FPGA Implementation of Central Pattern Generator
By James J Lin

Introduction:

Many analog implementations of CPG exist, typically using operational amplifier or transistor level circuits. These types of circuits have reduced power consumption and can be readily translated from the CPG dynamic equation. However, one primary disadvantage is fixed parameters. Without fixed parameters, it is difficult to create circuits that can generate various signals depending on sensory feedback. In this document, a digital implementation of CPG will be proposed and analyzed. It is expected that the CPG can be abstracted; thus, sensory feedback can tune the oscillatory parameters. The following is a list of the hardware and software tools used in this document.

Hardware:

Xilinx Virtex 4 ML403
Device/Package = 4vfx12 / ff668
SpeedGrade = -10

Software:

MATLAB v7.0.4
Simulink
Xilinx DSP System Generator v7

Central Pattern Generator:

The concept of CPG emerges from biological systems. Typically, in the base of the spine are a group of oscillatory neurons called the CPG. Together, these neurons generate the rhythmic patterns that map onto locomotive motion. The mathematical description of a CPG comes in numerous forms. For our implementation we will choose the Reaction-Diffusion CPG equation. This equation has four tunable parameters: alpha, beta, i_1, and i_2. Our CPG implementation will store these four values in registers, which will allow external controllers to modify these parameters. The overall RD CPG state response is described below:

$\partial x_1/\partial t = -x_1 + \text{alpha}*y_1 - \text{beta}*y2 + i_1$
$\partial x_2/\partial t = -x_2 + \text{alpha}*y_1 + \text{beta}*y_2 + i_2$
$y_i = f(x_i)$ $\quad f(x) = 0.5*[\text{abs}(x+1) - \text{abs}(x-1)]$

As background, the function f is called the activation function and numerous types exist including the sigmoid and arctan. Our piecewise linear activation function is a simple approximation. Using MATLAB Simulink, this CPG equation can be represented by the following simulation circuit in Figure 1.

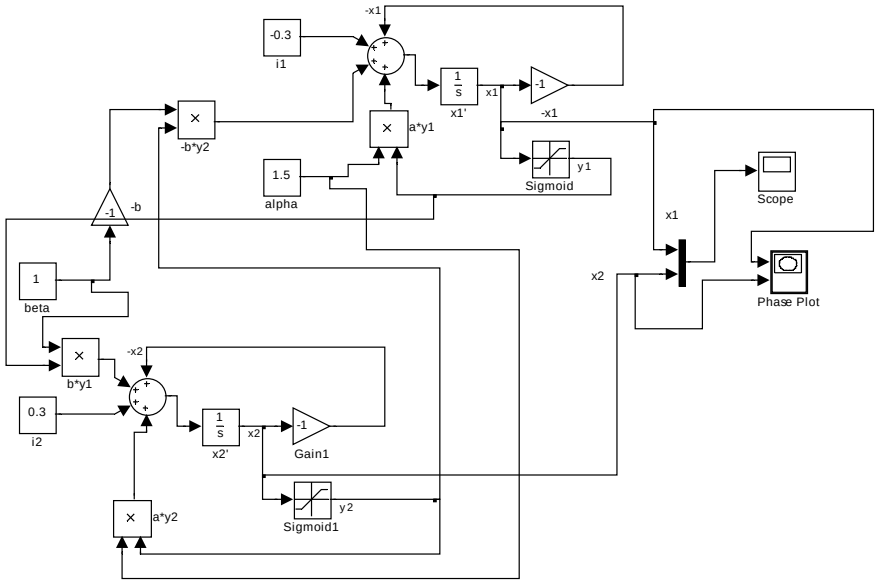

Figure 1: Continuous time implementation of RD-CPG in Simulink

Notice that there are two primary modules that are unique to the RD-CPG: the integrator and the sigmoid function. The interaction between these two modules forms the heart of the RD-CPG state response. A phase plot is shown in Figure 2 and output state response plot is shown in Figure 3. The input parameters are chosen as: alpha = 1.5; beta = 1; $i_1 = .3$; $i_2 = -.3$. These parameters were selected by a genetic algorithm programmed by Jonathan Tay. The details of this specific parameter selection will be discussed in another document.

