Workshop in Computer Architecture Education

PYRTL IN EARLY UNDERGRADUATE RESEARCH

DIBA MIRZA, DEEKSHA DANGWAL*, TIMOTHY SHERWOOD
UC SANTA BARBARA

*deeksha@cs.ucsb.edu

Benefits of Early Research

INCREASED CONFIDENCE

[1] L. Barker. Student and faculty perceptions of undergraduate research experiences in computing. Trans. Comput. Educ., 9(1):5:1–5:28, March 2009.
[2] Susan H. Russell, Mary P. Hancock, and James McCullough. The pipeline: Benefits of undergraduate research experiences.Science, 316(5824):548–549, 2007
[3] E. Seymour, A.-B. Hunter, S. L. Laursen, and T. Deantoni. Establishing the benefits of research experiences for undergraduates in the sciences: First findings from a three-year study.ScienceEducation, 88:493–534, 2004

Benefits of Early Research

INCREASED CONFIDENCE

INCREASED INTEREST AND RETENTION IN STEM

[1] L. Barker. Student and faculty perceptions of undergraduate research experiences in computing. Trans. Comput. Educ., 9(1):5:1–5:28, March 2009.
[2] Susan H. Russell, Mary P. Hancock, and James McCullough. The pipeline: Benefits of undergraduate research experiences.Science, 316(5824):548–549, 2007
[3] E. Seymour, A.-B. Hunter, S. L. Laursen, and T. Deantoni. Establishing the benefits of research experiences for undergraduates in the sciences: First findings from a three-year study.ScienceEducation, 88:493–534, 2004

Benefits of Early Research

INCREASED CONFIDENCE

INCREASED INTEREST AND RETENTION IN STEM

INCREASED LIKELIHOOD OF PURSUING GRADUATE STUDIES

[1] L. Barker. Student and faculty perceptions of undergraduate research experiences in computing. Trans. Comput. Educ., 9(1):5:1–5:28, March 2009.
[2] Susan H. Russell, Mary P. Hancock, and James McCullough. The pipeline: Benefits of undergraduate research experiences.Science, 316(5824):548–549, 2007
[3] E. Seymour, A.-B. Hunter, S. L. Laursen, and T. Deantoni. Establishing the benefits of research experiences for undergraduates in the sciences: First findings from a three-year study.ScienceEducation, 88:493–534, 2004

Benefits of Early Research

INCREASED CONFIDENCE

INCREASED INTEREST AND RETENTION IN STEM

INCREASED LIKELIHOOD OF PURSUING GRADUATE STUDIES

ESPECIALLY TRUE FOR UNDER REPRESENTED MINORITIES IN CS

Barriers to Early Research

(in computer architecture)

Barriers to Early Research

(in computer architecture)

LACK OF "HARDWARE THINKING"

Lack of "hardware thinking"

EARLY CS COURSES EMPHASIZE FUNDAMENTALS OF PROGRAMMING, DATA STRUCTURES, AND ALGORITHMS INSTEAD OF DIGITAL DESIGN.

HARDWARE IS INHERENTLY PARALLEL!

Barriers to Early Research

(in computer architecture)

LACK OF "HARDWARE THINKING"

COURSES DO NOT EXPLORE EXCITING OPEN PROBLEMS

Courses do not explore exciting open problems

UNDERGRADUATE COMPUTER ARCHITECTURE COURSES TEACH STUDENTS HOW A COMPUTER WORKS.

FAR FROM THE OPEN RESEARCH PROBLEMS AND INTERESTING APPLICATIONS REDEFINING OUR DISCIPLINE

Barriers to Early Research

(in computer architecture)

LACK OF "HARDWARE THINKING"

COURSES DO NOT EXPLORE EXCITING OPEN PROBLEMS

COMPLEX DESIGN TOOLCHAINS

Complex design toolchains

COMPARED TO SOFTWARE, DESIGNING, TESTING, DEBUGGING, AND PLAYING WITH HARDWARE INVOLVES GOING THROUGH MANY COMPLEX TOOLS.

WE NEED "JUPYTER NOTEBOOKS" FOR ARCHITECTURE

(who have never taken an architecture/digital design course before)

Can early undergraduate students succeed in conducting research in computer architecture?

(who have never taken an architecture/digital design course before)

Can early undergraduate students succeed in conducting research in computer architecture?

- When the problem is well scoped, and students have a support system that sets them up for success

- When tools for research build on top of what the students already know and are comfortable with

Research problem assigned

RESEARCH GOALS

Understand how the choice of neural network hyperparameters in software affect energy and latency at the custom hardware level.

Research problem assigned

RESEARCH GOALS

Understand how the choice of neural network hyperparameters in software affect energy and latency at the custom hardware level.

