NTNU 2013-02-04
Department of Electronics and Telecommunications

TFE4140 Modeling and Analysis of Digital Systems

SEMESTEROPPGAVE / TERM PROJECT

DESIGN OF COMMUNICATION SYSTEM FOR STUDENT SATELLITE

Introduction

You and your partner won the contract for designing a measuring module for the new NTNU student satellite. This satellite will orbit the Earth, measure atmospheric data, and deliver this data to a ground station.

Your commission is to design a fault tolerant system called Liaison. The main task is to exploit redundancy so that correct data is transmitted to the Earth. The system has the following specifications:

- Four (4) existing microcontrollers shall be used. Each performs the exact same computations. Every computation produces an 8-bit word.
- Due to cosmic radiation etc., one or more microcontrollers may produce incorrect data. In this situation, the majority shall decide what the correct data is. This process is known as *voting*.
- The voted data should be accompanied by a 3-bit status signal containing the number of microcontrollers that have failed so far, indicating whether the data can be trusted or not.
- Additionally, data may be corrupted during the journey to the earth. This problem should be solved with error-correcting codes (ECC.) The chosen scheme must be able to **correct single-bit errors**, and **detect double-bit errors**.

Overview of sub-problems to be solved

1. **State analysis**
   The following is a typical chain of events in the lifetime of the system:
   1. Initially, all four controllers are error free.
   2. One controller has errors. It needs to be tagged appropriately.
   3. Another controller breaks.
   4. The remaining two controllers disagree. The system must be shut down, as there is no way to tell which one is still working.

   Illustrate possible scenarios as a state transition diagram with transitions governed by error tagging (in our project, no controller will be declared error free again after it has failed once.) Propose a sensible state register to indicate controllers with errors.

2. **Design of 1-bit voter (separate assignment)**
   We start by developing a state machine that performs voting. The design of the 1-bit voter shall be done individually. (This is assignment 3 and 4 in the course.)
   - The voter has four inputs; *a*, *b*, *c* and *d* (one bit of data from each of the four microcontrollers) in addition to a clock signal, *clk* (signifying when new data is available and voting should be performed) and a synchronous reset, *reset*. The module has two outputs; a 1-bit voted output, *y*, and a 3-bits status signal, *status*.

- The design must be able to keep track of which controllers have failed in the past, and which are still working (*error tagging*.) *status* should be coded as follows:
  - *status* = 000: all OK.
  - *status* = 001: one controller is faulty.
  - *status* = 010: two controllers are faulty.
  - *status* = 111: more than two controllers faulty/system shut down.
- Whenever *clk* has a positive transition, voted data should appear on *y* and *status* should be updated. If the *reset*-signal is high at the time of the clock transition, all controllers should be marked as error free and *y* should be set to 0. Make any additional assumptions yourself.
- The circuit shall be **modeled** in VHDL, and **verified** thoroughly by simulation in *Active-HDL*, with your own test bench (Assignment 3). Finally, the circuit shall be **synthesized** with *Synplify Pro* to verify that the design is synthesizable and obtain numeric values for the area usage and maximum clock frequency of the design. (Assignment 4.)
- After working individually, compare your solutions in your group. Discuss pros and cons with the different designs. Select the best design, and explain why this solution is the best one. Assume that the most important design goal is minimum area, or in the case of synthesis for FPGAs, as few *LUTs* as possible.

3. Design of 8-bit voter

The next step is to design a module that reads one 8-bit word serially from each of the four microcontrollers, and performs voting and error tagging. You may want to use your 1-bit voter as part of the design.

- Discuss 2-3 possible ways of realizing the voter. Select one, and give reasons for why this is the preferred solution. The most important design goal is small chip area.
- As before the circuit shall be modeled in VHDL, verified with your own test benches in *Active-HDL*, and then finally synthesized with *Synplify Pro*.

4. Design of error correcting circuit

Read the papers on error correction (*Hamming codes*) on *it's learning*. Select a scheme that performs the desired coding (single bit error correction, two bit error detection). Write a condensed technical note on the selection, and your implementation of it. (This note will later be part of your final report).

- Expand the 8-bit voter with the appropriate error correcting logic.
- This expanded voter, i.e., our complete liaison system, shall also be modeled in VHDL, verified by simulation with your own test bench, and finally synthesized. Again, **min. area** is the most important goal.

5. Analysis of error probabilities

Given a fault probability of [1/(6 years)], reflecting the expectation that on average, a microcontroller will fail after 6 years. Work out mathematical expressions for the following probabilities as a function of time:

1. P(error in maximum one controller).
2. P(error in maximum two controllers).
3. P(error in at least three controllers), thus preventing proper data to be transmitted.
4. Finally, compute the MTTF (mean time to failure) for the whole system.

Detailed specification of the finished Liaison system

We want to be able to easily compare your designs (e.g., area and speed/performance). Therefore, we will here specify signal names and types for every I/O, as well as some general guidelines that must be followed.

