컴퓨터학과 석사과정
2023020959 유종상

Report : Hardware Accelerator for GEMM with Chisel

1) Introduce GEMM

GEMM은 General Matrix Multiply의 약자로 선형 대수학에서 일반적으로 사용되는 연산이다. 이는 본질적으로 두 개의 행렬을 함께 곱하여 세 번째 행렬을 얻는 것이다 (*<Eq.1>* 참조). 해당 연산은 여러 도메인에서 광범위하게 응용된다.

$$\begin{vmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{vmatrix} * \begin{vmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{vmatrix} = \begin{vmatrix} c_{0,0} & c_{0,1} & c_{0,2} \\ c_{1,0} & c_{1,1} & c_{1,2} \\ c_{2,0} & c_{2,1} & c_{2,2} \end{vmatrix}$$

<Eq.1>

하지만, 이러한 GEMM 연산의 시간 복잡도는 O(n^3)이다. 굉장히 계산 집약적이며 큰 행렬의 경우 계산하는 데 상당한 시간이 걸릴 수 있다. 본 Report에서는 이를 해결하기 위해 Chisel로 Hardware Accelerator를 구현한다.

2) Why Chisel?

Chisel은 종래의 Verilog와 VHDL의 대안으로 제시되는 HDL이다. 기존의 HDL보다 아래와 같은 장점이 있다.

1. High-Level language
   Chisel은 종래의 HDL보다 High-level language이다. Chisel을 이용하면 RTL엔지니어는 보다 쉽게 코드를 디버깅하고 유지보수 할 수 있다.

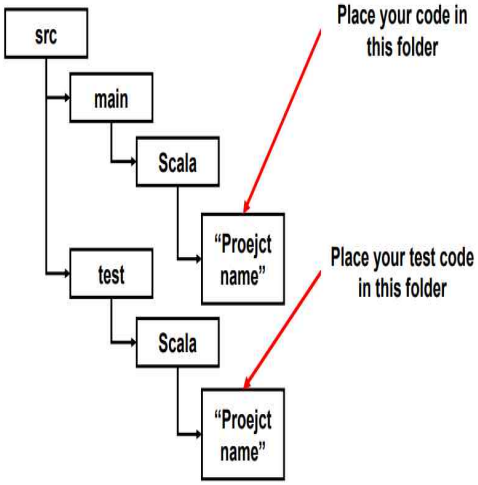
2. Flexible and modular design approach
   Chisel은 modular approach에 유리하다. Parameterizable components를 이용하여 재사용/확장할 수 있는 코드를 작성할 수 있다.

3. Modern software development tools
   기존의 HDL은 현대적인 SW 개발 툴을 사용하는 것을 상정하지 않고 개발되었다. 이는 Git과 같은 협업도구나 디버거 사용의 제한을 의미한다. 하지만, Chisel은 Scala기반 HDL이므로 이점에서 강점을 가진다.

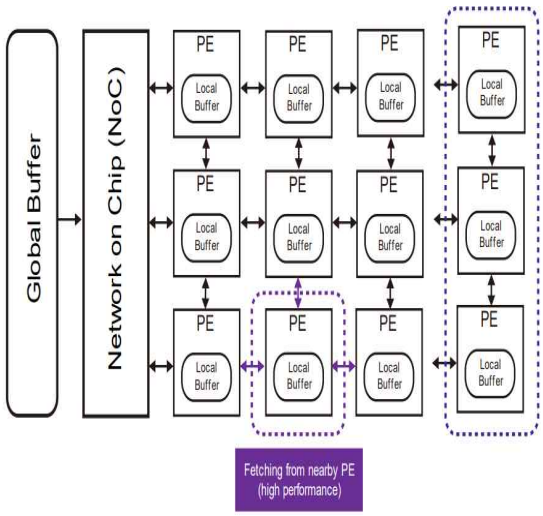
이러한 Chisel은 아래와 같은 프로젝트 *<fig.1>*과 같은 구조를 가진다.

