MPS

Memory Interfacing Lab Exercise

Interfacing Memory Chips on the 8051 Processor Bus

Student's name & ID (1):_____

Partner's name & ID (2): _____

Your Section number & TA's name _____

Notes:

You must work on this assignment with your partner.

Hand in a printer copy of your software listings for the team.
Hand in a neat copy of your circuit schematics for the team.

These will be returned to you so that they may be used for reference.

---------- do not write below this line ----------

	POINTS (1)	POINTS (2)	TA init.
Grade for performance verification, Part 1 & 2 (30% max.)			
Grade for performance verification, Part 3 (20% max.)			
Grade for answers to TA's questions (20% max.)			
Enhancement (5% max.)			
Grade for documentation and appearance (25% max.)			
			TOTAL

Grader's signature: _____

Date: _____

Interfacing Memory Chips to the 8051 Processor Bus

GOAL

By doing this lab assignment, you will learn to interface to the 8051 external memory:

1. The Am9128 2048 x 8-bit Static RAM and access it with software written in C.
2. The MCM6147 4096 x 1-bit or the Am914 1024 x 4-bit Static RAM and access it with software written in C.

PREPARATION

- References: *C8051F12x-13x Reference Manual*, Ch. 17

Hardware design

- Do a top-down design of the bus interface hardware. Select the appropriate IC chips.
- Provide logic expressions for the various control pins, address decoding, etc.
- Use LogicWorks, or another logic simulator, to validate your logic expressions.
- Include a full set of logic schematics in your report.

Software design

- Do a top-down design of your program. Provide a flowchart or equivalent.
- Write C programs for each module/subroutine. Provide planning documentation for each: flowchart and tests. Specify test inputs and expected output or other indictors of correct operation.
- Integrate your modules to get a C program that is free from syntax errors

1. INTRODUCTION TO BUS INTERFACING

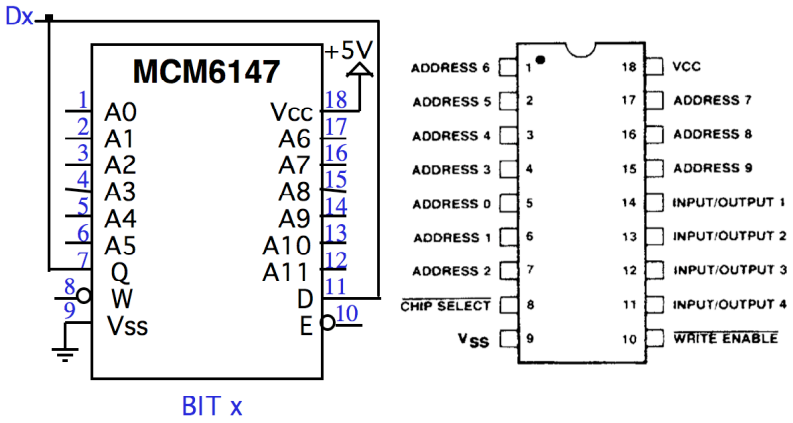
The 8051 features a 16-bit address bus for memory with the Lo-Address Bus (A0-A7) on P6 and the Hi-Address Bus (A8-A15) on P5. P7 acts as the 8-bit data bus. Pins P4.7 and P4.6 act as the write enable (/WR) and read enable (/RD) flags, respectively. These control pins provide the ability to enable a write or read to the external memory.

The /WR and /RD signals together with address decoding logic are used to select the time when data is to be put onto the DATA-bus for a processor memory read operation, e.g. a `MOVX A, val` instruction, or to get data from the DATA-bus for a processor write operation, e.g. a `MOVX val, A` instruction. For a memory read instruction, the memory chip must put the data onto the DATA-bus at the time labeled "put data." That is when both (Output) Enable and R/W* are high *and* the correct memory address is detected. For a memory write instruction, the memory chip must get data from the DATA-bus at the time labeled "get data." That is when both (Write) Enable and R/W* are low *and* the correct address is detected.

