Interfacing an External SRAM to the C8051F000

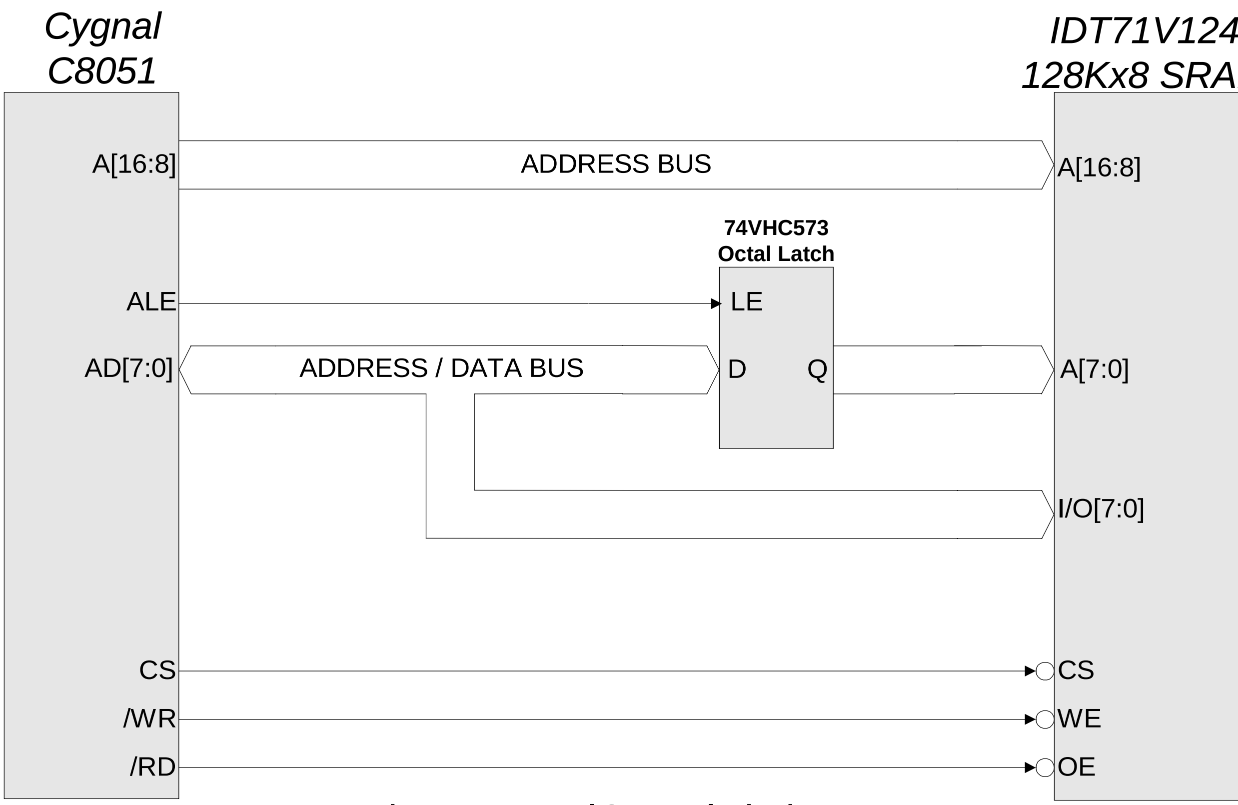

Figure 1. External SRAM Block Diagram.

Introduction

The purpose of this application note is to describe how to interface a generic SRAM or a memory mapped peripheral to a C8051 device using standard GPIO port pins. Hardware connections, schematics, timing diagrams, example code, and a performance review are provided.

The applications of this interface include acquiring ADC samples, data logging, or any other large data storage application.

Key Points

- This reference design assumes a 10 ns SRAM. If the SRAM access time is greater than 45 ns, it may be necessary to add NOP commands to increase the length of the address setup times and read/write strobes.
- The number of port pins required depends on the address space supported. This design's 128 Kbyte address space requires 21 port pins.
- If designing with an SRAM, double check product availability from your supplier. Manufacturers are phasing out many low-density SRAM devices.

Description

This example of an external SRAM interface uses an IDT71V124SA10PH (128K x 8-bit) 3V SRAM from Integrated Device Technologies (www.idt.com), although any generic SRAM will work in a similar fashion. The interface uses a multiplexed address and data bus to reduce the number of port pins required. The lower address bits are held in a latch while data is transferred. Figure 4 (page 5) shows the tested configuration of this implementation.

