# DANNA Neuromorphic Application Kit Demo

James S. Plank
Professor

2017 NICE Lightning Talk

THE UNIVERSITY OF TENNESSEE KNOXVILLE

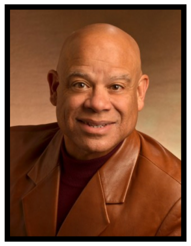
BIG ORANGE. BIG IDEAS.®

# This is really an amalgamation of 2 talks

- A **Vertical** Application Programming and Development **Framework** for Spike-Based Neuromorphic Computing Devices

James Plank
Catherine Schuman
Mark Dean
Garrett Rose

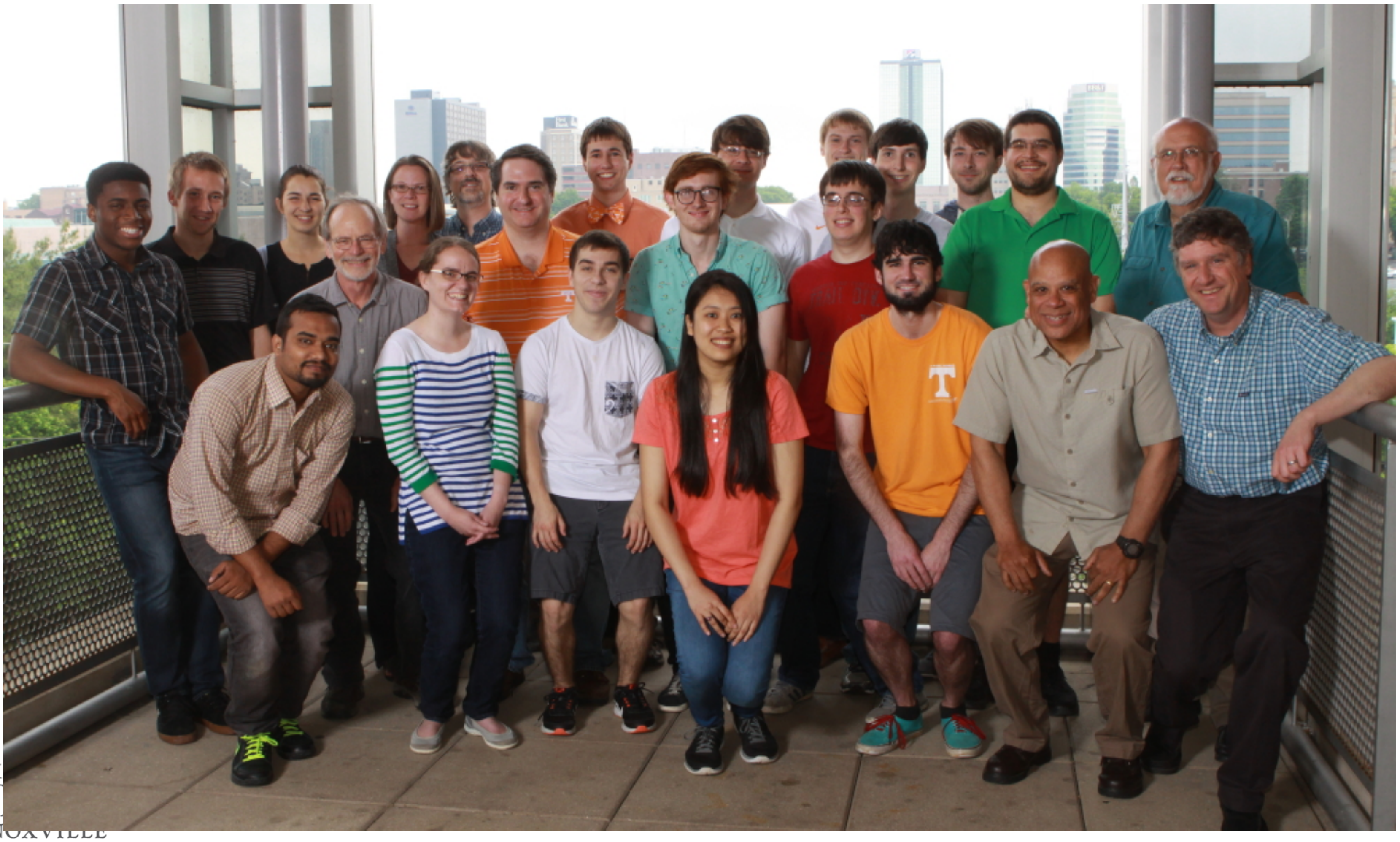
- DANNA Neuromorphic Application Development **Kit Demo**



