

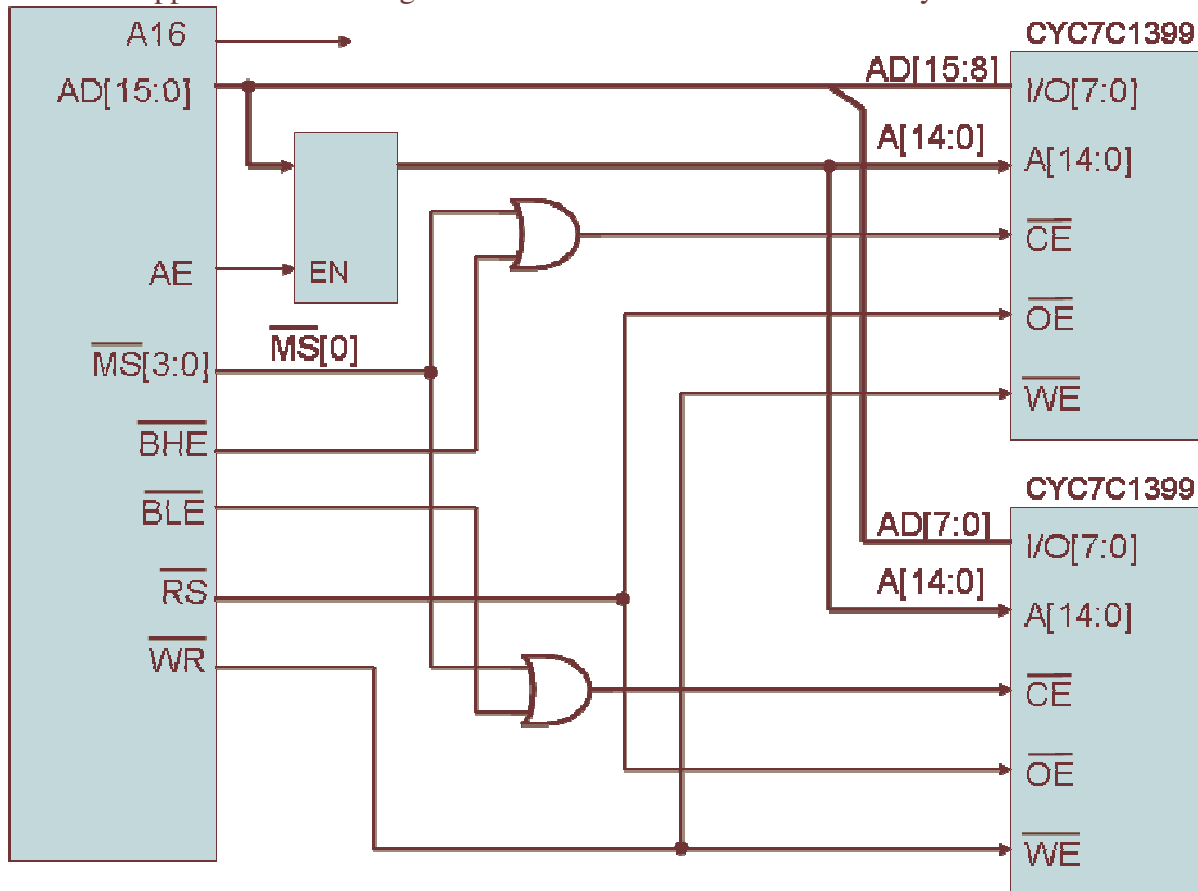
HOMEWORK ASSIGNMENT #5

Due Friday April 10th, 2009

**1. (20 points) Extending Memory Depth and/or Width**

Find the datasheet for a Cypress Semiconductor CY7C1399BN. Using 2 of these devices, you will design a 64KB memory bank for use with the ADuC7026's external memory interface. The memory bank is to be designed with a 16-bit width, and to support byte read/write capability.

Show all required connections to the SRAM devices, and how the system will connect to the ADuC7026 external memory interface. Use NANDs, NORs, Inverters as needed. Assume that a demultiplexed bus is already available in the system, as shown in the figure below. The memory should be mapped to the MS0 region. Points will be deducted for a messy schematic.