Bi-Directional Port Operation

'Data1' is used as a data input bus, output bus, and partial address bus. Multiplexing the bus requires dynamic port configuration changes to make the port an input or an output as needed.

To configure a port pin as an input, its associated Port Configuration Register bit (PRTnCF.x) must be set to a '0', which makes it's output mode 'open-drain', and it's register latch bit (Pn.x) must be set to a '1', which makes it's output state 'hi-z'. For example, the following code configures all the pins of Port 0 as inputs:

```
mov PRT0CF, #00h  ;Open-drain output mode
mov P0, #0ffh     ;high-impedance
```

This code configures all of Port 0's pins as push-pull outputs:

```
mov PRT0CF, #0ffh  ;Push-Pull output mode
```

The SRAM_Read routine (see the Example Code section, page 6) gives an example of changing the port direction. During the first phase of the routine, the 'DATA1' port is configured as an output to drive the least-significant address byte onto the port latch. In the second phase of the routine, the 'DATA1' port is configured as an input to read the value from the external SRAM.

Signals and Connections

Figure 1 shows a block diagram of the hardware connections between the C8051F000, SRAM, and address latch. The entire schematic is shown in Figure 4. The connections, designations, and signal names are as follows:

The multiplexed address/data bus 'AD[7..0]', designated 'DATA1' in the example code support the lower 8 bits of the address and the 8 bits of data. This configuration allows the lower address lines to be held by the '573 latch while the SRAM and C8051 transfer data, such that 8 additional ports for data transfer are not necessary.

'A[15..8]', designated 'ADDR' in the example code, supply the upper 8 bits of the address.

'A16', also designated 'A16' in the example code, acts as a bank select between the two 64 Kbyte banks. A '0' is bank one and '1' is bank two.

'RD', 'WR', 'ALE', and 'CS' are control signals and have the same corresponding names in the example code. 'RD' is the read strobe (operates active low). 'WR' is the write strobe (operates active low). 'ALE' is the address latch signal that holds the lower 8 address bits during data transfer. 'CS' is the SRAM chip select (operates active low).

Software Operation

'SRAM_Init', 'SRAM_Read', and 'SRAM_Write' are the three software routines used to access the external SRAM.

The 'SRAM_Init' routine initializes the SRAM interface logic and port configurations. This routine is only called in the initialization sequence of the device. This routine assumes that the crossbar has already been enabled (XBR2.6 = '1'). For example:

```
mov XBR@, #40h    ;enable the crossbar
acall SRAM_Init   ;initialize the SRAM
```

The 'SRAM_Read' routine reads a byte from the external SRAM. To use this routine, load DPTR with the sixteen-bit address to be read, call 'SRAM_Read', and the routine returns in ACC the data at the address pointed to by DPTR. For example:

```
mov DPH, #00h     ;load high byte of address
mov DPL, #00h     ;load low byte of address
acall SRAM_Read   ;perform read operation
                  ;data is returned in ACC
```

The 'SRAM_Write' routine writes the byte in ACC to the external SRAM at the address pointed to by DPTR. To use this routine, load ACC with the data to be written, load DPTR with the 16-bit address, and call 'SRAM_Write'. For example:

```
mov DPH, #00h     ;load high byte of address
mov DPL, #00h     ;load low byte of address
mov a, #55h       ;load value to write
acall SRAM_Read   ;perform read operation
```

The main program in the example code section outlines how to write to and read from every byte in the external 128 Kbyte SRAM. The program writes a byte to external RAM, reads that address location, and verifies the value read is the same as the written value. The program then proceeds to the next address space and continues until the entire 64K bank has been written to. Once the lower bank has been written the program switches to the upper bank by setting the 'A16' bit (see the "Constants and Declarations" section in Example Code). The routine then performs the same read, write, and verify operation for every byte in the upper bank.

Timing Description

Figures 2 and 3 show timing waveforms for reads and writes respectively, as implemented by the example code. Table 1 shows the timing values for these figures.

Read Timing Notes

't_{RDSU}' (Table 1) refers to the time period from when the read strobe is activated to when the data is valid. The corresponding code lines for this sequence are:

```
clr    RD         ; activate read strobe
;NOP              ; add NOPs here to extend tRDSU
mov a, DATA ; read the data
```

It may be necessary to add NOP instructions after the 'clr RD' instruction as shown above to extend 't_{RDSU}' in order to meet the setup time of the SRAM.

