112-1 SoC Design Laboratory

Lab6

Group 10 王語卉 吳至凌 高宇勝

Introduction

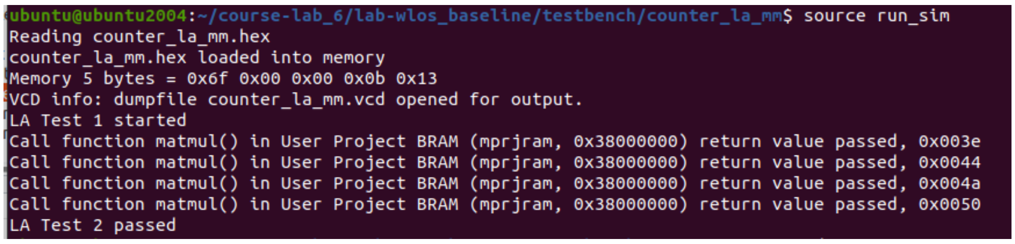
There are four workloads in this lab: matrix multiplication, quick sort, fir and uart. We get familiar with the uart behavior and observe the interruption in CPU.

In Lab6, we integrate the four workloads by modify the firmware code and the testbench. We also add a decoder in user project hardware to separate the user project and the uart request. After integrated the workloads, we synthesis and implement the design and use jupyter notebook to verify the results.

【Before integrated】

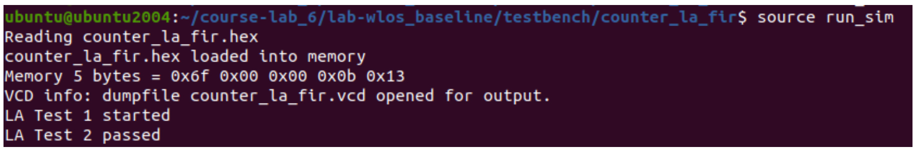
Simulation results

Matrix multiplication



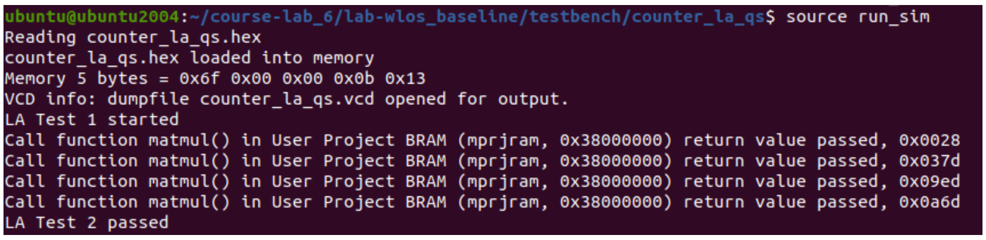
```
ubuntu@ubuntu2004:~/course-lab_6/lab-wlos_baseline/testbench/counter_la_mm$ source run_sim
Reading counter_la_mm.hex
counter_la_mm.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_mm.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
LA Test 2 passed
```

FIR



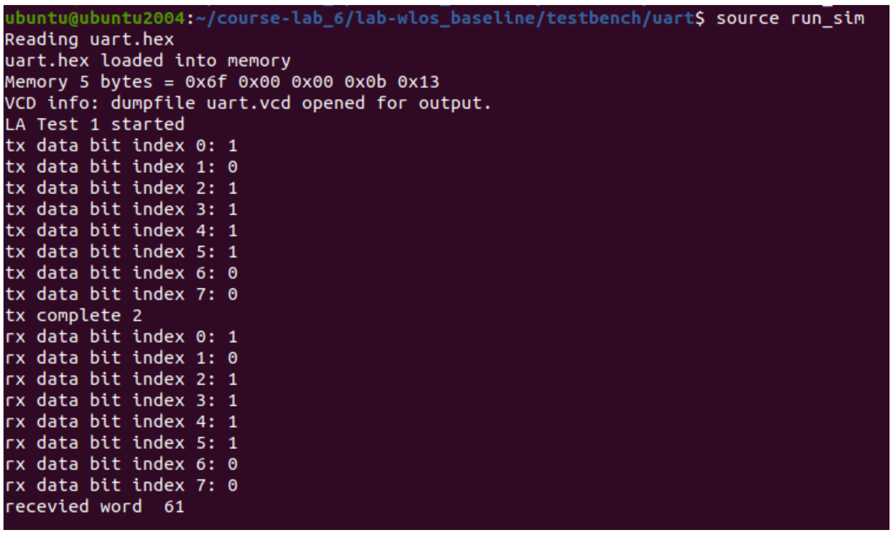
```
ubuntu@ubuntu2004:~/course-lab_6/lab-wlos_baseline/testbench/counter_la_fir$ source run_sim
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
LA Test 1 started
LA Test 2 passed
```

Quick sort

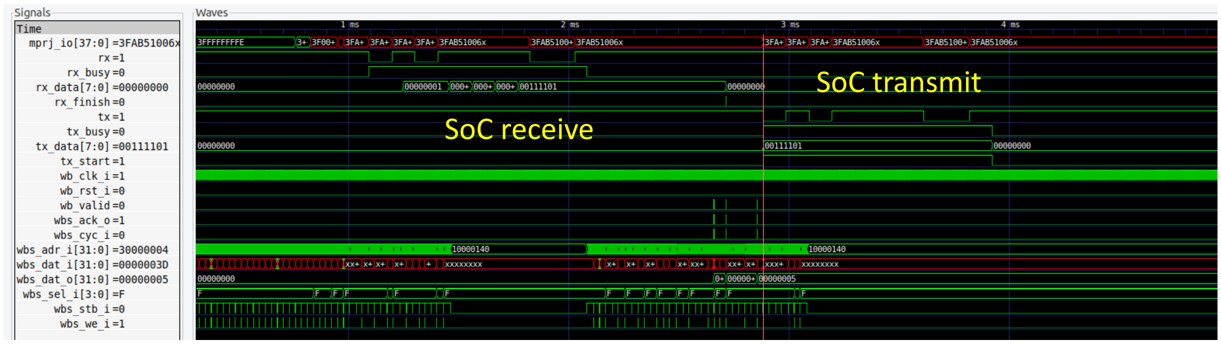


```
ubuntu@ubuntu2004:~/course-lab_6/lab-wlos_baseline/testbench/counter_la_qs$ source run_sim
Reading counter_la_qs.hex
counter_la_qs.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_qs.vcd opened for output.
LA Test 1 started
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
LA Test 2 passed
```

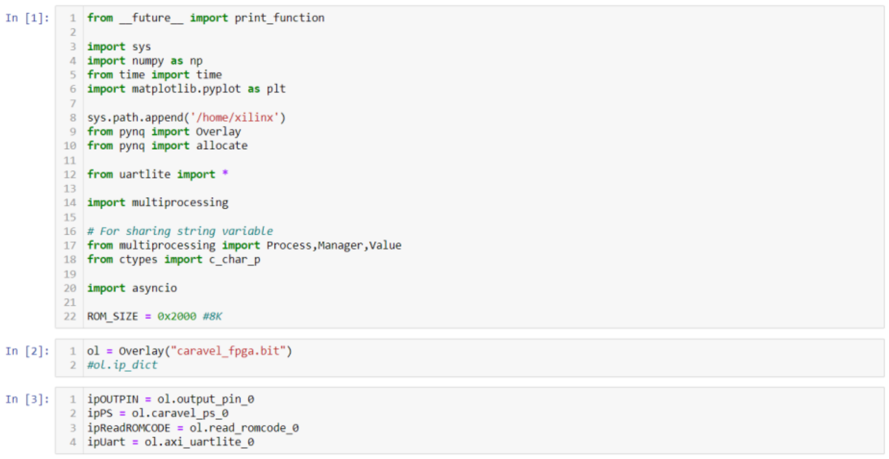
Uart



```
ubuntu@ubuntu2004:~/course-lab_6/lab-wlos_baseline/testbench/uart$ source run_sim
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
LA Test 1 started
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word   61
```