RESEARCH STEPS

- Design neural networks in software (PyTorch)
- Design hardware blocks in PyRTL
- Collect area and latency numbers from PyRTL

Research problem assigned

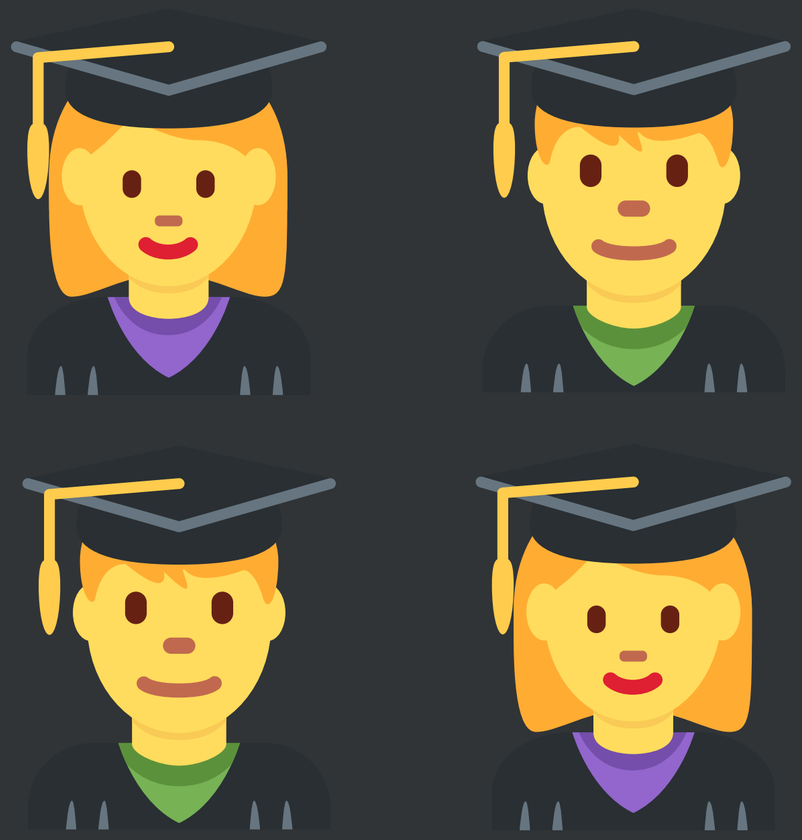
RESEARCH GOALS

Understand how the choice of neural network hyperparameters in software affect energy and latency at the custom hardware level.

RESEARCH STEPS

- Design neural networks in software (PyTorch)
- Design hardware blocks in PyRTL
- Collect area and latency numbers from PyRTL

In effect, come up with rules of thumb to inform neural network design in software to build energy-efficient systems.



CONTEXT

- Four second year undergraduates in computer science and computer engineering
- Early Research Scholars Program
- No prior experience in digital design/computer architecture
- No prior experience in machine learning/neural networks

