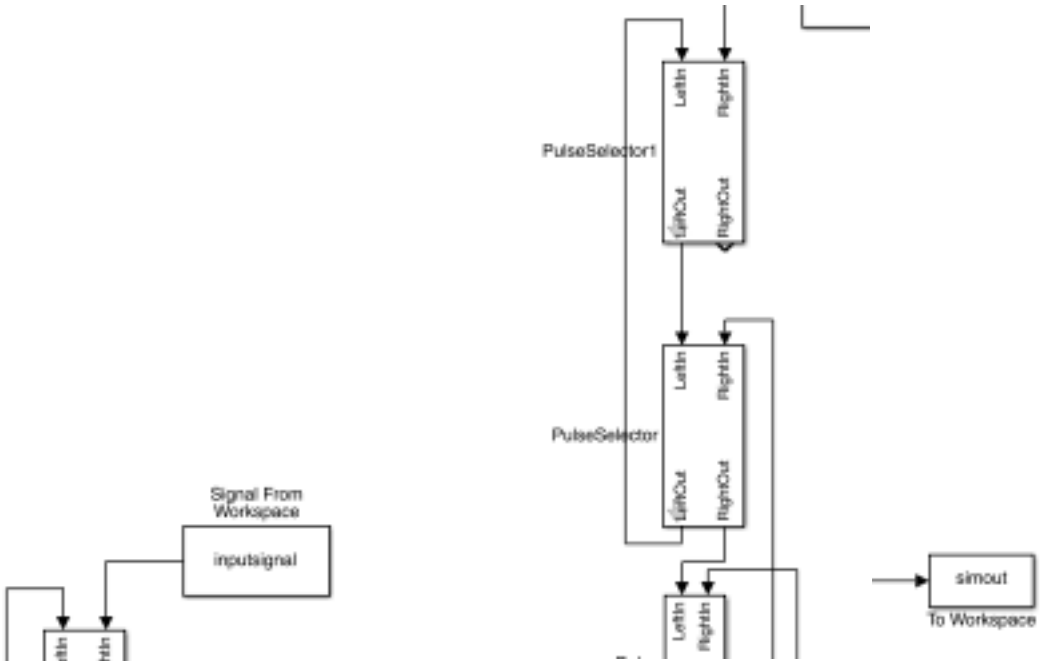
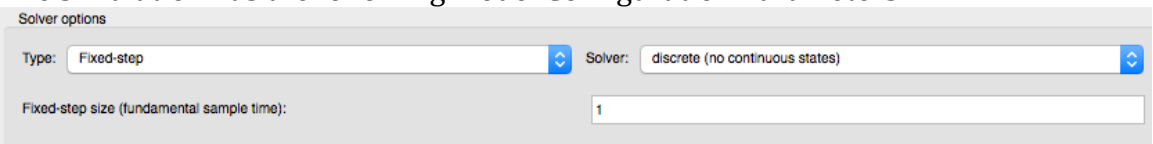
Basics of a Simulation Design

This document will describe the essential elements of running a simulation of an optical network. Please open the "Digitizing Design.slx" Simulink model for reference.

Designs require three essential pieces. The "Signal From Workspace" block (may need DSP toolbox, email kmenda@hmc.edu if this is a problem) which retrieves the inputsignal, which has the pulse stream. The components that pass the signal around, and final the "To Workspace" block, which stores the simulation output in a variable, called "simout", used by other scripts for plotting, etc.



The simulation has the following Model Configuration Parameters:



At each timestep, the next element in the vector "inputsignal" is fed into the model, and intermediate signals are fed forward to the next blocks. Timesteps are unit and Simulink is told not to do any continuous solving.

Please note the wiring of blocks. For a block taking a signal from the left and transmitting it right, the signal must enter at "LeftIn" and leave from "RightOut". We apologize for the fact that the inputs and outputs are not on the physical sides of the block they refer to – Simulink chose to not allow this. So, be careful with wiring!

Before continuing, please familiarize yourself with the wiring scheme of this design, and double click specifically on the PockelsCell and other blocks to see what the block parameter masks are like.

A design must run an initialization script to set up a few variables, such as "inputsignal", in the workspace. This can either be done manually or using a Callback function. See File>Model Properties>Callbacks>InitFcn, which runs the script "initialize.m" before the model starts running.

The script "**initialize**.m" is documented in detail, but the key components are:

- Clear any objects instantiate in previous simulations
- Define a few variables, such as N, T
- Define global variables that components refer to, such as "montecarlo"
- Set up "inputsignal"

Before running, you must make sure that you are in the folder containing BlockLib.slx, and the folder "Components" is in the path.
If they are, hitting run should work.

With the simulation terminates, it uses another callback, "StopFcn" to call "**stopscript**.m", which is commented. It does the following:

- Call "OutputPlotting" which makes the timing diagram
- Check for interference, and print results to the Simulink Model Diagnostics Viewer

If these pieces are in any design, I believe it should work so long as all the necessary information is in the workspace, and parameters for components are valid. Particular ones that could cause problems are PCTimings and Contol Powers for Pockels Cells, so please see type "help PockelsCell" for usage information. If you want to make your own design, it is probably easiest to make a copy of "DigitizingDesign.slx" and change the components between "inputsignal" and "simout", so Simulink parameters are copied over.

If "OutputPlotting" managed to run, there should be a vector called "IDs" in the workspace. Please take any pulse ID in that vector and type `Pulse.printStateHistory(ID)` and see this debug tool. If you see a pulse that shouldn't be somewhere, you can trace that pulse's history with knowledge of its ID. To figure out which pulse is with, it is easiest to use logical indexing with the "times" vector.

Another debug tool is EOM state diagrams. Please type `PC2 = PockelsCell.getComponent(2)` into the command window. Then please type `PC2.plotIO(T)` into the command window. In a few seconds, you should see the EOM state diagram.
Another document will discuss how to perform Monte Carlo simulations and get results other than timing diagrams.