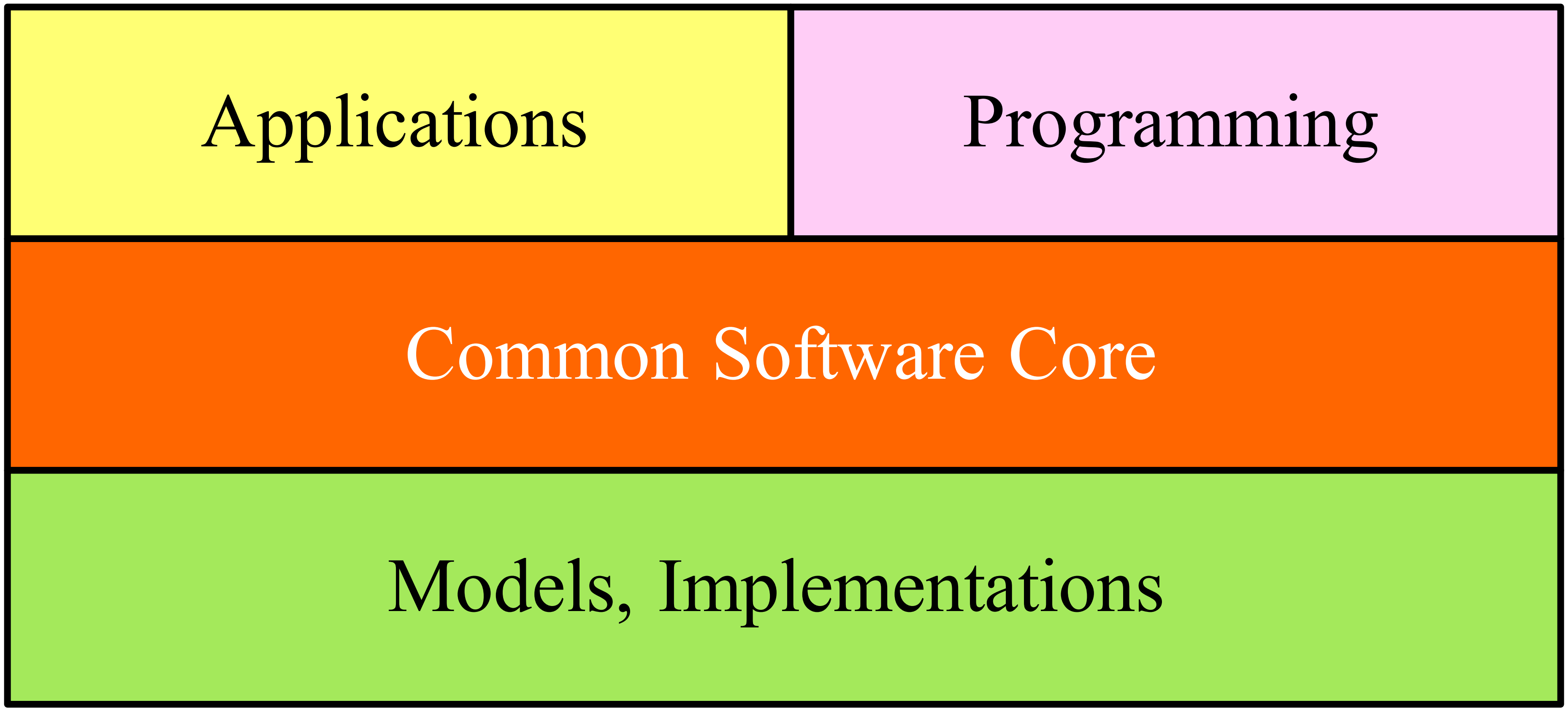
Young, Reynolds, Eckhart, Mitchell, Disney, Bruer + Adults.

THE UNIVERSITY OF TENNESSEE KNOXVILLE

## The Neuromorphic Group at Tennessee

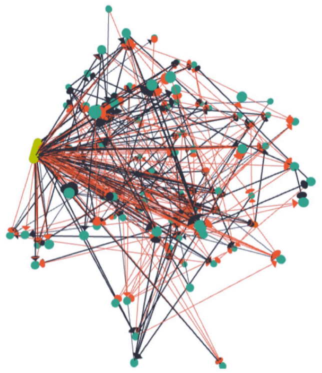


## The Vertical Framework



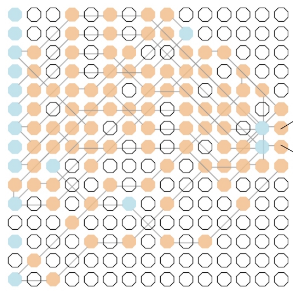
# The Vertical Framework: Models & Implementations

## NIDA



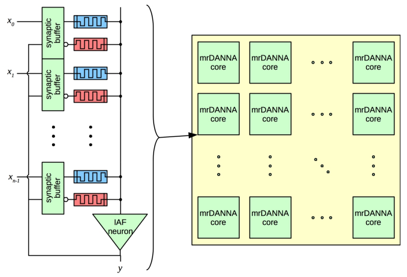
3D, Analog
Simulation
Viz * 2

## DANNA



2D, Discrete
Simulation * 3, (GPU Sim)
FPGA Implementation + Kit
VLSI design done
Visualization

## mrDANNA



2D, Analog
Simulation * 2
Memristors
Chip Fab w
SUNY Nanotech

## The Vertical Framework: Models & Implementations

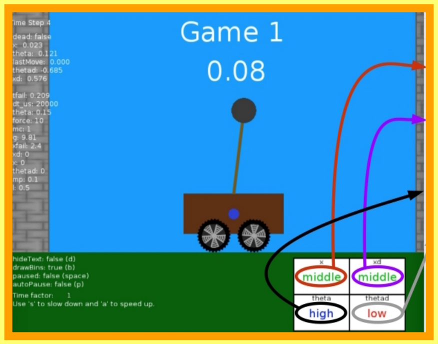
- All implement a common interface
- Include genetic operations
- Other models would be welcome
    - Reservoir
    - True North
    - Biomimetic

