In [1]:
```
from __future__ import print_function

import sys
import numpy as np
from time import time
import matplotlib.pyplot as plt

sys.path.append('/home/xilinx')
from pynq import Overlay
from pynq import allocate
from pynq import MMIO

ROM_SIZE = 0x2000 #8K

SOC_UP = 0x0000
SOC_LA = 0x1000
PL_AA_MB = 0x2000
PL_AA = 0x2100
SOC_IS = 0x3000
SOC_AS = 0x4000
SOC_CC = 0x5000
PL_AS = 0x6000
PL_IS = 0x7000
PL_DMA = 0x8000
```

In [2]:
```
ol = Overlay("/home/xilinx/jupyter_notebooks/PS/caravel_fpga.bit")
#ol.ip_dict
```

In [3]:
```
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
# ipReadROMCODE = ol.read_romcode_0

#Add for SPI
ip_QSPI = ol.axi_quad_spi_0
```

In [4]:
```
# ============================================
# AXI QuadSPI Control
# ============================================
XSP_DGIER_OFFSET = 0x1C
XSP_IISR_OFFSET = 0x20
XSP_IIER_OFFSET = 0x28
XSP_SRR_OFFSET = 0x40
XSP_CR_OFFSET = 0x60
XSP_SR_OFFSET = 0x64
XSP_DTR_OFFSET = 0x68
XSP_DRR_OFFSET = 0x6C
XSP_SSR_OFFSET = 0x70
XSP_TFO_OFFSET = 0x74
XSP_RFO_OFFSET = 0x78
XSP_REGISTERS = [0x40, 0x60, 0x64, 0x68, 0x6c, 0x70, 0x74, 0x78, 0x1c, 0x20, 0x28]

XSP_SRR_RESET_MASK = 0x0A
XSP_SR_TX_EMPTY_MASK = 0x04
XSP_SR_TX_FULL_MASK = 0x08
XSP_CR_TRANS_INHIBIT_MASK = 0x100
XSP_CR_LOOPBACK_MASK = 0x01
XSP_CR_ENABLE_MASK = 0x02
XSP_CR_MASTER_MODE_MASK = 0x04
XSP_CR_CLK_POLARITY_MASK = 0x08
XSP_CR_CLK_PHASE_MASK = 0x10
XSP_CR_TXFIFO_RESET_MASK = 0x20
XSP_CR_RXFIFO_RESET_MASK = 0x40
XSP_CR_MANUAL_SS_MASK = 0x80

SLAVE_NO_SELECTION = 0xFFFFFFFF

def cnfg(AxiQspi, clk_phase=0, clk_pol=0):
    print("Configure device")
    # Reset the SPI device
    AxiQspi.write(XSP_SRR_OFFSET, XSP_SRR_RESET_MASK)
    # Enable the transmit empty interrupt, which we use to determine progress on the transmission.
    AxiQspi.write(XSP_IIER_OFFSET, XSP_SR_TX_EMPTY_MASK)
    # Disable the global IPIF interrupt
    AxiQspi.write(XSP_DGIER_OFFSET, 0)
    # Deselect the slave on the SPI bus
    AxiQspi.write(XSP_SSR_OFFSET, SLAVE_NO_SELECTION)
    # Disable the transmitter, enable Manual Slave Select Assertion, put SPI controller into master mode, and enable it
    ControlReg = AxiQspi.read(XSP_CR_OFFSET)
    ControlReg = ControlReg | XSP_CR_MASTER_MODE_MASK | XSP_CR_MANUAL_SS_MASK | XSP_CR_ENABLE_MASK | XSP_CR_TXFIFO_RESET_MASK | X
    AxiQspi.write(XSP_CR_OFFSET, ControlReg)
    ControlReg = AxiQspi.read(XSP_CR_OFFSET)
    ControlReg = ControlReg & ~(XSP_CR_CLK_PHASE_MASK | XSP_CR_CLK_POLARITY_MASK)
    if clk_phase == 1:
        ControlReg = ControlReg | XSP_CR_CLK_PHASE_MASK
    if clk_pol == 1:
        ControlReg = ControlReg | XSP_CR_CLK_POLARITY_MASK
    AxiQspi.write(XSP_CR_OFFSET, ControlReg)

    return 0
```

```
    return 0

def write_tx_fifo(AxiQspi):
    #print("TransferData")
    ControlReg = AxiQspi.read(XSP_CR_OFFSET)
    ControlReg = ControlReg & ~XSP_CR_TRANS_INHIBIT_MASK
    AxiQspi.write(XSP_CR_OFFSET, ControlReg)

    StatusReg = AxiQspi.read(XSP_SR_OFFSET)
    while (StatusReg & XSP_SR_TX_EMPTY_MASK) == 0:
        StatusReg = AxiQspi.read(XSP_SR_OFFSET)

    #print('XSP_RFO_OFFSET  : 0x{0:08x}'.format(AxiQspi.read(XSP_RFO_OFFSET)))
    ControlReg = AxiQspi.read(XSP_CR_OFFSET)
    ControlReg = ControlReg | XSP_CR_TRANS_INHIBIT_MASK
    AxiQspi.write(XSP_CR_OFFSET, ControlReg)


def read_rx_fifo(bypass_length, AxiQspi):
    #print("ReadResponse")
    resp = list()
    RxFifoStatus = AxiQspi.read(XSP_SR_OFFSET) & 0x01

    # By pass the FIFO data during master issue command and address to slave device
    command_addr_length = bypass_length
    counter = 0

    while RxFifoStatus == 0:
        #temp = AxiQspi.read(XSP_RFO_OFFSET)
        #print('XSP_RFO_OFFSET  : 0x{0:08x}'.format(temp))
        temp = AxiQspi.read(XSP_DRR_OFFSET)
        #print('XSP_DRR_OFFSET  : 0x{0:08x}'.format(temp))

        counter = counter + 1
        if(counter > command_addr_length):
            resp.append(temp)

        RxFifoStatus = AxiQspi.read(XSP_SR_OFFSET) & 0x01

    return resp
```