2. INTERFACING STATIC RAM

In this lab exercise, two kinds of static RAM chips are used. The first kind is the Am9128, a 2K x 1-byte memory chip. It has 11 address pins (A10-A0), 8 bi-directional input-output pins, (IO8-IO1) and 3 control pins. The control pins are Chip Enable ($\overline{CE}$), Output Enable ($\overline{OE}$), and Write Enable ($\overline{WE}$) and are all active low. Notice that the 8 input-output pins are connected to the 8 DATA-bus pins. The control pins are driven by the "put data" and "get data" conditions and the address decoding of the address bits A15-A11.

The second kind of memory chip is either a 1-bit MCM6147 4K x 1-bit or a 4-bit Am9114 1K x 4-bit device. The 4K RAM has 12 address pins (A11-A0), 1 input pin (D), 1 output pin (Q), and 2 control pins. The control pins are Chip Enable ($\overline{E}$) and Write ($\overline{W}$) and are both active low. Notice both input D and output Q pins are connected together to the same DATA-bus pin. The 1K RAM has 10 address pins (A9-A0), 4 data pins (INPUT/OUTPUT n), and 2 control pins. The control pins are $\overline{\text{CHIP SELECT}}$ and $\overline{\text{WRITE ENABLE}}$ and are both active low. The control pins are driven by the "put data" and "get data" conditions and the address decoding for the 4 or 6 high order address bits A15-A12 or A15-A10. Due to limited supplies, the 1-bit RAM may not be available, in which case the 4-bit RAM should be used



Pin-outs for the Am9128, Pin-outs for the MCM6147. Pin-outs for the Am91L14.

This lab uses the 8051 as the processor to which you will be adding external memory. The 8051 features 16-bit addresses, which means that 64 Kbytes of memory can be accessed over an 8-bit wide data bus. The /WR and /RD are used with the address lines to select the chip and the desired operation.

3. LAB EXERCISE

This lab exercise has two parts. The first part is to interface two Am9128 static RAM chips to the 8051 bus. The second part is to interface 2 (depending on availability) MCM6147 static RAM chips or a single Am91L14 static RAM chip to form a 4K x 2-bit or a 1K x 4-bit (1-nibble) memory array.

1. Initially, properly wire a single Am9128 chip and connect it to the 8051 with additional glue logic to start at address 0x2000. Next, write a simple C program that will start at address location 0x1FF0 to accesses 16 bytes (the last 16 bytes of the 8k block that is already there and should always work) of available address space on the 8051 on-board memory, the block of on-chip external memory starting at 0x0000 and ending at 0x1FFF. Then write the same value to every location from address 0x2000 to 0x27FF. An additional write to location 0x2800 will be past the range of the 2k chip and cannot properly be read back. Verify that the memory works as desired by reading from the address locations and ensuring that the value written is the same as the value read. Execute the code while the external RAM is without the +5 V power attached and again after it is powered up. If the memory chip is correctly interfaced, the commands should be able to change values at memory locations in the range when the RAM is powered up and fail if the RAM is powered down. Note that missing or unpowered memory bits usually float high and return a value of 0xFF.

   All 16 address lines MUST be used either directly to the chip or in the chip selection logic so that the chip is enabled ONLY in the correct address range (0x2000-0x27FF). Make sure you turn in all complete logic circuit diagrams.

Address Range	**Description**	**Location**
0x0000 – 0x1FFF	XRAM – 8192 Bytes	Internal on C8051F120 chip
0x2000 – 0x27FF	Added RAM block – 2048 Bytes	External (Off-chip)
0x2800 – 0xFFFF	Remaining available space	External (Off-chip)

"External" Memory space of the C8051F120 processor

   Your program will need to adjust the values in the EMI0CF and EMI0TC SFRs to slow down the processor's access to these older, relatively slow RAM devices. (That is a digit '0' in the SFR names.) Read the description of the registers in Ch. 17, "External Data Memory Interface and On-Chip XRAM" of the Technical Reference Manual. Set the register to insert the maximum number of SYSCLK cycles in the ALE pulse width. Do not worry about optimizing the program for speed at this point. Make sure it works reliably in all cases for the Am9128, MCM6147 or Am91L14 chips.