3) GEMM Accelerator Architecture

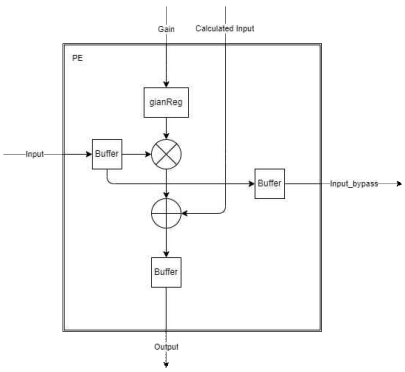
Systolic Array를 이용하면 GEMM 연산이 가능하다. Systolic Array구조는 *<fig.2>*과 같다. Systolic Array의 PE는 *<fig.3>*와 같은 구조로 이뤄져있다. 이러한 PE들은 <fig.4>와 같이 배치되어 Systolic Array구조를 이룬다.



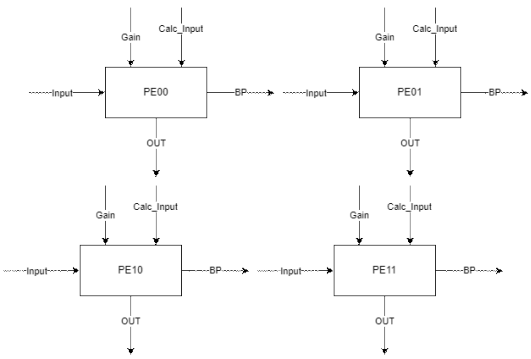
<fig.1>



<fig.2>



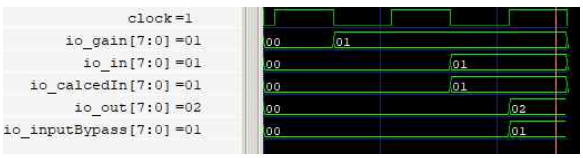
<fig.3>



<fig.4>

4) PE Implement test

*<Appendix.1-a>*는 Chisel로 구현 PE모듈이다. *<Appendix.1-b>* 는 해당 모듈을 테스트 하는 코드이다. 이를 실행하면 Gain 1이 bypassing되어야 하며 계산 결과는 2로 out되어야한다. 아래*<fig.5>*는 *<Appendix.1-b>*에 대한 Waveform이다.



<fig.5>

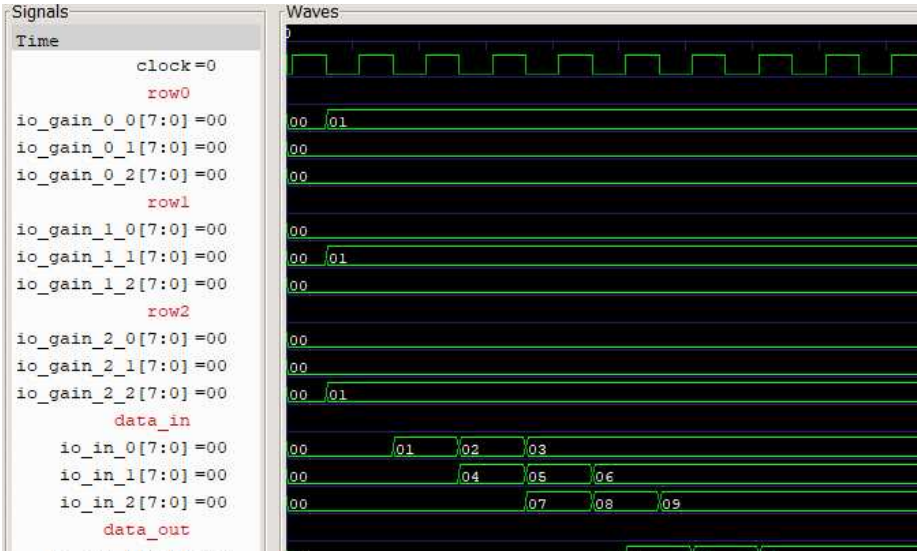
4) Parameterizable GEMM Accelerator Implement test

*<Appendix.2-a>*는 *<Appendix.1-a>*로 구현한 PE를 Parameterizable components로서 활용하여 확장가능하게 한 설계이다. 각 GEMM module의 PE간 Wiring관계는 *<fig.4>*와 *<tabel.1>*를 참조 바란다.