In [5]:
```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0x0
0x20 =  0x0
0x34 =  0xffffff7
0x38 =  0x3f
```

In [6]:
```
# ============================================
# Release Reset First before passthrough mode
# ============================================
# Release Caravel reset
# 0x10 : Data signal of outpin_ctrl
#        bit 0  - outpin_ctrl[0] (Read/Write)
#        others - reserved
print (ipOUTPIN.read(0x10))
ipOUTPIN.write(0x10, 1)
print (ipOUTPIN.read(0x10))
```

```
0
1
```

In [7]:
```
# ============================================
# Load firmware (fsic.hex) to memory npROM
# ============================================

# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
npROM = np.zeros(ROM_SIZE >> 2, dtype=np.uint32)

npROM_index = 0
npROM_offset = 0
fiROM = open("/home/xilinx/jupyter_notebooks/PS/fsic.hex", "r+")
```

```
for line in fiROM:
    # offset header
    if line.startswith('@'):
        # Ignore first char @
        npROM_offset = int(line[1:].strip(b'\x00'.decode()), base = 16)
        npROM_offset = npROM_offset >> 2 # 4byte per offset
        #print (npROM_offset)
        npROM_index = 0
        continue
    #print (line)

    # We suppose the data must be 32bit alignment
    buffer = 0
    bytecount = 0
    for line_byte in line.strip(b'\x00'.decode()).split():
        buffer += int(line_byte, base = 16) << (8 * bytecount)
        bytecount += 1
        # Collect 4 bytes, write to npROM
        if(bytecount == 4):
            npROM[npROM_offset + npROM_index] = buffer
            # Clear buffer and bytecount
            buffer = 0
            bytecount = 0
            npROM_index += 1
            #print (npROM_index)
            continue
    # Fill rest data if not alignment 4 bytes
    if (bytecount != 0):
        npROM[npROM_offset + npROM_index] = buffer
        npROM_index += 1

fiROM.close()
```