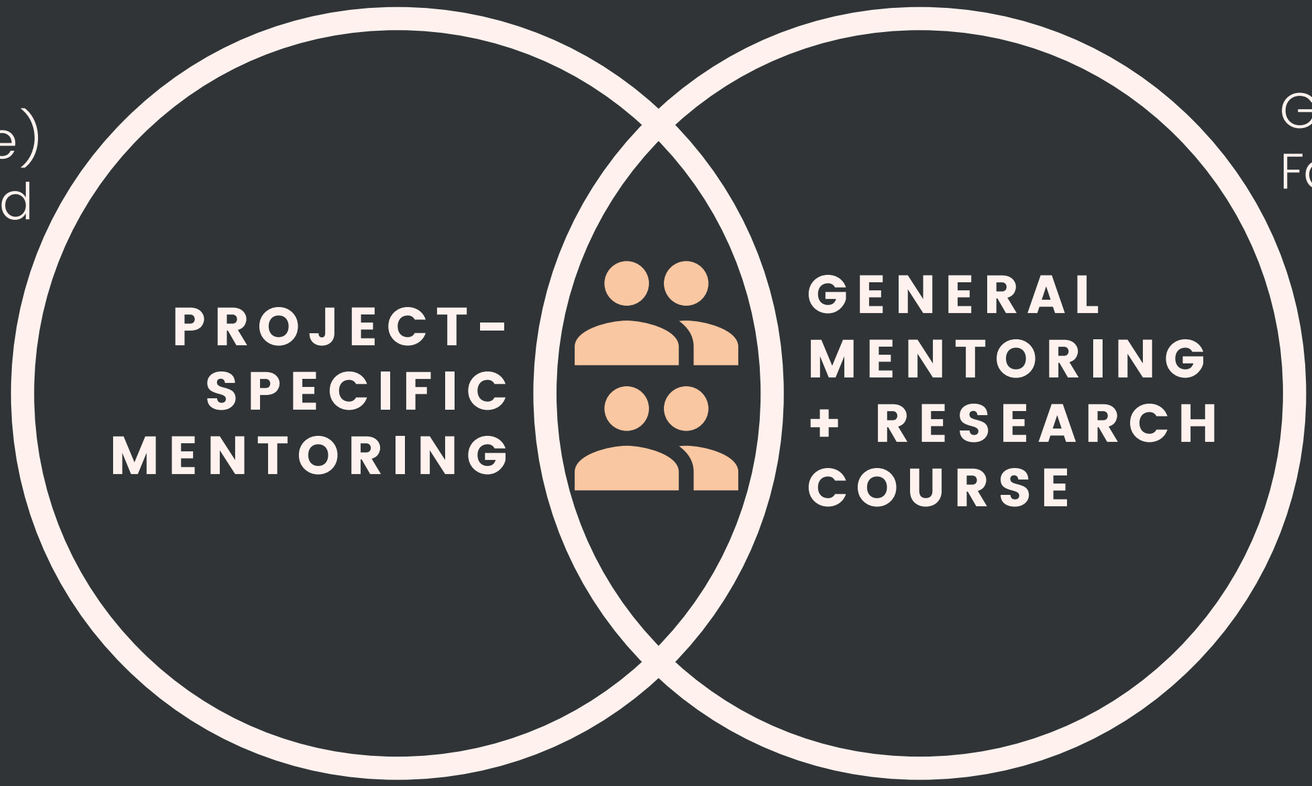
Early Research Scholars Program

ERSP MATCHES STUDENTS TO RESEARCH FACULTY AND A GRADUATE STUDENT MENTOR FOR A YEAR-LONG APPRENTICESHIP

Early Research Scholars Program

ERSP MATCHES STUDENTS TO RESEARCH FACULTY AND A GRADUATE STUDENT MENTOR FOR A YEAR-LONG APPRENTICESHIP

Research team:
Mentor - Deeksha (me)
Faculty - Tim Sherwood



General advisory team:
Faculty - Diba Mirza

Early Research Scholars Program

TIMELINE

FALL 2018

Research methods course
+
literature review, getting started with PyTorch, PyRTL

Early Research Scholars Program

TIMELINE

FALL 2018

Research methods course + literature review, getting started with PyTorch, PyRTL

RESEARCH PROPOSAL

Early Research Scholars Program

TIMELINE

FALL 2018

Research methods course + literature review, getting started with PyTorch, PyRTL

RESEARCH PROPOSAL

WINTER 2019 SPRING 2019

Conducting research, weekly lab meetings, weekly meetings with advisory team

Early Research Scholars Program

TIMELINE

FALL 2018

Research methods course + literature review, getting started with PyTorch, PyRTL

RESEARCH PROPOSAL

WINTER 2019 SPRING 2019

Conducting research, weekly lab meetings, weekly meetings with advisory team

RESEARCH POSTER

✓ When the problem is well scoped, and students have a support system that sets them up for success

- When tools for research build on top of what the students already know and are comfortable with

PyRTL: elaboration through execution in Python

- rapid prototyping of complex digital hardware
- make all hardware decisions explicit and concrete (rather than inferred, as is the case with HLS)
- allow complex hardware design patterns to promote reuse beyond just hardware blocks
- lower the barrier of entry to digital design (for both students and software engineers)

PyRTL: elaboration through execution in Python

- rapid prototyping of complex digital hardware
- make all hardware decisions explicit and concrete (rather than inferred, as is the case with HLS)
- allow complex hardware design patterns to promote reuse beyond just hardware blocks
- lower the barrier of entry to digital design (for both students and software engineers)

PyRTL, a user-friendly hardware design language

PYRTL IS A COLLECTION OF PYTHON CLASSES PROVIDING SIMPLE RTL SPECIFICATION, SIMULATION, SYNTHESIS, TRACING, AND TESTING.

CONCISE

BROADLY UNDERSTANDABLE

FAMILIAR SYNTAX

PyRTL, a user-friendly hardware design language

PYRTL IS A COLLECTION OF PYTHON CLASSES PROVIDING SIMPLE RTL SPECIFICATION, SIMULATION, SYNTHESIS, TRACING, AND TESTING.

CONCISE

BROADLY UNDERSTANDABLE

FAMILIAR SYNTAX

PyRTL treats hardware design like a software problem, it leverages what students have learned in their early classes.