   Design the hardware top-down. Then validate the operation of the chips bottom-up. You can do this without the computer being connected. Thoroughly test each chip before integrating them. This way you can correct any hardware errors as you are assembling the modules.

   Integrating and debugging working modules is much simpler than trying to debug an ensemble of untried chips. Your grade will depend on how well you follow these guidelines.

2. A second (although this may be tedious) Am9128 chip may be added to the 8051 such that the

processor accesses this chip for addresses 0x2800 to 0x2FFF with modified Chip Enable logic (the first chip is still at addresses 0x2000 to 0x27FF). It is acceptable to modify the Chip Enable glue logic for the existing RAM so that it now responds in the 0x2800 to 0x2FFF range. Write a short C program to check the correct operation of the memory chips. Do this by writing and then reading the pattern 0xAA at all 2048 locations starting at 0x2000 (or 0x2800 for the 2nd range). To be sure that address decoding is being performed correctly, you may want to be certain that other address values outside the block are NOT being affected by writes to values in the block. Next write and read the pattern 0x55 at all 2048 locations. Record any memory location addresses that fail to read the pattern that is written. Use a buffer of size 512 for the errors. This will need to be in XRAM because of the small 128-byte variable space on the 8051 (`unsigned static int __xdata count[512]`). If the buffer becomes full, empty the buffer to the screen and continue. Otherwise, display the buffer contents when the program finishes. Note that using pointers in C is a convenient way to access specific memory addresses.

Repeat this test for a replacement Am9128 memory chip if you suspect a defective chip. If you get the same error pattern, check your wiring for errors. It is highly unlikely that two chips have failed in exactly the same manner. After completing the exercise with the Am9128 chips, have the TA or instructor check you off on parts 1 and 2. It will be easier to proceed with part 3 using a different part of the protoboard. You may want to keep the memory chips from parts 1 and 2 for enhancements. Parts of the address decoding and chip select logic can be used again with modifications.

Helpful Hint:

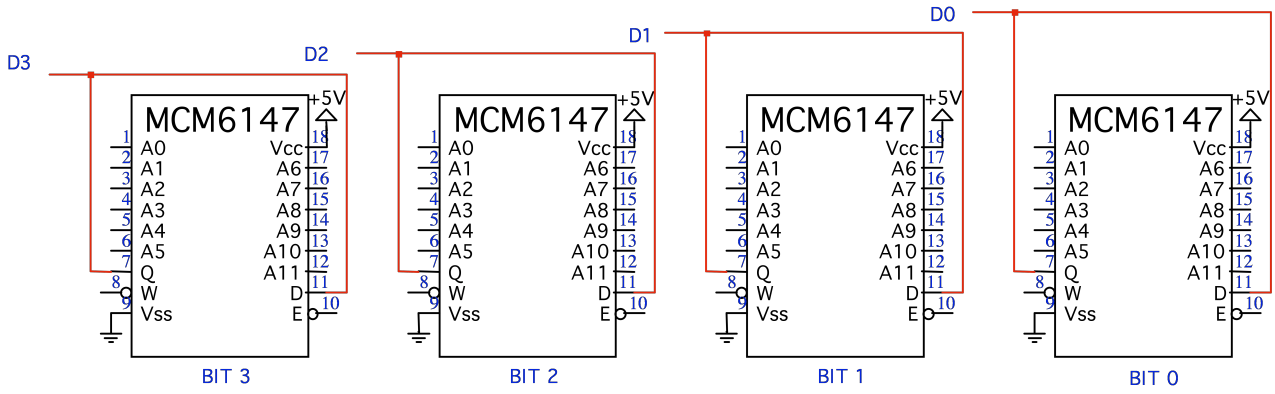
The Am9128 memory chip only uses 11 address lines, while the 8051 has 16. Use all the additional address lines to decode which chip of the two should be selected for a read or write.