Write Timing Notes

As shown in Table 1, 't_{WR}' refers to '/WR' pulse width. The following code sequence executes the pulse.

```
clr    WR      ; activate WRITE strobe
;NOP           ; add NOPs here to extend tWRSU
setb WR        ; de-assert WRITE strobe
```

It may be necessary to add NOP instructions after the 'clr WR' instruction as shown above to extend 't_{WR}' in order to meet the setup time of the SRAM.

Performance

This multiplexed parallel interface implementation achieves high throughput performance with moderate Port I/O consumption. A byte-read operation or byte-write operation, each takes 34 SYSCLK cycles from procedure entry point to return-from-call inclusive, which takes 1.7 µs with a 20 MHz SYSCLK. This achieves a maximum transfer rate of 588K bytes per second. A 64K bank can be filled in 137 µs.

Symbol	Parameter	Cycles	SYSCLK=20MHz
READ CYCLE			
t_{ALE}	Latch Pulse Width	2	100ns
t_{RDSU}	Data Setup Time	2	100ns
WRITE CYCLE			
t_{ALE}	Latch Pulse Width	2	100ns
t_{WASU}	Address Setup Time	3	300ns
t_{WDSU}	Data Setup Time	4	200ns
t_{WR}	Write Pulse Width	2	100ns

Table1. Read and write cycle timing

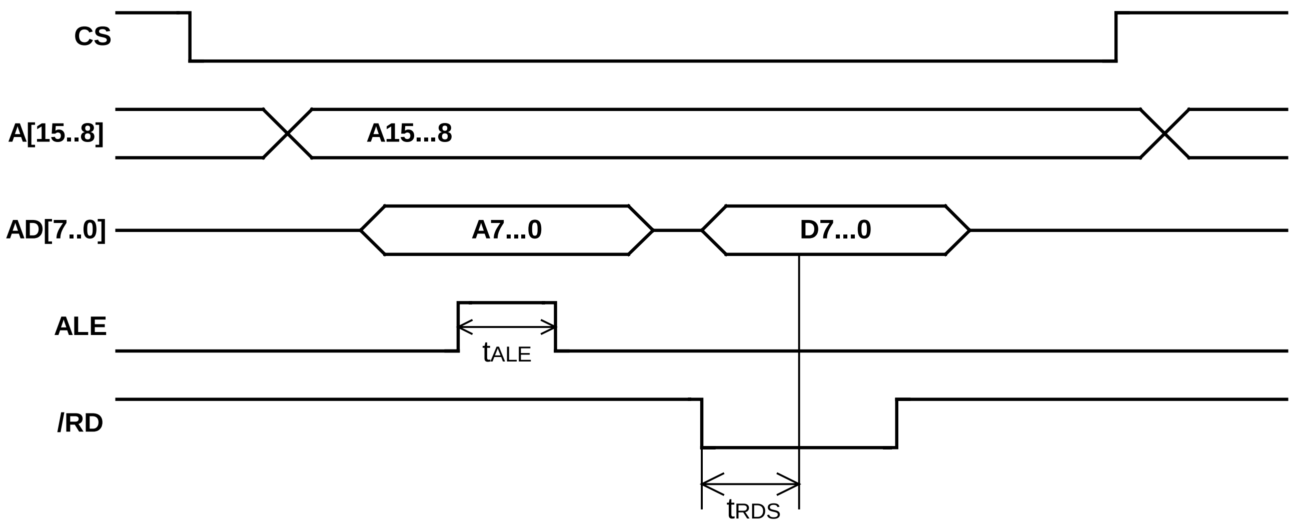

Figure 2. Read Cycle Timing Waveform.