FPGA results



In [1]:
```python
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 uartlite import *

import multiprocessing

# For sharing string variable
from multiprocessing import Process,Manager,Value
from ctypes import c_char_p

import asyncio

ROM_SIZE = 0x2000 #8K
```

In [2]:
```python
ol = Overlay("caravel_fpga.bit")
#ol.ip_dict
```

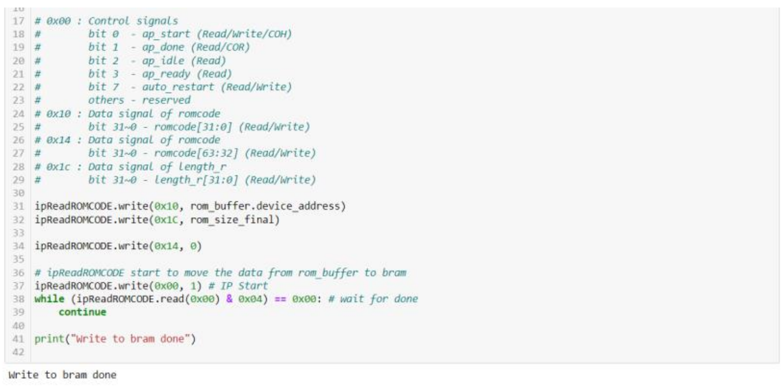
In [3]:
```python
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
ipUart = ol.axi_uartlite_0
```

In [4]:
```python
ol.interrupt_pins
```
Out[4]:
```
{'axi_uartlite_0/interrupt': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_uartlite_0/interrupt'},
 'axi_intc_0/intr': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_intc_0/intr'}}
```
In [5]:
```python
# See what interrupts are in the system
#ol.interrupt_pins

# Each IP instances has a _interrupts dictionary which lists the names of the interrupts
#ipUart._interrupts

# The interrupts object can then be accessed by its name
# The Interrupt class provides a single function wait
# which is an asyncio coroutine that returns when the interrupt is signalled.
intUart = ipUart.interrupt
```
In [6]:
```python
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

npROM = np.zeros(ROM_SIZE >> 2, dtype=np.uint32)
npROM_index = 0
npROM_offset = 0
fiROM = open("uart.hex", "r+")
#fiROM = open("counter_wb.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)
```
In [6]:
```python
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

npROM = np.zeros(ROM_SIZE >> 2, dtype=np.uint32)
npROM_index = 0
npROM_offset = 0
fiROM = open("uart.hex", "r+")
#fiROM = open("counter_wb.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)
```

```python
        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
```

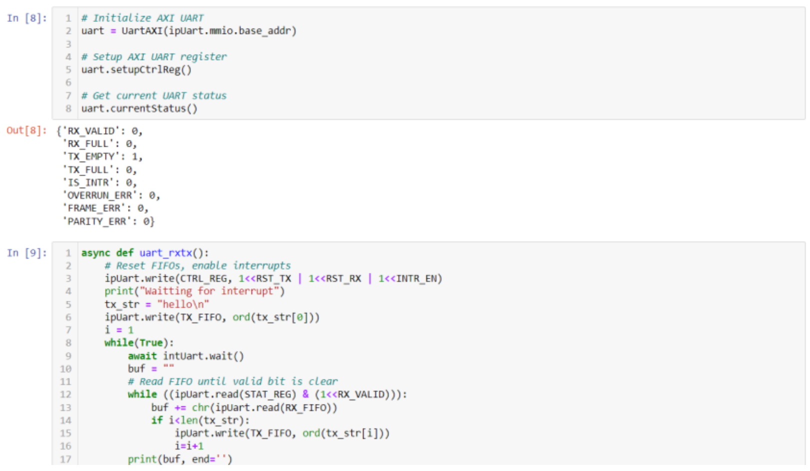
```python
            npROM_index += 1
            #print (npROM_index)
            continue
```

```python
    # Fill rest data if not alignment 4 bytes
    if (bytecount != 0):
        npROM[npROM_offset + npROM_index] = buffer
        npROM_index += 1

fiROM.close()

rom_size_final = npROM_offset + npROM_index
#print (rom_size_final)

#for data in npROM:
#    print (hex(data))

```
In [7]:
```python
# Allocate dram buffer will assign physical address to ip ipReadROMCODE

#rom_buffer = allocate(shape=(ROM_SIZE >> 2,), dtype=np.uint32)
rom_buffer = allocate(shape=(rom_size_final,), dtype=np.uint32)

# Initial it by npROM
#for index in range (ROM_SIZE >> 2):
for index in range (rom_size_final):
    rom_buffer[index] = npROM[index]

#for index in range (ROM_SIZE >> 2):
#    print ("0x{0:08x}".format(rom_buffer[index]))

# Program physical address for the romcode base address


# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
```



```python
# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)

ipReadROMCODE.write(0x10, rom_buffer.device_address)
ipReadROMCODE.write(0x1C, rom_size_final)

ipReadROMCODE.write(0x14, 0)

# ipReadROMCODE start to move the data from rom_buffer to bram
ipReadROMCODE.write(0x00, 1) # IP Start
while (ipReadROMCODE.read(0x00) & 0x04) == 0x00: # wait for done
    continue

print("Write to bram done")

```

```
Write to bram done
```



In [8]:

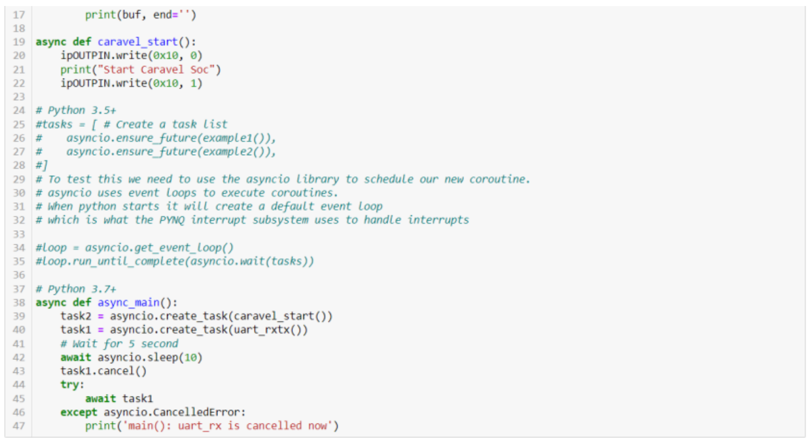
```python
# Initialize AXI UART
uart = UartAXI(ipUart.mmio.base_addr)

# Setup AXI UART register
uart.setupCtrlReg()

# Get current UART status
uart.currentStatus()
```

Out[8]:

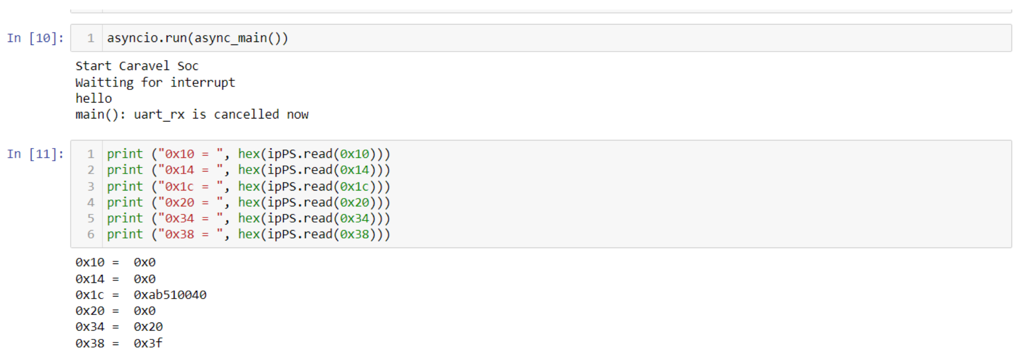
```
{'RX_VALID': 0,
 'RX_FULL': 0,
 'TX_EMPTY': 1,
 'TX_FULL': 0,
 'IS_INTR': 0,
 'OVERRUN_ERR': 0,
 'FRAME_ERR': 0,
 'PARITY_ERR': 0}
```