In [8]:
```
# ============================================
# Enabling passthrou mode
# ============================================
cnfg(ip_QSPI)
# Passthrou mode - Write command
ip_QSPI.write(XSP_DTR_OFFSET, 0xC4) # Pass-Through (management)
ip_QSPI.write(XSP_DTR_OFFSET, 0x02) # Command: Write data to memory
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte0
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte1
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte2

print('XSP_TFO_OFFSET  : 0x{0:08x}'.format(ip_QSPI.read(XSP_TFO_OFFSET)))

ip_QSPI.write(XSP_SSR_OFFSET, 0xFFFFFFFE)
write_tx_fifo(ip_QSPI)

print('XSP_TFO_OFFSET  : 0x{0:08x}'.format(ip_QSPI.read(XSP_TFO_OFFSET)))
```

```
Configure device
XSP_TFO_OFFSET  : 0x00000004
XSP_TFO_OFFSET  : 0x00000000
```

In [9]:
```
# ============================================
# Writing FW into SPIROM
# ============================================
# Fill up Tx_FIFO (16) for each write_tx_fifo
for index in range (ROM_SIZE >> 2):
     # 4 bytes alignment in npROM
    for byte_shift in range(4):
        tmp = int((npROM[index] >> (byte_shift * 8)) & 0xFF)
        ip_QSPI.write(XSP_DTR_OFFSET, tmp) # Write_data
    # TX_FIFO = 16, 4 * 4 = 16
    if((index % 3) == 3):
        write_tx_fifo(ip_QSPI)

# If rest data is not enough 16 bytes. Tx_FIFO is not empty
    StatusReg = ip_QSPI.read(XSP_SR_OFFSET)
    if ((StatusReg & XSP_SR_TX_EMPTY_MASK) == 0):
         write_tx_fifo(ip_QSPI)
```

In [10]:
```
# ============================================
# Read SPIROM for testing
# ============================================
cnfg(ip_QSPI)
```

```
Configure device
```

Out[10]:
```
0
```

In [11]:
```
# Test Passthrou mode - Read command
ip_QSPI.write(XSP_DTR_OFFSET, 0xC4) # Pass-Through (management)
ip_QSPI.write(XSP_DTR_OFFSET, 0x03) # Command: Read data from memory
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte0
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte1
ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Address_byte2
# Write dummy data
data_length = 0x8
for index in range(data_length):
    ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Dummy data

print('XSP_TFO_OFFSET  : 0x{0:08x}'.format(ip_QSPI.read(XSP_TFO_OFFSET)))
ip_QSPI.write(XSP_SSR_OFFSET, 0xFFFFFFFE)
```

```
XSP_TFO_OFFSET  : 0x0000000c
```

In [12]:
```
# Issue SPI master cycle
write_tx_fifo(ip_QSPI)

# Read the Rx data
rx_final = read_rx_fifo(5, ip_QSPI)
for data in rx_final:
    print (hex(data))
```

```
0x6f
0x0
0x0
0xb
0x13
0x0
0x0
0x0
```

In [13]:
```
# Write dummy data
data_length = 0x8
for index in range(data_length):
    ip_QSPI.write(XSP_DTR_OFFSET, 0x00) # Dummy data

print('XSP_TFO_OFFSET  : 0x{0:08x}'.format(ip_QSPI.read(XSP_TFO_OFFSET)))
```

```
XSP_TFO_OFFSET  : 0x00000007
```

In [14]:
```
# Issue SPI master cycle
write_tx_fifo(ip_QSPI)

# Read the Rx data
rx_final = read_rx_fifo(0, ip_QSPI)
for data in rx_final:
    print (hex(data))
```

```
0x13
0x0
0x0
0x0
0x13
0x0
0x0
0x0
```