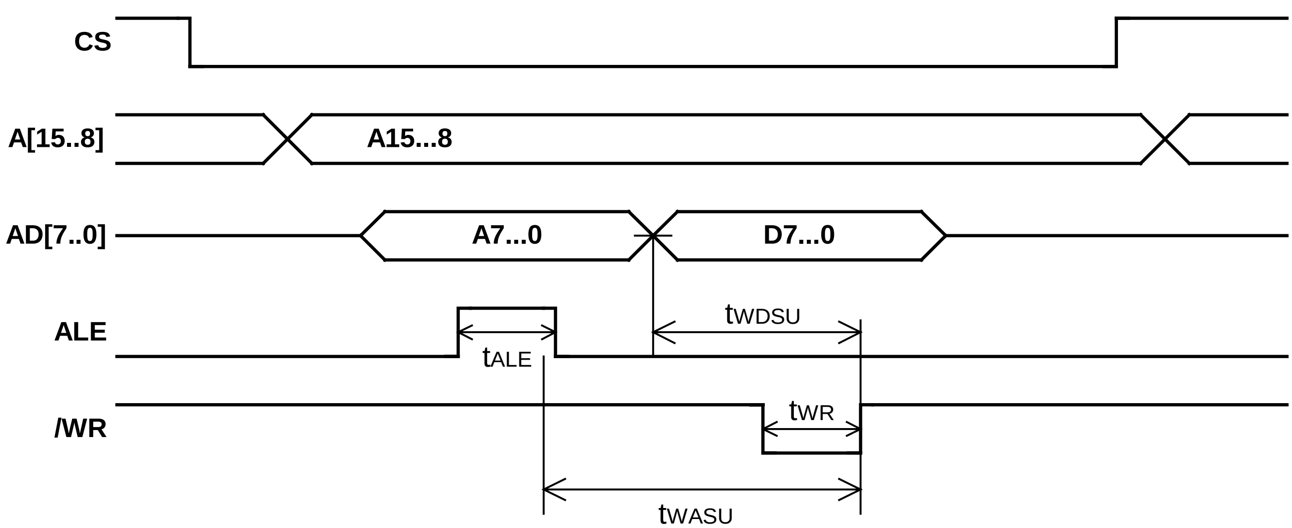

Figure 3. Write Cycle Timing Waveform.