In [9]:

```python
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')
```



```python
        print(buf, end='')

async def caravel_start():
    ipOUTPIN.write(0x10, 0)
    print("Start Caravel Soc")
    ipOUTPIN.write(0x10, 1)

# Python 3.5+
#tasks = [ # Create a task list
#    asyncio.ensure_future(example1()),
#    asyncio.ensure_future(example2()),
#]
# To test this we need to use the asyncio library to schedule our new coroutine.
# asyncio uses event loops to execute coroutines.
# When python starts it will create a default event loop
# which is what the PYNQ interrupt subsystem uses to handle interrupts

#loop = asyncio.get_event_loop()
#loop.run_until_complete(asyncio.wait(tasks))

# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    task1 = asyncio.create_task(uart_rxtx())
    # Wait for 5 second
    await asyncio.sleep(10)
    task1.cancel()
    try:
        await task1
    except asyncio.CancelledError:
        print('main(): uart_rx is cancelled now')
```



In [10]:

```python
asyncio.run(async_main())
```

```
Start Caravel Soc
Waitting for interrupt
hello
main(): uart_rx is cancelled now
```

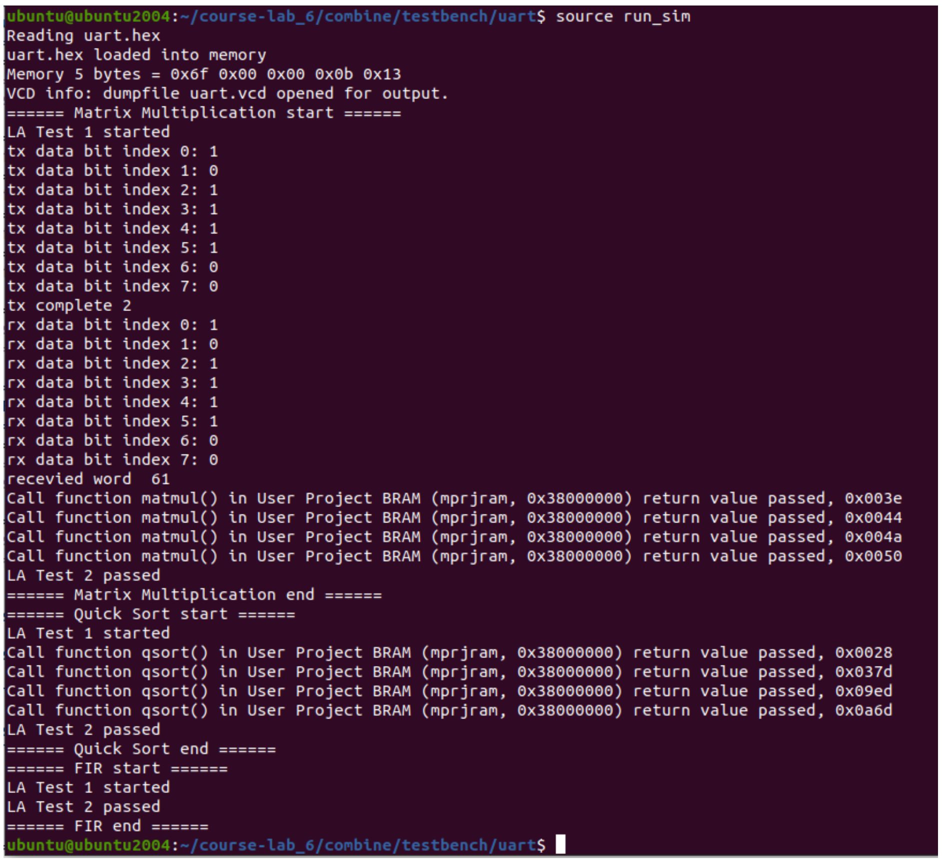
In [11]:

```python
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)))
```

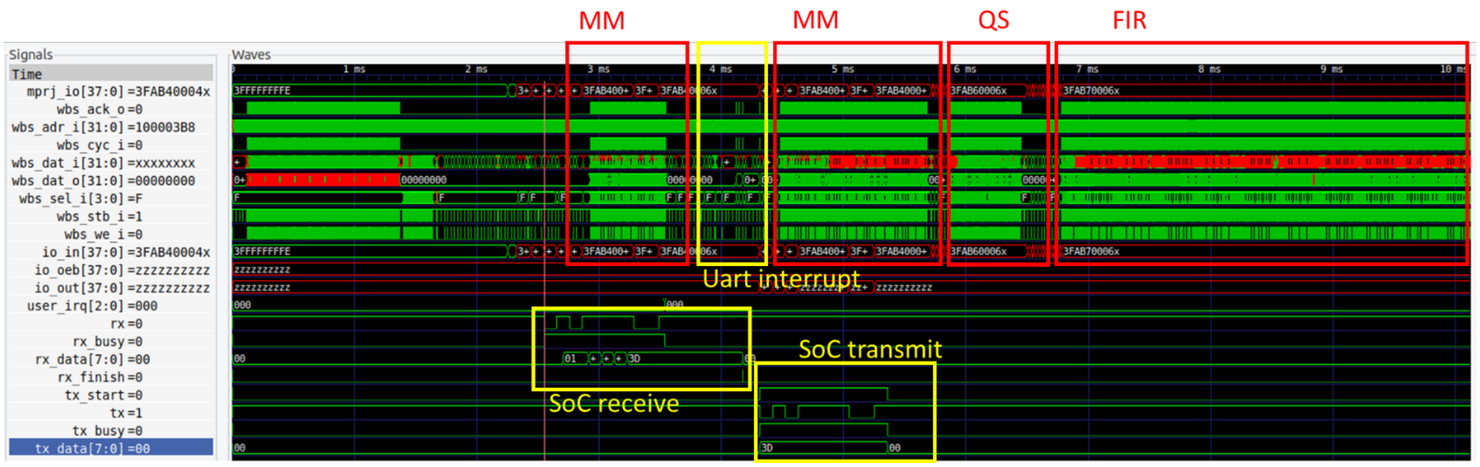
```
0x10 =  0x0
0x14 =  0x0
0x1c =  0xab510040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

【After integrated】

Simulation results



```
ubuntu@ubuntu2004:~/course-lab_6/combine/testbench/uart$ source run_sim
Reading uart.hex
uart.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile uart.vcd opened for output.
====== Matrix Multiplication start ======
LA Test 1 started
tx data bit index 0: 1
tx data bit index 1: 0
tx data bit index 2: 1
tx data bit index 3: 1
tx data bit index 4: 1
tx data bit index 5: 1
tx data bit index 6: 0
tx data bit index 7: 0
tx complete 2
rx data bit index 0: 1
rx data bit index 1: 0
rx data bit index 2: 1
rx data bit index 3: 1
rx data bit index 4: 1
rx data bit index 5: 1
rx data bit index 6: 0
rx data bit index 7: 0
recevied word  61
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x003e
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0044
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x004a
Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0050
LA Test 2 passed
====== Matrix Multiplication end ======
====== Quick Sort start ======
LA Test 1 started
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0028
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x037d
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x09ed
Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x0a6d
LA Test 2 passed
====== Quick Sort end ======
====== FIR start ======
LA Test 1 started
LA Test 2 passed
====== FIR end ======
ubuntu@ubuntu2004:~/course-lab_6/combine/testbench/uart$ 
```



FPGA results

In [1]:
```python
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 uartlite import *

import multiprocessing

# For sharing string variable
from multiprocessing import Process,Manager,Value
from ctypes import c_char_p

import asyncio

ROM_SIZE = 0x2000 #8K
```

In [2]:
```python
ol = Overlay("caravel_fpga.bit")
#ol.ip_dict
```

In [3]:
```python
ipOUTPIN = ol.output_pin_0
ipPS = ol.caravel_ps_0
ipReadROMCODE = ol.read_romcode_0
ipUart = ol.axi_uartlite_0
```

In [4]:
```python
ol.interrupt_pins
```

Out[4]:
```
{'axi_intc_0/intr': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_intc_0/intr'},
 'axi_uartlite_0/interrupt': {'controller': 'axi_intc_0',
  'index': 0,
  'fullpath': 'axi_uartlite_0/interrupt'}}
```

In [5]:
```python
# See what interrupts are in the system
#ol.interrupt_pins

