Pytorch to Amaranth Communication

Hardware System Design

Spring, 2023

Schedule

- 5/24 (today) Pytorch ~ Amaranth communication
- 5/31 zero-skipping hardware + Q&A for final
- 6/5 Final exam
- 6/7 Project Q&A
- Online claim about final on 6/12, online project Q&A on 6/14
- Project detail will be announced on ETL later

Usability of Amaranth simulator

Amaranth simulator is declared as a Python class

- allowing utilization in various way

In our implementation, Amaranth simulator is declared as attribute of another class

```
class ComunicationSimulator():
  def __init__(self, width=32, num_bits = 8):
      self.output = 0

      self.width = width
      self.num_bits = num_bits
      signed = True
      cnt_bits = 5

      self.dut = PEStack(self.num_bits, self.width,
                         cnt_bits=cnt_bits, signed=signed)
      self.dut = ResetInserter(self.dut.in_rst)(self.dut)

      # make amaranth simulator as attribute of our simulator
      self.sim = Simulator(self.dut)
      self.sim.add_clock(1e-6)

      self.i_stack = []
      self.j_stack = []
      self.count = 0
```

Communication method

```python
def test_case(self, dut, in_a, in_b, in_init):
    yield dut.in_a.eq(in_a)
    yield dut.in_b.eq(in_b)
    yield dut.in_init.eq(in_init)
    yield
    out_data = yield dut.out_d
    return out_data


def bench(self):
    # initialize
    yield from self.test_case(self.dut, 0, 0, self.count)
    # feed
    for i in range(self.count):
        yield from self.test_case(self.dut, self.i_stack[i], self.j_stack[i],0)
    # get output
    self.output = yield from self.test_case(self.dut, 0, 0, 0)
```

Amaranth simulator functions are declared in our simulator class

During tiling operation, call simulator function
Progress simulation and get output from hardware simulation

```python
def mmul_tiling(matA, matB, t,simulator):
    a, c = matA.size()
    _, b = matB.size()
    matC = torch.zeros(a, b).type(torch.int8)

    if simulator is not None:
      for j in range((b + t - 1)//t):
        for i in range((a + t - 1)//t):
          ##  TODO  ##
          # Hint: use simulator.set_input
          simulator.sim.add_sync_process(simulator.bench)
          simulator.sim.run()
          tileC = matC[i*t:(i+1)*t, j*t:(j+1)*t]
          tileC += simulator.output
```

Implementation detail

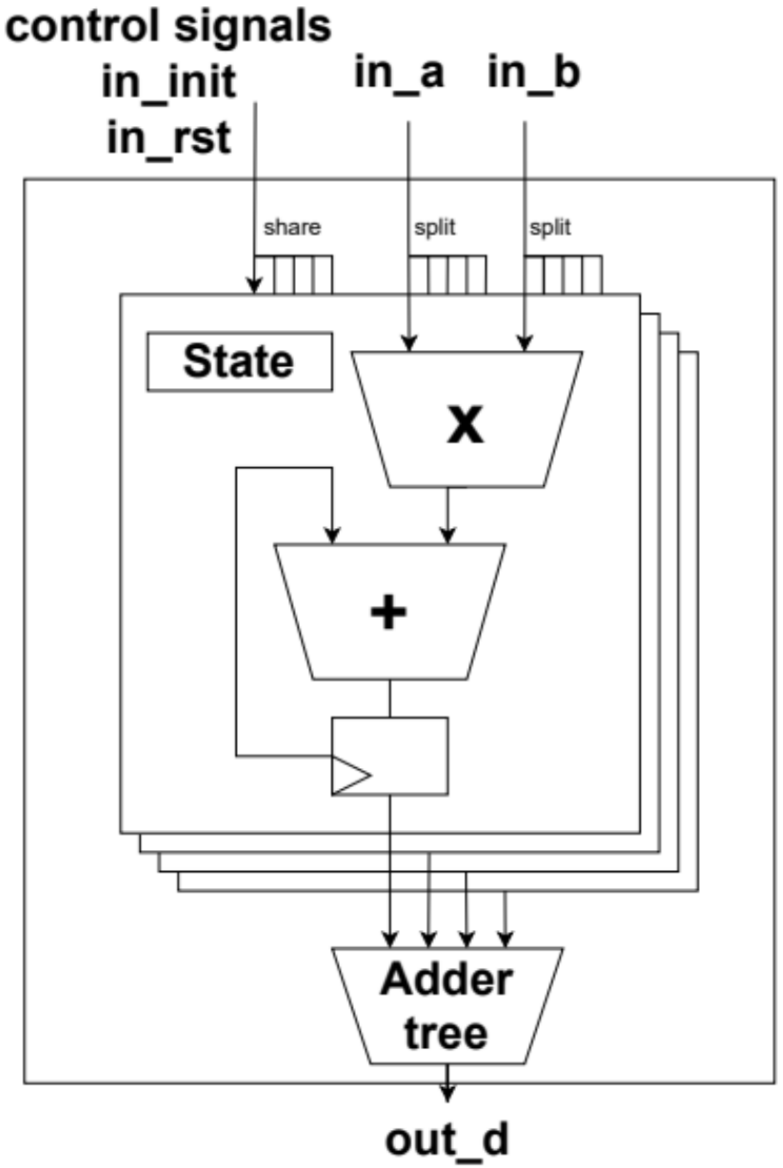
Use PEstack hardware composed of 4 Pes
PEstack get 32bits input and split into 8bits
Each PE has int8 input and output