In [15]:
```
# ========================================
# Exit passthrou mode, FW will be fetched
# ========================================
ip_QSPI.write(XSP_SSR_OFFSET, SLAVE_NO_SELECTION)
```

In [16]:
```
# Check MPRJ_IO input/out/en
# 0x10 : Data signal of ps_mprj_in
#        bit 31~0 - ps_mprj_in[31:0] (Read/Write)
# 0x14 : Data signal of ps_mprj_in
#        bit 5~0 - ps_mprj_in[37:32] (Read/Write)
#        others  - reserved
# 0x1c : Data signal of ps_mprj_out
#        bit 31~0 - ps_mprj_out[31:0] (Read)
# 0x20 : Data signal of ps_mprj_out
#        bit 5~0 - ps_mprj_out[37:32] (Read)
#        others  - reserved
# 0x34 : Data signal of ps_mprj_en
#        bit 31~0 - ps_mprj_en[31:0] (Read)
# 0x38 : Data signal of ps_mprj_en
#        bit 5~0 - ps_mprj_en[37:32] (Read)
#        others  - reserved

print ("0x10 = ", hex(ipPS.read(0x10)))
print ("0x14 = ", hex(ipPS.read(0x14)))
print ("0x1c = ", hex(ipPS.read(0x1c)))
print ("0x20 = ", hex(ipPS.read(0x20)))
print ("0x34 = ", hex(ipPS.read(0x34)))
print ("0x38 = ", hex(ipPS.read(0x38)))
```

```
0x10 =  0x0
0x14 =  0x0
0x1c =  0x0
0x20 =  0x0
0x34 =  0x3ffff6
0x38 =  0x10
```

In [17]:
```
IP_BASE_ADDRESS = 0x60000000
ADDRESS_RANGE = 0x9000
mmio = MMIO(IP_BASE_ADDRESS, ADDRESS_RANGE)
```

In [18]:
```
# ================================================================================== #
# ================================================================================== #
# PL_FSIC Side Configuration
# ================================================================================== #
# ================================================================================== #
```

In [19]:
```
# PL_IS Config
ADDRESS_OFFSET = PL_IS #0x7000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x0
```

In [20]:
```
mmio.write(ADDRESS_OFFSET, 0x12345671)
```

```
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x1
```

In [21]:
```
mmio.write(ADDRESS_OFFSET, 0x12345673)
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x3
```

In [22]:
```
# PL_AS Config
ADDRESS_OFFSET = PL_AS # 0x6000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
mmio.write(ADDRESS_OFFSET, 0x12345676)
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x6
mmio.read(ADDRESS_OFFSET):  0x6
```

In [23]:
```
# PL_AA Config
ADDRESS_OFFSET = PL_AA # 0x2100
print("mmio.read(ADDRESS_OFFSET+0x00): ", hex(mmio.read(ADDRESS_OFFSET+0x00)))
print("mmio.read(ADDRESS_OFFSET+0x04): ", hex(mmio.read(ADDRESS_OFFSET+0x04)))
```

```
mmio.read(ADDRESS_OFFSET+0x00):  0x0
mmio.read(ADDRESS_OFFSET+0x04):  0x0
```

In [24]:
```
mmio.write(ADDRESS_OFFSET+0x00, 0x11111111)
mmio.write(ADDRESS_OFFSET+0x04, 0x22222222)
```