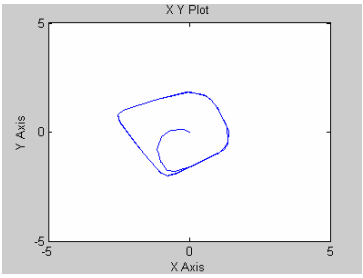

Figure 2: Phase plot

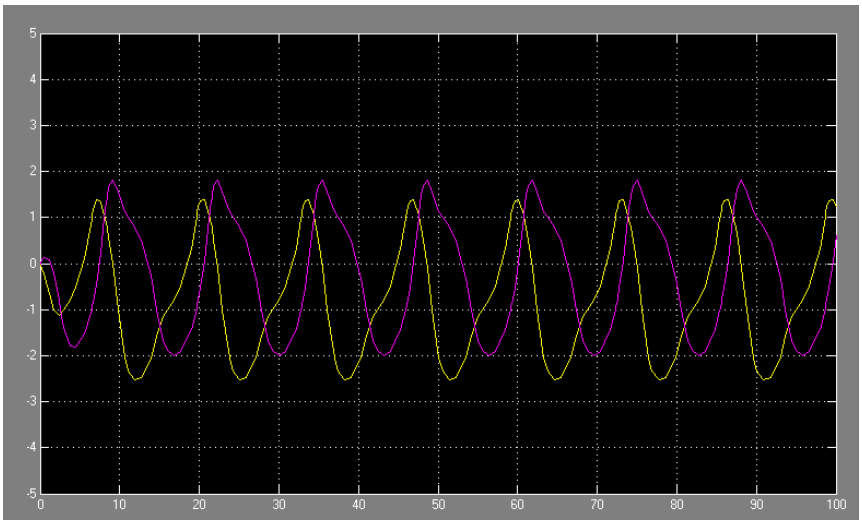

Figure 3: Output state response waveform

Notice that the phase plot forms a limit cycle. This implies that the output state response will generate stable oscillatory signals. With the desired output waveforms and RD CPG circuit defined, we can now implement a digital circuit and verify that it produces similar properties.

FPGA Implementation of CPG:

There are numerous ways to implement CPG digitally. We have chosen to use Xilinx DSP System Generator to program our CPG. Using this software tool, we have abstracted key modules typically used in digital programming and may now focus on the design of our specific application. This produces more robust and efficient code. For our digital implementation of CPG using FPGA, we can divide our labor into three tasks. The first task is to create the CPG integrator and sigmoid block using the System Generator. The second task is to create the RD CPG neuron for Verilog generation. The third task is to configure the FPGA to run the module.

DSP System Generator:

Xilinx produces many software tools to simplify the design tasks for hardware programmers. One such program is the DSP System Generator. It interfaces with MATLAB Simulink and provides efficient implementations of digitally realizable Simulink blocks and commonly used digital programming blocks. This includes the Adder, Multiplier, Negate, Constant, Register, Mux, and Comparator. Another important feature of System Generator is the Gateway In/Out blocks. These provide the interface between the double precision of Simulink with the floating point architecture of the FPGA. Using these basic blocks, higher-order blocks and modules can be created for our RD CPG.

Integrator and Sigmoid:

We begin our System Generator RD CPG module using a bottom-up approach. At the heart of the RD CPG is the integrator and sigmoid block. These blocks have simple digital representations. Integration can be approximated with a Riemann sum. This implies that an Adder and Register blocks can approximate integration. In Figure 4, the integrator block is shown using these two System Generator blocks.