3. Start the address of the bit memory array above the address as the Am9128 address space at the next 4K boundary (0x3000). If you are organized and neat, you should easily allow the 1-bit RAM to reside at memory location starting at 0x3000 and coexist with the 8-bit RAM simultaneously. If the 4-bit (nibble) Am91L14 is used, its address should start at 0x4000. There are 3 variations of the 4-bit RAM chips from different manufacturers. They all have identical pin-outs to the Am91L14 chip. The other two part numbers are P114A (Intel) and L2114 (GTE).

   A full implementation of the bit wide 4k MCM6147 requires 8 chips to add 4k of RAM (4k x 1 byte). The same amount of space with the Am91L14 would also require 8 chips. This can be tedious to wire and debug so only 2 bits (2 chips) or 1 nibble (1 chip) are required for this part of the exercise. Verify that the bit (or nibble) memory works correctly by copying a list of 16 2- to 4-bit digits into the memory array beginning at the first address. Display side-by-side the original digits copied to and the ones read from the nibble memory array. If the array is interfaced correctly, the

numbers should match (except for the missing data bits). A partial schematic for configuring a nibble memory array out of the 1-bit chips is shown below. Note, except for the data pins, the address and control pins are all connected the same way. That is, all A0s are connected together, all $\overline{E}$s, all $\overline{W}$s, etc.

Again, design the hardware top-down. Then validate the operation of the chips bottom-up. A single bit can be tested with a simple C program.



Partial schematic for 4-bit memory array.

ADDITIONAL NOTES:

1. Use P5 and P6 for the address bus, and P7 for the data bus.
2. Make sure your program contains a _sdcc_external_startup() function which disables the watchdog time. A sample has been included in the memory.c file.
3. Use the /WR and /RD bus control signals to direct $\overline{WE}$, $\overline{CE}$, $\overline{OE}$, $\overline{W}$, and $\overline{E}$.
4. Read the manual carefully for a complete diagram and description of how external memory works on the 8051.
5. Some implementations may need to slow the 8051 system clock (SYSCLK) way down (3 – 4MHz) to operate reliably, if the RAM chips are slow and the circuit wiring is long. Changes in the BAUD rate generator setup will also be required to keep the RS232 serial interface to the terminal working.
6. The C8051F120 boards used in the class all have extra external memory added to them on the plug-in board on the J24 edge connector. That memory normally occupies XRAM addresses from 0x2000 to 0xFFFF. In order to make some of these address locations available to the memory being added in this exercise, the J1 jumper on the memory board must be moved so that it doesn't short the 2 pins together, if it hasn't been moved already.
7. Do not use any 54XX series logic chips for the glue logic. Make sure you use only 74XX series chips. The older 54XX gates seem to have compatibility problems with the 3.3V CMOS 8051 logic levels and do not switch properly.
8. Print values using %2X or %4X to make it easier to recognize bit patterns and problems.

9. There may be some unresolved differences between the 8-bit and 1-bit chips as far as timing and rapid sequential access to data stored in memory. For now the cause seems to be related to how clean the wiring is, the way the chip select logic is implemented, and they mode used to access memory. Writing arrays to the 1-bit MCM6147 memory may result in corrupted data and loss of bits. Single byte read and write seems to avoid the problem. Bypass capacitors across the chip power and ground should always be used.

10. The SDCC compiler may be doing some unexpected optimizations on memory access. Arithmetic operations on pointers that use temporary calculated results may not work correctly [ex. *(p + q) may not work while *r will, even if r = p + q]. Following the example in the following item 11. should give correct results. When in doubt, follow the example **exactly**.

