EECE 6017C - Final Exam

Max Thrun

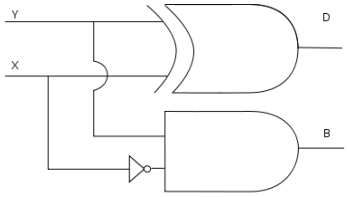
December 12, 2012

Problem 1

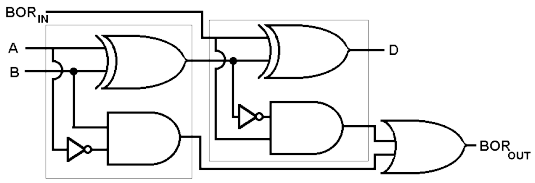
Use Verilog to specify a 2-bit subtractor. Inputs are $A = A_0A_1, B = B_0B_1$ and BorrowIn. Outputs are $Diff = Diff_0Diff_1$ and BorrowOut.

A) Structural, using 1-bit subtractors as components

Using two 1-bit half subtractors, shown in Figure 1(a), we can create a 1-bit full subtractor, shown in Figure 1(b). We can then combine two 1-bit full subtractors to achieve the desired 2-bit full subtractor.



(a) 1-Bit Half Subtractor[1]



(b) 1-Bit Full Subtractor[2]

Figure 1: Gate Level Schematic of Subtractors

We can then translate the subtractor circuits shown in Figure 1 into a Verilog structural model.

Listing 1: Structural 2-Bit Subtractor

```
// 1-bit half subtractor
module half_sub(
    input a,
    input b,
    output diff,
    output bout
);
    wire b_not;

    xor g1(diff, a, b);
    not g2(b_not, b);
    and g3(bout, a, b_not);

endmodule

// 1-bit full subtractor
module sub_one_bit(
    input a,
    input b,
    input bin,
    output diff,
    output bout
);
    wire s1_diff;
    wire s1_bout;
    wire s2_bout;

    half_sub s1(a, b, s1_diff, s1_bout);
    half_sub s2(bin, s1_diff, diff, s2_bout);
    or g1(bout, s1_bout, s2_bout);

endmodule
```

```
33
34  // 2-bit full subtractor
35  module sub_two_bit{
36      input [1:0] a,
37      input [1:0] b,
38      input bin,
39      output [1:0] diff,
40      output bout
41  );
42      wire s1_bout;
43
44      sub_one_bit s1(a[0], b[0], bin, diff[0], s1_bout);
45      sub_one_bit s2(a[1], b[1], s1_bout, diff[1], bout);
46
47  endmodule
```

B) Behavioral

Subtraction can be easily achieved by simply subtracting all three inputs and letting Verilog figure out the two's complement arithmetic. The most significant bit of the result is the borrow out.

Listing 2: Behavioral 2-Bit Subtractor

```
1   // 2-bit full subtractor
2   module sub_two_bit{
3       input [1:0] a,
4       input [1:0] b,
5       input bin,
6       output reg [1:0] diff,
7       output reg bout
8   );
9
10      always @(*) begin
11          {bout, diff} = a - b - bin;
12      end
13
14  endmodule
```

Problem 2

Use a Hamming code to provite 1-bit error correction (set the overall parity to EVEN) for 16-bit data words. Use the minimum number of additional bits.