# Each IP instances has a _interrupts dictionary which lists the names of the interrupts
#ipUart._interrupts

# The interrupts object can then be accessed by its name
# The Interrupt class provides a single function wait
# which is an asyncio coroutine that returns when the interrupt is signalled.
intUart = ipUart.interrupt
```

In [6]:
```python
# Create np with 8K/4 (4 bytes per index) size and be initiled to 0
rom_size_final = 0

npROM = np.zeros(ROM_SIZE >> 2, dtype=np.uint32)
npROM_index = 0
npROM_offset = 0
fiROM = open("uart.hex", "r+")
#fiROM = open("counter_wb.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)
```

```python
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()

rom_size_final = npROM_offset + npROM_index
#print (rom_size_final)

#for data in npROM:
#    print (hex(data))
```

In [7]:

```python
# Allocate dram buffer will assign physical address to ip ipReadROMCODE

#rom_buffer = allocate(shape=(ROM_SIZE >> 2,), dtype=np.uint32)
rom_buffer = allocate(shape=(rom_size_final,), dtype=np.uint32)

# Initial it by npROM
#for index in range (ROM_SIZE >> 2):
for index in range (rom_size_final):
    rom_buffer[index] = npROM[index]

#for index in range (ROM_SIZE >> 2):
#    print ("0x{0:08x}".format(rom_buffer[index]))

# Program physical address for the romcode base address


# 0x00 : Control signals
#        bit 0  - ap_start (Read/Write/COH)
#        bit 1  - ap_done (Read/COR)
#        bit 2  - ap_idle (Read)
#        bit 3  - ap_ready (Read)
#        bit 7  - auto_restart (Read/Write)
#        others - reserved
# 0x10 : Data signal of romcode
#        bit 31~0 - romcode[31:0] (Read/Write)
# 0x14 : Data signal of romcode
#        bit 31~0 - romcode[63:32] (Read/Write)
# 0x1c : Data signal of length_r
#        bit 31~0 - length_r[31:0] (Read/Write)

ipReadROMCODE.write(0x10, rom_buffer.device_address)
ipReadROMCODE.write(0x1C, rom_size_final)

ipReadROMCODE.write(0x14, 0)

# ipReadROMCODE start to move the data from rom_buffer to bram
ipReadROMCODE.write(0x00, 1) # IP Start
while (ipReadROMCODE.read(0x00) & 0x04) == 0x00: # wait for done
    continue

print("Write to bram done")
```

```
Write to bram done
```

In [8]:

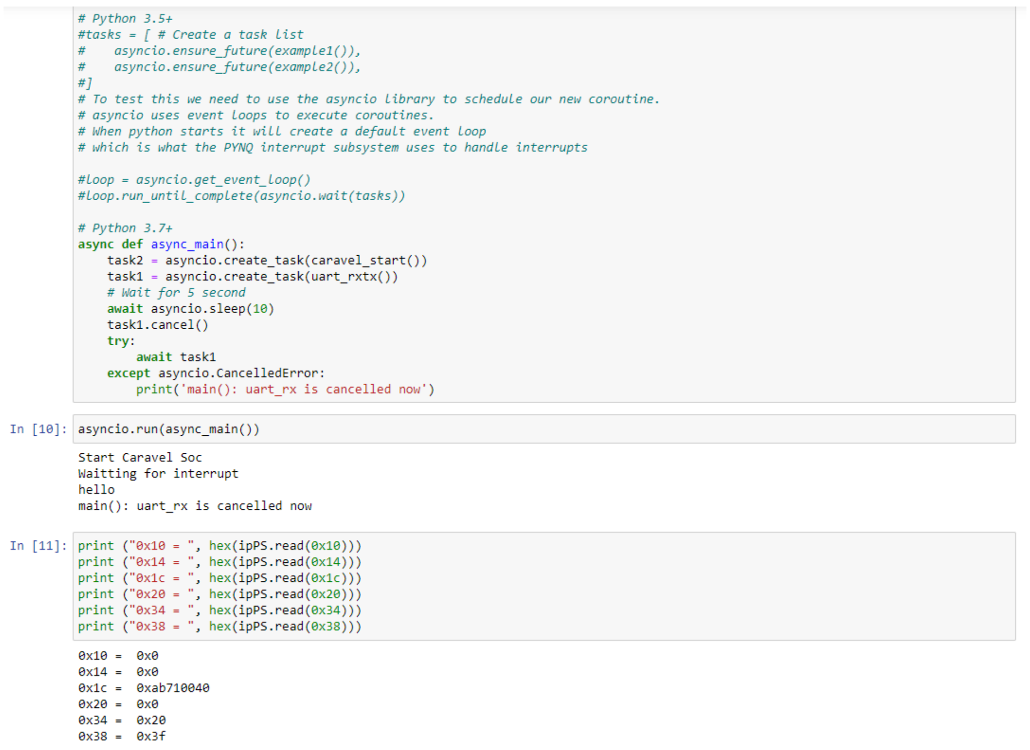
```python
# Initialize AXI UART
uart = UartAXI(ipUart.mmio.base_addr)

# Setup AXI UART register
uart.setupCtrlReg()

# Get current UART status
uart.currentStatus()
```

Out[8]:

```
{'RX_VALID': 0,
 'RX_FULL': 0,
 'TX_EMPTY': 1,
 'TX_FULL': 0,
 'IS_INTR': 0,
 'OVERRUN_ERR': 0,
 'FRAME_ERR': 0,
 'PARITY_ERR': 0}
```

In [9]:

```python
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')

async def caravel_start():
    ipOUTPIN.write(0x10, 0)
    print("Start Caravel Soc")
    ipOUTPIN.write(0x10, 1)

# Python 3.5+
#tasks = [ # Create a task list
#    asyncio.ensure_future(example1()),
#    asyncio.ensure_future(example2()),
```



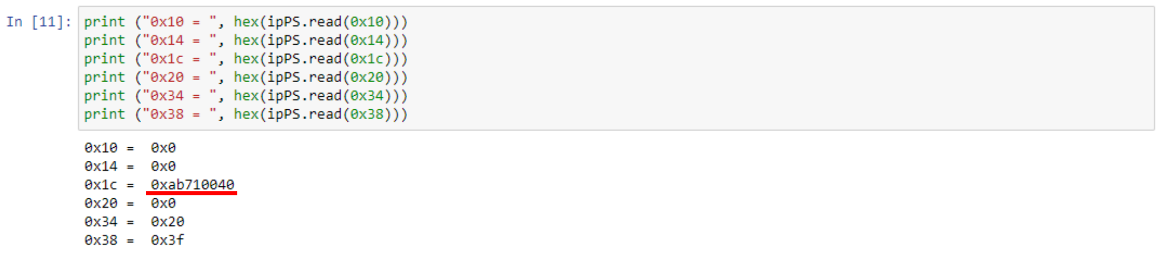
```
# Python 3.5+
#tasks = [ # Create a task list
#    asyncio.ensure_future(example1()),
#    asyncio.ensure_future(example2()),
#]
# To test this we need to use the asyncio library to schedule our new coroutine.
# asyncio uses event loops to execute coroutines.
# When python starts it will create a default event loop
# which is what the PYNQ interrupt subsystem uses to handle interrupts

#loop = asyncio.get_event_loop()
#loop.run_until_complete(asyncio.wait(tasks))

# Python 3.7+
async def async_main():
    task2 = asyncio.create_task(caravel_start())
    task1 = asyncio.create_task(uart_rxtx())
    # Wait for 5 second
    await asyncio.sleep(10)
    task1.cancel()
    try:
        await task1
    except asyncio.CancelledError:
        print('main(): uart_rx is cancelled now')
```

```
In [10]: asyncio.run(async_main())

Start Caravel Soc
Waitting for interrupt
hello
main(): uart_rx is cancelled now
```

```
In [11]: 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 =  0xab710040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

1. How do you verify your answer from notebook

Since I use different check bits to indicate different workloads, I can trace if the engine completes all the tasks after being interrupted by isr.