- Recursion
- Object oriented design
- Hardware comprehension
- List slicing
- Dynamic types

Programming in PyRTL

ONE BIT ADDER

```
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout
```

ONE BIT ADDER

```python
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout
```

RIPPLE CARRY ADDER

```python
def ripple_add(a, b, cin=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, cout = one_bit_add(a, b, cin)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], cin)
        msbits, cout = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, cout
```

ONE BIT ADDER

```python
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout
```

RIPPLE CARRY ADDER

```python
def ripple_add(a, b, cin=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, cout = one_bit_add(a, b, cin)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], cin)
        msbits, cout = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, cout
```

INSTANTIATE

```python
# instantiate an adder into a 3-bit counter
counter = pyrtl.Register(bitwidth=3, name='counter')
sum, cout = ripple_add(counter, pyrtl.Const("1'b1"))
counter.next <<= sum
```

```python
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout

def ripple_add(a, b, cin=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, cout = one_bit_add(a, b, cin)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], cin)
        msbits, cout = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, cout

# instantiate an adder into a 3-bit counter
counter = pyrtl.Register(bitwidth=3, name='counter')
sum, cout = ripple_add(counter, pyrtl.Const("1'b1"))
counter.next <<= sum

# simulate the instantiated design for 15 cycles
sim_trace = pyrtl.SimulationTrace()
sim = pyrtl.Simulation(tracer=sim_trace)
for cycle in range(15):
    sim.step({})
sim_trace.render_trace()
```

ONE BIT ADDER

RIPPLE CARRY ADDER

INSTANTIATE

SIMULATE

```python
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout

def ripple_add(a, b, cin=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, cout = one_bit_add(a, b, cin)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], cin)
        msbits, cout = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, cout

# instantiate an adder into a 3-bit counter
counter = pyrtl.Register(bitwidth=3, name='counter')
sum, cout = ripple_add(counter, pyrtl.Const("1'b1"))
counter.next <<= sum

# simulate the instantiated design for 15 cycles
sim_trace = pyrtl.SimulationTrace()
sim = pyrtl.Simulation(tracer=sim_trace)
for cycle in range(15):
    sim.step({})
sim_trace.render_trace()
```

Python assert

Python built-in function

Python recursion
Python bit slicing

```python
def one_bit_add(a, b, cin):
    assert len(a) == len(b) == 1  # len returns the bitwidth
    sum = a ^ b ^ cin  # operators on WireVectors build the hardware
    cout = a & b | a & cin | b & cin
    return sum, cout

def ripple_add(a, b, cin=0):
    a, b = pyrtl.match_bitwidth(a, b)
    if len(a) == 1:
        sumbits, cout = one_bit_add(a, b, cin)
    else:
        lsbit, ripplecarry = one_bit_add(a[0], b[0], cin)
        msbits, cout = ripple_add(a[1:], b[1:], ripplecarry)
        sumbits = pyrtl.concat(msbits, lsbit)
    return sumbits, cout

# instantiate an adder into a 3-bit counter
counter = pyrtl.Register(bitwidth=3, name='counter')
sum, cout = ripple_add(counter, pyrtl.Const("1'b1"))
counter.next <<= sum

# simulate the instantiated design for 15 cycles
sim_trace = pyrtl.SimulationTrace()
sim = pyrtl.Simulation(tracer=sim_trace)
for cycle in range(15):
    sim.step({})
sim_trace.render_trace()
```

