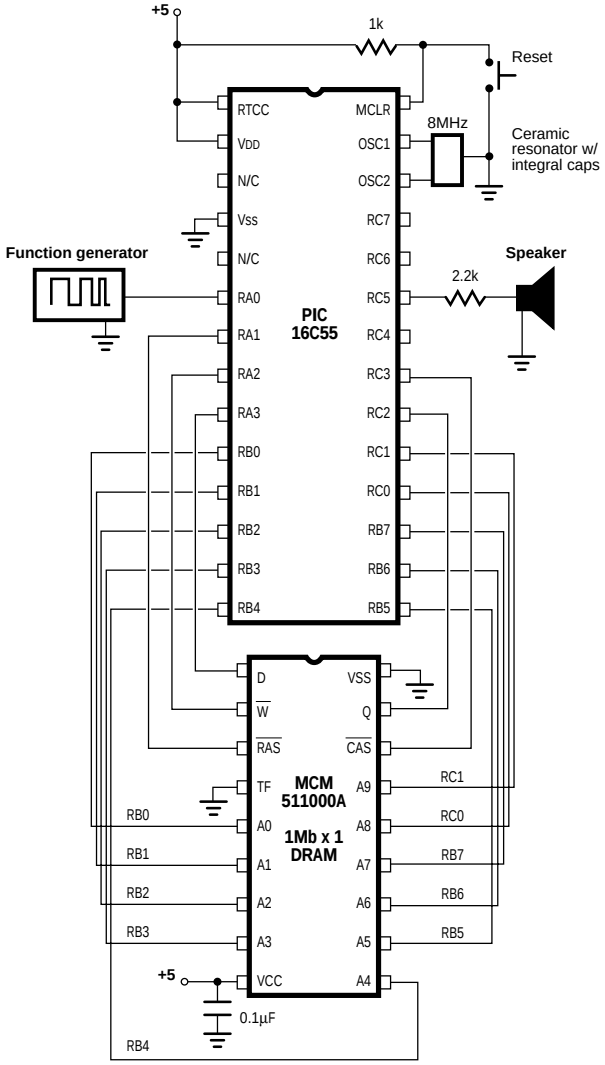
Using Dynamic RAM

Introduction. This application note covers the use of dynamic RAM (DRAM) with PIC16C5x series PICs. It presents a circuit using a Motorola 1-Mb x 1 DRAM IC, and a program in TechTools assembly language showing how to refresh, write, and read the DRAM's 1,048,576 bits of storage.



Background. When a project requires a large amount of memory at the lowest possible cost and in the smallest available package, dynamic RAM (DRAM) is currently the only choice. Unfortunately, DRAM has a bad reputation for being difficult to work with. That reputation is built on the experiences of designers squeezing the greatest possible speed out of the large memory arrays used in general-purpose computers. In PIC applications, where speeds are more leisurely and 1 kByte is still a lot of memory, DRAMs can be downright mild-mannered.

A review of the basic differences between DRAM and static RAM (SRAM) is in order. The basic storage unit in an SRAM is a flip-flop circuit made up of several transistors. Once set or cleared, the flip-flop remains in that state until set or cleared again, or until power is removed. This is a convenient form of temporary storage, since it requires no attention from the processor except when it's being read or written.

DRAM uses even simpler storage cells, consisting of just a capacitor and a transistor. The state of the cell, set or cleared, is stored as a charge on the capacitor. However, the capacitor inevitably leaks and loses its charge over time. When this happens, the stored data is lost.

To prevent the loss of data, each cell in a DRAM must be periodically read and rewritten before its charge fades completely. This ensures that the storage capacitors always have a "fresh" charge, so the process is called *refreshing* the DRAM. The responsibility for refreshing the DRAM is shared between the external processor and circuitry inside the DRAM, as we will see later.

SRAM and DRAM also use different addressing schemes. To access a particular cell of an SRAM, you place the bits of its address on the SRAM's address pins. The number of address pins must be sufficient for each cell to have a unique binary address. A 256-cell SRAM requires 8 address lines, since 8 binary digits are required to express 256 addresses ($2^8 = 256$). By the same token, a 64k SRAM needs 16 address lines ($2^{16} = 65{,}536$) and a 1M SRAM, 20 address lines ($2^{20} = 1M$).

DRAMs use a two-dimensional addressing scheme. Each cell has two addresses, a row and a column. A 256-cell DRAM would require a 4-bit row address and a 4-bit column address ($2^4 \times 2^4 = 256$). A 64k DRAM

would need 8 rows and 8 columns (2^8 x 2^8 = 65,536) and a 1M DRAM, 10 rows and 10 columns (2^{10} x 2^{10} = 1M).

Therefore, a 1-Mb DRAM has a 10-bit wide address bus. To access a memory cell, you put the row address on the bus and activate the DRAM's row-address strobe (RAS), then put the column address onto the bus and activate the column-address strobe (CAS). At the expense of a two-step addressing processing, the number of address lines is cut in half.