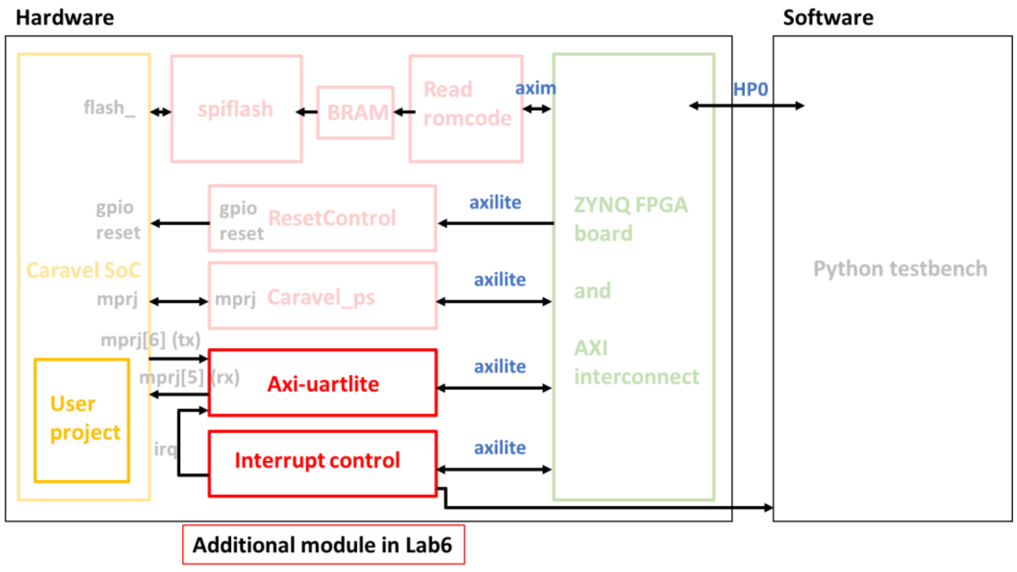
```
In [11]: print ("0x10 = ", hex(ipPS.read(0x10)))
         print ("0x14 = ", hex(ipPS.read(0x14)))
         print ("0x1c = ", hex(ipPS.read(0x1c)))
         print ("0x20 = ", hex(ipPS.read(0x20)))
         print ("0x34 = ", hex(ipPS.read(0x34)))
         print ("0x38 = ", hex(ipPS.read(0x38)))

0x10 =  0x0
0x14 =  0x0
0x1c =  0xab710040
0x20 =  0x0
0x34 =  0x20
0x38 =  0x3f
```

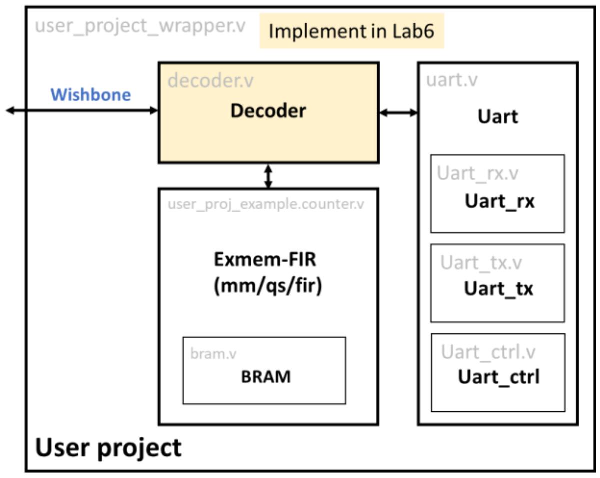
Workload	Start/end	Check bits
Matrix multiplication	Start	16’hAB40
	End	16’hAB51
Quick sort	Start	16’hAB60
	End	16’hAB61
FIR	Start	16’hAB70
	End	16’hAB71

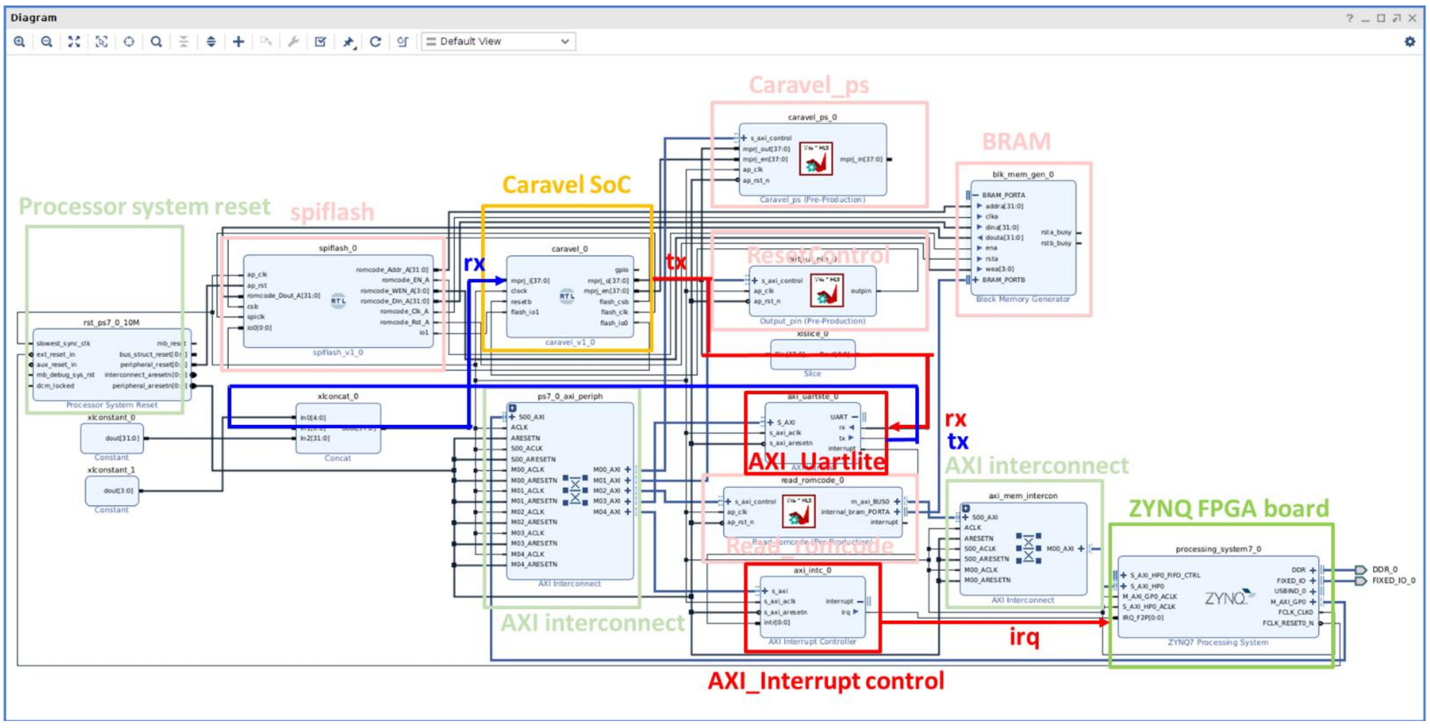
If there’re some tasks didn’t complete or have some mistakes, the check bits will stop at the check bits that the error happened.