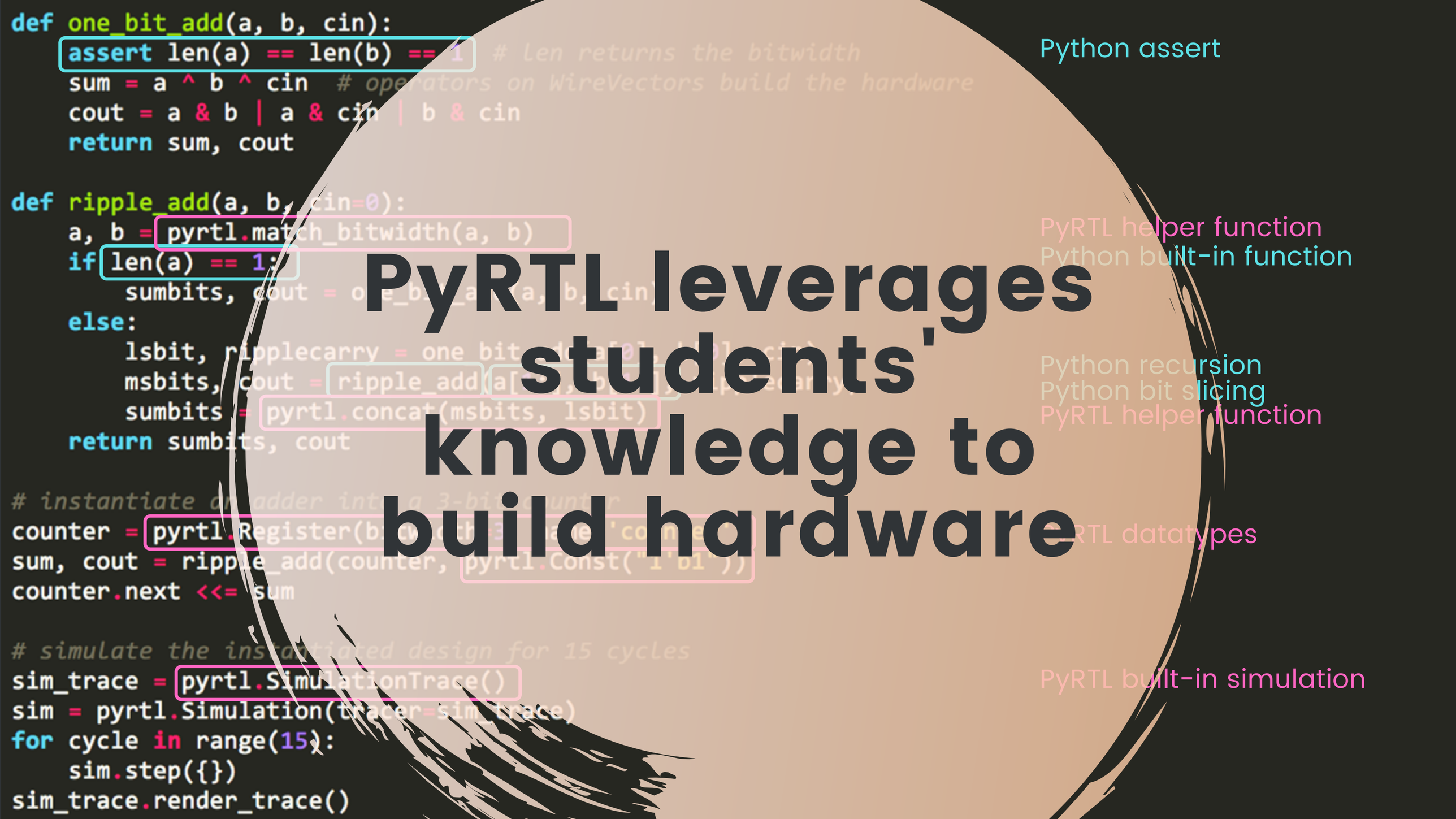
Python assert

PyRTL helper function
Python built-in function

Python recursion
Python bit slicing
PyRTL helper function

PyRTL datatypes

PyRTL built-in simulation

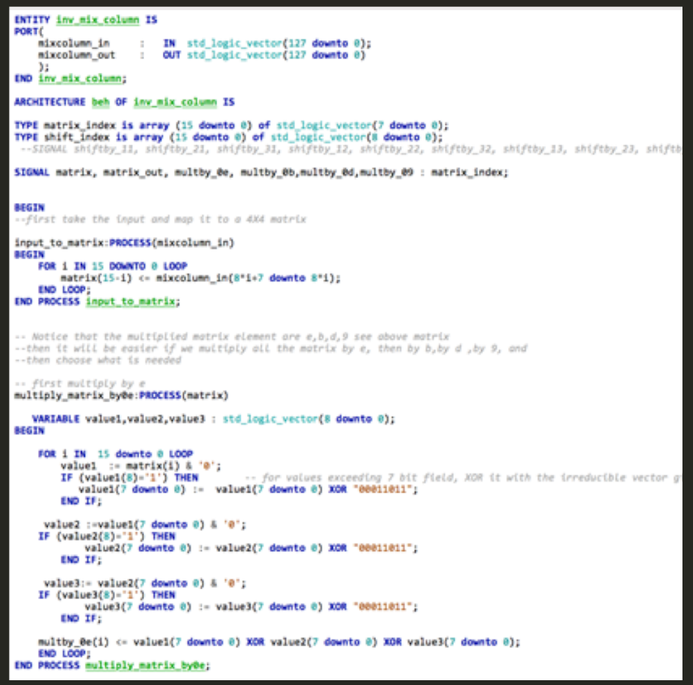


PyRTL leverages students' knowledge to build hardware

```
def inv_mix_columns(in_vector):
    igm_divisor = [14, 11, 13, 9]
    def inv_mix_single(index):
        mult_items = [inv_galois_mult(a[mod_add(index, loc, 4)], mult_table)
                      for loc, mult_table in enumerate(igm_divisor)]
        return mult_items[0] ^ mult_items[1] ^ mult_items[2] ^ mult_items[3]

    inverted = [inv_mix_single(index) for index in range(15)]
    return pyrtl.concat(*inverted)
```