# The Vertical Framework: Applications

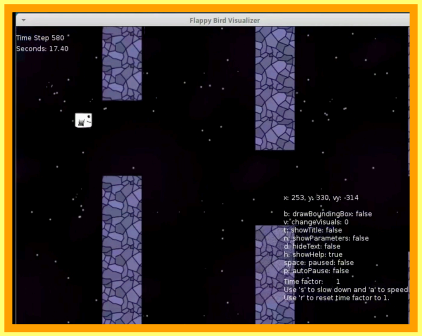
Control

Classification

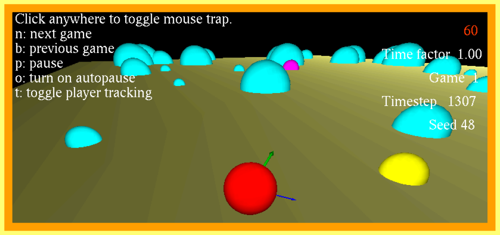
Microapps



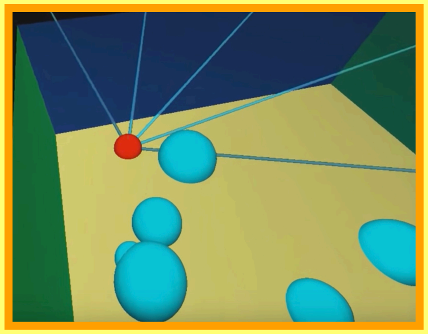
```
Time Step 4
dead: false
x:  0.023
theta:  0.121
lastMove:  0.000
thetad: -0.685
xd:  0.576

tfail: 0.209
dt_us: 20000
theta: 0.15
force: 10
mc: 1
g: 9.81
xfail: 2.4
xd: 0
x: 0
thetad: 0
mp: 0.1
l: 0.5

hideText: false (d)
drawBins: true (b)
paused: false (space)
autoPause: false (p)
Time factor:      1
Use 's' to slow down and 'a' to speed up.
```

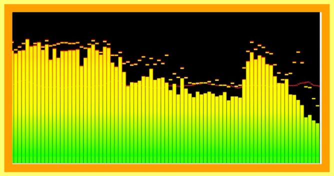


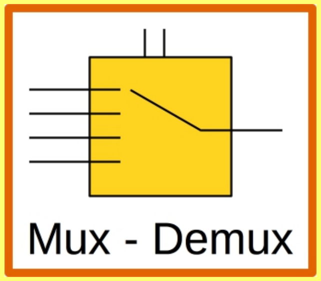
```
Time Step 580
Seconds: 17.40

x: 253, y: 330, vy: -314

b: drawBoundingBox: false
v: changeVisuals: 0
t: showTitle: false
n: showParameters: false
d: hideText: false
h: showHelp: true
space: paused: false
p: autoPause: false
Time factor:      1
Use 's' to slow down and 'a' to speed
Use 'r' to reset time factor to 1.
```

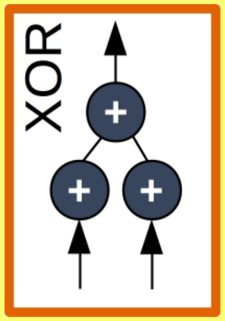


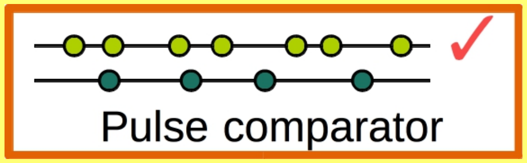
```
Click anywhere to toggle mouse trap.
n: next game
b: previous game
p: pause
o: turn on autopause
t: toggle player tracking
60
Time factor  1.00
Game  1
Timestep  1307
Seed 48
```











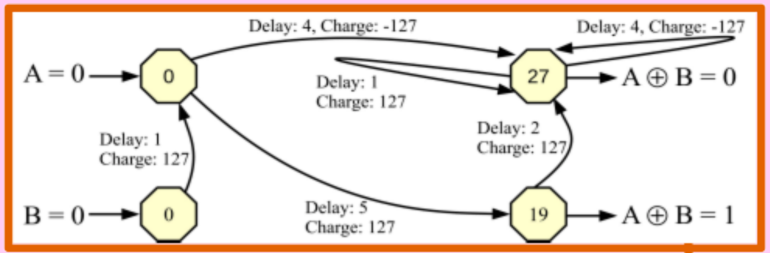
## The Vertical Framework: Applications

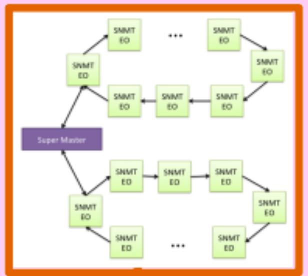
- All program to a common interface
- Include genetic operations (fitness)
- Compose a suite of scalable benchmarks
- Other applications are welcome:
    - Helicoptor Brownout
    - Mix/match with Deep Learning