The RAS and CAS lines play an important role in the refresh process. DRAM manufacturers know that refresh is a pain in the neck, so they add features to make it easier. For example, just cycling through the DRAM's row addresses will satisfy refresh requirements. The DRAM's internal circuitry takes care of the rest. This is known as RAS-only refresh, because all the user has to do is put a sequential row address on the bus, and blip the RAS line. The MCM 511000A DRAM shown in the figure supports RAS-only refresh. Although it has 10 bits worth of row addresses, the DRAM requires that only the lower 9 bits (512 addresses) be accessed in RAS-only refresh.

An even simpler refresh scheme, the one illustrated in the program listing, is known as CAS-before-RAS refresh. All the processor has to do in this case is activate CAS, activate RAS, and then deactivate both strobes. Each time it does this, it refreshes a row. In a normal access, RAS is activated before CAS, so reversing the order serves as a signal to the DRAM. When the DRAM sees this signal, it uses its own internal row-refresh counter to supply row addresses for the refresh process, ignoring the external address bus.

The MCM 511000A DRAM requires refresh every 8 milliseconds, but it doesn't care whether it is performed all at once (burst refresh), or spread out over time (distributed refresh).

Some applications don't require a separate refresh routine at all. Any application that accesses 512 row addresses every 8 milliseconds doesn't need any further refresh. If the application doesn't access all rows in the required time, but does read or write the DRAM 512 or more times every 8 milliseconds, a CAS-before-RAS refresh cycle can be

tacked to the end of each read or write. This is known as hidden refresh.

Not all DRAMs support all refresh schemes, so it's important to review the manufacturer's specifications before finalizing a design. Data on the MCM 511000A for this application note came from the *Motorola Memory Data book*, DL113/D, Rev 6, 1990.

How it works. The sample application shown in the figure and program listing accepts a stream of input bits, such as the output from a square-wave generator, and records them into sequential addresses in the DRAM. At the same time, it echoes these bits to a speaker. When the DRAM is full, the program plays back the recorded bits through the speaker until the circuit is shut off or reset.

To demonstrate the circuit, reset the PIC and twiddle the tone-generator frequency knob. It will take about 52 seconds for the PIC to fill the DRAM's 1,048,576 bits of storage. When it does, you will hear the sequence of tones played back through the speaker. Because the playback routine requires fewer instructions than recording, the pitch of the tones will be slightly higher and the playback time shorter.

The write loop takes up to 120 program cycles to execute; 60 microseconds when the PIC is operating from an 8-MHz clock. The exact number of cycles for a trip through the loop depends on how long the program spends in the subroutine *inc_xy*, which increments 10-bit row and column counters. At 60 microseconds per loop, the PIC samples the input pin approximately 17,000 times per second. If you use the circuit to record frequencies higher than half the sampling rate (approximately 8.5 kHz), you'll hear some interesting squawks. The output tone will become lower as you adjust the input tone higher.

What you're hearing is known as *aliasing noise* or *foldover*. The only way to faithfully reproduce an input frequency is to record two or more samples per cycle. You will hear this referred to as the Nyquist theorem or Nyquist limit. If you are interested in modifying this application to build a recording logic probe or tapeless audio recorder, keep Nyquist in mind.

Modifications. This application will work fine (albeit at faster rates of sampling and playback) if you remove *call refresh* from the *:record* and

:playback loops. The reason is that with *call refresh* removed the program accesses row addresses sequentially every 30 microseconds or so. This means that 512 rows are refreshed every 15 milliseconds. Although this exceeds the maximum refresh time allowed by the DRAM specification, DRAMs are rated conservatively. Further, you are unlikely to hear the data errors that may be occurring because of inadequate refresh!

If you want to organize data stored in the DRAM as bytes (or nibbles, 16-bit words, etc.) instead of bits, consult the DRAM specs on "fast page mode" reads and writes. Using this method of accessing the DRAM, you issue the row address just once, then read or write multiple columns of the same row. This would be faster and simpler than eight calls to the single-bit read and write routines presented here.