A) Show how the message 1100 0001 1001 1010 would be encoded

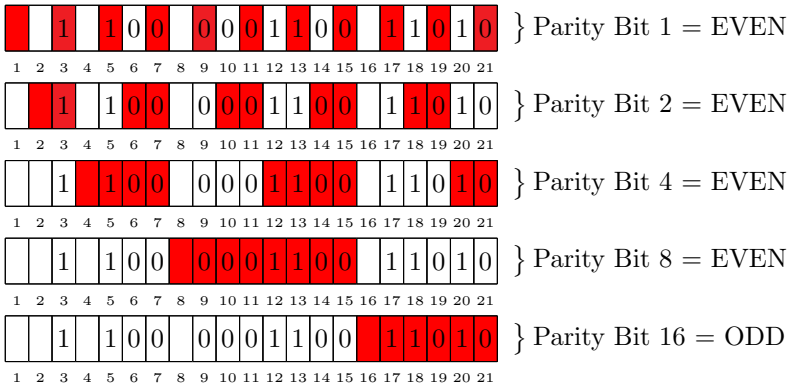
1. Original message

1	1	0	0	0	0	0	0	1	1	0	0	1	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	

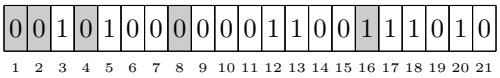
2. Add space for parity bits at every 2^n

		1		1	0	0		0	0	0	1	1	0	0		1	1	0	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

3. Calculate each of the parity bits using the simple check n skip n technique.



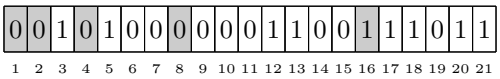
4. After calculating the parity for each of the parity bit positions we can form our final bitstring



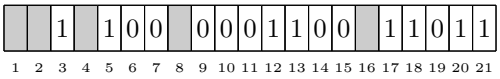
B) Suppose the received message has one error, in the right most bit position. Show how this error will be detected.

The error can be detected by recalculating the parity bits and summing the indexes of those which are wrong. Using our answer for part A we can show how we'd go about detecting an error if the last bit was incorrect

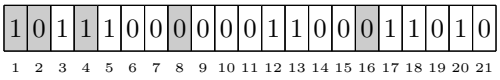
1. Encoded message from part A with an error in right-most bit



2. Remove received parity bits at every 2^n



3. Calculate new parity bits using the method shown in part A



4. Add up the parity bit positions whose calculated value does not match the received value

$$P_1 + P_4 + P16 = Bit_{21}$$

The newly calculated parity shows that Bit 21, the right-most bit, is incorrect.

Problem 3

Give four specific examples of optimization techniques which are likely to be useful in programming an embedded system. In each case, give a criterion that could be used to decide when this optomization technique should be employed.

1. **Loop Unrolling** is one technique that can be used to increase the execution rate of code contained within a loop. By unrolling a loop you remove the looping logic, typically a conditional branch and increment, and simply rewrite the body of the loop multiple times replacing the looping variable with constants. Loop unrolling should be employed when code size is not of concern or the execution rate is deemed more important.

2. **Inlining Functions** is another technique that can be used to increase the execution rate of called functions by removing the call and return logic typically involved with a subroutine and placing a copy of the body of the function at the location where it was originally called from. Like loop unrolling, inlining functions should be employed when code size is not of concern or the cost of manipulating the stack is high.

3. **Enabling Compiler Optomizations** is a simple way to gain numerous code optimizations for free. Most compilers typically have different levels of speed and size optimizations that you can choose from depending on the desired result. For instance, if you are targeting speed and do not care about code size you could enable one of the few speed oriented flags. If code size is important because you are limited in program memory space or you will be loading the executable off some kind of storage media then the size oriented optimization flags would be desirable.

4. **Local and Global Variables** play a bigger role in embedded systems than in a tradition computing system. Using local, small scoped, variables allows the compiler a better chances of using registers to store their value instead of expensive global RAM. Using global variables however can be more efficient than passing a parameter to a function as it does not need to touch the stack. The small scoped variable optimization is more of a good programming practice and can be used regardless of the system requirements. The global variable over parameters can be used in any situation where manipulating the stack is deemed expensive.

Problem 4

A) Are there any problems with the scheme described above? If so, identify what they are.

