

This homework is due either **at the beginning of class** on Tuesday, November 1. *No group work is allowed on this assignment.*

Note: A correct answer without adequate explanation will have points deducted. To get full credit, (a) write legibly/type, and (b) show all work (label relevant items, show derivations, include explanations).

1. (36 points) **Low Power Design: Precomputation-Based Sequential Logic Optimization.** In this problem, you are to create two different low-power versions of a design of an **8-to-3 priority encoder** stage in a pipelined sequential system. Refer to Handout #21 and the corresponding class lectures.

Setup. A *priority encoder* block is described in Brown/Vranesic, ch. 6.3.2 (3rd edition). The book's example is a 4-to-2 priority encoder, with 4 inputs (w_3, w_2, w_1, w_0) and 3 outputs (y_1, y_0, z). As indicated, w_3 is highest priority input and w_0 is lowest priority input, the resulting asserted output has 2 bits ($y_1 y_0$), which is the binary representation of the highest priority requesting input. The flag output z is de-asserted to 0 when no input requests are 1, otherwise is asserted to 1 along with the encoder output result on $y_1 y_0$. For this problem, extend this priority encoder to handle 8 inputs (w_7 to w_0), an output code on the 3 bits ($y_2 y_1 y_0$), as well as a z output. This 8-to-3 priority encoder will be used in a pipelined system architecture as shown in Figure 1 of Handout #21, and will serve as "Block A". Note that this block (unlike "Block A" in the paper) has *multiple outputs*.

- (a) **First Precomputation Architecture.** Design a low-power sequential optimized version of Figure 1, using the 8-to-3 priority encoder as "Block A", based on the first precomputation architecture shown in Figure 2.

(i) *Show all work:* label your figures, and add explanation to justify the components of your design. (A minimally-labelled or explained correct design may have points deducted.) (ii) *Clearly derive the "hit rate" of your precomputation blocks:* assuming random input vectors, for what percentage of vectors will the register disabling be activated and the bypassed precomputed outputs be generated?

Requirement: Your "predictor" block (containing g_1 and g_2) should examine exactly 3 different w_i inputs: w_7, w_6 and w_5 .

Hint: Two critical parts of your design will be to show (i) how you handle multiple outputs in "Block A", and (ii) how you design g_1 and g_2 "predictor" functions.

- (b) **Second Precomputation Architecture.** Repeat part (a) above, but now based on the second precomputation architecture shown in Figure 3.

(i) *Show all work:* label your figures, and add explanation to justify the the components of your design. (A minimally-labelled or explained correct design may have points deducted.) (ii) *Clearly derive the "hit rate" of your precomputation blocks:* assuming random input vectors, for what percentage of vectors will the register disabling be activated?

Requirement: Your "predictor" block (containing g_1 and g_2) should examine exactly 3 different w_i inputs: w_7, w_6 and w_5 .

Hint: Three critical parts of your design will be to show (i) how you handle multiple outputs in "Block A", (ii) how you design g_1 and g_2 "predictor" functions, and (iii) how you partition inputs into registers $R1$ vs. $R2$.

2. (30 points) **Low Power Design: Bus Invert Coding.** Refer to Handout #25 and the corresponding class lectures. In this problem, we will assume 8-bit data words, which you will encode through bus invert coding. We refer to *three alternative* bus invert codes for the 8-bit data:

Code #1: Monolithic Code. One 8-bit data field followed by 1 invert bit.

Code #2: Partitioned Code A. 8-bit data divided into two 4-bit fields, each of which is followed by 1 invert bit.

Code #3: Partitioned Code B. 8-bit data divided into four 2-bit fields, each of which is followed by 1 invert bit.

- (a) **Simple Encoding Example.** You are given the following sequence of five 8-bit data bytes transmitted in order: 00000000, 01001000, 00101101, 11010110, 10111101. Assume that the first data word (00000000) is transmitted with invert bit(s) set all to 0. *Write out* the corresponding transmitted code sequence for this five data byte transmission using (i) Code #1, (ii) Code #2, and (iii) Code #3.
- (b) **Cost Evaluation: Code #1.** Using the methods presented in Handout #25 and in class lectures, derive “average power”, “peak power”, and “coding efficiency”, for Code #1. *For full credit, show all work and clearly label and explain each step.*
Note: By “average power”, we mean “average number of bit transitions per transmission”, *assuming random input bytes*. By “peak power”, we mean “maximum possible number of bit transitions per transmission”. By “coding efficiency”, we mean “number of bits/number of wires”.
- (c) **Cost Evaluation: Code #2.** Repeat part (b), but now for Code #2.
- (d) **Cost Evaluation: Code #3.** Repeat part (b), but now for Code #3.
- (e) **Cost Evaluation: Summary.** Provide a 3-4 sentence summary, clearly evaluating the tradeoffs in the above 3 cost functions, between these three different bus invert codes (Code #1, Code #2, Code #3).

3. (18 points) **VHDL for Sequential Logic: Master-Slave JK Flipflop.**

In this problem, you are to model the master-slave JK Flipflop implementation shown in B/V Fig. P7.4 (p. 479) (3rd edition).

This master-slave JK flipflop is based on SR latches; it is an alternative implementation to the edge-triggered implementation in B/V Fig. 7.17(a) (3rd edition) based on a DFF. The former SR-latch based design is known to have a “latchup” problem: in some scenarios, input changes made *before the sampling rising clock edge* can eventually result in incorrect final Q output changes, hence the flipflop is not properly edge-triggered. (For optional extra reading on this phenomenon, see the Katz book on reserve.)

Do the following:

- (a) *Write VHDL (entity/architecture) for this JKFF.* Your top-level architecture should use structural VHDL, modelling the 5 distinct gates/components shown in Fig. P7.4, and their interconnections. In turn, the combinational gates should be modelled using dataflow VHDL, and the SR latches using sequential VHDL.
- (b) *Latchup Scenario.* Write a simple and short timing diagram, indicating all primary inputs/outputs (J, K, Clock, Q, Q’) on the y-axis, demonstrating the problematic “latchup” scenario, where input changes made before the rising clock edge can eventually incorrectly propagate to the JKFF Q output. *Also include 2-3 sentences of discussion, that clearly explains how the problem arises.*

4. (16 points) **VHDL for Sequential Logic: LFSR Design.**

In this problem, you are to model a 5-bit LFSR design, following the approach presented in class and in Handout #6. You should add the appropriate modifications, as discussed, to generate the “all-0” state.

You will *not* model any individual gates or structural composition of cells. For this problem, you will make a simple and compact architecture based mostly on behavioral VHDL, but with some dataflow statements included.

- (a) *Block-level design.* Draw a block-level view of the design, showing all 5 stages, as well as appropriately-added XOR and other gate-level logic. Your figure should show a black box for each stage, and the interconnection of the 5-stages plus the gate-level added logic. The LFSR should include appropriate active-low asynchronous reset. Clearly label all signals in this figure.
- (b) *VHDL model.* Write VHDL code (entity/architecture) for this 5-bit LFSR. Your code should be *entirely using sequential and dataflow VHDL*: do *not* use any structural VHDL! In particular, for full credit, use a single “FOR-LOOP” sequential construct to generate the 5 LFSR cells. Be sure to model appropriate asynchronous active-low reset for the LFSR, as well as the computation of the new left serial input bit, and the handling of the all-0 state. The FOR-LOOP is used for replication for sequential VHDL, the same way that the “for-generate” is used for dataflow and structural VHDL. (See B/V Appendix A.9.4 and A.9.7 for discussion and examples.)