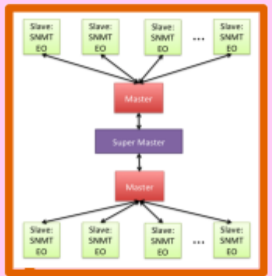
# The Vertical Framework: Programming

**Evolutionary Optimization**

- Applications define fitness
- Models define operations
- Evolution of parameters and structure







**Exploiting Parallelism**

**Kernel Development**

## The Vertical Framework: Common Software Core

- DeviceModule
- NetworkModule
- I/O Module
- Job Module
- Programming Module

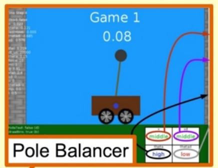
THE UNIVERSITY OF TENNESSEE KNOXVILLE

# The Vertical Framework

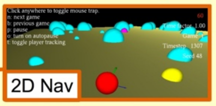
We have developed a software environment for programming applications on spike-based neuromorphic computing systems. The environment is **vertical**, from applications and programming methodologies at the top, to software support at the middle, to various neuromorphic computing models, simulators and devices at the bottom.

## Applications

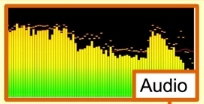
Flappy Bird



Pole Balancer

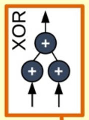

2D Nav

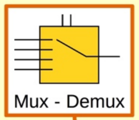
Control


Audio

UCI Data base

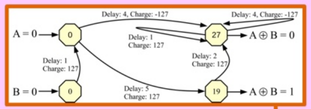
Classification




Mux - Demux

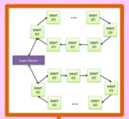
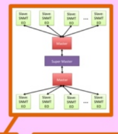
Microapplications

## Programming



**Evolutionary Optimization**

- Applications define fitness
- Models define operations
- Evolution of parameters and structure

Exploiting Parallelism

Kernel Development

## Common Software Core

Implementation in C++

**Network Module**

- Create
- Serialize
- De-serialize
- Mutate
- Reproduce (Crossover)

**Device Module**

- Instantiate
- Load network
- Pull network
- Apply inputs
- Read outputs
- Run

**I/O Module**

- Transform inputs
  ← numbers, strings
  → pulses, neurons, values
- Transform outputs
  ← counts, neurons
  → numbers, decisions

**Job Module**

- Define job
- Batch multiple jobs
- (Support for GPU)
- (Support for threading)

**EO Module**

- Define complexity
- Fitness evalution
- Sliding fitness window
- Reproduction
- Job support

## Neuromorphic Models

**NIDA**

- 3D RISC model
- Analog neurons & synapses

Simulator in C++

3D Visualization

ASIC Design Completed

**DANNA**

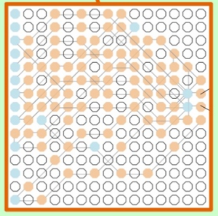
- 2D grid of programmable elements
- See our demo/poster on DANNA