The main module of the design should be named *liaison*. The interface to *liaison* is presented in Figure 1 and Figure 2. Table 1 shows a description of the I/O signals. The finished system shall read one 8-bit word serially from each of the four microcontrollers, perform voting and error tagging and finally compute and append error correcting code.

The voted result (8 bit), status (3 bits) and error correcting code (m bits) add up to a sequence of (11+m bits), to be transmitted serially from the system. This sequence is illustrated in Figure 3.

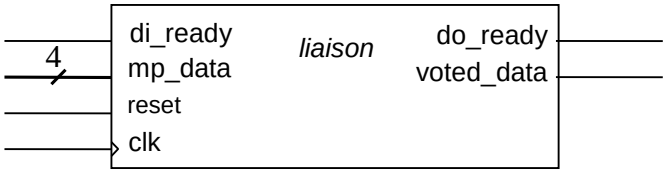

Figure 1: The interface to liaison.

```
ENTITY liaison IS
      PORT(
      clk         : IN  std_logic;
      mp_data     : IN  std_logic_vector(3 DOWNTO 0);
      reset       : IN  std_logic;
      di_ready    : IN  std_logic;
      do_ready    : OUT std_logic;
      voted_data  : OUT std_logic
      );
```

Figure 2: Entity declaration of liaison in VHDL.

Table 1: Description of signals.

Signal	Description
di_ready	A pulse on *di_ready* (data in ready) indicates that data starts arriving from the microcontrollers. Active high.
mp_data	Data from the four microcontrollers arrive serially on *mp_data* (*microprocessor data*). MSB arrives first.
reset	Synchronous reset of the system. Active high.
clk	Clock. All flip-flops are clocked on positive edge of *clk*.
do_ready	A pulse on *do_ready* (*data out ready*) indicates that data starts exiting liaison through the *voted_data* output. Active high.
voted_data	On this output, the voted result (8 bits) is clocked out serially followed by status (3 bits) and finally the error correction code (m bits.) MSB first.

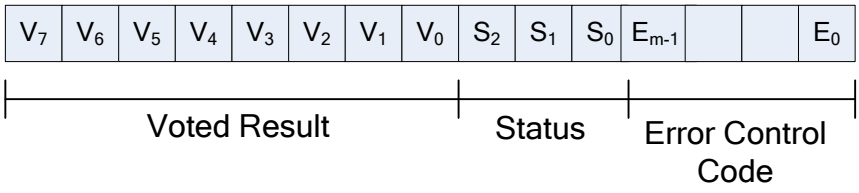

Figure 3: Format of data to be sent from the system.

Assume that the microcontrollers output data on the negative edge of *clk*, so that liaison can read data on the positive edge. Liaison should also output data on the positive edge of *clk*, so that the data can be read on the next negative edge.

Figure 4 shows liaison operating with the shortest possible delay between incoming consecutive words, namely 11+m clock cycles between each assertion of *di_ready*. Your implementation must be able to handle this scenario.

Unless reset is activated, a pulse on *di_ready* must always result in a pulse on *do_ready* and transmission of voted data on *voted_data*. The delay between these events is not important, however, and may be chosen by you.

In Figure 5, Figure 6 and Figure 7, various details of the protocol are illustrated.

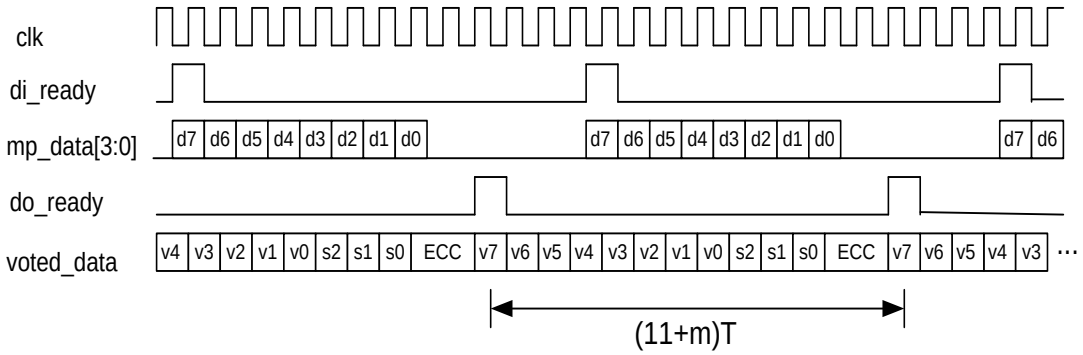

Figure 4: Relation between input and output signals.

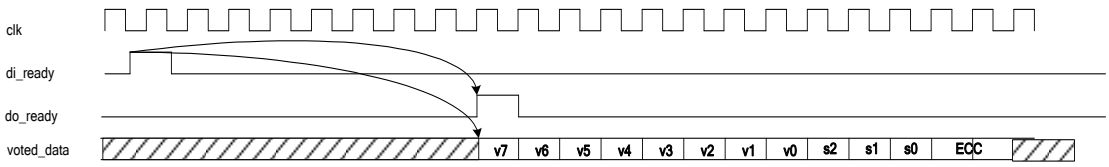

Figure 5: A pulse on di_ready must always result in a pulse on do_ready at some point in the future, but the number of clock cycles between these events can be chosen by you.