2. Block design

The block diagram of the Lab6 is like below:







The part that we design is: decoder, firmware code and the testbench.

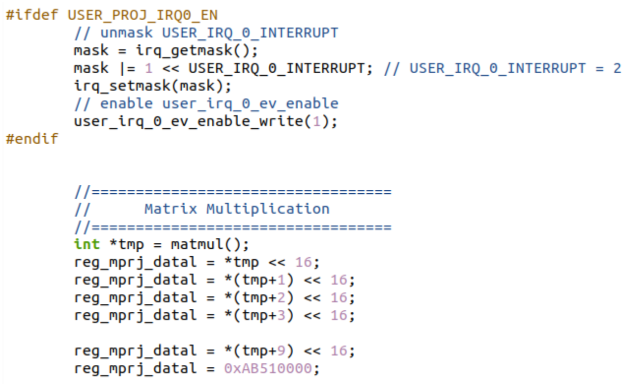
Decoder

All the input signals are sent to both uart and user project, but the respond signals will connect to uart or user project base on the wishbone address. If the address begins with 38, then the output ports will connect to the user project. Otherwise, they will connect to the uart.

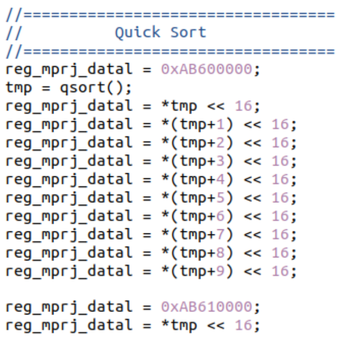
The exceptions are that, for the output ports io_out, io_oeb and user_irq, they are always connect to uart.

Firmware code

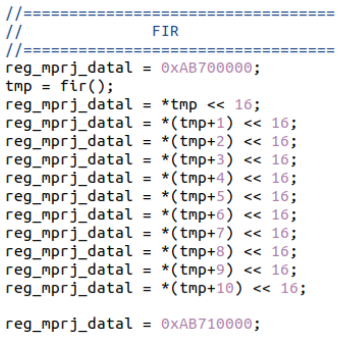
Four workloads are integrated into a .c file:



```c
#ifdef USER_PROJ_IRQ0_EN
        // unmask USER_IRQ_0_INTERRUPT
        mask = irq_getmask();
        mask |= 1 << USER_IRQ_0_INTERRUPT; // USER_IRQ_0_INTERRUPT = 2
        irq_setmask(mask);
        // enable user_irq_0_ev_enable
        user_irq_0_ev_enable_write(1);
#endif


        //=================================
        //      Matrix Multiplication
        //=================================
        int *tmp = matmul();
        reg_mprj_datal = *tmp << 16;
        reg_mprj_datal = *(tmp+1) << 16;
        reg_mprj_datal = *(tmp+2) << 16;
        reg_mprj_datal = *(tmp+3) << 16;

        reg_mprj_datal = *(tmp+9) << 16;
        reg_mprj_datal = 0xAB510000;
```



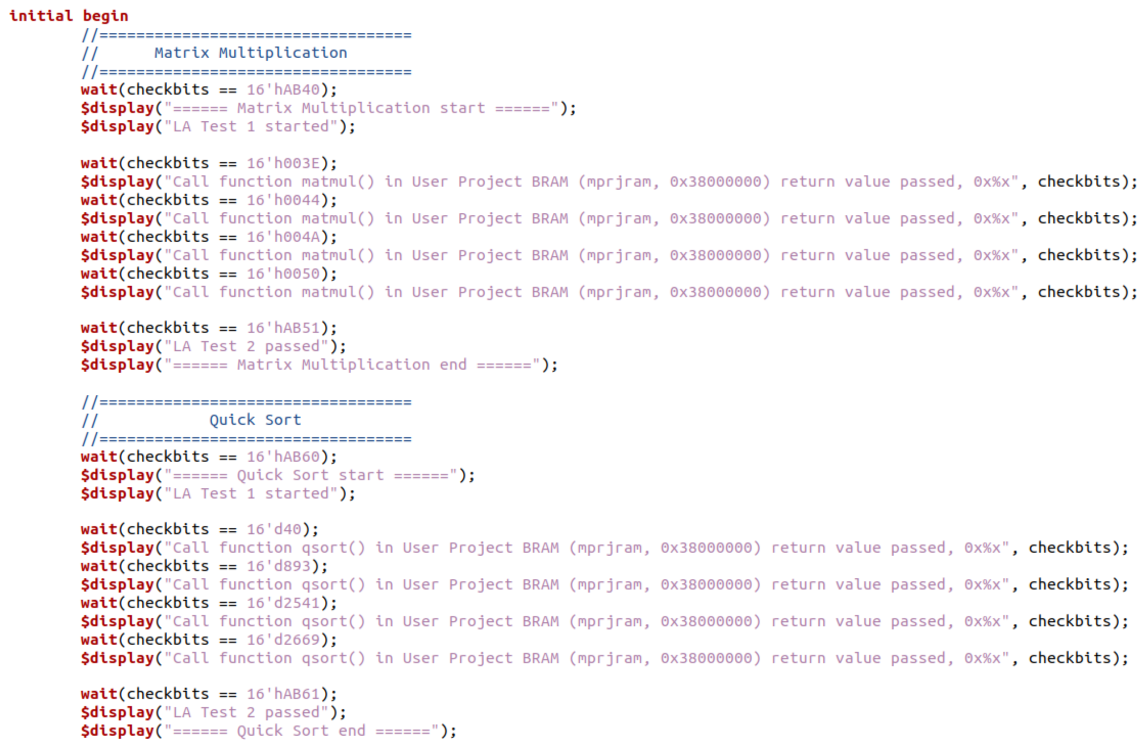
```c
//=================================
//          Quick Sort
//=================================
reg_mprj_datal = 0xAB600000;
tmp = qsort();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;

reg_mprj_datal = 0xAB610000;
reg_mprj_datal = *tmp << 16;
```



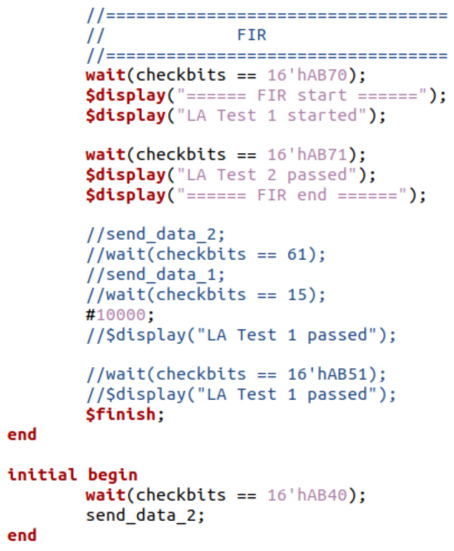
```c
//=================================
//              FIR
//=================================
reg_mprj_datal = 0xAB700000;
tmp = fir();
reg_mprj_datal = *tmp << 16;
reg_mprj_datal = *(tmp+1) << 16;
reg_mprj_datal = *(tmp+2) << 16;
reg_mprj_datal = *(tmp+3) << 16;
reg_mprj_datal = *(tmp+4) << 16;
reg_mprj_datal = *(tmp+5) << 16;
reg_mprj_datal = *(tmp+6) << 16;
reg_mprj_datal = *(tmp+7) << 16;
reg_mprj_datal = *(tmp+8) << 16;
reg_mprj_datal = *(tmp+9) << 16;
reg_mprj_datal = *(tmp+10) << 16;

reg_mprj_datal = 0xAB710000;
```

Testbench

The testbench will check if the four workloads are complete successfully by checking the check bits that we design in firmware. To ensure the uart functions correctly, we let the uart sending data in parallel with other workloads.



