**Microprocessor Systems Lab 5**

Memory Interfacing

Shane Conaboy

Justin Yost

ECSE-4790 Microprocessor Systems

November 16, 2015

**Introduction**

Lab 5 explored the interfacing of external memory with the 8051 microcontroller. Two different types of external RAM chips, the AM9128 and MCM6147, were wired to the 8051, and data was written to and read from each chip to verify the wiring and that the chip was functioning properly. Since the 8051 supports 16 bits of addressing of external memory and the RAM chips had less than 16 address bits, the extra address lines were used to implement glue logic to control which memory chips were active in specific address ranges. By the end of the lab, four external RAM chips were used and the glue logic allowed them to be used with the external addressing scheme of the 8051.

**Procedure**

The first part of the lab was to add a single AM9128 RAM chip as external memory and verify that it could be written to and read from. The AM9128, shown in Figure1 below, is a 2k long, 8 bit wide (2k addressed, 8 bits per address) memory chip, having 11 address lines and 8 data lines. The address range desired for this chip was 0x2000 – 0x27FF. Since the 8051 has 16 address lines, 5 address lines were used to create glue logic that was connected to the chip enable pin of the external RAM so it could only be accessed in the desired memory range. The remaining address lines, data lines, and the write and read enables were wired directly to the EVB, with P6 being the Lo-Address Bus, P5 the Hi-Address Bus, and P7 the data bus. P4.6 and P4.7 were the read and write enable pins respectively. After the chip was fully wired, data was written to and then read from each address to verify that both the wiring worked and that the chip was functioning properly.



Figure : AM9128 RAM Chip

The second part of the lab was to add a second AM9128 RAM chip in conjunction with the first one and have it be addressable in the 0x2800 – 0x2FFF range. The glue logic had to be modified for this chip, however the rest of the wires from the first AM9128 were simply jumped over to the second chip. The final glue logic for both these chips is shown in Figure 2 below. Once the two chips were connected, more testing was done by reading and writing to and from each RAM chip and verifying that the data was correct. If any errors occurred, the address was recorded and output to the screen to aid in debugging.

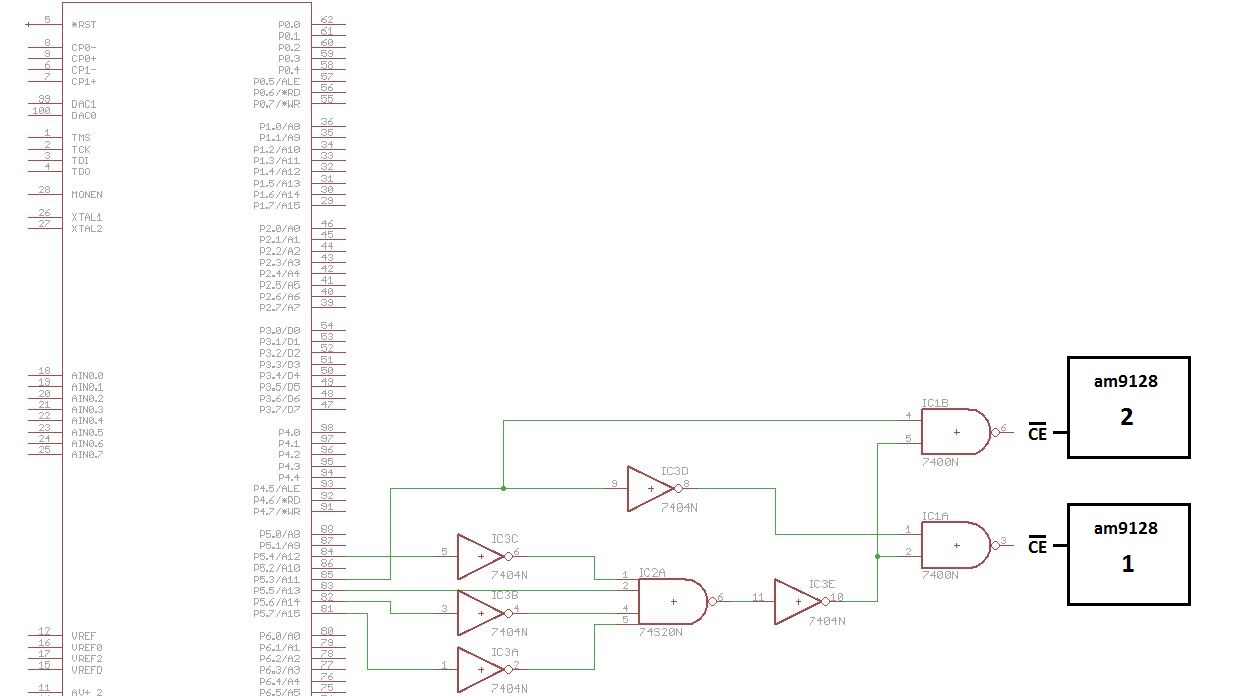
****

Figure 2: Glue Logic to the AM9128 Chips

Part 3 of the lab was to implement 2 MCM6147 RAM chips. The MCM6147, as seen in Figure 3 below, is a 4k long, 1 bit wide memory with 12 address lines. The desired address range for these chips was 0x3000 – 0x3FFF, so more glue logic had to be added using the 4 upper address lines. Both chips had the same chip enable logic, shown in Figure 4, to create a 4k by 2-bit memory structure. Like the previous part, the remaining wires were jumped over from the previous chips, with the 12th address line coming from the EVB. Once these 1-bit wide RAMs were connected, 16 2-bit numbers were written to and then read from the memory. The original numbers and the numbers read back from memory were displayed side by side on the terminal to compare and verify that the circuit and chips were working correctly.



