright (c) 1992
1040 ;parsec inc po box 111
1050 ;salem ma 01970-0111 usa
1060 :
1070 program name 'ifr.src'
1080 :
1090 :
1100 ;power assembler (buddy128)
1110 :
1120 *= $84d0
1130 :
1140 ;address = 34000 decimal
1150 :
1160 .bank 12
1170 :
1180 ;some assemblers might not allow a
1190 ;rom bank. if so, assemble to ram 0
1200 ;and use mlm to transfer to bank 4
1210 :
1220 .mem
1230 :
1240 ;move to/from internal function ram
1250 ;in bank 4 (normally eeprom or rom)
1260 :
1270 ;.a=bank (0-3,14,15), .x=0-to
1280 :
1290 setup cmp #$10 ;host bank <16
1300 : bcc xcp ;yes
1310 :
1320 ;call basic illegal quantity
1330 ;using jmpfar
1340 :
1350 illqty lda #$0f ;bank15
1360 : ldx #$7d ;addr hi
1370 : ldy #$28 ;addr lo
1380 : sta $02
1390 : stx $03
1400 : sty $04
1410 : jmp $02e3 ;jmpfar
1420 :
1430 xcp cmp #$0e
1440 : bcs ok ;banks 4-13
1450 : cmp #$04 ;not allowed
1460 : bcs illqty
1470 ok stx $cf ;save direction
1480 : tax
1490 : jsr $ff6b ;get config.
1500 : sta $ce ;store it
1510 :
1520 ;copy code to low end of stack
1530 :
1540 : ldx #$0b
1550 cs1 lda stack1,x
1560 : sta $0110,x
1570 : dex
1580 : bpl cs1
1590 : jsr $0110 ;call it
1600 :
1610 : sty $ac ;host start lo
1620 : sta $ad ;host start hi
1630 : jsr $0110 ;get host end
1640 : cmp #$ff
1650 : beq illqty ;page ff no good
1660 : sty $ae
1670 : sta $af
1680 : jsr $eeb7 ;start < end
1690 : bcs illqty
1700 : jsr $0110 ;get ifr start
1710 : cmp #$ff
1720 : beq illqty
1730 : cmp #$80
1740 : bcc illqty ;<$8000 n.g.
1750 mlalt sty $c3
1760 : sta $c4
1770 : ldx #$c3
```