FPGA Implementation

Simulator in C++

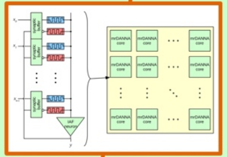
GPU Simulator for batched runs

Hardware kit



**MrDANNA**

- Memristive Synapses
- See our poster on MrDANNA



Simulator in C++

Chip fabrication with SUNY Nanotech

**Others**

Reservoir
Biomimetic

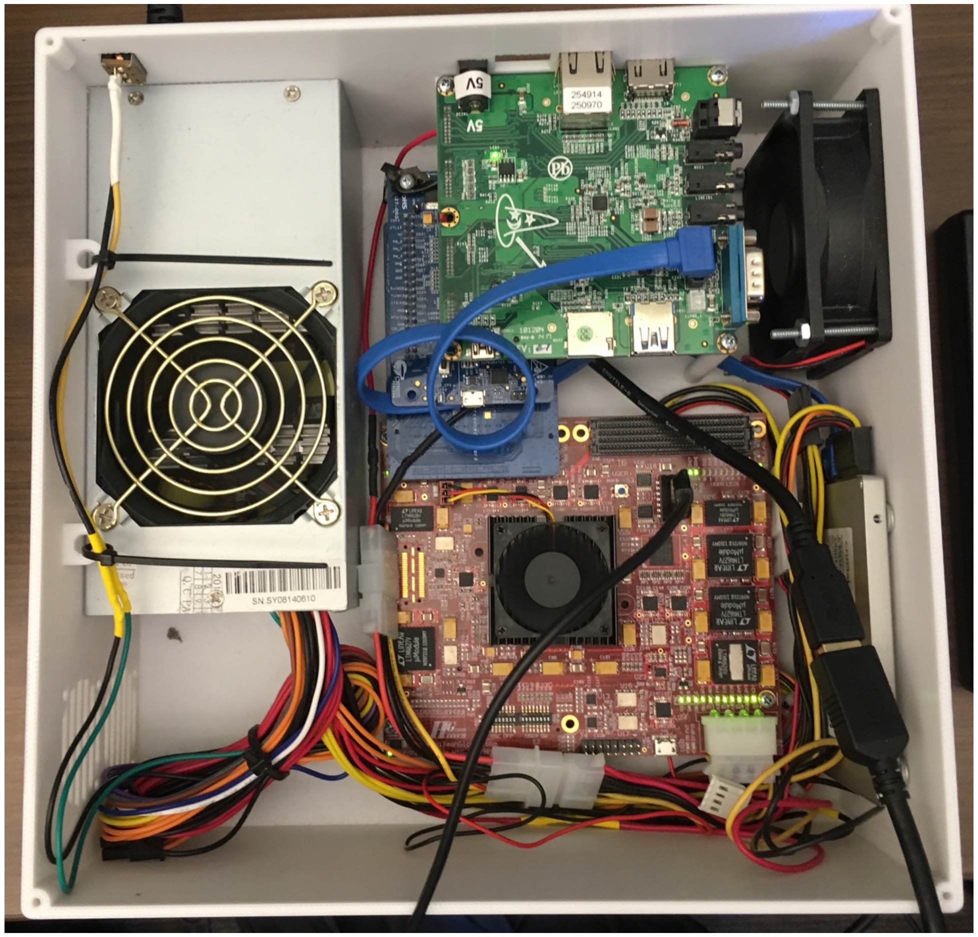
## Acknowledgements

This work was funded in part by the Air Force Research Laboratory, Information Directorate under award number FA8750-16-1-0065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

Visit us @
neuromorphic.eecs.utk.edu

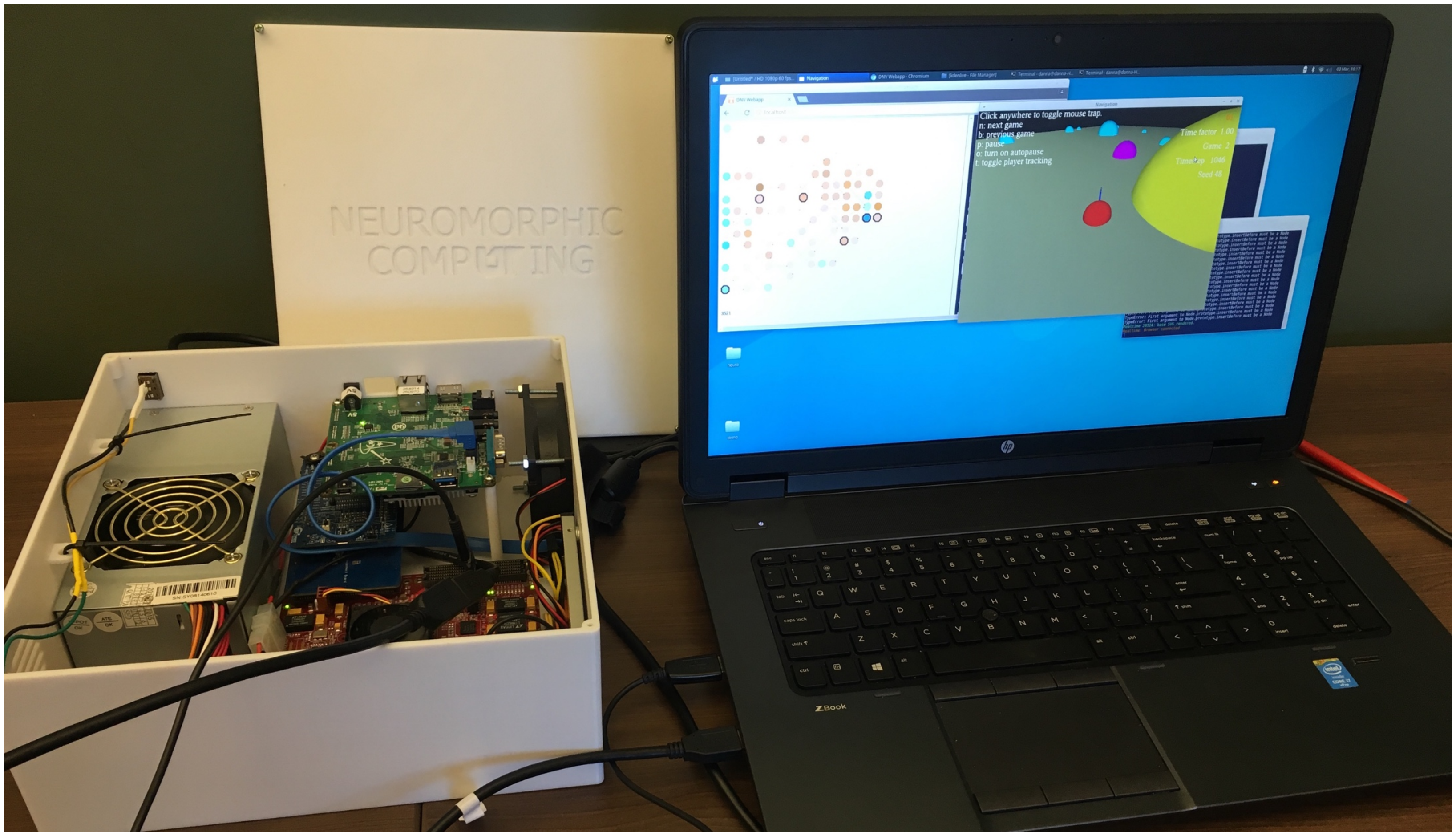
## The Kit Demo:

- DANNA FPGA
- Xilinx Virtex-7 XC7V690T
- Also XC7V2000T
- Cypress FX3 Board
- USB 3.0 to host
- Crappy ARM processor
  (not using in demo)

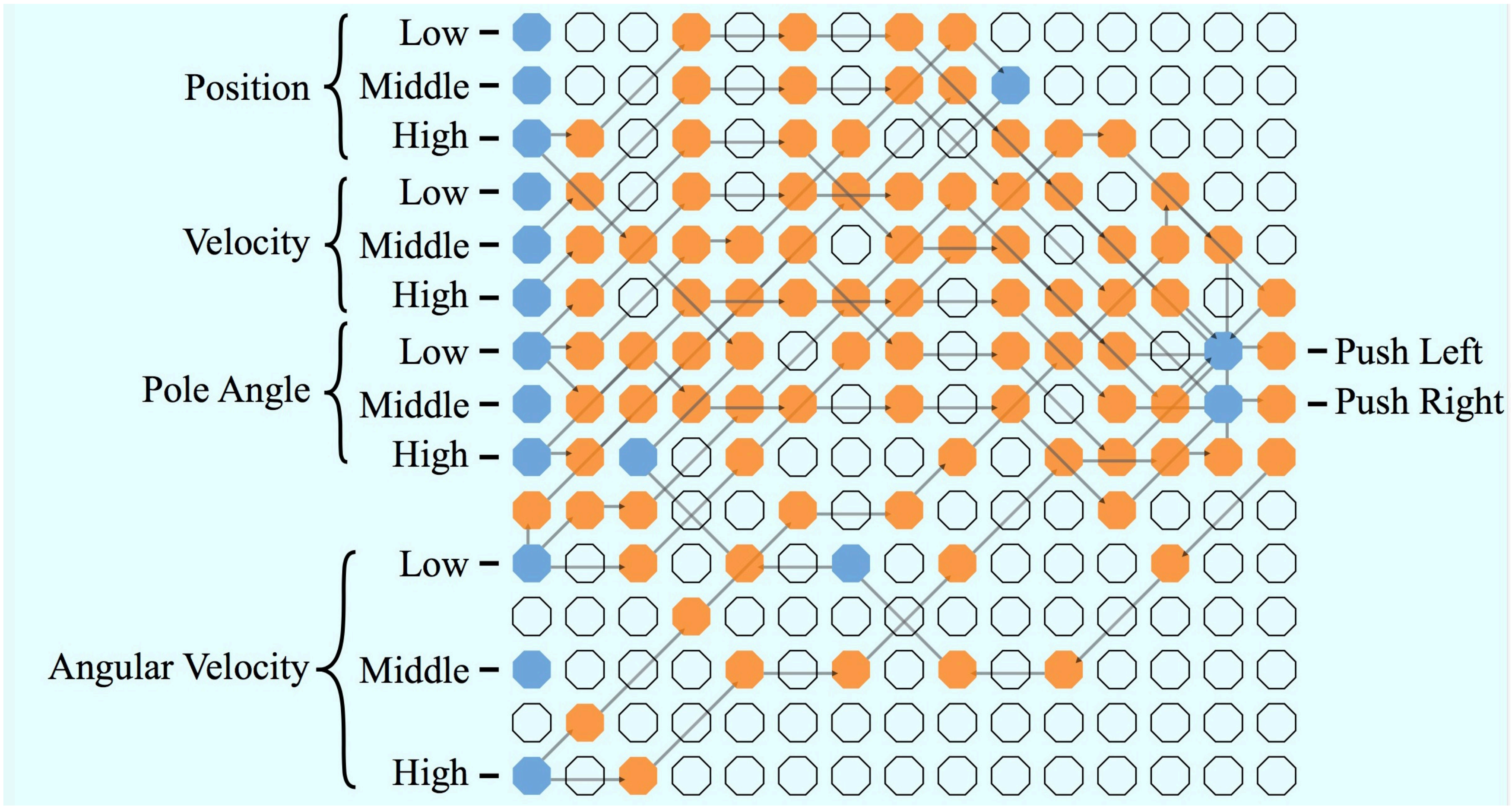


THE UNIVERSITY OF TENNESSEE KNOXVILLE

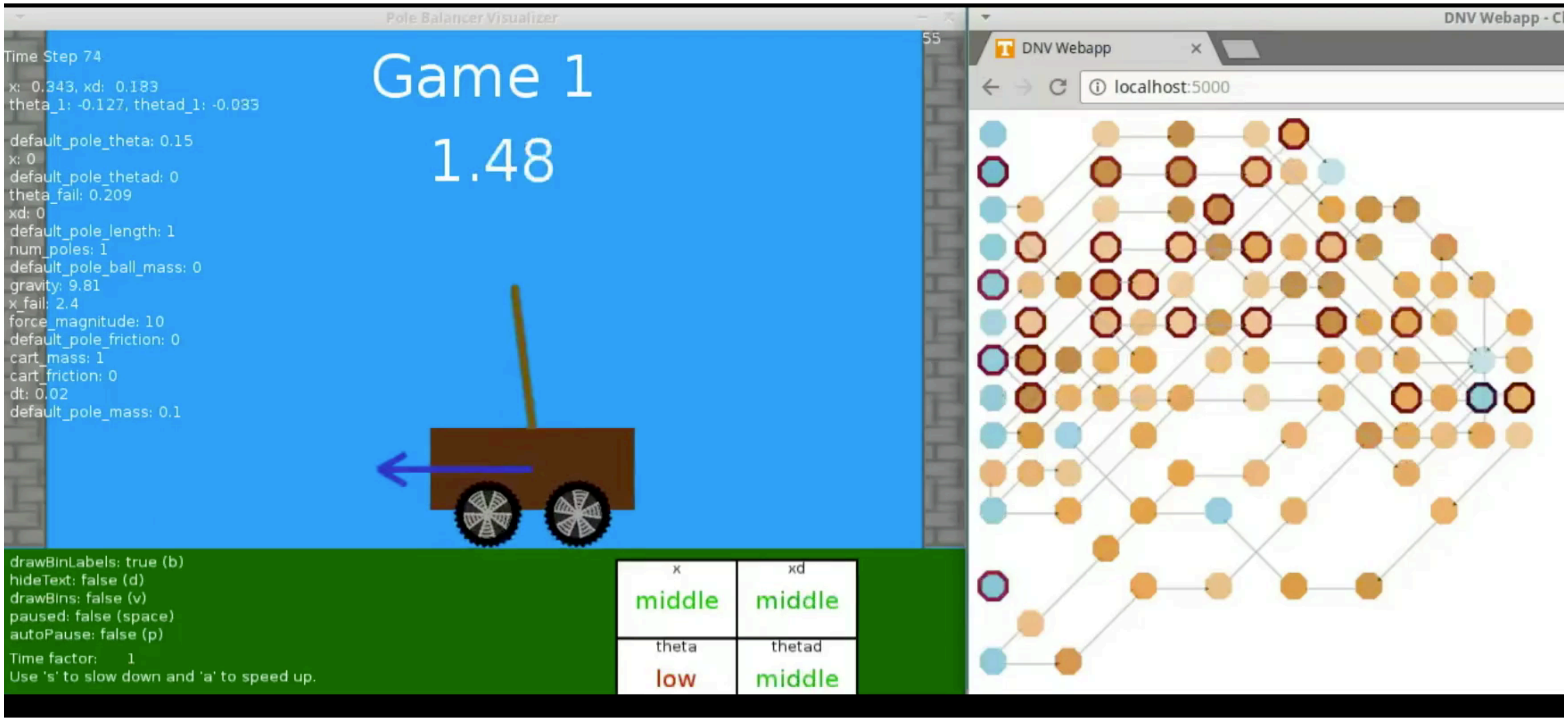
## The Kit Demo:


NEUROMORPHIC
COMPUTING
Click anywhere to toggle mouse trap.
n: next game
b: previous game
p: pause
o: turn on autopause
t: toggle player tracking

# The Kit Demo: The Pole Balancer



# The Kit Demo:



```
Time Step 74
x:  0.343, xd:  0.183
theta_1: -0.127, thetad_1: -0.033

default_pole_theta: 0.15
x: 0
default_pole_thetad: 0
theta_fail: 0.209
xd: 0
default_pole_length: 1
num_poles: 1
default_pole_ball_mass: 0
gravity: 9.81
x_fail: 2.4
force_magnitude: 10
default_pole_friction: 0
cart_mass: 1
cart_friction: 0
dt: 0.02
default_pole_mass: 0.1
```

Game 1

1.48

```
drawBinLabels: true (b)
hideText: false (d)
drawBins: false (v)
paused: false (space)
autoPause: false (p)
Time factor:     1
Use 's' to slow down and 'a' to speed up.
```

| x | xd |
|---|---|
| middle | middle |
| **theta** | **thetad** |
| low | middle |

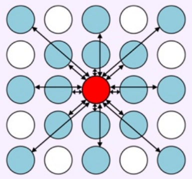
# The Kit Demo

## Introduction

Dynamic Adaptive Neural Network Array (DANNA) is a neuromorphic implementation meant for conventional digital hardware that is currently implemented on field programmable gate array (FPGA) with a VLSI implementation in progress. In this demo, we will demonstrate the FPGA implementation of DANNA running in real-time on at least three applications: pole balancing, one-dimensional navigation with gravity (a game based on the mobile application Flappy Bird), and two-dimensional navigation with obstacle avoidance. We will also demonstrate a visualization of the DANNA neuromorphic implementation running in real-time alongside the simulation of the hardware on the FPGA.

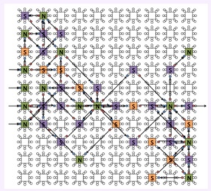
## Danna Element

- A neuromorphic structure which can act as a neuron or a synapse.
- Each element can be configured as either a neuron or as synapse.
- Connection scheme connects elements together to form array.