Program listing. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

; PROGRAM: Dynamic RAM Basics (DRAM.SRC)

```
; Upon power up or reset, this program starts sampling pin ra.0 and recording its
; state in sequential addresses of a 1Mb dynamic RAM. When the DRAM is full, the
; program plays back the recorded bits in a continuous loop.

; Remember to change device info when programming part.
                device      pic16c55,xt_osc,wdt_off,protect_off
                reset       start

Sin             =           ra.0            ; Signal generator input
RAS             =           ra.1            ; DRAM row-address strobe
WR              =           ra.2            ; DRAM write line (0 = write)
Dout            =           ra.3            ; DRAM data line
adrb_lo         =           rb              ; Low bits of address bus
adrb_hi         =           rc              ; High bits of address bus
Din             =           rc.2            ; DRAM Q line
CAS             =           rc.3            ; DRAM column-address strobe
Sout            =           rc.5            ; Signal out to speaker

; Put variable storage above special-purpose registers.
                org         8
```

```
r_ctr       ds      1               ; Refresh counter
row_lo      ds      1               ; Eight LSBs of row address
row_hi      ds      1               ; Two MSBs of row address
col_lo      ds      1               ; Eight LSBs of column address
col_hi      ds      1               ; Two MSBs of column address
flags       ds      1               ; Holder for bit variable flag
flag        =       flags.0         ; Overflow flag for 20-bit address

; Set starting point in program ROM to zero
            org     0

start       setb    RAS             ; Disable RAS and CAS before
            setb    CAS             ; setting ports to output.
            mov     !ra,#1          ; Make ra.0 (Sin) an input.
            mov     !rb,#0          ; Make rb (low addresses) output.
            mov     !rc,#00000100b  ; Make rc.2 (Din) an input.
            clr     flags           ; Clear the variables.
            clr     row_lo
            clr     row_hi
            clr     col_lo
            clr     col_hi
            call    refresh         ; Initialize DRAM.

:record     call    refresh         ; Refresh the DRAM.
            call    write           ; Write Sin bit to DRAM.
            call    inc_xy          ; Increment row and col
                                    ; addresses.
            jnb     flag,:record    ; Repeat until address overflows.

:play       call    refresh         ; Refresh the DRAM.
            call    read            ; Retrieve bit and write to Sout)
            call    inc_xy          ; Increment row and col
                                    ; addresses.
            goto    :play           ; Loop until reset.

write       mov     adrb_lo,row_lo  ; Put LSBs of row addr on bus
(rb).
            AND     adrb_hi,#11111100b ; Clear bits adrb_hi.0 and .1.
            OR      adrb_hi,row_hi  ; Put MSBs of row adr on bus (rc).
            clrb    RAS             ; Strobe in the row address.
            movb    Dout,Sin        ; Supply the input bit to the
DRAM,
            movb    Sout,Sin        ; and echo it to the speaker.
            mov     adrb_lo,col_lo  ; Put LSBs of col addr on bus (rb).
            AND     adrb_hi,#11111100b ; Clear bits adrb_hi.0 and .1.
            OR      adrb_hi,col_hi  ; Put MSBs of col addr on bus
(rc).
            clrb    WR              ; Set up to write.
            clrb    CAS             ; Strobe in the column address.
            setb    WR              ; Conclude the transaction by
```

```
                setb    RAS             ; restoring WR, RAS, & CAS high
                setb    CAS             ; (inactive).
                ret

read            mov     adrb_lo,row_lo  ; Put LSBs of row addr on bus
(rb).
                AND     adrb_hi,#11111100b ; Clear bits adrb_hi.0 and .1.
                OR      adrb_hi,row_hi  ; Put MSBs of row addr on bus
(rc)
                clrb    RAS             ; Strobe in the row address.
                mov     adrb_lo,col_lo  ; Put LSBs of col addr on bus (rb).
                AND     adrb_hi,#11111100b ; Clear bits adrb_hi.0 and .1.
                OR      adrb_hi,col_hi  ; Put MSBs of col addr on bus
(rc).
                clrb    CAS             ; Strobe in the column address.
                movb    Sout,Din        ; Copy the DRAM data to the
                                        ; speaker.
                setb    RAS             ; Conclude transaction by
                                        ; restoring
                setb    CAS             ; RAS and CAS high (inactive).
                ret

; This routine implements a CAS-before-RAS refresh. The DRAM has a counter to
; keep track of row addresses, so the status of the external address bus doesn't
; matter. The DRAM requires 512 rows be refreshed each 8 ms, so this routine
must
; be called once every 125 microseconds. Changing the initial value moved into
r_ctr
; will alter the refresh schedule. To refresh the entire DRAM, move #0 into r_ctr
(256
; loops) and call the routine twice in a row (512).

refresh         mov     r_ctr,#8
:loop           clrb    CAS             ; Activate column strobe.
                clrb    RAS             ; Activate row strobe.
                setb    CAS             ; Deactivate column strobe.
                setb    RAS             ; Deactivate row strobe.
                djnz    r_ctr,:loop     ; Repeat.
                ret

; This routine increments a 20-bit number representing the 1,048,576 addresses of
; the DRAM. The number is broken into two 10-bit numbers, which are each stored
; in a pair of byte variables. Note that wherever the routine relies on the carry bit to
; indicate a byte overflow, it uses the syntax "add variable,#1" not "inc variable." Inc
; does not set or clear the carry flag.

inc_xy          add     row_lo,#1       ; Add 1 to eight LSBs of row addr.
                sc                      ; If carry, add 1 to MSB, else
```

```
                                                    ; return.
                        ret
                        inc     row_hi              ; Increment MSBs of row address.
                        sb      row_hi.2            ; If we've overflowed 10 bits, clear
                        ret                         ; row_ hi and add 1 to column
                        clr     row_hi              ; address, else return.
                        add     col_lo,#1
                        sc                          ; If carry, add 1 to MSBs, else
                                                    ; return.
                        ret
                        inc     col_hi              ; Increment MSBs of col address.
                        sb      col_hi.2            ; If we've overflowed 10 bits, clear
                        ret                         ; col_ hi and set the flag to signal
                        clr     col_hi              ; main program that we've
reached
                        setb    flag                ; the end of memory, then return.
                        ret
```