By calling the set_input function, save inputs for hardware as attribute of simulator

```
def set_input(self, input_a, input_b):
    self.i_stack = []
    self.j_stack = []
    self.count = len(input_a)//4

    #input_a and input_b are lists of tile size tensor
    for i in range(self.count):
        tmp = 0
        for l in range(self.width // self.num_bits):
          if int(input_a[i*4 + l].item())>=0:
            tmp = (tmp << self.num_bits) +\
                  int(input_a[i*4 + l].item())
          else:
            tmp = (tmp << self.num_bits) +\
                  int(2**self.num_bits +input_a[i*4 + l].item())
        self.i_stack.append(tmp)

        tmp = 0
        for l in range(self.width // self.num_bits):
          if int(input_b[i*4 + l].item())>=0:
            tmp = (tmp << self.num_bits) +\
                  int(input_b[i*4 + l].item())
          else:
            tmp = (tmp << self.num_bits) +\
                  int(2**self.num_bits +input_b[i*4 + l].item())
        self.j_stack.append(tmp)
```



TODO

```python
def mmul_tiling(matA, matB, t,simulator):
    a, c = matA.size()
    _, b = matB.size()
    matC = torch.zeros(a, b).type(torch.int8)

    if simulator is not None:
      for j in range((b + t - 1)//t):
        for i in range((a + t - 1)//t):
          ##  TODO  ##
          # Hint: use simulator.set_input
          simulator.sim.add_sync_process(simulator.bench)
          simulator.sim.run()
          tileC = matC[i*t:(i+1)*t, j*t:(j+1)*t]
          tileC += simulator.output
```

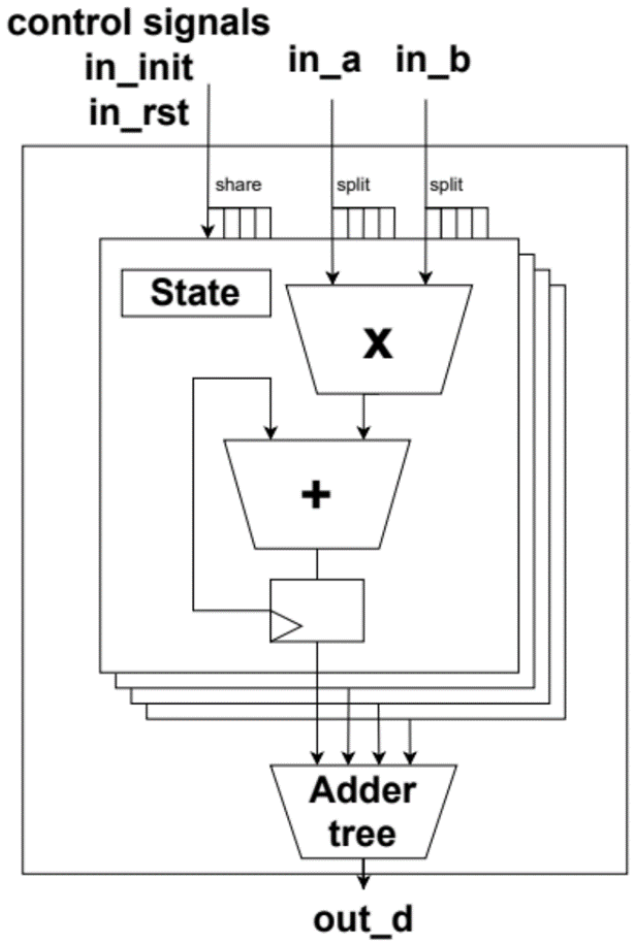
Implement code to
get input values in tiling function

```python
def set_input(self, input_a, input_b):
    self.i_stack = []
    self.j_stack = []
    self.count = len(input_a)//4

    #input_a and input_b are lists of tile size tensor
    for i in range(self.count):
        tmp = 0
        for l in range(self.width // self.num_bits):
          if int(input_a[i*4 + l].item())>=0:
            tmp = (tmp << self.num_bits) +\
                  int(input_a[i*4 + l].item())
          else:
            tmp = (tmp << self.num_bits) +\
                  int(2**self.num_bits +input_a[i*4 + l].item())
        self.i_stack.append(tmp)

        tmp = 0
        for l in range(self.width // self.num_bits):
          if int(input_b[i*4 + l].item())>=0:
            tmp = (tmp << self.num_bits) +\
                  int(input_b[i*4 + l].item())
          else:
            tmp = (tmp << self.num_bits) +\
                  int(2**self.num_bits +input_b[i*4 + l].item())
        self.j_stack.append(tmp)
```

Setting values to
input for hardware



Get output
from simulation

```python
def test_case(self, dut, in_a, in_b, in_init):
    yield dut.in_a.eq(in_a)
    yield dut.in_b.eq(in_b)
    yield dut.in_init.eq(in_init)
    yield
    out_data = yield dut.out_d
    return out_data


def bench(self):
    # initialize
    yield from self.test_case(self.dut, 0, 0, self.count)
    # feed
    for i in range(self.count):
        yield from self.test_case(self.dut, self.i_stack[i], self.j_stack[i],0)
    # get output
    self.output = yield from self.test_case(self.dut, 0, 0, 0)
```

Run hardware
in simulation

For project?

Project hardware has memory and instruction sets

Various functions for simulation are needed
(set Memory of hardware, instruction setting, etc.)

Explanation of project hardware and skeleton code
is uploaded in eTL week 8 module