Figure : MCM6147 RAM Chip

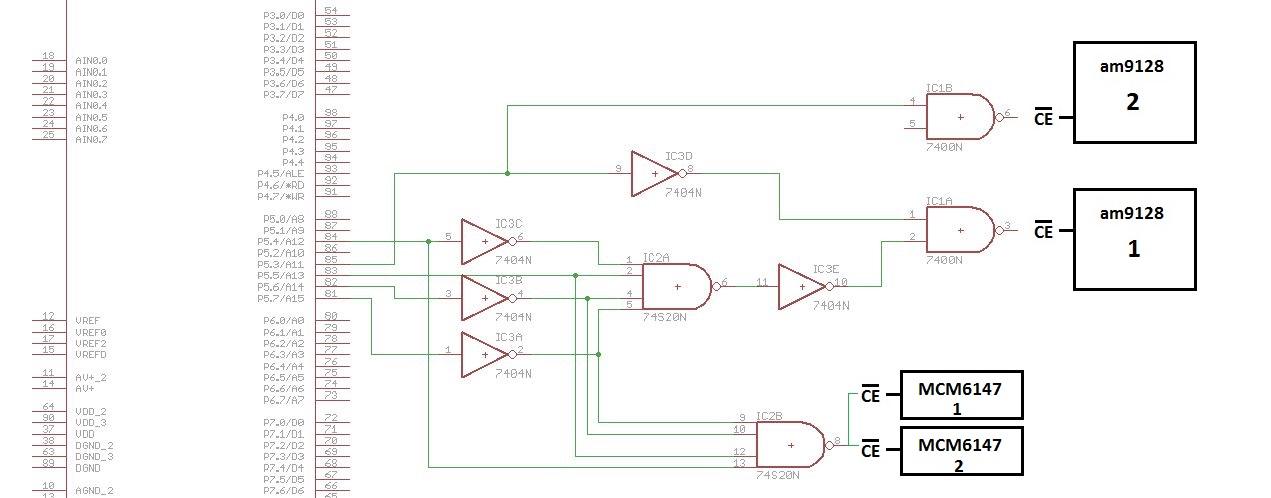


Figure 4: Complete Glue Logic Circuit

For an enhancement to this lab, a simple parity check was implemented for the data stored in memory. Even parity was used, with the most significant bit (MSB) being used as the parity bit. The number of 1’s in the remaining 7 bits of data were counted by starting with 0, XORing it with the first data bit, taking the result and XORing it with the next bit, and so on. The final result, 0 or 1, would then be added as the parity bit before writing the data to memory. When reading the data back, the parity was computed in the same fashion on all 8 bits of data, with even parity (the result being a 0) indicating valid data and odd parity (the result being a 1) indicating that there was an error in the data. To test the functionality, data was written to and read from the memory both with all the data lines connected and then one data line removed. With all the lines connected, there should be no errors, and with the single line removed, there should be an error in the data when that line is supposed to be lo but is default hi when disconnected.

**Results**

To generate glue logic for each address range, the bits that were used within that address were examined. For the first AM9128 chip, the range 0x2000 – 0x27FF corresponds to 0010 0000 0000 0000 – 0010 0111 1111 1111. In this range, the 5 MSBs are always 00100, so logic was made to generate an active-lo signal only when the bits were in these states. The same process was used for the second AM9128 chip and the MCM6147 chips.