11. There are several different ways of accessing the memory through C statements. The example listing below on p. 8 demonstrates the most basic approach. Other ways are through arrays or pointers.

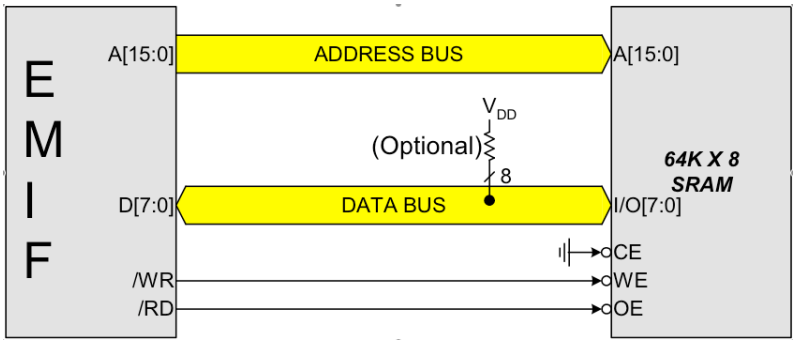
    Pointers to external memory:

```
__xdata unsigned char *ext_ram;
ext_ram = (__xdata unsigned char *)(0x2000);
*ext_ram = 'a';
```

    Arrays in external memory:

```
xdata unsigned char *ext_ram;
ext_ram = (__xdata unsigned char *)(0x2000);

ext_ram[0] = 'a';
ext_ram[1] = 'b';
...
ext_ram[i] = 'z';
```



Memory Interfacing Example. (Only works for 64k-byte block.)

Possible lab enhancements:

Add parity checking to memory read & write with 7 bits of data and use the MSbit as the parity check bit (in software). Disconnect one data line of memory to verify operation.

Add glue logic so that the 2k memory also responds to addresses beginning at 0x9000, 0xA000, and 0xA800 simultaneously. The same data will appear at multiple addresses.

Wire up a D flip-flop to an LED (with current limiting resistor) and add glue logic so that the LED is controlled by writing to bit 0 at address 0x9000

Determine the maximum clock frequency with minimum wait states for reliable data access to the 2k x 8-bit memory.

Disable the built-in internal (extended) 8k memory block and modify the glue logic so that the 2k memory starts at 0x0000 (active only from 0x0000 – 0x0800). Then disable all the 4k x 1-bit memory chips and change the glue logic so that the 2k memory is active for the **bottom half** of every 4k block in the 64k range but not top half (0xX000 - 0xX7FF but not 0xX800 - 0xXFFF where 'X' is any hex digit 0 through F).

Investigate protecting memory by disabling writes (but not reads) with logic and a Port 0 control bit.