Hardware comprehension w/ list comprehension

```python
def inv_mix_columns(in_vector):
    igm_divisor = [14, 11, 13, 9]
    def inv_mix_single(index):
        mult_items = [inv_galois_mult(a[mod_add(index, loc, 4)], mult_table)
                      for loc, mult_table in enumerate(igm_divisor)]
        return mult_items[0] ^ mult_items[1] ^ mult_items[2] ^ mult_items[3]

    inverted = [inv_mix_single(index) for index in range(15)]
    return pyrtl.concat(*inverted)
```

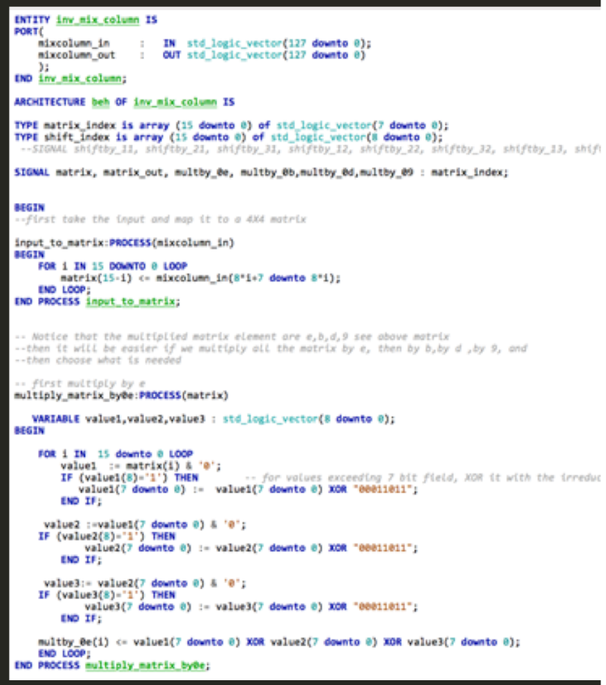


```
ENTITY inv_mix_column IS
PORT(
    mixcolumn_in     :   IN  std_logic_vector(127 downto 0);
    mixcolumn_out    :   OUT std_logic_vector(127 downto 0)
    );
END inv_mix_column;

ARCHITECTURE beh OF inv_mix_column IS

TYPE matrix_index is array (15 downto 0) of std_logic_vector(7 downto 0);
TYPE shift_index is array (15 downto 0) of std_logic_vector(8 downto 0);
 --SIGNAL shiftby_11, shiftby_21, shiftby_31, shiftby_12, shiftby_22, shiftby_32, shiftby_13, shiftby_23, shiftb

SIGNAL matrix, matrix_out, multby_0e, multby_0b,multby_0d,multby_09 : matrix_index;


BEGIN
--first take the input and map it to a 4X4 matrix

input_to_matrix:PROCESS(mixcolumn_in)
BEGIN
    FOR i IN 15 DOWNTO 0 LOOP
        matrix(15-i) <= mixcolumn_in(8*i+7 downto 8*i);
    END LOOP;
END PROCESS input_to_matrix;


-- Notice that the multiplied matrix element are e,b,d,9 see above matrix
--then it will be easier if we multiply all the matrix by e, then by b,by d ,by 9, and
--then choose what is needed

-- first multiply by e
multiply_matrix_by0e:PROCESS(matrix)

   VARIABLE value1,value2,value3 : std_logic_vector(8 downto 0);
BEGIN

    FOR i IN  15 downto 0 LOOP
        value1  := matrix(i) & '0';
        IF (value1(8)='1') THEN      -- for values exceeding 7 bit field, XOR it with the irreducible vector g
            value1(7 downto 0) :=  value1(7 downto 0) XOR "00011011";
        END IF;

     value2 :=value1(7 downto 0) & '0';
    IF (value2(8)='1') THEN
            value2(7 downto 0) := value2(7 downto 0) XOR "00011011";
        END IF;

     value3:= value2(7 downto 0) & '0';
    IF (value3(8)='1') THEN
            value3(7 downto 0) := value3(7 downto 0) XOR "00011011";
        END IF;

    multby_0e(i) <= value1(7 downto 0) XOR value2(7 downto 0) XOR value3(7 downto 0);
    END LOOP;
END PROCESS multiply_matrix_by0e;
```

Hardware comprehension w/ list comprehension

```python
def inv_mix_columns(in_vector):
    igm_divisor = [14, 11, 13, 9]
    def inv_mix_single(index):
        mult_items = [inv_galois_mult(a[mod_add(index, loc, 4)], mult_table)
                      for loc, mult_table in enumerate(igm_divisor)]
        return mult_items[0] ^ mult_items[1] ^ mult_items[2] ^ mult_items[3]

    inverted = [inv_mix_single(index) for index in range(15)]
    return pyrtl.concat(*inverted)
```