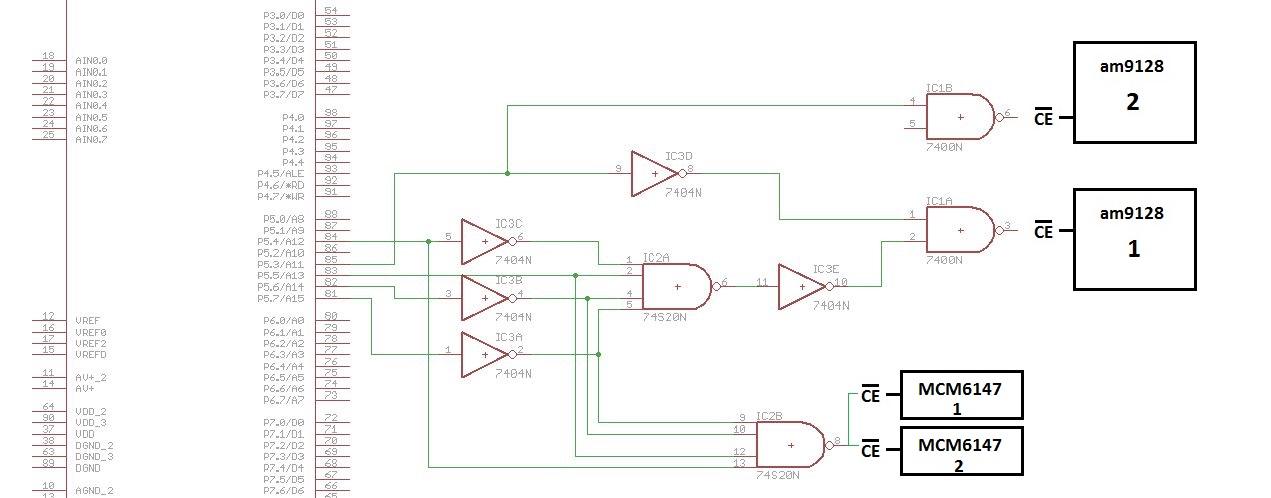
All parts of the lab were successful in their implementation, with little time needed for debugging and the most time spent on wiring. To verify the AM9128 chips, the patterns 0xAA and 0x55 were written to and read from the chips. 0xAA is an alternating 1-0 pattern while 0x55 is an alternating 0-1 pattern, both of which help test for possible errors resulting from connected bit locations. For both chips, no errors were found when running normally. As a sanity check, the power was shut off before running the code again, at which point all the addresses had errors, demonstrating that our program and chips were functioning correctly. To verify the 2-bit wide memory structure created by the MCM6147 chips, 16 numbers were written to and read from memory and then compared. The numbers matched as expected, except for the missing data bits.

Verifying the parity check was similar to how the MCM6147 chips were verified – 16 numbers were written to and read from memory, with the original number (with the parity bit added) and the received number displayed side by side, as well as an output displaying whether the data was valid or invalid. The numbers were then inspected to confirm that the parity bit was a 0 for numbers with an even count of 1’s, and a 1 for numbers with an odd count of 1’s. Running the program with a data line disconnected caused certain numbers to fail the parity check, and it was visually confirmed that the program was running correctly.

**Conclusion**

This lab investigated the interfacing of external memory to the 8051 microcontroller, as well as techniques to verify the functionality of memory chips. Emphasis was put on the hardware portion of the lab, with careful attention and organization needed when connected the RAM chips due to the large amount of wires used. Glue logic was also an important aspect since multiple chips needed to be interfaced side by side in the addressing space. Overall, this lab gave a good overview of different memory structures, how to connect them, and how to interface with them.

**Appendix A: Circuit Schematic**



**Appendix B: Code**

//------------------------------------------------------------------------------------

// memory.c

//------------------------------------------------------------------------------------

// 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

//------------------------------------------------------------------------------------

//This is special function called by the system BEFORE main() is executed

// 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)

{

unsigned int i;

volatile \_\_xdata unsigned char \*ext\_ram;

volatile \_\_xdata unsigned char \*ext\_ram2;

unsigned static int \_\_xdata failure[512];

unsigned int count;

ext\_ram = (\_\_xdata unsigned char \*)(0x1FF0);

ext\_ram2 = (\_\_xdata unsigned char \*)(0x2800);

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");

count = 0;

for(i=0; i<2064; i++)

{

ext\_ram[i] = 0xAA;

}

for(i=0; i<2064; i++)

{

printf("%2X", ext\_ram[i]);

if(ext\_ram[i] != 0xAA)

{

failure[count] = i+8176;

count++;

if(count == 512)

{

for(count=0; count<512; count++)

{

if(failure[count] != 0)

printf("%4X\r\n", failure[count]);

}

count = 0;

}

}

}

for(i=0; i<2048; i++)

{

ext\_ram2[i] = 0xAA;

}

for(i=0; i<2048; i++)

{

printf("%2X", ext\_ram2[i]);

if(ext\_ram2[i] != 0xAA)

{

failure[count] = i+10240;

count++;

if(count == 512)

{

for(count=0; count<512; count++)

{

if(failure[count] != 0)

printf("%4X\r\n", failure[count]);

}

count = 0;

}

}

}

for(i=0; i<2048; i++)

{

ext\_ram2[i] = 0x55;

}

for(i=0; i<2048; i++)

{

printf("%2X", ext\_ram2[i]);

if(ext\_ram2[i] != 0x55)

{

failure[count] = i+10240;

count++;

if(count == 512)

{

for(count=0; count<512; count++)

{

if(failure[count] != 0)

printf("%4X\r\n", failure[count]);

}

count = 0;

}

}

}

for(count=0; count<512; count++)

{

if(failure[count] != 0)

printf("%4X\r\n", failure[count]);

}

printf("Memory Test Completed");

while(1);

}

//------------------------------------------------------------------------------------

// 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

}