Other possibilities…

```
//-------------------------------------------------------------------------------------
// memory.c
//-------------------------------------------------------------------------------------
// This software writes a character to a specific address in external memory
//     NOTES:
// (1) /WR    = P4.7 (
// (2) /RD    = P4.6
// (3) D0-D7  = P7.0-P7.7 (DATA bus)
// (4) A0-A7  = P6.0-P6.7 (ADR bus lo byte)
// (5) A8-A15 = P5.0-P5.7 (ADR bus hi byte)
//-------------------------------------------------------------------------------------
// Includes
//-------------------------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include "putget.h"

//-------------------------------------------------------------------------------------
// Global Constants
//-------------------------------------------------------------------------------------
#define EXTCLK      22118400     // External oscillator frequency in Hz
#define SYSCLK      22118400     // Output of crystal oscillator
#define BAUDRATE 28800           // UART baud rate in bps

//-------------------------------------------------------------------------------------
// Function Prototypes
//-------------------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);
void PORT_INIT(void);
void UART0_INIT(void);
unsigned char _sdcc_external_startup(void);

//-------------------------------------------------------------------------------------
// _sdcc_external_startup
//-------------------------------------------------------------------------------------
//
// Disable watchdog timer before normal initialization - needed for memory
//
unsigned char _sdcc_external_startup(void)
{
   WDTCN = 0xDE;                 // Disable the watchdog timer
   WDTCN = 0xAD;

   return 0;                     // init everything else normally
}
//-------------------------------------------------------------------------------------
// MAIN Routine
//-------------------------------------------------------------------------------------
void main(void)
{
   volatile xdata at 0x2002 unsigned char p;

   SYSCLK_INIT();                // Initialize the oscillator
   PORT_INIT();                  // Initialize the Crossbar and GPIO
   UART0_INIT();                 // Initialize UART0

   SFRPAGE = UART0_PAGE;         // Direct output to UART0

   printf("\033[2J");            // Erase ANSI terminal & move cursor to home position
   printf("Memory test\n\n\r");
   p = 'a';
   while(1)
   {
      printf("Enter a character to write to memory address 0x2002:", p);
      p=getchar();
      printf("\r\nCharacter stored in memory: %c\r\n", p);
   }
}
```

```
//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// Initialize the system clock to use a 22.1184MHz crystal as its clock source
//
void SYSCLK_INIT(void)
{
   int i;
   char SFRPAGE_SAVE;

   SFRPAGE_SAVE = SFRPAGE;          // Save Current SFR page   SFRPAGE = CONFIG_PAGE;
    SFRPAGE   = CONFIG_PAGE;

   OSCXCN = 0x67;                   // Start ext osc with 22.1184MHz crystal
   for(i=0; i < 3000; i++);         // Wait for the oscillator to start up
   while(!(OSCXCN & 0x80));
   CLKSEL = 0x01;                   // Switch to the external crystal oscillator
   OSCICN = 0x00;                   // Disable the internal oscillator

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_INIT(void)
{
   char SFRPAGE_SAVE = SFRPAGE;     // Save Current SFR page
   SFRPAGE = CONFIG_PAGE;

   XBR0    = 0x04;                  // Enable UART0
   XBR1    = 0x00;
   XBR2    = 0x40;                  // Enable Crossbar and weak pull-up

   P0MDOUT |= 0x01;                 // Set TX0 pin to push-pull
   P4MDOUT = 0xFF; // Output configuration for P4 all pushpull
   P5MDOUT = 0xFF; // Output configuration for P5 pushpull EM addr
   P6MDOUT = 0xFF; // Output configuration for P6 pushpull EM addr
   P7MDOUT = 0xFF; // Output configuration for P7 pushpull EM data

   P5 = 0xFF;
   P6 = 0xFF;
   P7 = 0xFF;

   // EMI_Init, split mode with no banking

   SFRPAGE = EMI0_PAGE;
   EMI0CF = 0x3b;  //34
   EMI0TC = 0xFF;

   SFRPAGE = SFRPAGE_SAVE;          // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1
//
void UART0_INIT(void)
{
   char SFRPAGE_SAVE;

   SFRPAGE_SAVE = SFRPAGE;          // Save Current SFR page
```

```
    SFRPAGE = TIMER01_PAGE;

    TCON    = 0x40;
    TMOD   &= 0x0F;
    TMOD   |= 0x20;               // Timer1, Mode 2, 8-bit reload
    CKCON  |= 0x10;               // Timer1 uses SYSCLK as time base
//  TH1     = 256 - SYSCLK/(BAUDRATE*32)  Set Timer1 reload baudrate value T1 Hi Byte
    TH1     = 0xE8;               // 0xE8 = 232
    TR1     = 1;                  // Start Timer1

    SFRPAGE = UART0_PAGE;
    SCON0   = 0x50;               // Mode 1, 8-bit UART, enable RX
    SSTA0   = 0x00;               // SMOD0 = 0, in this mode
                                  // TH1 = 256 - SYSCLK/(baud rate * 32)

    TI0 = 1;                      // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;       // Restore SFR page
}
```