AD[14:0] from the ADuC should be connected to A[14:0] of the SRAM. Remember that when run in 16-bit mode the LSB of the address is dropped and the address bits are shifted up one, so AD[14:0] actually carry address information for [15:1]. The BHE, and BLE can be simply combined with the MS[0] pin to form the CE.

**2. (25 points) ADuC7026 Timing Analysis**

Now, analyze the memory system that you designed in problems #1. Assume that the propagation delays are as follows: latch (from any input) - 10ns, gates (NANDs, NORs, Inverters) (from any input) - 5ns. The ADuC7026 is assumed to be running with a CPU core clock of (1275 x 32.768kHz) at 3.3V. You are to determine if the CY7C1399BN device is compatible with the ADuC7026 external memory interface. Evaluate the read & write timing to determine if the SRAM is compatible with the ADuC7026 system. To do this, verify that all the timing requirements for both the ADuC7026 (is data driven in time and held long enough, and are there any contention issues) and the SRAM are met (there are 3 timing variants available use the 20ns access time data set). Determine what if any wait states are required, and tell the values you would program into the XMxPAR register.

For the ADuC7026, assume that  $t_{\text{DATA\_SETUP}} = 10\text{ns}$  and  $t_{\text{DATA\_HOLD}} = 0\text{ns}$ . Use the timing relationships given on the in-class handout, not those on the ADuC7026 datasheet timing diagram.

First consider  $t_{\text{AA}}$  vs  $t_{\text{AVDV}}$ .

$$t_{\text{AVDV}} = 2.5\text{CLK} - t_{\text{DATA\_SETUP}} - t_{\text{LATCH}} = 71.8\text{ns} - 10\text{ns} - 10\text{ns} = 51.8\text{ns}$$

$$t_{\text{AA}} = 20\text{ns}; \quad \text{So } t_{\text{AA}} < t_{\text{AVDV}}$$

Now let's consider time from the CE signal. Longest path for generating CE timing to the SRAM is from BHE/BLE through the OR gate. Call that  $t_{\text{DECODE}}$

$$t_{\text{CE}} = 1\text{CLK} - t_{\text{DATA\_SETUP}} - t_{\text{DECODE}} = 23.9\text{ns} - 10\text{ns} - 5\text{ns} = 8.9\text{ns}$$

$t_{\text{ACE}} = 20\text{ns}$  (SRAM's access from chip enable low to Data valid)... So we don't make it. Will have to add an extra state using XMxPAR[10] or XMxPAR[9].

$$t_{\text{OE}} = 1\text{CLK} - t_{\text{DATA\_SETUP}} = 23.9\text{ns} - 10\text{ns} = 13.9\text{ns}$$

$t_{\text{DOE}} = 7\text{ns}$  (From SRAM spec) so  $t_{\text{DOE}} < 13.9\text{ns}$  so we are fine here with no extra delays.

Now looking at tDF (Data Float). Timing for when Data from SRAM goes high impedance.

Need data to be held longer than hold time of ADuC. Since the ADuC has a hold time of zero than any number greater or equal to zero for  $t_{HZOE(\min)}$  will work. Actually a minimum number is not given for this spec in the data sheet, but it is safe to assume the number is greater than zero (the SRAM cannot predict that OE was going to rise and float the bus before it was instructed to).

On the other side of this we want to make sure the bus was put to high impedance before the next memory access occurs and a new address is driven out. If the SRAM was still driving data when the ADuC tried to drive the next address then we would have a conflict. So we want to look if  $t_{HZOE(\max)}$  is less than the amount of time till the next address would be driven.

Time from OE (RS) to next address =  $1\text{CLK} = 23.9\text{ns}$   
 $t_{HZOE} = 6\text{ns}$

So we are fine here.

Now lets look at a couple of write parameters...

First look at CE to WR deasserted.

CE is gated by the BHE, BLE signals which are the last to arrive. There is only 1 clock period with involved, and the BHE, BLE need to propagate through the decode logic.

$t_{CW} < 1\text{CLK} - t_{LATCH} = 23.9\text{ns} - 5\text{ns} = 18.9\text{ns}$

$t_{SCE} = 12\text{ns}$  (this parameter is termed  $t_{SCE}$  in the Cypress data sheet, and is 12ns) so we are fine

Next lets look at the minimum WR active pulse width ( $t_{WR}$ ). The no wait state case for this is 1CLK period or 23.9ns.