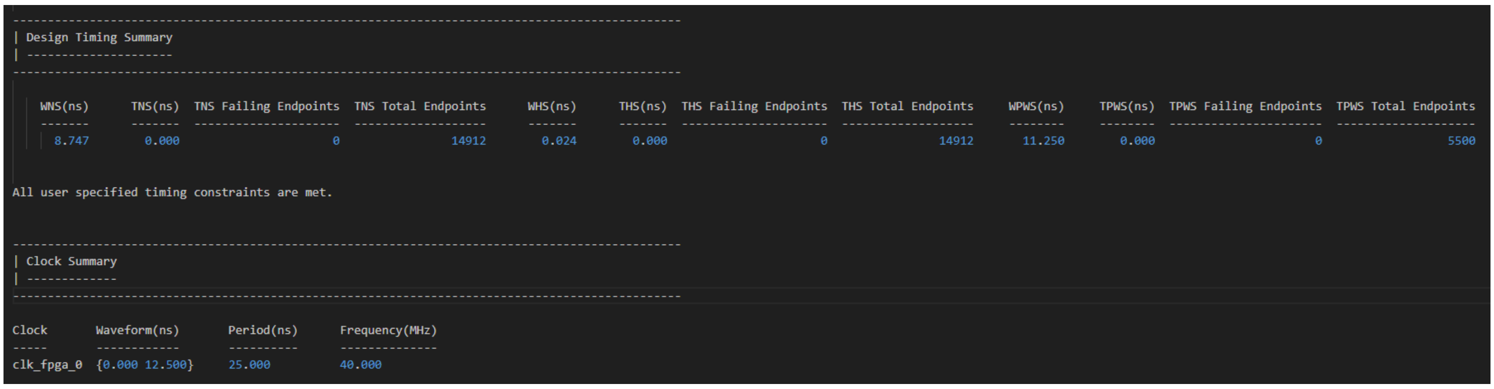
```
initial begin
        //================================
        //      Matrix Multiplication
        //================================
        wait(checkbits == 16'hAB40);
        $display("====== Matrix Multiplication start ======");
        $display("LA Test 1 started");

        wait(checkbits == 16'h003E);
        $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'h0044);
        $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'h004A);
        $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'h0050);
        $display("Call function matmul() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);

        wait(checkbits == 16'hAB51);
        $display("LA Test 2 passed");
        $display("====== Matrix Multiplication end ======");

        //================================
        //            Quick Sort
        //================================
        wait(checkbits == 16'hAB60);
        $display("====== Quick Sort start ======");
        $display("LA Test 1 started");

        wait(checkbits == 16'd40);
        $display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'd893);
        $display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'd2541);
        $display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);
        wait(checkbits == 16'd2669);
        $display("Call function qsort() in User Project BRAM (mprjram, 0x38000000) return value passed, 0x%x", checkbits);

        wait(checkbits == 16'hAB61);
        $display("LA Test 2 passed");
        $display("====== Quick Sort end ======");
```

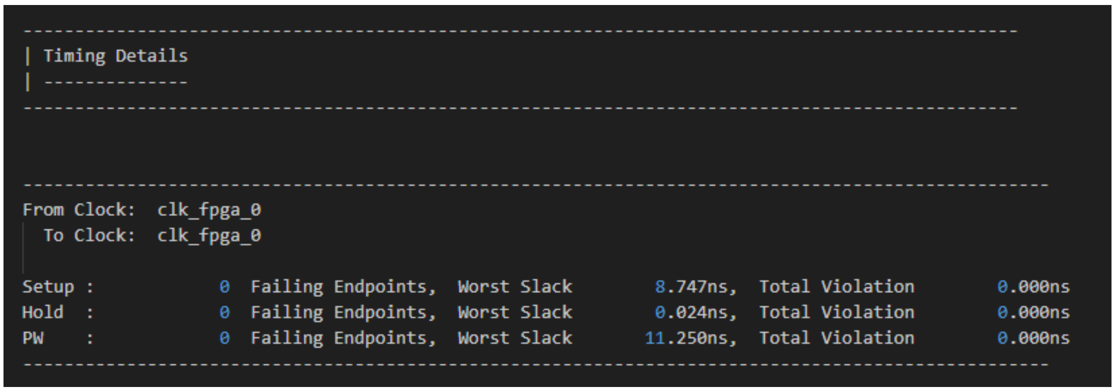


```
        //================================
        //              FIR
        //================================
        wait(checkbits == 16'hAB70);
        $display("====== FIR start ======");
        $display("LA Test 1 started");

        wait(checkbits == 16'hAB71);
        $display("LA Test 2 passed");
        $display("====== FIR end ======");

        //send_data_2;
        //wait(checkbits == 61);
        //send_data_1;
        //wait(checkbits == 15);
        #10000;
        //$display("LA Test 1 passed");

        //wait(checkbits == 16'hAB51);
        //$display("LA Test 1 passed");
        $finish;
end

initial begin
        wait(checkbits == 16'hAB40);
        send_data_2;
end
```

3. Timing report/ resource report after synthesis

Timing report


```
-----------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
-----------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------  ----------------------  --------------------
      8.747        0.000                      0                14912        0.024        0.000                      0                14912       11.250        0.000                       0                  5500


All user specified timing constraints are met.


-----------------------------------------------------------------------------------------
| Clock Summary
| -------------
-----------------------------------------------------------------------------------------

Clock       Waveform(ns)       Period(ns)      Frequency(MHz)
-----       ------------       ----------      --------------
clk_fpga_0  {0.000 12.500}     25.000          40.000
```

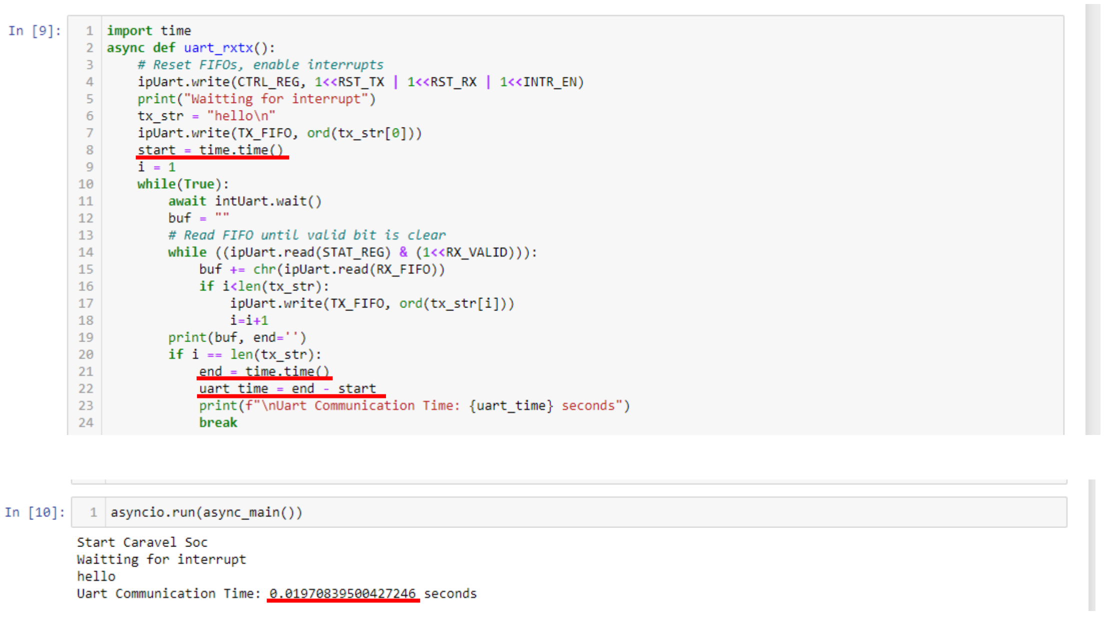

```
-----------------------------------------------------------------------------------------
| Timing Details
| --------------
-----------------------------------------------------------------------------------------


-----------------------------------------------------------------------------------------
From Clock:  clk_fpga_0
  To Clock:  clk_fpga_0

Setup :            0  Failing Endpoints,  Worst Slack        8.747ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.024ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack       11.250ns,  Total Violation        0.000ns
-----------------------------------------------------------------------------------------
```