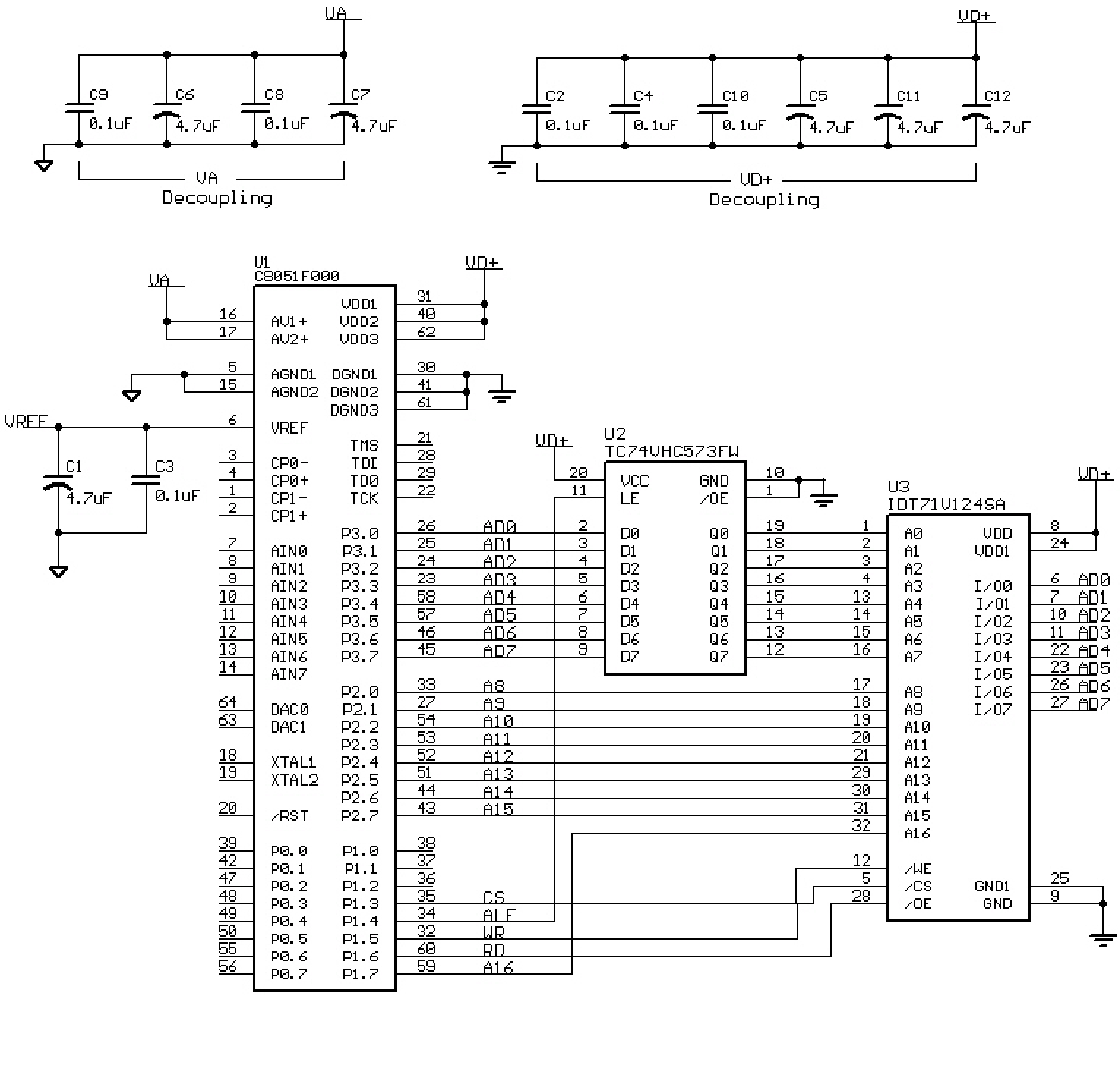

Figure4. Tested configuration of C8051F000, 128k x 8 SRAM, and address latch.