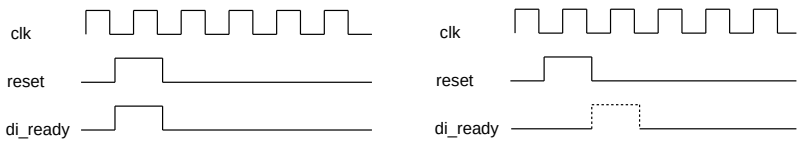

Figure 6: If reset and di_ready are activated simultaneously, reset shall get priority. Liaison must be ready to receive data immediately after reset is deactivated.

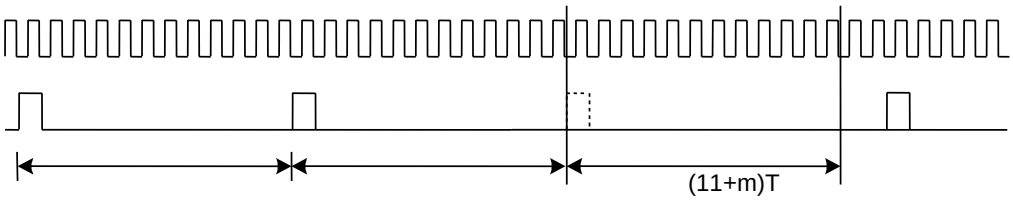

Figure 7: Pulses on di_ready may arrive at any time, but the distance between two pulses will never be less than 11+m clock cycles.

It is important that you describe any assumptions you make during the design process.

You are (presumably) novice designers. An experienced designer knows how to design (and write VHDL code) such that efficient circuits are obtained. For you to gain such experience, it is recommended to synthesize small modules at a time and study the resulting circuits. Make small changes to the code, and re-synthesize to observe the effect of these changes. Practice makes expert.

Please log the amount of work *each of you* do during the process. (This info will be used to estimate this year's workload.)

Milestones / plan for delivery

All tasks described in the *Overview of sub-problems to be solved*-section shall be performed during the semester. One preliminary report, giving the overall concept and architecture, shall be subject to "peer evaluation". This involves swapping of the report with another group, doing evaluation, and receiving evaluation.

- **Assignment 3: February 14, 2013:**
  Individual solutions for the 1-bit voter. They serve as basis for the solution chosen in the group for the 8-bit voter.
- **Technical note: March 14, 2013:**
  This note contains your preliminary solutions to sub-problems 1, 2 and 3. Your "peer group" will evaluate this note. When handing in your technical note on *it's learning*, also send it to your peer group by e-mail. A list of peer groups can be found on *it's learning*.
- **Peer evaluation: March 18, 2013:**
  Deadline for turning in the completed evaluation form. Send a copy by e-mail to your peer group.
- **Presentation: May 2, 2013:**
  Oral presentation, 6 minutes. Only one person will present from each group, as determined by a coin toss before your presentation. This way you both have to practice.
- **Final report: May 3, 2013:**
  Delivery of final report, with answers/solutions to all sub-problems. The contents of the final report are described below.

Contents of final report

- Complete description of the problems to be solved. In this way, the report will be self-sustained, i.e., understandable for a reader who has not read the specification.
- Answers to all sub-problems. The report shall include an evaluation of at least two different possible solutions, and argumentation for why the chosen solution is preferred. A thorough description of your design.
- The designed circuit must be properly verified. You shall convince the reader (customer) that your design will comply 100% with the given specification. Discuss how you chose your test vectors, and why they guarantee that the design works. Include simulation timing diagrams, annotated for readability.
- Synthesis results: number of LUTs and flip-flops used, and the maximum clock frequency of your design. Put schematics in the appendix.
- All source code produced, including test benches.
- Any feedback to the teachers and the assistants, and how much time you spent on the project. Please be honest.

Some words on how to write your report

- The report should have a front page, abstract (summary), table of contents, introduction, and a conclusion at the end. You may otherwise choose how to organize the report.
- Try to use the *what, why, how* scheme everywhere. That is, in each sub-section describe what you are going to present, why it is important, and how you solved the problem at hand.
- Spend time on making good figures. Annotate timing diagrams to make it easy to see what is happening. E.g., add descriptive labels and arrows pointing to notable events.
- Use a consistent layout on the source code, and avoid too much whitespace (blank lines.) Write descriptive comments. If you use LaTeX, use the *listings*-package to display source code in the report.

You may write your report in any document processor or typesetting program, such as Microsoft Word or LaTeX. It is important to produce a consistent and well-integrated final report. Writing good reports is important both in academic and professional contexts, and here is your chance to practice!

Normally, the members of each group will receive the same grade on the project. In some (extreme) cases, one of the members in a group ends up doing considerably more work than his/her peer(s), in which case it is possible to distribute the points given to your group unevenly between the group members. Please set up a meeting with Snorre Aunet if you feel this applies to you.

Good luck!