$t_{PWE} = 12\text{ns}$  (Cypress calls this spec  $t_{PWE}$ ) and the minimum value is 12ns, so we are fine.

Now for address setup prior to deassert of write.

$t_{A\_SETUP} = 2.5\text{CLK} - t_{LATCH} = 49.8\text{ns}$   
 $t_{AW} = 12\text{ns}$  (From SRAM spec)  
So we are fine here  $12\text{ns} < 49.8\text{ns}$

Now look at data setup to write end.

Data is driven on the bus 1 clock cycle prior to the deassert of WR.

$t_{D\_SETUP} = 1\text{CLK} - \text{some time for bus to transition (lets say } 5\text{ns)} = 18.9\text{ns}$   
 $t_{SD} = 10\text{ns}$  (From SRAM spec)  
So we are fine here  $10\text{ns} < 18.9\text{ns}$

Next one would look at data and address hold times...but looking at the specs for the SRAM we can see both the data and address hold times are zero so we know we are going to be fine.

$t_{HD} = 0ns$  &  $t_{HA} = 0ns$

Now what would you write to XM1PAR?

Since we only needed to setup for longer chip enable to data valid timing have to setup for either a XMPAR[10] state (longer address hold time), or a longer XMPAR[9] state (data turn around time). I will choose XMPAR[10]. Keep in mind XM1PAR[10:8] have inverted meanings (for some strange reason) so..

```
LDR R0, =(XM_BASE)
MOV R1, #0x0300 ; All at minimum time expect address hold to extend CE to DV
STR R1, [R0,#XM1PAR]
```

### 3. (10 points) Memory Types. (fill in this table) (handwritten is fine)

Memory Type:	Description:	Describe application it is suited for
ROM	ROM is a Read Only Memory. It is programmed via mask data (either diffusion or via(contact) programmed. Its contents cannot be easily changed. It is high density (1 transistor/cell) and can be made on any CMOS logic process.	Best suited for applications where constant data is needed. Can be used for program (firmware) storage in high volume running $\mu$ Controller based product where the firmware is known bug free. Can be used to store complex relationships for look up tables (like dew point given RH and temperature) (or an inverse trig function).
SRAM	Can read and write with equal ease. Does not wear out. Can be made on any standard CMOS logic process. Rather large cell (6-transistor/cell) and uses differential read/write mechanism. Typically single cycle reads are possible if the SRAM is on chip.	Best used for local (on chip) data storage (working space for alorithms). Most $\mu$ Controllers will have an on chip SRAM for this purpose (our ADuC has 8kB).
Flash	A non-volatile memory that can be read from fast, but writing involves erasing an entire block and then programming byte by byte, and is slow. Requires a double layer poly process to manufacture. High voltages required for programming/erasing are usually derived on chip by charge pumps	Very useful for code storage in mController applications. Non-volatile so program always present, but code can easily be erased and re-programmed when a bug is discovered. Has also found great use as low power, high shock resistance memory in applications like MP3, Memory sticks, digital camera's, ...
	Dynamic Random Access Memory	Good for large areas of data working space

DRAM	can be read and written with ease, but is volatile. It stores the bits as charge on a capacitor, which leaks over time, so it must be refreshed periodically. It is manufactured on a specialized process with trenches forming the capacitance.	in micro-processor systems. Typically off chip from the processor since it is made on a specialized process. Access is slower since it is offchip. Reads are typically done in bursts (CAS after RAS).
Cache	Memory architecture made using SRAM as its building blocks. Typically made on chip and single cycle access. Processor will request a memory location and the Cache will read and determine if it has that memory location present and valid. If so it provides it in 1 cycle. If not the processor will have to perform a bus cycle to fetch it from external memory (typically DRAM based).	Used in high performance micro-processors. Most processors will separate into code vs data caches, however, there are unified caches.

#### 4. (10 points) Memory Nuances

Explain the differences between the following 4 memories in the non-volatile family (Flash, EEPROM, EPROM, and OTP).

They are all built from the same type of cell with a floating poly gate that traps charges. They are also all programmed with a similar mechanism (hot electron injection of charge on the floating gate using high potential on the control gate and a medium voltage on the drain). The main differences are in how they are erased.

Flash is erased electrically in blocks. A high potential is placed on all the drains in the block, and zero or a negative voltage is placed on the control gates. The negative charge on the floating gate is attracted to the drain and tunnels through the oxide.