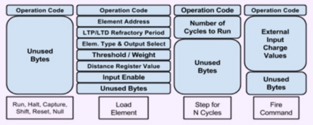


## Danna Network

- Two-dimensional grid of elements.
- Elements can be neurons or synapses.
- 16 nearest neighbor connectivity.
- Programmable synaptic delays.
- FPGA implementation deployed.
- VLSI implementation designed.



## Communication



## Key Features

### Commands

**Run, Step, Halt**
Control the clock that drives the cycles in the array.

**Reset**
Reset the network back to the initial state.

**Fire, Null**
Fire into an input neuron or do nothing for cycle.

**Load Element**
Set the configuration of an element.

**Capture Shift**
Capture and read the internal state of the network.

### Major Capabilities

**Array Status**
The internal status of the array can be determined while the array is running with the capture shift commands.

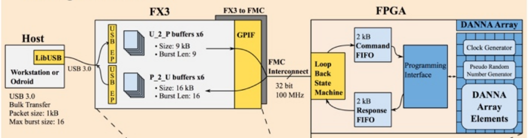
**Compressed Shift and Null**
In order to reduce communication traffic, single commands like step, null, and shift, can be can be repeated multiple times with a single packet.

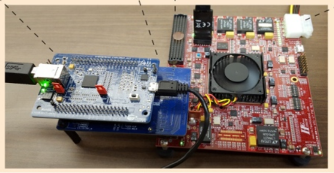
**Element Programmability**
Each element is identical. The load command specifies the type of the element and sets the parameters. Neurons have a firing threshold. Synapses have a weight, distance, and refractory period.

## Hardware

For the DANNA model, we have implemented the model on the Xilinx Virtex-7 XC7V690T (supporting DANNA arrays up to 47 X 47), and XC7V2000T (supporting arrays up to 75 X 75). We have verified the functionality of these implementations with a cycle-level simulator.

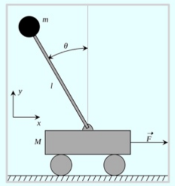
FPGA DANNA is the original implementation of the model. Currently we have DANNA running on Xilinx Virtex-7 models @ 1 million DANNA cycles per second (1 MHz). Communication with the FPGA is implemented with a Cypress FX3 over USB 3.0. Currently we have the 690T model FPGA implementing an array size of 45 by 45 using over 99% of the slices. The 2000T model FPGA is implementing an array size of 70 by 70 using 87% of the slices.



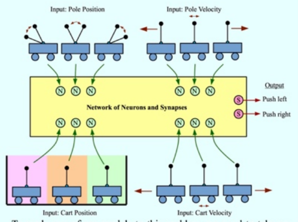


## Applications

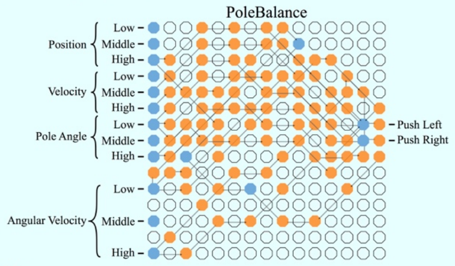
### Pole Balance



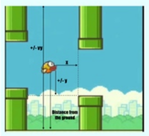
The inverted pendulum is a classic application from control theory. The goal of the system is to apply periodic forces to move the cart left or right, to keep the pole from falling, and to keep the cart from moving beyond its boundaries.

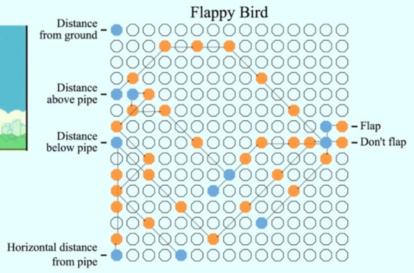


To apply one of our models to this problem, we need to take an instance of the inverted pendulum problem and map it to input pulses. We need to map the output pulses of our model to cart movements. If our model is programmed correctly, it will keep the pole balanced without having the cart go beyond its fixed boundaries.



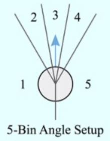
### Flappy Bird

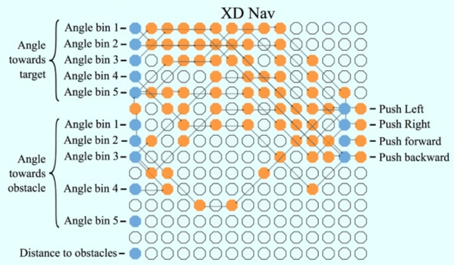




The Flappy Bird cell phone game consists of a bird constantly being forced to the right on screen and gravity is pulling the bird downward. The player controls when the bird flaps to counter the gravitational pull downward. As the bird advances forward, there are barriers with gaps that the bird must make it through. Any collision with a barrier ends the game.

### XD Nav





The player is tasked with navigating to a target in X-dimensional space, where X is 2 or 3. There are obstacles in its path that it must avoid. Inputs to the 2DNav network are based on the angle relative to the direction the player is facing. Each angle is put into 5 bins, each bin corresponds to the angle relative to the player. The network then decides if the player should move left, right, forward, or back.

Visit us @ neuromorphic.eecs.utk.edu

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-16-1-0065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

## The Neuromorphic Group at Tennessee