The first issue occurs when both trains simultaneously change the other signal and then go back and try to change their own. For example, Train 1 changes Signal 2 to blue and at the same time Train 2 changes Signal 1 to red. Then Train 1 goes and trys to change Signal 1 to blue but it is now red so it is not allowed to change it. Similarly Train 2 cannot change Signal 2 to red because it is blue.

B) Will such a scheme prevent collisions? Justify your answer. If not, propose a solution that will.

This scheme *will* prevent collisions as the trains need to acquire the signals before proceeding. The worst case in this system is that both trains deadlock and wait forever.

C) Will such a scheme prevent deadlocks? Justify your answer. If not, propose a solution that will.

Yes, as explained in part A, if both trains simultaneously acquire the their first lights they will not be able to acquire their second lights because the other train has it. In this situation both trains are deadlocked waiting to acquire each others lights. A possible solution to this problem would be to give priority to one train over the other. For example, if it is detected that one light is blue and the other light is red the light system could force the blue light to red which will allow Train 2 to go. After Train 2 leaves the platform it will change the lights to yellow and allow Train 1 to try and reacquire the lights.

Problem 5

As embedded systems continue to be used for more and more critical tasks the issues surrounding embedded system security become extremely important. We can now find embedded systems commonly used in critical sectors such as defense, medical, and power. A compromised system in any of these fields could result in severe consequences. Additionally, with the rise of smart phones practically everyone is now carrying an embedded device in their pocket which, for many, is a window into the rest of their life such as their personal identity and financial information. With many of these devices sharing the same underlying technology it's not hard to imagine a scenario where failed security could lead to a major disaster.

One of the biggest issues that most embedded devices face is that the device is usually out there in the field and an attacker can easily get physical access. Unlike other computer systems where the attacker might only have access through the internet, with most embedded systems an attacker can physically analyze and measure properties of the system that would otherwise be unknown. For instance, an attacker could use a side channel to gain addition information such as the current draw of a device. The attacker can then use this side channel information, correlate it with the input and output of the device, and determine the internal state and process of the system. This technique can be used to recover encryption keys and other sensitive information. One recent example of this technique was shown at Black Hat [3] in which the bootstrap loader password on a MSP430 was determined by watching the power draw of the device. When checking the password the program would take a different branch depending on if the current character it was checking was correct or not. The two branches were a different number of instructions and therefore drew more current for different amounts of time. By watching how much current the device was drawing the attacker was able to determine how many characters of his attempted password were correct and from there could keep changing the next character until the whole password was correct. While this example is simple, it shows how physical access to a system can allow it to be compromised.

The need for security in embedded systems is often met with numerous challenges that are unique to an embedded device such as processing capabilities and the need for low power solutions to allow for low battery life [4]. Traditional computer systems such as servers and even regular desktop PCs have more than enough processing power to allow the use of complex and secure encryption and protocols. With an embedded system having a much less powerful processor other techniques such as using dedicated hardware crypto-accelerators must be used. The use of this dedicated hardware helps with battery life as a hardware implementation can be much more efficient. A problem arises however if a flaw is found in this custom hardware because unless the system is designed using an FPGA there is essentially no way to update and fix the issue. This fact results in the need for much more up front testing when it comes to embedded systems.

Overall, the biggest issues with security in embedded systems is the fact that an attack can easily gain physical access and that the processors typically found in embedded systems are not powerful enough to handle hugely complex cryptography. The ladder of these two issues seems to be less and less of a problem as embedded processors become faster and more feature rich in terms of security devices such as cryto-accelerators. The issue of physical access is purely a design issue which needs to be addressed up front and carefully thought through during the design process.

References

[1] http://ustudy.in/sites/default/files/381px-HalfSubtractor.svg_.png

[2] http://ustudy.in/sites/default/files/fullsub.gif

[3] http://www.blackhat.com/presentations/bh-usa-08/Goodspeed/BH_US_08_Goodspeed_Side-channel_Timing_Attacks_Slides.pdf

[4] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.586&rep=rep1&type=pdf