EEPROM is erased in a similar fashion to Flash except it can be done on a byte by byte basis. The extra routing of high voltage potentials needed to erase byte at a time is what makes EEPROM memory large, and therefore EEPROM chips are typically lower capacity than Flash chips.

EPROM can only be erased by UV light. Typically in a ceramic package with a quartz crystal window to let the UV light in. Very expensive package.

OTP = One Time Programmable. Essentially EPROM in a plastic package, hence it cannot be erased.

Explain the difference between via programmed and diffusion programmed ROM. What are advantages/disadvantage of each. Via or contact programming takes place at the metal

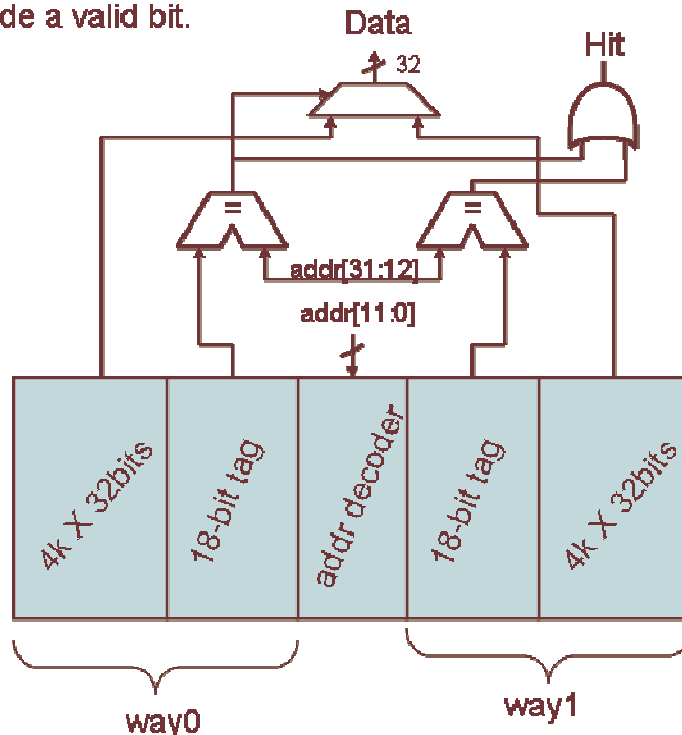
interconnect level. That is much later in the process flow, so if a change needs to be made than a boat of staged wafers can be released in the manufacturing flow with a new contact mask, and you will have new silicon in < 2 weeks.

Diffusion programming takes place way deep in the silicon by having a diffused drain connection or not to a transistor. This programming method results in a smaller cell size than contact programming, but since it is early in the process flow it will take over 6 weeks to get new silicon if you need to make a change.

Imagine a 2-way set associative cache memory for an ARM processor (32-bit addressing). Each way contains 4k entries of 32-bit words. How many bits wide would the tag portion of the array need to be? Show how you derived this & sketch out the basic structure.

Since we are accessing 32-bit words we don't need the lower 2 bits of address.

4k entries requires 12-bits. So tag width is  $32 - 2 - 12 = 18$  bits. It would be 19 if you include a valid bit.



### 5. (10 points) EEPROM Memory System Usage

A. Erasing an EEPROM cells is a destructive process, and so EEPROMs have limited endurance. Suppose that you use the 128kB AT28LV010 EEPROM memory subsystem in a product that needs to maintain some information in non-volatile memory, and that the information is updated once per second. Assume that you need to store 120 bytes of information at a time. Nothing else needs to be stored in the EEPROMs. Describe how you could use the 128kB EEPROM memory system to store this information so as to maximize the length of time the system could be expected to operate correctly. How long would the system be guaranteed to operate correctly? You cannot assume that the product will be left on continuously. **Note:** obviously you are going

to distribute the 120 bytes over the 128kB memory to distribute wear evenly. The challenge here is to think of an addressing mechanism to know which block contains the current valid set of data.