Example Code

```
;-----------------------------------------------------------------------------
;      Copyright (C) 2000 CYGNAL INTEGRATED PRODUCTS, INC.
;      All rights reserved.
;
;
;      FILE NAME   : XSRAM.ASM
;      TARGET MCU  : C8051F000
;      DESCRIPTION : External SRAM read/write verification routine for
;                    IDT 71V124SA.
;
;-----------------------------------------------------------------------------
; EQUATES
;-----------------------------------------------------------------------------

$NOLIST
$MOD8F000
$LIST

;---------------------------------------
; Constants and Declarations
;---------------------------------------

DATA1  EQU  P3                 ; port for DATA pins(AD7..0)
DATACF EQU  PRT3CF             ; port configuration register for DATA
ADDR   EQU  P2                 ; port for ADDR pins(A15..8)
ADDRCF EQU  PRT2CF             ; port configuration register for ADDR
A16    EQU  P1.7               ; upper address bit(address bank select)
RD     EQU  P1.6               ; READ strobe (active low)
WR     EQU  P1.5               ; WRITE strobe (active low)
ALE    EQU  P1.4               ; address latch signal(active low)
CS     EQU  P1.3               ; SRAM chip select(active low)

;-----------------------------------------------------------------------------
; VARIABLES
;-----------------------------------------------------------------------------

;-----------------------------------------------------------------------------
; RESET and INTERRUPT VECTORS
;-----------------------------------------------------------------------------
; Reset Vector

      org   00h
      ljmp  Main

;-----------------------------------------------------------------------------
; MAIN PROGRAM CODE
;-----------------------------------------------------------------------------

      org   0B3h
Main:
      ; Disable the WDT. (IRQs not enabled at this point.)
      ; if interrupts were enabled, we would need to explicitly disable
      ; them so that the following two instructions were guaranteed to
      ; to execute within 4 clock cycles of each other.
```

```
        mov   WDTCN, #0DEh
        mov   WDTCN, #0ADh

        ; Set up the XBar.
        mov   XBR2, #40h        ; Weak pull-ups, XBAR enabled.
        acall SRAM_Init         ; Initialize SRAM


        mov   R0, #0ffh         ; R0 holds value to write
        mov   DPH, #00h         ; initialize 16bit address to start of bank (0000h)
        mov   DPL, #00h         ;
        mov   a, R0             ; load write value

        ; bank1 will write a value to ram, read it, and then verify the value
        ; bank1 will perform this operation for every byte in the 64Kbyte bank
bank1:
        acall SRAM_Write        ; write to sram
        clr   a                 ; clear load value
        acall SRAM_Read         ; read same address
        cjne  a, 00h, error     ; verify read value is the same as the loaded
                                ; value 00h=R0
        inc   dptr              ; next address
        mov   a, DPH            ; check dptr for finished (dptr=0000h)
        orl   a, DPL            ;
        jz    b1done            ; we are finished with the first 64k bank if
                                ; dptr rolls over
        mov   a, R0             ; reload write value
        jmp   bank1             ; write; read; and verify again

        ; once bank1 is done we will perform the same operation on bank2
b1done:
        setb  A16               ;  change to bank 2
        mov   R0, #0ffh
        mov   DPL, #00h         ; initialize 16bit address
        mov   DPH, #00h         ;
        mov   a, R0             ; load write value

        ; bank2 will write a value to ram, read it, then verify the value
bank2:
        acall SRAM_Write        ;write to sram
        clr   a                 ; clear load value
        acall SRAM_Read         ; read same address
        cjne  a, 00h, error     ; verify read value is the same as the loaded
                                ; value 00h=R0
        inc   dptr              ; next address
        mov   a, DPH            ; check dptr for finished (dptr=0000h)
        orl   a, DPL            ;
        jz    b2done            ; we are finished with the first 64k bank if
                                ; dptr rolls over
        mov   a, R0             ; reload write value
        jmp   loop1             ; write; read; and verify again

b2done:
        jmp   $                 ;
error:
        jmp   $                 ; a verification error has occurred
```

```
;-------------------------------------
; SRAM_Init
;-------------------------------------
; This routine initializes the SRAM interface logic.  Must be called once
; before any SRAM_Read or SRAM_Write operations, typically as part of the
; reset sequence.  This routine assumes that the crossbar has already been
; enabled (XBR2.6 = '1').
;

SRAM_Init:
      mov   DATACF, #00h      ; Enable Port3 (DATA) as an input bus
      mov   DATA1, #0ffh
      mov   ADDRCF, #0ffh     ; Enable Port2 (ADDR) as an output
      mov   ADDR, #0ffh       ;  driven high ($ff)
      orl   PRT1CF, #11111000b ; enable P1.7..3 as outputs
      clr   A16               ; select bank 0
      setb  RD                ; READ initialized
      setb  WR                ; WRITE initialized
      clr   ALE               ; address latch disabled
      setb  CS                ; SRAM de-selected
ret


;-------------------------------------
; SRAM_Read
;-------------------------------------
; This routine reads from the external SRAM.  Specifically, it returns
; in ACC the data at the address pointed to by DPTR.  Bank select
; (manipulation of A16) is not handled here.
;
SRAM_Read:
      clr   CS                ; select external SRAM
      mov   ADDR, DPH         ; force external address A15..A8
      mov   DATACF, #0ffh     ; enable AD7..0 as outputs
      mov   DATA1, DPL        ; force external address A7..A0
      setb  ALE               ; latch the address
      ;NOP                    ; add NOPs here to extend tALE
      clr   ALE
      mov   DATACF, #00h      ; enable AD7..0 as inputs
      mov   DATA1, #0ffh
      clr   RD                ; activate READ strobe
      ;NOP                    ; add NOPs here to extend tRDSTU
      mov   a, DATA1          ; read the data (note: setup time for OE-based
                              ;  reads is 10ns for this SRAM.  At SYSCLK
                              ;  = 20MHz, this instruction takes 2 clock
                              ;  cycles, or 50ns * 2 = 100ns.
      setb  RD                ; de-assert READ strobe
      setb  CS                ; de-select SRAM
      ret

;Totals for a read are:
;30 bytes, 34 cycles.


;-------------------------------------
; SRAM_Write
;-------------------------------------
```

```
; This routine writes a byte to the external SRAM.  Specifically, it writes
; the byte in ACC to the address pointed to by DPTR.  Bank select
; (manipulation of A16) is not handled here.
;
SRAM_Write:
      clr   CS                  ; select external SRAM
      mov   ADDR, DPH           ; force external address A15..A8
      mov   DATACF, #0ffh       ; enable AD7..0 as outputs
      mov   DATA1, DPL          ; force external address A7..A0
      setb  ALE                 ; latch the address
      ;NOP                      ; add NOPs here to extend tALE
      clr   ALE
      mov   DATA1, a            ; present the data to the DATA bus
      clr   WR                  ; activate WRITE strobe
      ;NOP                      ; add NOPs here to extend tWR
      setb  WR                  ; de-assert WRITE strobe
                                ; note: this results in a write pulse width
                                ;  of 100ns with a 20MHz SYSCLK.  The minimum
                                ;  width for this SRAM is 7ns.
      mov   DATACF, #00h        ; enable AD7..0 as inputs
      mov   DATA1, #0ffh
      setb  CS                  ; de-select SRAM
      ret
;-----------------------------------------------------------------------------
; End of file.

END
```