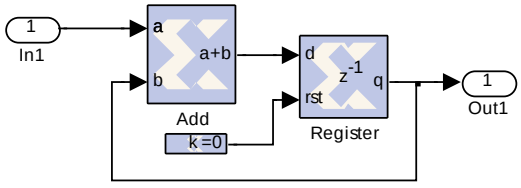

Figure 4: Integrator block

With regard to the sigmoid block, the output is 1 and -1 when the input is above or below 1 and -1, otherwise, the output equals the input. Comparator and Mux blocks are used to implement this function. Shown Figure 5 is the sigmoid block.

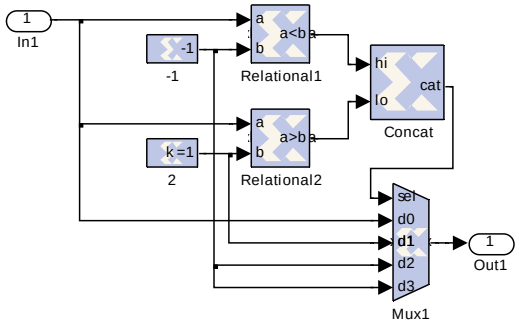

Figure 5: Sigmoid block

Notice that higher-order blocks can be quickly created using the fundamental blocks provided by System Generator. This reduces our programming burden and allows the Xilinx tools to optimize our module for specific FPGA platforms. With these System Generator blocks for integration and sigmoid, we have all the blocks necessary to implement the RD CPG module.

System Generator Model of CPG:

Keeping in mind that our goal is to abstract the RD CPG neuron to provide tunable parameters, we intend to create four input ports that modify the alpha, beta, i_1, and i_2 registers. Shown in Figure 6 is our RD CPG System Generator module.

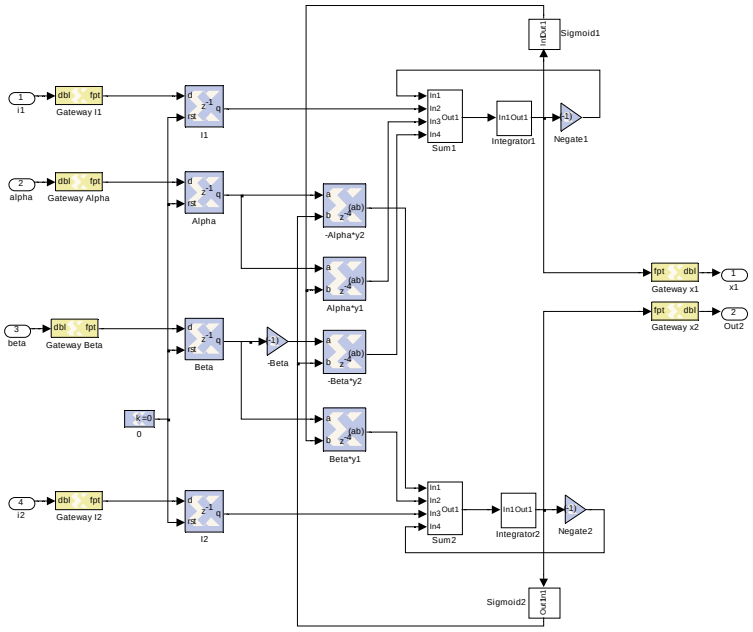

Figure 6: RD CPG module

For our digital implementation of RD CPG, we have chosen a 2's complement and 20 bit floating point number with a binary point of 15. This choice was arbitrary and further analysis will determine the optimal floating point representation. Figure 7 shows the simulation test-bench and Figure 8 shows the output waveform produced by the RD CPG circuit.

