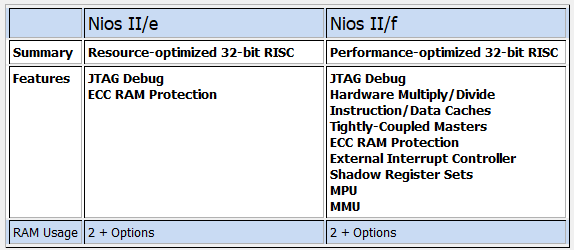
1. What are the differences between the Nios II/e and Nios II/f CPUs?
   1. e stands for Economy, and f stands for Fast. Economy is the free version that lacks a few features.



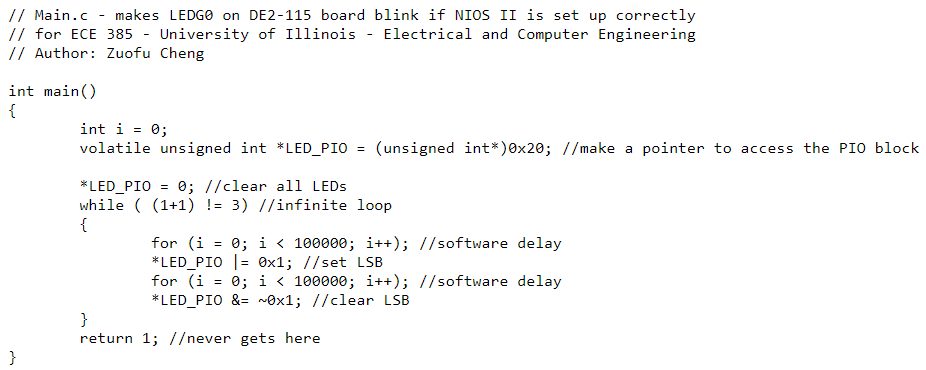
1. What advantage might on-chip memory have for program execution?
   1. It has the highest throughput and lowest latency. Having the latency of only one clock cycle, it makes program execution faster because it does not take that long to get instructions.
2. Note the bus connections coming from the NIOS II; is it a Von Neumann, “pure Harvard”, or “modified Harvard” machine and why?
   1. It is a modified Harvard machine. There are separate sections storing the data and the instructions, which makes it like a Havard machine. However, both the data and the instructions use the same memory bus to the SDRAM and on-chip memory, which transforms this into a modified Harvard machine.
3. Note that while the on-chip memory needs access to both the data and program bus, the led peripheral only needs access to the data bus. Why might this be the case?
   1. The LED peripheral only needs access to the data bus because it is only used to display data. There is no need for it to access the program bus because it is not storing or executing any code, just displaying data when it needs to.
4. Why does SDRAM require constant refreshing?
   1. Because it is dynamic, it needs constant refreshing to maintain its contents. If it did not refresh, then the stored memory could be lost as the charge in the memory cell leaks away. Additionally, reads for each row in dynamic memory can be destructive, so the rows refresh after each read. However, since this is not dependable because not all rows will be refreshed with a read, a constant refresh is needed.

|  |  |  |
| --- | --- | --- |
| SDRAM parameter | Short name | Parameter value (fill in from datasheet) |
| Data Width | [width] | 16 \* 2 = 32 in total |
| # of Rows | [nrows] | 8K (13 address pins) for each bank |
| # of Columns | [ncols] | 1K (10 address pins) for each bank |
| # of Chip Selects | [ncs] | 1 for each chip |
| # of Banks | [nbanks] | 4 for each chip |

1. Note that there are two 32M\*16 chips, so the total amount of memory should be 1Gbit (128 Mbytes), make sure this is consistent with your above numbers; you will need to justify how you came up with 1 Gbit to your TA.
   1. There are 4 banks on one chip according to the datasheet. Each bank has 13 row pins and 10 column pins. This amounts to 8,192 (8K) possible rows and 1,024 (1K) possible columns in a bank. In total, there are 2^23, or 8,388,608 (8M), possible row-column combinations for each bank. Accounting for all 4 banks, this gives you 2^25, or 33,554,432 (32M), possible addresses. Because we are using 2 chips, each with a width of 16 bits, if both chips use the same addressing, each address can correspond to a total for 32 bits, with one chip holding the upper bits and the other chip holding the lower bits. With both chips, you get a grand total of 2^30, or 1,073,741,824 (1G), bits of data within memory. This corresponds to 2^27, or 134,217,728 (128M), bytes of data in memory.