For ease of addressing, we will store data as 128-byte blocks (120 bytes of data with 8 bytes left over). We have 128 KB of memory divided up into 128-byte blocks, so we have  $131072/128 = 1024$  blocks. Each block can be written at least 10,000 times, so we can write a total of  $1024 * 10,000 = 10,240,000$  times. At one update per second, our EEPROM would last  $(10,240,000 \text{ updates})(1 \text{ sec/update})(1 \text{ min}/60 \text{ sec})(1 \text{ hour}/60 \text{ min}) = 2844.44$  hours. Since the internal write buffer of the part is 128 bytes in length, we can write the whole 128-byte block in one write operation. It takes a maximum of 10ms to write one page to the EEPROM. This is much less than the one second interval between writes, so the write process will not affect the time interval. **So, the system can be guaranteed to operate correctly for 2844.44 hours.**

Unlike Flash devices, this EEPROM hides the erase cycle from the user. After we write data to its buffer, it will automatically handle erasing and writing those byte(s) into the EEPROM cells. An obvious scheme to manage the EEPROM data would be to treat the EEPROM as a circular buffer of size 128 KB. The head pointer would then give us a run-time indication of where the most recent data was. The only problem is that we can't just store that information in a fixed EEPROM location, or we would wear out that location first. So, we need a scheme that distributes the information into the data blocks as they are written. One way to do it is as follows: A reasonable assumption is that the system leaves the factory with the EEPROM erased, so all bytes are initially 0xFF. To identify the data blocks, we can use 3 or more bytes to store an increasing serial number in each data block. (Three bytes can represent  $2^{24}$  unique values, which exceeds our requirement of 10,240,000 unique blocks of data.) To make things easier, we would probably want to store serial numbers in complemented form, so a stored value of 0xFFFFF would be interpreted as serial number 0. When we need to write a block of data, we search the EEPROM to find the next available block. The next available block would be easily identified as the first block that has a serial number that is not greater than the preceding block's serial number. (If we got to the end of the EEPROM and didn't find a block meeting the criteria, it means that we need to write the first block in the EEPROM.) Note that this search process would only need to be done on startup, and then during run-time we could just keep track of the next block location in data memory. The value of the serial number would also allow us to estimate how much operating life is remaining.

### 6. (15 points) Building Programming Proficiency

Implement a bubble sort algorithm in a subroutine *bsearch*. The subroutine will be passed the starting address of an array of unsigned words in R0, and the unsigned length of the array in R1. The length of the array may be any value, including 0. The array should be sorted in ascending order. Ensure that *bsearch* is exported from the file. Do not corrupt any of the caller's registers. We will test your subroutine by linking it with a driver program.

**Important:** Submit ONLY your program source code file *teamX\_bsearch.s* using the homework 5 dropbox in Learn@UW. Also, submit a **paper copy** of *teamX\_bsearch.s* with the rest of the assignment.

## Krishna's solution below:

```

bbsort
    PUSH  {R0-R6,R10,LR}
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    CMP  R1,#0
    BEQ  error          ; if the length of the array is zero, it exits the subroutine.
    MOV  R3,#0          ; R3 is loaded with 0 because it starting element that has compared with the rest n-1 elements
    MOV  R5,#1          ; R5 is loaded with 1 which gets incremented by 1 in the inner loop so that each element i is
    MOV  R6,R1
    SUB  R6,R6,#1

bubble_sort
    LDRB R2,[R0,R3]     ; loading R2 with the ith element(given by the offset in R3) to compare it with rest of the i-1
    LDRB R4,[R0,R5]     ; R5 contains the offset for the elements being compared with the ith element
    CMP  R2,R4          ; check whether ith element is smaller than rest of the i-1 elements for each i-1 elements.
    BLT  exit           ; if yes, branch out to exit, i.e. you don't have to change the element position in the array.
    STRE R4,[R0,R3]     ; if no, swap the elements.
    STRE R2,[R0,R5]

exit  ADD  R5,R5,#1      ; increment R5 by 1 till the end of the array.
    CMP  R5,R1          ; check whether R5(count) has reached the end of the loop.
    BNE  bubble_sort    ; if not, repeat the procedure.
    ADD  R3,R3,#1       ; if yes, increment the outer loop( count)  by one.
    ADD  R5,R3,#1       ; inner loop( count) should always be one more than the outer loop count when entering the loop
    CMP  R3,R6
    BNE  bubble_sort

error
    POP  {R0-R6,R10,PC}

wait
    B    wait

    ALIGN

array DCB 19,48,7,66,53,40,13,24,12,05

```