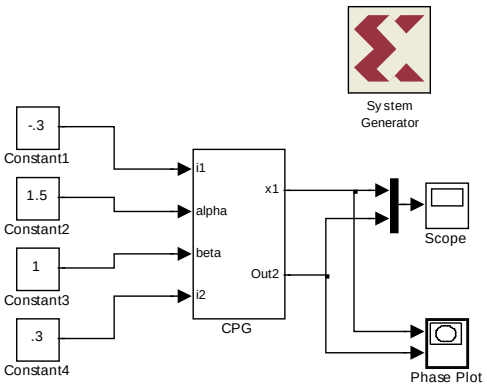

Figure 7: Simulation test-bench

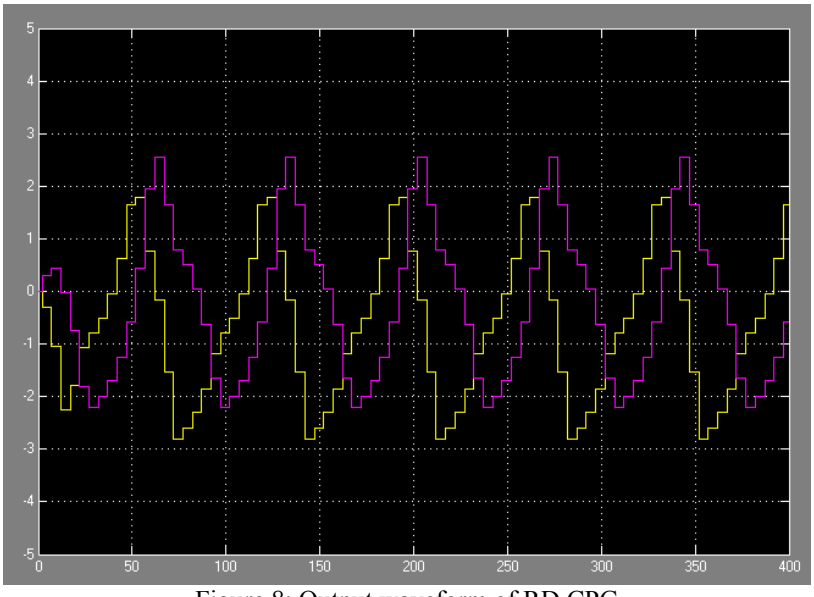

Figure 8: Output waveform of RD CPG

Notice that the output waveform is a discrete version of Figure 3 and that the time axis has been scaled. These variations are expected since the FPGA module will sample values according to the system clock, which produces the step shape of the waveform and accounts for the time

scale. With this successful simulation model of RD CPG, we will generate the Verilog module. Figure 9 shows the System Generator wizard and our Virtex 4 FPGA board parameters.

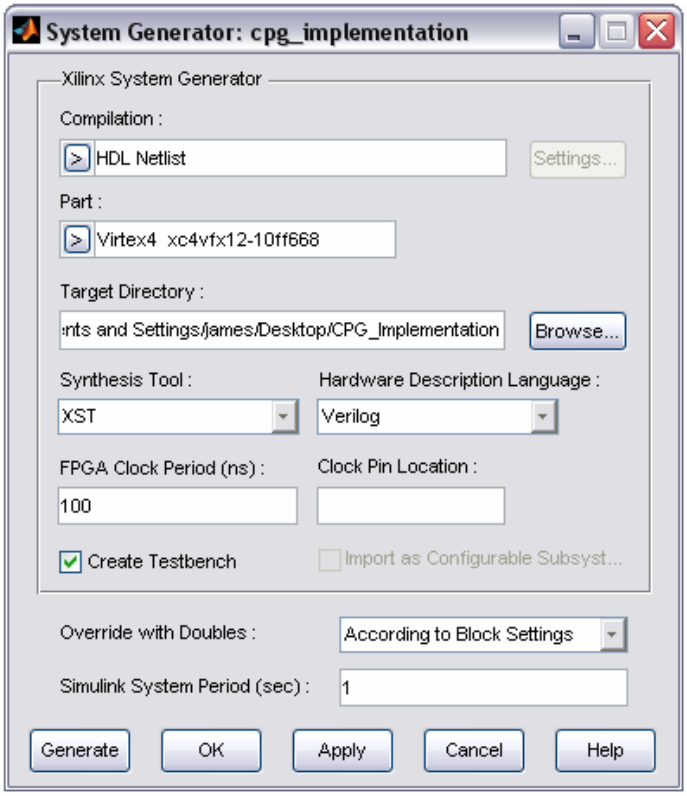

Figure 9: System Generator Wizard