Resource report

```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs                 | 5985 |     0 |          0 |     53200 | 11.25 |
|   LUT as Logic             | 5797 |     0 |          0 |     53200 | 10.90 |
|   LUT as Memory            |  188 |     0 |          0 |     17400 |  1.08 |
|     LUT as Distributed RAM |   18 |     0 |            |           |       |
|     LUT as Shift Register  |  170 |     0 |            |           |       |
| Slice Registers            | 6269 |     0 |          0 |    106400 |  5.89 |
|   Register as Flip Flop    | 6269 |     0 |          0 |    106400 |  5.89 |
|   Register as Latch        |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                   |  169 |     0 |          0 |     26600 |  0.64 |
| F8 Muxes                   |   47 |     0 |          0 |     13300 |  0.35 |
+----------------------------+------+-------+------------+-----------+-------+


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 283   |          Yes |           - |          Set |
| 1031  |          Yes |           - |        Reset |
| 130   |          Yes |         Set |            - |
| 4825  |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
2. Slice Logic Distribution
---------------------------

+--------------------------------------------+------+-------+------------+-----------+-------+
|                  Site Type                 | Used | Fixed | Prohibited | Available | Util% |
+--------------------------------------------+------+-------+------------+-----------+-------+
| Slice                                      | 2573 |     0 |          0 |     13300 | 19.35 |
|   SLICEL                                   | 1773 |     0 |            |           |       |
|   SLICEM                                   |  800 |     0 |            |           |       |
| LUT as Logic                               | 5797 |     0 |          0 |     53200 | 10.90 |
|   using O5 output only                     |    0 |       |            |           |       |
|   using O6 output only                     | 4313 |       |            |           |       |
|   using O5 and O6                          | 1484 |       |            |           |       |
| LUT as Memory                              |  188 |     0 |          0 |     17400 |  1.08 |
|   LUT as Distributed RAM                   |   18 |     0 |            |           |       |
|     using O5 output only                   |    0 |       |            |           |       |
|     using O6 output only                   |    2 |       |            |           |       |
|     using O5 and O6                        |   16 |       |            |           |       |
|   LUT as Shift Register                    |  170 |     0 |            |           |       |
|     using O5 output only                   |   43 |       |            |           |       |
|     using O6 output only                   |   81 |       |            |           |       |
|     using O5 and O6                        |   46 |       |            |           |       |
| Slice Registers                            | 6269 |     0 |          0 |    106400 |  5.89 |
|   Register driven from within the Slice    | 3091 |       |            |           |       |
|   Register driven from outside the Slice   | 3178 |       |            |           |       |
|     LUT in front of the register is unused | 2002 |       |            |           |       |
|     LUT in front of the register is used   | 1176 |       |            |           |       |
| Unique Control Sets                        |  320 |       |          0 |     13300 |  2.41 |
+--------------------------------------------+------+-------+------------+-----------+-------+


3. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |   70 |     0 |          0 |       140 | 50.00 |
|   RAMB36/FIFO*    |   67 |     0 |          0 |       140 | 47.86 |
|     RAMB36E1 only |   67 |       |            |           |       |
|   RAMB18          |    6 |     0 |          0 |       280 |  2.14 |
|     RAMB18E1 only |    6 |       |            |           |       |
+-------------------+------+-------+------------+-----------+-------+


4. DSP
------

+-----------+------+-------+------------+-----------+-------+
| Site Type | Used | Fixed | Prohibited | Available | Util% |
+-----------+------+-------+------------+-----------+-------+
| DSPs      |    0 |     0 |          0 |       220 |  0.00 |
+-----------+------+-------+------------+-----------+-------+


5. IO and GT Specific
---------------------

+-----------------------------+------+-------+------------+-----------+--------+
|          Site Type          | Used | Fixed | Prohibited | Available |  Util% |
+-----------------------------+------+-------+------------+-----------+--------+
| Bonded IOB                  |    0 |     0 |          0 |       125 |   0.00 |
| Bonded IPADs                |    0 |     0 |          0 |         2 |   0.00 |
| Bonded IOPADs               |  130 |   130 |          0 |       130 | 100.00 |
| PHY_CONTROL                 |    0 |     0 |          0 |         4 |   0.00 |
| PHASER_REF                  |    0 |     0 |          0 |         4 |   0.00 |
| OUT_FIFO                    |    0 |     0 |          0 |        16 |   0.00 |
| IN_FIFO                     |    0 |     0 |          0 |        16 |   0.00 |
| IDELAYCTRL                  |    0 |     0 |          0 |         4 |   0.00 |
| IBUFDS                      |    0 |     0 |          0 |       121 |   0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |     0 |          0 |        16 |   0.00 |
| PHASER_IN/PHASER_IN_PHY     |    0 |     0 |          0 |        16 |   0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |     0 |          0 |       200 |   0.00 |
| ILOGIC                      |    0 |     0 |          0 |       125 |   0.00 |
| OLOGIC                      |    0 |     0 |          0 |       125 |   0.00 |
+-----------------------------+------+-------+------------+-----------+--------+
6. Clocking
-----------

+------------+------+-------+------------+-----------+-------+
|  Site Type | Used | Fixed | Prohibited | Available | Util% |
+------------+------+-------+------------+-----------+-------+
| BUFGCTRL   |    6 |     0 |          0 |        32 | 18.75 |
| BUFIO      |    0 |     0 |          0 |        16 |  0.00 |
| MMCME2_ADV |    0 |     0 |          0 |         4 |  0.00 |
| PLLE2_ADV  |    0 |     0 |          0 |         4 |  0.00 |
| BUFMRCE    |    0 |     0 |          0 |         8 |  0.00 |
| BUFHCE     |    0 |     0 |          0 |        72 |  0.00 |
| BUFR       |    0 |     0 |          0 |        16 |  0.00 |
+------------+------+-------+------------+-----------+-------+
```

4. Latency for a character loop back using UART

After modifying the .ipynb file, we get the latency of the uart communication is about 0.02 seconds.



```
In [9]:
import time
async def uart_rxtx():
    # Reset FIFOs, enable interrupts
    ipUart.write(CTRL_REG, 1<<RST_TX | 1<<RST_RX | 1<<INTR_EN)
    print("Waitting for interrupt")
    tx_str = "hello\n"
    ipUart.write(TX_FIFO, ord(tx_str[0]))
    start = time.time()
    i = 1
    while(True):
        await intUart.wait()
        buf = ""
        # Read FIFO until valid bit is clear
        while ((ipUart.read(STAT_REG) & (1<<RX_VALID))):
            buf += chr(ipUart.read(RX_FIFO))
            if i<len(tx_str):
                ipUart.write(TX_FIFO, ord(tx_str[i]))
                i=i+1
        print(buf, end='')
        if i == len(tx_str):
            end = time.time()
            uart_time = end - start
            print(f"\nUart Communication Time: {uart_time} seconds")
            break
```

```
In [10]:
asyncio.run(async_main())

Start Caravel Soc
Waitting for interrupt
hello
Uart Communication Time: 0.01970839500427246 seconds
```

5. Suggestion for improving latency for UART loop back

To improve the uart latency, we can increase the baud rate to have more uart communication in one second. Also, we can add a FIFO to store the words rather than sending one word at one time, to reduce the number of interruptions.