In [25]:
```
# PL_AA_MB Mailbox
ADDRESS_OFFSET = PL_AA_MB # 0x2000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
print("mmio.read(ADDRESS_OFFSET+0x04): ", hex(mmio.read(ADDRESS_OFFSET+0x04)))
print("mmio.read(ADDRESS_OFFSET+0x08): ", hex(mmio.read(ADDRESS_OFFSET+0x08)))
print("mmio.read(ADDRESS_OFFSET+0x0C): ", hex(mmio.read(ADDRESS_OFFSET+0x0C)))
print("mmio.read(ADDRESS_OFFSET+0x10): ", hex(mmio.read(ADDRESS_OFFSET+0x10)))
print("mmio.read(ADDRESS_OFFSET+0x14): ", hex(mmio.read(ADDRESS_OFFSET+0x14)))
print("mmio.read(ADDRESS_OFFSET+0x18): ", hex(mmio.read(ADDRESS_OFFSET+0x18)))
print("mmio.read(ADDRESS_OFFSET+0x1C): ", hex(mmio.read(ADDRESS_OFFSET+0x1C)))
```

```
mmio.read(ADDRESS_OFFSET):  0x0
mmio.read(ADDRESS_OFFSET+0x04):  0x0
mmio.read(ADDRESS_OFFSET+0x08):  0x0
mmio.read(ADDRESS_OFFSET+0x0C):  0x0
mmio.read(ADDRESS_OFFSET+0x10):  0x0
mmio.read(ADDRESS_OFFSET+0x14):  0x0
mmio.read(ADDRESS_OFFSET+0x18):  0x0
mmio.read(ADDRESS_OFFSET+0x1C):  0x0
```

In [26]:
```
mmio.write(ADDRESS_OFFSET, 0x11111112)
mmio.write(ADDRESS_OFFSET+0x04, 0x22222223)
mmio.write(ADDRESS_OFFSET+0x08, 0x33333334)
mmio.write(ADDRESS_OFFSET+0x0C, 0x44444445)
mmio.write(ADDRESS_OFFSET+0x10, 0x55555556)
mmio.write(ADDRESS_OFFSET+0x14, 0x66666667)
mmio.write(ADDRESS_OFFSET+0x18, 0x77777778)
mmio.write(ADDRESS_OFFSET+0x1C, 0x88888889)
```

In [27]:
```
# ================================================================================== #
# ================================================================================== #
# PL_Caravel Side Configuration
# ================================================================================== #
# ================================================================================== #
```

In [28]:
```
# Caravel-IS Config
ADDRESS_OFFSET = SOC_IS # 0x3000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x1
```

In [29]:
```
# Caravel-IS Config
ADDRESS_OFFSET = 0x3000
mmio.write(ADDRESS_OFFSET, 0x00000003)
```

In [30]:
```
# Caravel-AS Config
ADDRESS_OFFSET = SOC_AS # 0x4000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x6
```

In [31]:
```
# Caravel-AS Config
ADDRESS_OFFSET = SOC_AS # 0x4000
mmio.write(ADDRESS_OFFSET, 0x00000006)
```

In [32]:
```
# Caravel-CC Config
ADDRESS_OFFSET = SOC_CC # 0x5000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

```
mmio.read(ADDRESS_OFFSET):  0x1f
```

In [33]:
```
# Caravel-CC Config
ADDRESS_OFFSET = SOC_CC # 0x5000
mmio.write(ADDRESS_OFFSET, 0x00000000)
```

In [34]:
```
# Caravel-UP Config
ADDRESS_OFFSET = SOC_UP # 0x0000
print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET+0x04)))
```

```
mmio.read(ADDRESS_OFFSET):  0x280
```

In [35]:
```
# select target UP
ADDRESS_OFFSET = SOC_CC # 0x5000
mmio.write(ADDRESS_OFFSET, 0x00000001)
#print("mmio.read(ADDRESS_OFFSET): ", hex(mmio.read(ADDRESS_OFFSET)))
```

TPU application

Address def

In [36]:
```
TPU_CTRL_OFFSET = 0x00