PE(i)(j).out	→	PE(i+1)(j).calcedIn
PE(i+1)(j).inputBypass	→	PE(i+1)(j).in

<tabel.1>

<*Appendix.2-b*>는 3*3 Matrix GEMM를 수행하는 테스트 코드이다. Gain에는 단위행렬처럼 작동할 수 있게 값을 입력해줬다. 해당 테스트를 수행하면 io_in[2:0][7:0]와 io_out[2:0][7:0]는 위상 차이만 가지고 값은 같아야한다. 아래<*fig.6*>는 <*Appendix.2-b*>에 대한 Waveform이다.

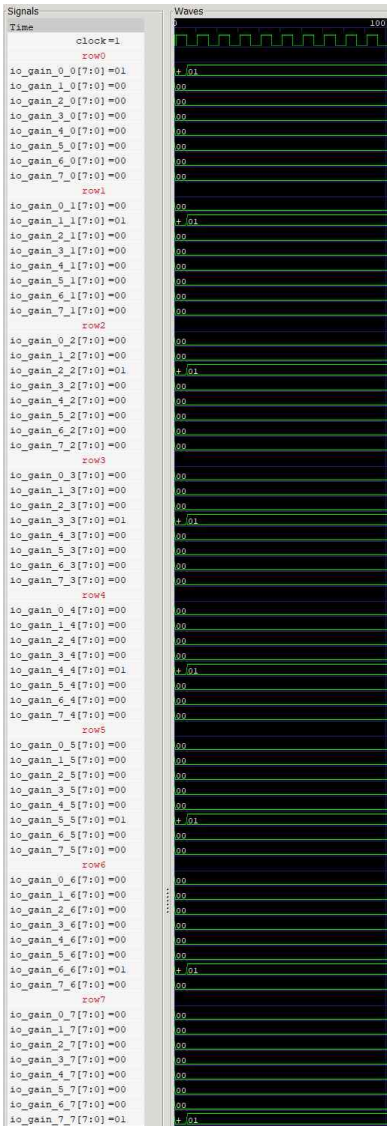


<fig.6>

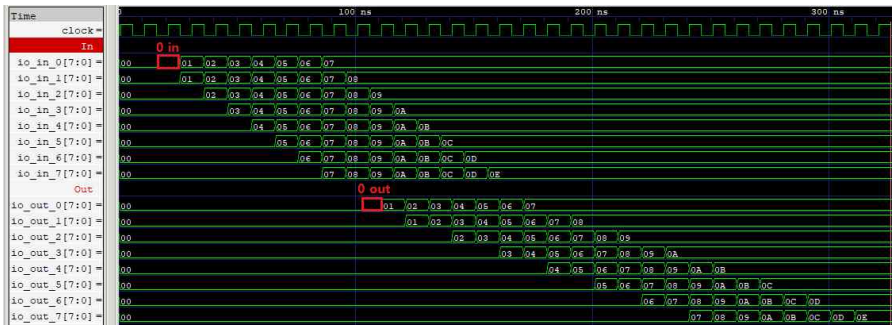
예상 실험결과와 같은 것을 알 수 있다.

5) Parameterizable GEMM Accelerator Implement test (8x8 Matrix Case)

<*Appendix.2-c*>는 <*Appendix.2-a*>를 8x8 matrix GEMM 가능하게 Parameter를 준 Tes tcase이다. <*Appendix.2-b*>와 마찬가지로 단위행렬로서 작동할 수 있게 Gain을 입력해주었다. <fig.7>을 참조바란다. 해당 테스트를 수행하면 io_in[7:0][7:0]와 io_out[7:0][7:0]는 위상 차이만 가지고 값은 같아야한다. <*fig.8*>를 참조 바란다.