(32 \* (2^13) \* (2^10) \* 4 = 1,073,741,824).

1. What is the maximum theoretical transfer rate to the SDRAM according to the timings given?
   1. Access time is 5.5ns
   2. Units for theoretical transfer rate is in bytes/second
   3. (1 / (5.5 \* 10 ^ -9 seconds)) \* (32 bits) \* ((1 byte) / (8 bits)) = .727272 \* 10 ^ 9 bytes/second = 727.272 \* 10 ^ 6 bytes/second = 727.272 Mbytes/second
2. The SDRAM also cannot be run too slowly (below 50 MHz). Why might this be the case?
   1. If the SDRAM is ran too slowly, it will not refreshing properly. Additionally, our system is running at 50MHz, and since we want our SDRAM to be synced up with our system, the clock for SDRAM should also be running at 50MHz to ensure that our accesses are valid and not misaligned.
3. This puts the clock going out to the SDRAM chip (clk c1) 3ns behind of the controller clock (clk c0). Why do we need to do this? Hint, check Altera Embedded Peripheral IP datasheet under the SDRAM controller.
   1. The address, data, and control signals are only valid in a certain time frame. As such, in order to access these signals from the SDRAM, we must toggle the SDRAM clock appropriately. When this delay is calculated, it turns out that the SDRAM clock needs a 3ns delay behind the controller clock in order for the address, data, and control signals to be valid and stable when we access them.
4. What address does the NIOS II start execution from? Why do we do this step after assigning the addresses?
   1. NIOS start execution starts from the reset vector, which in our code is 0x10000000. We assign the reset and exception vectors after assigning the addresses because these vector addresses will change after assigning the base addresses and we want the most updated addresses when assigning the vectors.
5. You must be able to explain what each line of this (very short) program does to your TA. Specifically, you must be able to explain what the volatile keyword does (line 8), and how the set and clear functions work by working out an example on paper (lines 13 and 16).
   1. *(int i …)* Make a variable for the loops for software delay
   2. *(volatile unsigned int …)* Make a pointer to the LEDs; volatile means that the system will always reload the variable because it can be changed elsewhere
   3. *(\*LED\_PIO = 0 …)* Initialize the LEDs
   4. *(while …)* Loop infinitely so that the code is always run on the board
   5. *(for …)* Software delay
   6. *(\*LED\_PIO |= …)* Set the least significant bit (0x00 | 0x01 = 0x01)
   7. *(for …)* Software delay
   8. *(\*LED\_PIO &= …)* Clear the least significant bit (0x01 & 0xFE = 0x00)
   9. *(return …)* Should never return because of infinite loop



1. Look at the various segment (.bss, .heap, .rodata, .rwdata, .stack, .text), what does each section mean? Give an example of C code which places data into each segment, e.g. the code: const int my\_constant[4] = {1, 2, 3, 4} will place 1, 2, 3, 4 into the .rodata segment
   1. .bss (block started by symbol) is used for uninitialized statically allocated objects (file scope variables and static local variables)
      1. *static int counter;* allocates space in the .bss segment for an integer and sets it to zero
   2. .heap is used for a dynamically allocated memory
      1. *int\* ptr = malloc(sizeof(int));* allocates an integer in the .heap segment
   3. .rodata is used for constant data
      1. *const int bitmask = 0x0001;* will store the value of bitmask in the .rodata segment
   4. .rwdata is used for initialized static variables (global variables and static local variables)
      1. *static int num = 42;* will store the value of num in the .rwdata segment
   5. .stack is used for storing local variables and stack frames for functions
      1. *int i = 0;* at the start of a function will make room for an integer in the .stack segment and initialize it to 0
   6. .text is used for storing code that runs on the machine
      1. *a = b + c;* is stored in the .text segment