**Binary Phase Shift Keying**

**Transmitter: Optimization Exercise (55X)**

Spencer Brady

This work is produced by The Connexions Project and licensed under the Creative Commons Attribution License

---

**Abstract**

Students are to implement and optimize a pseudo-noise (PN) sequence generator and a binary phase shift keying (BPSK) transmitter.

In this lab you are to implement and optimize a pseudo-noise (PN) sequence generator and a Binary Phase Shift Keying (BPSK) transmitter to encode and transmit the PN sequence. For the lab grade, you will be judged on the execution time of your system (memory usage need not be minimized).

1 **Reference Implementation**

You will find the source code for this lab in the following folder `v:\ece420\55x\bpsk.tx`. After taking a look at the source code for the BPSK transmitter reference implementation, you will likely discover inefficiencies. This implementation is provided as the “reference implementation” of the optimization process and to define the expected input and output of the application. The computational efficiency of your code will be judged against this implementation. While the given source code might serve as a starting point, you should do whatever you need to do to make your code as efficient as possible, while operating in an equivalent manner as the given code.

The exact portion of the code to be optimized is defined below. You may write in C, assembly, or any combination of the two; choose whatever will allow you to write the fastest code. The optimization process will be smoother if you plan for optimization before you begin any programming.

2 **Optimization**

Since a primary purpose of this lab is to learn optimization and efficient code techniques, your lab grade will be based primarily on the total execution time of your system. You are not required to optimize memory use. Note that by execution time we mean cycle count, not the number of instructions in your program. Remember that some of the TMS320C55xx instructions take more than one cycle. However, most operations take only one cycle and can be placed in parallel with other operations. Branch and repeat statements are the most common instructions that require several cycles to execute. Most C instructions take more than one cycle. The debugger can be used to determine the exact number of cycles used by your code. The instructions on how to do this can be found in Cycle Counts (m14415).

---

†http://creativecommons.org/licenses/by/2.0/
We will grade you based on the number of cycles used between the profile\_me = 1; and profile\_me = 0; statements. Thus, you can optimize code within this interval, but optimizing code outside of this interval will not help. For grading simplicity, your code should not have modifications except between these two instructions. If the number of cycles between these two points is variable, the maximum possible number of cycles will be counted. We reserve the right to test your code by modifying the inputs.

3 Routine Specific Optimization Tips

If you are programming the PN generator in assembly, you may wish to refer to the description of assembly instructions for logical operations in the C55x Mnemonic Instruction Set reference. Initialize the shift register to one. You can debug the PN output by comparing it to the output of the MATLAB code. Be prepared to prove to a TA that your PN generator works properly as part of your quiz. For more information about the PN generator, see Lab 4 Prelab.

Your BPSK transmitter can be debugged by looking at the output of the transmitter along with the corresponding PN values on channels one and two of the oscilloscope. The transmitter output corresponding to PN=1 should be 180 degrees out of phase with the transmitter output corresponding to PN=0. To verify that you are starting with a decent sine wave, you might try overwriting the PN sequence to a constant zero or one, which should produce a smooth sinusoid with no phase shifts at the transmitter's output.

4 Grading

Grading for this lab will be a bit different from past labs:

- 2 points: Working code, implemented from scratch in assembly language and/or C.
- 5 points: Optimization. These points will be assigned based on your cycle counts and the optimization you have made.
- 3 points: Oral quiz.

http://cnx.org/content/m15202/1.1/