Signals
Waves
Time
clock=1
row0
io_gain_0_0[7:0]=01
io_gain_1_0[7:0]=00
io_gain_2_0[7:0]=00
io_gain_3_0[7:0]=00
io_gain_4_0[7:0]=00
io_gain_5_0[7:0]=00
io_gain_6_0[7:0]=00
io_gain_7_0[7:0]=00
row1
io_gain_0_1[7:0]=00
io_gain_1_1[7:0]=01
io_gain_2_1[7:0]=00
io_gain_3_1[7:0]=00
io_gain_4_1[7:0]=00
io_gain_5_1[7:0]=00
io_gain_6_1[7:0]=00
io_gain_7_1[7:0]=00
row2
io_gain_0_2[7:0]=00
io_gain_1_2[7:0]=00
io_gain_2_2[7:0]=01
io_gain_3_2[7:0]=00
io_gain_4_2[7:0]=00
io_gain_5_2[7:0]=00
io_gain_6_2[7:0]=00
io_gain_7_2[7:0]=00
row3
io_gain_0_3[7:0]=00
io_gain_1_3[7:0]=00
io_gain_2_3[7:0]=00
io_gain_3_3[7:0]=01
io_gain_4_3[7:0]=00
io_gain_5_3[7:0]=00
io_gain_6_3[7:0]=00
io_gain_7_3[7:0]=00
row4
io_gain_0_4[7:0]=00
io_gain_1_4[7:0]=00
io_gain_2_4[7:0]=00
io_gain_3_4[7:0]=00
io_gain_4_4[7:0]=01
io_gain_5_4[7:0]=00
io_gain_6_4[7:0]=00
io_gain_7_4[7:0]=00
row5
io_gain_0_5[7:0]=00
io_gain_1_5[7:0]=00
io_gain_2_5[7:0]=00
io_gain_3_5[7:0]=00
io_gain_4_5[7:0]=00
io_gain_5_5[7:0]=01
io_gain_6_5[7:0]=00
io_gain_7_5[7:0]=00
row6
io_gain_0_6[7:0]=00
io_gain_1_6[7:0]=00
io_gain_2_6[7:0]=00
io_gain_3_6[7:0]=00
io_gain_4_6[7:0]=00
io_gain_5_6[7:0]=00
io_gain_6_6[7:0]=01
io_gain_7_6[7:0]=00
row7
io_gain_0_7[7:0]=00
io_gain_1_7[7:0]=00
io_gain_2_7[7:0]=00
io_gain_3_7[7:0]=00
io_gain_4_7[7:0]=00
io_gain_5_7[7:0]=00
io_gain_6_7[7:0]=00
io_gain_7_7[7:0]=01

<fig.7>


Time
clock
In
0 in
io_in_0[7:0]
io_in_1[7:0]
io_in_2[7:0]
io_in_3[7:0]
io_in_4[7:0]
io_in_5[7:0]
io_in_6[7:0]
io_in_7[7:0]
Out
0 out
io_out_0[7:0]
io_out_1[7:0]
io_out_2[7:0]
io_out_3[7:0]
io_out_4[7:0]
io_out_5[7:0]
io_out_6[7:0]
io_out_7[7:0]
100 ns
200 ns
300 ns

<fig.8>

Appendix.1-a

```
class PE(width:Int)extends Module{ // 얘가 그거임 단위 모듈이랄까...
  val io = IO(new Bundle() {
    val gain = Input(UInt(width.W))

    val in = Input(UInt(width.W))
    val calcedIn = Input(UInt(width.W))

    val inputBypass = Output(UInt(width.W))
    val out = Output(UInt(width.W))
  })

  val gainReg = RegInit(0.U(width.W))
  val inputBuffer = RegInit(0.U(width.W))
  val bypassBuffer = RegInit(0.U(width.W))
  val Buffer  = RegInit(0.U(width.W))

  gainReg := io.gain
  inputBuffer := io.in
  //Buffer := (gainReg * io.in) + io.calcedIn
  Buffer := (gainReg * inputBuffer) + io.calcedIn

  //bypassBuffer := io.in
  bypassBuffer := inputBuffer

  io.out := Buffer
  io.inputBypass := bypassBuffer
}
```

<Appendix.1-a>

Appendix.1-b

```
class PEtest extends AnyFlatSpec with ChiselScalatestTester {
  behavior of "PE"
  it should "PE ex" in {
    test(new PE(width = 8)).withAnnotations(Seq(WriteVcdAnnotation)) { p =>
      p.io.gain.poke(1.U(8.W))
      p.clock.step()

      p.io.in.poke(1.U(8.W))
      p.io.calcedIn.poke(1.U(8.W))

      p.clock.step()

      p.io.out.expect(2.U(8.W))
      p.io.inputBypass.expect(1.U(8.W))

    }
  }
}
```

<Appendix.1-a>

Appendix.2-a

```
class GEMM(size : Int, width: Int) extends Module {
  val io = IO(new Bundle {
    val in = Input(Vec(size, UInt(width.W)))
    val gain = Input(Vec(size, Vec(size, UInt(width.W))))
    val out = Output(Vec(size, UInt(width.W)))
  })

  val PEVec = Array.tabulate(size, size)((i, j) => Module(new PE(width)))

  //wiring
  //세로
  for (i <- 1 until size)
    for (j <- 0 until size)
      PEVec(i)(j).io.calcedIn := PEVec(i-1)(j).io.out

  //가로 : Bypass
  for (i <- 0 until size)
    for (j <- 1 until size)
      PEVec(i)(j).io.in := PEVec(i)(j-1).io.inputBypass

  //output
  for (i <- 0 until size)
    io.out(i) := PEVec(size-1)(i).io.out

  //data_injection
  //first row calced data Init
  for (j <- 0 until size)
    PEVec(0)(j).io.calcedIn := (0.U(width.W))

  //gain
  for (i <- 0 until size)
    for (j <- 0 until size + 0)
      PEVec(i)(j).io.gain := io.gain(i)(j)

  //data
  for (i <- 0 until size)
      PEVec(i)(0).io.in := io.in(i)

}
```

<Appendix.2-a>

Appendix.2-b

```
class GEMMtest extends AnyFlatSpec with ChiselScalatestTester {
  behavior of "GEMM"
  it should "GEMM ex" in {
    test(new GEMM(size = 3, width = 8)).withAnnotations(Seq(WriteVcdAnnotation)) { p =>
      p.io.gain(0)(0).poke(1.U(8.W))
      p.io.gain(0)(1).poke(0.U(8.W))
      p.io.gain(0)(2).poke(0.U(8.W))

      p.io.gain(1)(0).poke(0.U(8.W))
      p.io.gain(1)(1).poke(1.U(8.W))
      p.io.gain(1)(2).poke(0.U(8.W))

      p.io.gain(2)(0).poke(0.U(8.W))
      p.io.gain(2)(1).poke(0.U(8.W))
      p.io.gain(2)(2).poke(1.U(8.W))

      p.clock.step()

      p.io.in(0).poke(1.U(8.W))
      p.clock.step()

      p.io.in(0).poke(2.U(8.W))
      p.io.in(1).poke(4.U(8.W))
      p.clock.step()

      p.io.in(0).poke(3.U(8.W))
      p.io.in(1).poke(5.U(8.W))
      p.io.in(2).poke(7.U(8.W))
      p.clock.step()

      p.io.in(1).poke(6.U(8.W))
      p.io.in(2).poke(8.U(8.W))
      p.clock.step()

      p.io.in(2).poke(9.U(8.W))
      p.clock.step()

      //add clk
      p.clock.step()
      p.clock.step()
      p.clock.step()
      p.clock.step()
      p.clock.step()

    }
  }
}
```

<Appendix.2-b>

Appendix.2-c

```
class GEMMtest_8x8 extends AnyFlatSpec with ChiselScalatestTester {
  behavior of "GEMM_8x8"
  it should "GEMM ex" in {
    test(new GEMM(size = 8, width = 8)).withAnnotations(Seq(WriteVcdAnnotation)) { p =>
      // 공치행렬 선언
      for (i <- 0 until 8) {
        for (j <- 0 until 8) {
          if (i == j) p.io.gain(i)(j).poke(1.U(8.W))
          else p.io.gain(i)(j).poke(0.U(8.W))
        }
      }
      p.clock.step()

      // data injection
      for (i <- 0 until 8) {
        for (j <- 0 until i+1) {
          p.io.in(j).poke(i.U(8.W)) // in 피크
        }
        p.clock.step()
      }

      for (i <- 8 until 16) {
        for (j <- i - 7 until 8) {
          p.io.in(j).poke(i.U(8.W)) // in 피크
        }
        p.clock.step()
      }

      // add clk
      for (i <- 0 until 30)
        p.clock.step()
    }
  }
}
```

<Appendix.2-c>