Hardware comprehension w/ list comprehension



```
ENTITY inv_mix_column IS
PORT(
    mixcolumn_in     :   IN  std_logic_vector(127 downto 0);
    mixcolumn_out    :   OUT std_logic_vector(127 downto 0)
    );
END inv_mix_column;

ARCHITECTURE beh OF inv_mix_column IS

TYPE matrix_index is array (15 downto 0) of std_logic_vector(7 downto 0);
TYPE shift_index is array (15 downto 0) of std_logic_vector(0 downto 0);
--SIGNAL shiftby_11, shiftby_21, shiftby_31, shiftby_12, shiftby_22, shiftby_32, shiftby_13, shift

SIGNAL matrix, matrix_out, multby_0e, multby_0b,multby_0d,multby_09 : matrix_index;


BEGIN
--first take the input and map it to a 4X4 matrix

input_to_matrix:PROCESS(mixcolumn_in)
BEGIN
    FOR i IN 15 DOWNTO 0 LOOP
        matrix(15-i) <= mixcolumn_in(8*i+7 downto 8*i);
    END LOOP;
END PROCESS input_to_matrix;


-- Notice that the multiplied matrix element are e,b,d,9 see above matrix
--then it will be easier if we multiply all the matrix by e, then by b,by d ,by 9, and
--then choose what is needed

-- first multiply by e
multiply_matrix_by0e:PROCESS(matrix)

   VARIABLE value1,value2,value3 : std_logic_vector(8 downto 0);
BEGIN

    FOR i IN  15 downto 0 LOOP
        value1  := matrix(i) & '0';
        IF (value1(8)='1') THEN      -- for values exceeding 7 bit field, XOR it with the irreduc
            value1(7 downto 0) :=  value1(7 downto 0) XOR "00011011";
        END IF;

     value2 :=value1(7 downto 0) & '0';
    IF (value2(8)='1') THEN
            value2(7 downto 0) := value2(7 downto 0) XOR "00011011";
        END IF;

     value3:= value2(7 downto 0) & '0';
    IF (value3(8)='1') THEN
            value3(7 downto 0) := value3(7 downto 0) XOR "00011011";
        END IF;

    multby_0e(i) <= value1(7 downto 0) XOR value2(7 downto 0) XOR value3(7 downto 0);
    END LOOP;
END PROCESS multiply_matrix_by0e;
```

```
    VARIABLE value1, value2, value3 : std_logic_vector(8 downto 0);
BEGIN

    FOR i IN  15 downto 0 LOOP
        value1  := matrix(i) & '0';
        IF (value1(8)='1') THEN      -- for values exceeding 7 bit field, XOR it with the irreductible vec
            value1(7 downto 0) :=  value1(7 downto 0) XOR "00011011";
        END IF;

     value2 :=value1(7 downto 0) & '0';
    IF (value2(8)='1') THEN
            value2(7 downto 0) := value2(7 downto 0) XOR "00011011";
        END IF;

     value3 :=  value2(7 downto 0) & '0';
    IF (value3(8)='1') THEN
            value3(7 downto 0) := value3(7 downto 0) XOR "00011011";
        END IF;

    multby_0b(i) <= matrix(i) XOR value1(7 downto 0) XOR value3(7 downto 0);
    END LOOP;
END PROCESS multiply_matrix_by0b;

- first multiply by 0d
ultiply_matrix_by0d:PROCESS(matrix)

    VARIABLE value1, value2, value3 : std_logic_vector(8 downto 0);
BEGIN

    FOR i IN  15 downto 0 LOOP
        value1  := matrix(i) & '0';
        IF (value1(8)='1') THEN      -- for values exceeding 7 bit field, XOR it with the irreductible vec
            value1(7 downto 0) :=  value1(7 downto 0) XOR "00011011";
        END IF;

     value2 :=value1(7 downto 0) & '0';
    IF (value2(8)='1') THEN
            value2(7 downto 0) := value2(7 downto 0) XOR "00011011";
        END IF;

     value3 :=  value2(7 downto 0) & '0';
    IF (value3(8)='1') THEN
            value3(7 downto 0) := value3(7 downto 0) XOR "00011011";
        END IF;

    multby_0d(i) <= matrix(i) XOR value2(7 downto 0) XOR value3(7 downto 0);
    END LOOP;
END PROCESS multiply_matrix_by0d;

- first multiply by 9
ultiply_matrix_by09:PROCESS(matrix)

    VARIABLE value1, value2, value3 : std_logic_vector(8 downto 0);
BEGIN

    FOR i IN  15 downto 0 LOOP
        value1  := matrix(i) & '0';
        IF (value1(8)='1') THEN      -- for values exceeding 7 bit field, XOR it with the irreductible vec
            value1(7 downto 0) :=  value1(7 downto 0) XOR "00011011";
        END IF;

     value2 :=value1(7 downto 0) & '0';
    IF (value2(8)='1') THEN
            value2(7 downto 0) := value2(7 downto 0) XOR "00011011";
        END IF;

     value3:= value2(7 downto 0) & '0';
    IF (value3(8)='1') THEN
            value3(7 downto 0) := value3(7 downto 0) XOR "00011011";
        END IF;

    multby_09(i) <= matrix(i) XOR value3(7 downto 0);
    END LOOP;
END PROCESS multiply_matrix_by09;
```