TPU_M_OFFSET = 0x10
TPU_K_OFFSET = 0x14
TPU_N_OFFSET = 0x18

TPU_BUFF_A_ADDR_OFFSET = 0x20
TPU_BUFF_A_DIN_OFFSET = 0x24
TPU_BUFF_B_ADDR_OFFSET = 0x30
TPU_BUFF_B_DIN_OFFSET = 0x34
TPU_BUFF_C_ADDR_OFFSET = 0x40
TPU_BUFF_C_DOUT_0_OFFSET = 0x44
TPU_BUFF_C_DOUT_1_OFFSET = 0x48
TPU_BUFF_C_DOUT_2_OFFSET = 0x4c
TPU_BUFF_C_DOUT_3_OFFSET = 0x50
```

In [127]:
```
DEBUG = False
```

Function def

In [132]:
```
import struct

def matrix_mul(matA, matB, k, m, n):
    # wait ap_idle
    while (mmio.read(SOC_UP + TPU_CTRL_OFFSET) & 0x04) == 0:
        continue
    # write config
    mmio.write(SOC_UP + TPU_M_OFFSET, m)
    mmio.write(SOC_UP + TPU_K_OFFSET, k)
    mmio.write(SOC_UP + TPU_N_OFFSET, n)
    # write matA
    nrow = int(k * ((m >> 2) + 1)) if m % 4 != 0 else int(k * (m >> 2))
    #print(f'nrow A:{nrow}')
    for i in range(nrow):
        mmio.write(SOC_UP + TPU_BUFF_A_ADDR_OFFSET, i)
        mmio.write(SOC_UP + TPU_BUFF_A_DIN_OFFSET, matA[i]) # TODO
    # write matB
    nrow = int(k * ((n >> 2) + 1)) if n % 4 != 0 else int(k * (n >> 2))
    #print(f'nrow B:{nrow}')
    for i in range(nrow):
        mmio.write(SOC_UP + TPU_BUFF_B_ADDR_OFFSET, i)
        mmio.write(SOC_UP + TPU_BUFF_B_DIN_OFFSET, matB[i]) # TODO
    # ap_start
    mmio.write(SOC_UP + TPU_CTRL_OFFSET, 0x01)
    # wait ap_done

    while (mmio.read(SOC_UP + TPU_CTRL_OFFSET) & 0x02) == 0:
        continue
    calign = int((n+3)/4)*4
    matC_temp = np.zeros((m, calign), dtype=np.int32)

    nrow = int(m * ((n >> 2) + 1)) if n % 4 != 0 else int(m * (n >> 2))
    for i in range(nrow):
        mmio.write(SOC_UP + TPU_BUFF_C_ADDR_OFFSET, i)
        c_temp0 = mmio.read(SOC_UP + TPU_BUFF_C_DOUT_0_OFFSET) # TODO
        c_temp1 = mmio.read(SOC_UP + TPU_BUFF_C_DOUT_1_OFFSET) # TODO
        c_temp2 = mmio.read(SOC_UP + TPU_BUFF_C_DOUT_2_OFFSET) # TODO
        c_temp3 = mmio.read(SOC_UP + TPU_BUFF_C_DOUT_3_OFFSET) # TODO
        if DEBUG:
            print(f'{hex(c_temp3)},{hex(c_temp2)},{hex(c_temp1)},{hex(c_temp0)}')

        m_index = int(i % m)
        n_index = int(i / m)

        matC_temp[m_index][n_index*4 + 0] = int(c_temp3)
        matC_temp[m_index][n_index*4 + 1] = int(c_temp2)
        matC_temp[m_index][n_index*4 + 2] = int(c_temp1)
        matC_temp[m_index][n_index*4 + 3] = int(c_temp0)

    matC = matC_temp[:,0:n]

    return matC
```

In [133]:
```
import numpy as np
import re

def read_numbers(filepath):
```

```
    numbers = []
    with open(filepath, 'r') as file:
        for line in file:
            # Split line into individual hexadecimal numbers
            for hex_str in line.split():
                # Convert each hex string to an integer and store it
                number = int(hex_str, 16)
                numbers.append(number)
        return number  # Use yield to return one number at a time

def read_input_file(filepath):
    with open(filepath, 'r') as file:
        first_line = file.readline()  # Read the first line
        # Use regular expression to find and extract numbers
        numbers_str = re.split(r'\W+',first_line)
        # Convert each string number to integer and store in a list
        numbers = [int(num, 16) for num in numbers_str[:-1]]

