Lab 1

Introduction to Digital System Design

Nicholas Warner

22 September 2014

CSCE 312-501

Problem 1

1. Tag 1 is the declaration of the variable “int\_var,” which is later used to find the size of an int data type on the system in which the program is run. Tag 2 is the printf function, which is printing to the console (using the sizeof function) the number of bytes or bits int\_var takes in memory.
2. The sizeof function returns the size in bytes of the data type given to it. It’s a standard ANSI C library function.
3. Ran on linux.cse.tamu.edu
   1. Execution time: -295419701
4. Ran on sun.cse.tamu.edu
   1. Execution time: 1104479197
5. I was able to get the time ran to be negative. By changing the data type of time\_stamp from double to a long int, I was able to return a positive time: 1410308921. This discrepancy was, I believe, the result of a sort of bad casting (bad in this case): this\_instant.tv\_sec returns a long int, not a double. Assigning it to a long int preserves what it returns. No other data type gave me consistent returns while testing.
6. There was definitely a change on linux.cse.tamu.edu: it ran about the same time consistently, and never negative (~1410308921). The sun server ran better as well, consistently returning about ~1410311844. On a final note, where they returned two very different times, the servers are now returning about the same times.
7. timeval uses longs in its struct, and is part of the POSIX standards. It’s platform specific. The interpretation of the longs is up to the platform.

Problem 2

1. See lab1\_prob2.c and lab1\_prob2\_output.txt.
2. There are definitely multiple data types that occupy the same storage size – long, long long and double all occupy 8 bytes of memory. unsigned int and float both occupy 4 bytes of memory. This tells me that though they occupy the same amount of storage, how those bits are interpreted must be different between them, if only slightly (for example, signed versus unsigned), to deal with different situations. The biggest surprise was timeval taking up 16 bytes. I wasn’t really expecting it to be so large, since it only deals with time. However, it makes sense when you’re dealing with something that may need to be very accurate, such as time.

Problem 3

1. Boolean expressions for each requirement (the first three requirements are for BELL, the fourth is for DLA and the fifth for BA).
   1. For the BELL actuator
      1. if ( ( engine\_running == 1 ) && ( driver\_seat\_belt\_fastened == 0 ) )
         1. bell = 1;
      2. else if ( ( engine\_running == 1 ) && ( doors\_closed == 0 ) )
         1. bell = 1;
      3. else if ( engine\_running == 0 )
         1. bell = 0;
      4. Equation: F = abc’ + ab’c’
   2. For the DLA
      1. if ( ( driver\_on\_seat == 0 ) && ( key\_in\_car == 1 ) )
         1. door\_lock\_actuator = 0;
      2. else if ( door\_lock\_lever == 1 )
         1. door\_lock\_actuator = 1;
      3. Equation: abc + ab’c + a’b’c
   3. For the BA
      1. if ( brake\_pedal && car\_moving )
         1. brake\_actuator = 1;
      2. else
         1. brake\_actuator = 0;
      3. Equation: F = ab

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ER | DSBF | DC | DOS | KIC | DCL | BP | CM | BELL | DLA | BA |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | X | X | 0 | 0 | X |
| 0 | 1 | 0 | 0 | 1 | 0 | X | X | 0 | 0 | X |
| 0 | 0 | 1 | 0 | 0 | 1 | X | X | 0 | 1 | X |
| 0 | 0 | 0 | 0 | 0 | 0 | X | X | 0 | 0 | X |

1. See lab1\_prob3.c, lab1\_prob3\_input.c, lab1\_prob3\_output.c
2. See lab1\_prob3.c, lab1\_prob3\_input.c, lab1\_prob3\_output.c

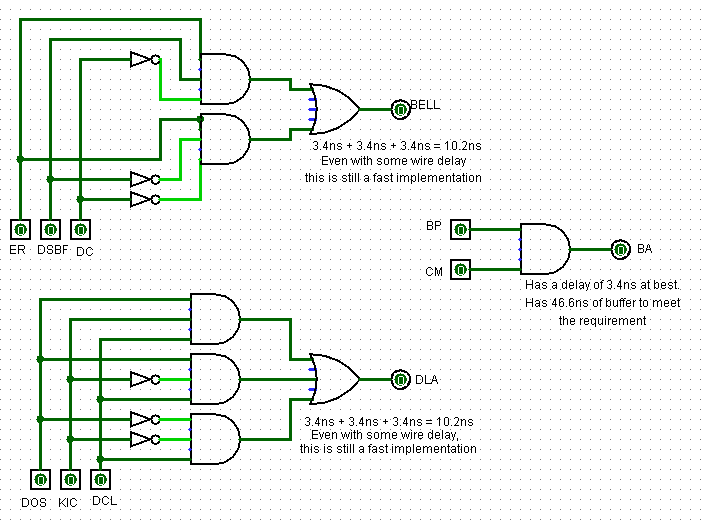
Problem 4

1. See lab1\_prob4.c, lab1\_prob4\_input.c, lab1\_prob4\_output.c
2. I only used three masks in my code to target the specific bits that pertained to the three actuators. I would assign a new mask to this same variable when I reached the next actuator. I did this in an effort to save space. However, were I to have many more actuators and many more sensors, this would quickly become very cumbersome and unreadable. Using enum in the control function to assign unique masks would improve readability and help distinguish the logic behind why and where each mask is used.

Problem 5

1. See lab1\_prob5.c, lab1\_prob5\_output(linux2 run).txt, and lab1\_prob5\_output(sun run).txt
   1. The first test always takes the longest on both machines. Every test after runs a good bit faster, and more consistently. The linux2 machine is slower than the sun machine on the first test, but is significantly faster than the sun machine every time after that.
2. See lab1\_prob5prob3code.c, See lab1\_prob5(prob3code)\_output(linux2 run).c, and See lab1\_prob5(prob3code)\_output(sun run).c
   1. I’m definitely convinced the code I wrote for problem 4 was much faster than the code I wrote for problem 3. Using two variables to keep track of sensors and actuators instead of 11, and only running bitwise comparisons, seem to be much quicker than having tons of variables and logical comparisons over many variables.
3. I’m fairly certain that yes, you could. However, the linux2 server runs 16 3GHz CPUs and even that could barely squeak past 50ns (and then only after the first run). No car would have that kind of computing power. So doubling CPU speed (6 GHz) is not an option for a microprocessor.
4. It would not be cost effective to build an even faster computer to run this code when you could simply make a dedicated hardware logic circuit. It’ll run much quicker, and you’re only dealing with a handful of sensors and actuators.

Problem 6



1. The speed taking into account just the 74XX logic gates (since we do not know wire times) would be at most 10.2ns for the DLA and BELL. The BA is only 3.4ns (again without wires accounted for), leaving 46.6 ns of buffer between the implementation and the system test engineer’s 50ns requirement.
2. Because of these impressive run times, it’s fairly apparent that to meet the safety requirement for the brakes a pure hardware solution would be the best option.