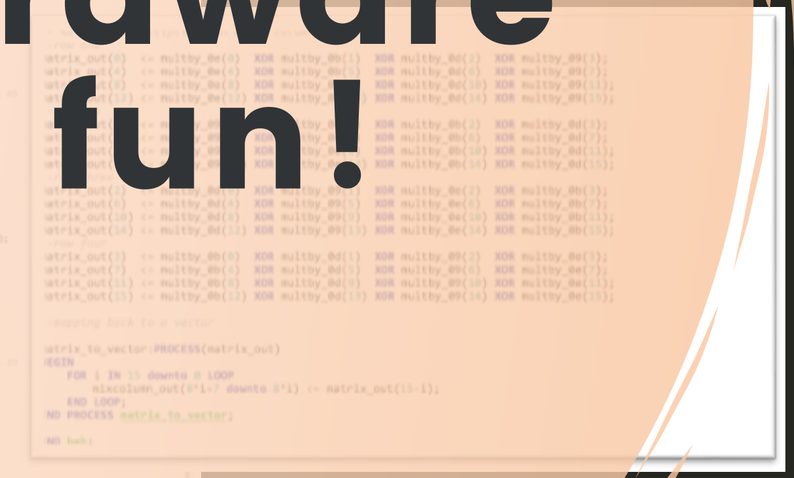
```python
def inv_mix_columns(in_vector):
    igm_divisor = [14, 11, 13, 9]
    def inv_mix_single(index):
        mult_items = [inv_galois_mult(a[mod_add(index, loc, 4)], mult_table)
                      for loc, mult_table in enumerate(igm_divisor)]
        return mult_items[0] ^ mult_items[1] ^ mult_items[2] ^ mult_items[3]

    inverted = [inv_mix_single(index) for index in range(15)]
    return pyrtl.concat(*inverted)
```

Hardware comprehension w/ list comprehension

```
def inv_mix_columns(in_vector):
    igm_divisor = [14, 11, 13, 9]
    def inv_mix_single(index):
        mult_items = [inv_galois_mult(a[mod_add(index, loc, 4)], mult_table)
                      for loc, mult_table in enumerate(igm_divisor)]
        return mult_items[0] ^ mult_items[1] ^ mult_items[2] ^ mult_items[3]

    inverted = [inv_mix_single(index) for index in range(15)]
    return pyrtl.concat(*inv
```

Hardware comprehension

PyRTL is concise and tries to make hardware design fun!



- When the problem is well scoped, and students have a support system that sets them up for success
- When tools for research build on top of what the students already know and are comfortable with

Research problem assigned

RESEARCH GOALS

Understand how the choice of neural network hyperparameters in software affect energy and latency at the custom hardware level.

RESEARCH STEPS

- Design neural networks in software (PyTorch)
- Design hardware blocks in PyRTL
- Collect area and latency numbers from PyRTL

In effect, come up with rules of thumb to inform neural network design in software to build energy-efficient systems.

RESEARCH OUTCOME

- a **hardware design pattern**, **PyRTLMatrix class**, to concisely instantiate neural network primitives in hardware

- instantiate designs with varying parameters and study their effect of on area and latency

Lessons learned

- more online material to get started
- PyRTL errors vs. conceptual errors
- keeping up with undergraduate rhythm:
  - Teams, ERSP advisory team
  - Overheads
- Personally, enriching mentoring experience

[most rewarding part of research]

The moment when we had collected data from a fully functioning model, and were able to correctly interpret it...

[most rewarding part of research]

I had gone into this project with almost no knowledge about any of the related fields. It was incredibly rewarding to analyze and develop new knowledge from something we had built.

[most challenging experience]

Encountering errors generated by backend PyRTL code, which were mostly based on our misunderstanding of how hardware worked on the lowest level.

Working through several levels of highly abstracted code to pinpoint source of error.

[thoughts on research]

It's definitely made me feel more capable of diving into problems I've never experienced before and learning advanced skills on the fly...

[thoughts on research]

I think research has taught me that things take time. Several times during the project, we have hit stopping blocks. I think by working through them we got better but it still takes time...

[thoughts on research]

I think research is more about motivation than knowledge. There is not always one answer to a problem. We can usually understand what the answer to a problem should be, but sometimes it is outside our expectations. It's made me realize that there are a lot of problems that don't have straightforward answers or ways of getting those answers, which is a little discouraging but also very exciting.


THANKYOU
NSF