        # Read PATNUM
        PATNUM = numbers[0]
        K_golden = numbers[1]
        M_golden = numbers[2]
        N_golden = numbers[3]

        for patcount in range(PATNUM):
            # Read K, M, N

            # Read A Matrix
            A_matrix = read_matrix(file, K_golden, M_golden, 0) # mode: 0//int8, 1//int32

            # Read B Matrix
            B_matrix = read_matrix(file, K_golden, N_golden, 0) # mode: 0//int8, 1//int32

            # Read Golden Matrix
            golden_matrix = read_matrix(file, M_golden, N_golden, 1)

            if DEBUG:
                print(f"K_golden: {K_golden}, M_golden: {M_golden}, N_golden: {N_golden}")
                print("A_matrix:", A_matrix)
                print("B_matrix:", B_matrix)
                print("Golden_matrix:", golden_matrix)
    return A_matrix, B_matrix, K_golden, M_golden, N_golden

def read_matrix(file, rows, cols, mode):
    nrow = (cols & 0x3 != 0) * rows * ((cols >> 2) + 1) + (cols & 0x3 == 0) * rows * (cols >> 2)
    matrix = []

    for i in range(nrow):
        while(1):
            line = file.readline().strip().split()
            if line !=[]:
                break
        rbuf = [int(x, 16) for x in line]
        if mode:
            int128_value = ((rbuf[0] & 0xFFFF) << 96) | ((rbuf[1] & 0xFFFF) << 64) | ((rbuf[2] & 0xFFFF) << 32) | ((rbuf[3] & 0xF
            matrix.append(int128_value)
        else:
            int32_value = ((rbuf[0] & 0xFF) << 24) | ((rbuf[1] & 0xFF) << 16) | ((rbuf[2] & 0xFF) << 8) | ((rbuf[3] & 0xFF) << 0)
            matrix.append(int32_value)

    return matrix
```

In [148]:
```
# Pattern 0
filepath = "/home/xilinx/jupyter_notebooks/PS/input0.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[148]:
```
array([[10508, 26042],
       [12391, 28813]])
```

In [149]:
```
# Pattern 1
filepath = "/home/xilinx/jupyter_notebooks/PS/input1.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[149]:
```
array([[42655, 46402, 52966, 48211],
       [72284, 77356, 76244, 54911],
       [54083, 64382, 61155, 32507],
       [92613, 55246, 67097, 44616]])
```

In [150]:
```
# Pattern 2
filepath = "/home/xilinx/jupyter_notebooks/PS/input2.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[150]:
```
array([[257544, 282056, 247302, 273224],
       [217173, 322973, 318826, 321325],
       [215332, 305971, 271182, 275131],
       [144806, 211321, 186570, 178815]])
```

In [151]:
```
# Pattern 3
filepath = "/home/xilinx/jupyter_notebooks/PS/input3.txt"
matA, matB, k, m, n = read_input_file(filepath)
matrix_mul(matA, matB, k, m, n)
```

Out[151]:
```
array([[191603, 148591, 180584, 143078, 123093, 215895, 160457, 102397,
        181006],
```

```
[302761, 196791, 261155, 144023, 207904, 266165, 191987, 211105,
 215770],
[298663, 229956, 305909, 195132, 233333, 302714, 207584, 227801,
 212764],
[257556, 175886, 213371, 171674, 194718, 245988, 208329, 205362,
 215941]])
```

In [ ]: