CS-476
Real-Time Embedded Systems

Lab 3: Multiprocessors design

Elias De Smijter
SCIPER : 366670

Erik Wilhelm Widlund Mellergård
SCIPER : 361948

May 12, 2023

Contents

Introduction

This is the report for the third lab of Real-Time Embedded Systems (CS476). The goal of this lab is to understand how a multiprocessor system can regulate access to shared resources. In this lab, three methods are tested: a hardware mutex, a hardware mailbox and a hardware counter. To ensure that each subsystem can function independently of the other, the method tests are preceded by some initial single-core tests of the subsystems, conducted in parallel.

1. System description

The system, shown in figure 1, mainly consists of two subsystems that are more precisely described in section 1.1. Both of these subsystems act independently of each other, and have access to some shared resources: LED lights, switches, SDRAM, a custom counter and a clock. To regulate access to these resources, some hardware exclusion primitives are included in the system: a mutex and a mailbox. To ensure that operations on both subsystems are started simultaneously, the parallel port connected to the switches generates interrupts that are sent to both subsystem CPUs. In each subsystem, the interrupt triggers the corresponding functions to be started.

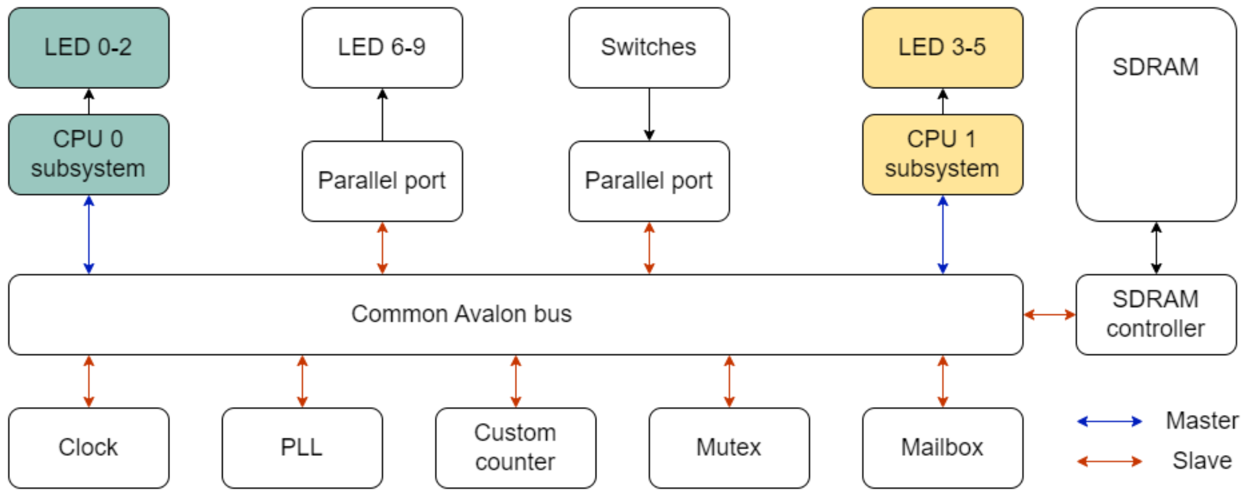


Figure 1: Overall system

1.1 Single-core subsystems

Both subsystems have the same architecture. Their components can be seen in Figure 2. There is an on-chip memory to store the program of each processor, a JTAG module to print to the console, some counters/timers and a parallel port connected to LEDs. The idea is for each subsystem to be able to perform certain actions completely independently of the other, for the purposes of the tests described in this report.

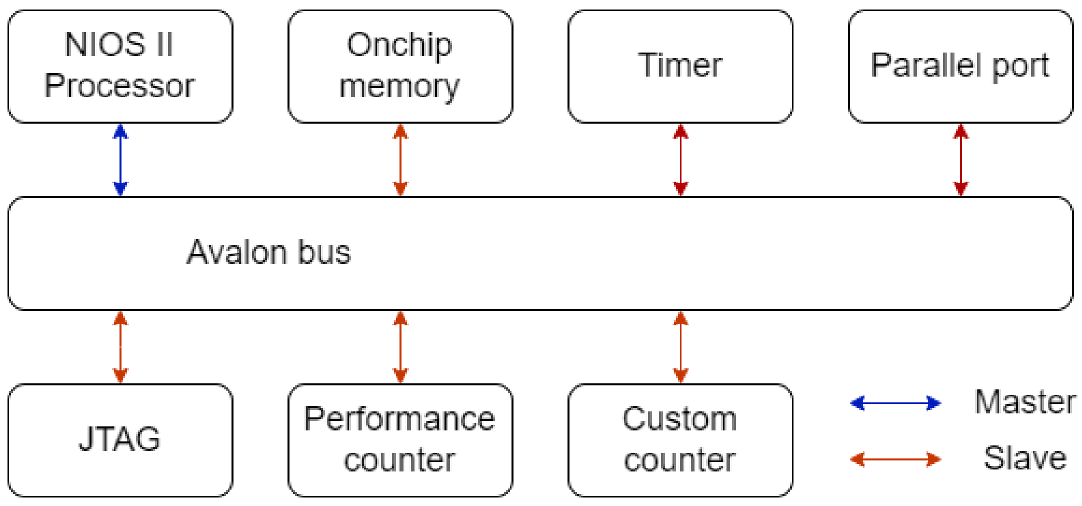


Figure 2: Single-core subsystem

1.2 Custom counter

The implemented custom counter has the key functionality of enabling incrementation and decrementation by an arbitrary amount. This is done by a write access to specific registers in the counter. Beyond this functionality, the custom counter is a straightforward up-counter that may be started, stopped, reset and loaded with a user-chosen target value. There is also the possibility to generate interrupts upon reaching the target value. The register map can be seen in Table 1, and the vhdl code in Section 5.1.

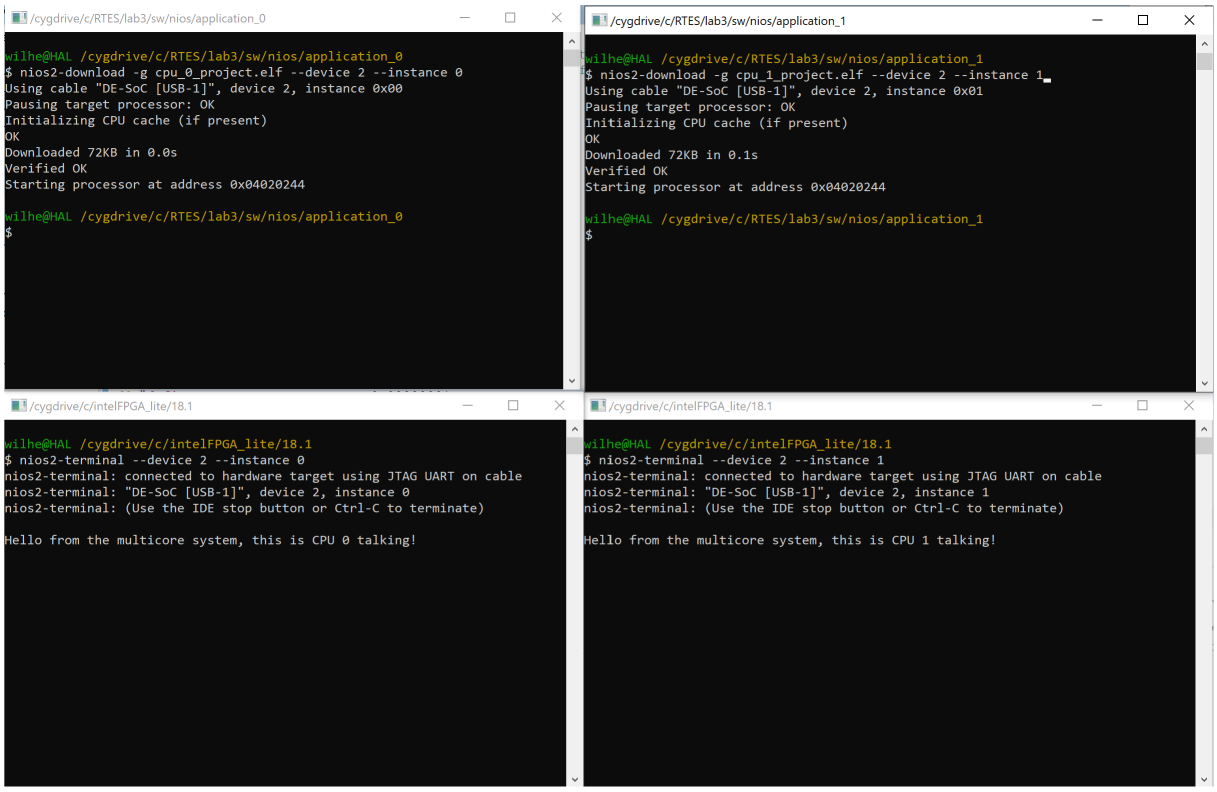
Address	Read	ReadData	Write	WriteData
000	Counter	iCounter	Decrement command	Decrementation amount
001	-	-	Reset command	DONTCARE
010	-	-	Start command	DONTCARE
011	-	-	Stop command	DONTCARE
100	IRQEn	iIRQEn	IRQEn	iIRQEn
101	Status	iEN, iEOT	ClrEOT	iClrEOT
110	Target	iTarget	Target	iTarget
111	-	-	Increment command	Incrementation amount

Table 1: Custom counter register map

1.3 Initial tests

To test if the system is all set up correctly, two tests are carried out on both of the processors. The tests only use the processor itself and the components in their subsystem, so there is no need for synchronization. Indeed, one purpose of the tests is to assert that both can work independently of each other at the same time.

In a first test, the ***printf*** function was used on both processors. Both call the argument in their *system.h* file that says which process number they have and they print it to their respective screens, see Figure 3. The test demonstrates that each processor can print messages to its terminal.



/cygdrive/c/RTES/lab3/sw/nios/application_0

```
wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$ nios2-download -g cpu_0_project.elf --device 2 --instance 0
Using cable "DE-SoC [USB-1]", device 2, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 72KB in 0.0s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$
```

/cygdrive/c/RTES/lab3/sw/nios/application_1

```
wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$ nios2-download -g cpu_1_project.elf --device 2 --instance 1
Using cable "DE-SoC [USB-1]", device 2, instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 72KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$
```

/cygdrive/c/intelFPGA_lite/18.1

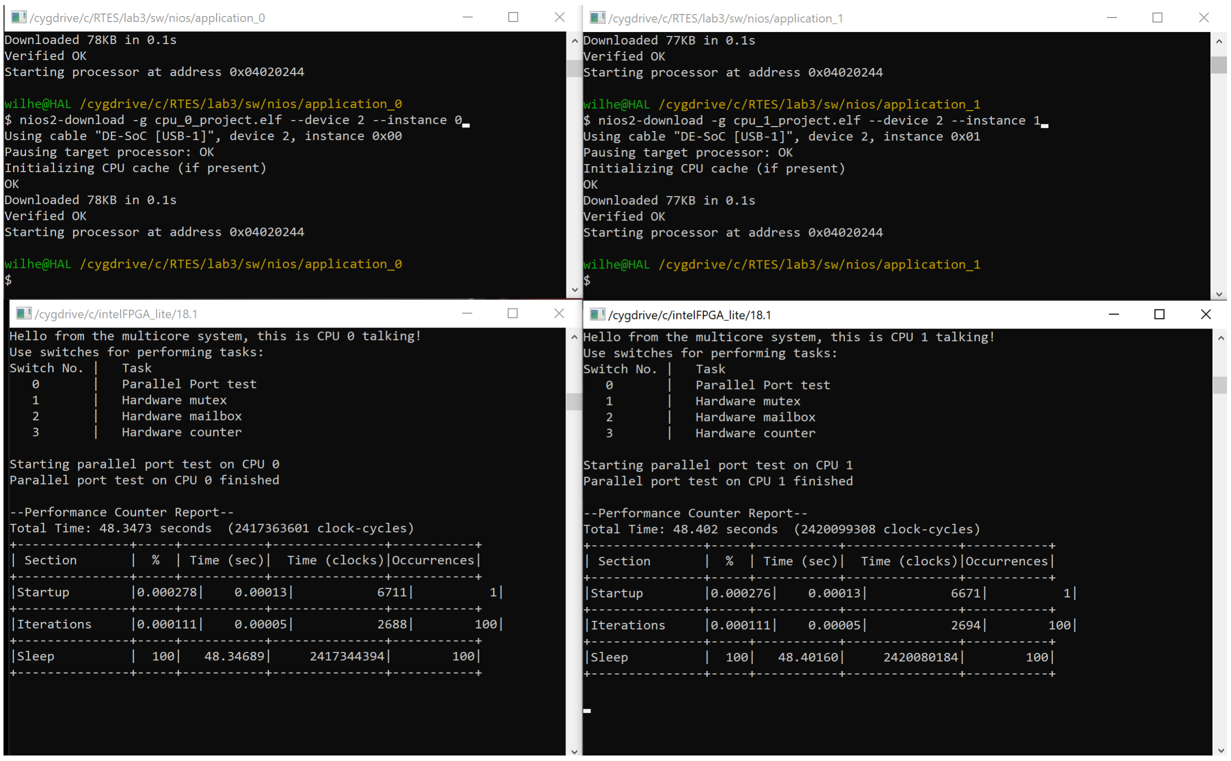
```
wilhe@HAL /cygdrive/c/intelFPGA_lite/18.1
$ nios2-terminal --device 2 --instance 0
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE-SoC [USB-1]", device 2, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello from the multicore system, this is CPU 0 talking!
```

/cygdrive/c/intelFPGA_lite/18.1

```
wilhe@HAL /cygdrive/c/intelFPGA_lite/18.1
$ nios2-terminal --device 2 --instance 1
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "DE-SoC [USB-1]", device 2, instance 1
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hello from the multicore system, this is CPU 1 talking!
```

Figure 3: System realization

A second test of parallel port access was written, to assert that each processor could access its own connected LED lights independently of the other, at the same time. The test also asserts the manual incrementation functionality of the custom counter. Each subsystem goes through the exact same steps, first initializing an instance of the custom counter, and then once every millisecond incrementing the counter and accessing the parallel port connected to the LEDs. This iterative portion of the test goes on for 100 iterations. A stock performance counter is used to make a profiling of the test, which can be seen in Figure 4. Since the same interrupt generated by flicking a switch on the board triggers both subsystems to start the parallel port test, they can be assured to start at the same time. As the figure demonstrates, both subsystems go through the test sequence in virtually the same time, spending an almost identical number of clock cycles in each portion. This verifies that the subsystems are capable of independent, parallel operation when not accessing any shared resources.



/cygdrive/c/RTES/lab3/sw/nios/application_0

```
Downloaded 78KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$ nios2-download -g cpu_0_project.elf --device 2 --instance 0
Using cable "DE-SoC [USB-1]", device 2, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 78KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$
```

/cygdrive/c/RTES/lab3/sw/nios/application_1

```
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$ nios2-download -g cpu_1_project.elf --device 2 --instance 1
Using cable "DE-SoC [USB-1]", device 2, instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$
```

/cygdrive/c/intelFPGA_lite/18.1

```
Hello from the multicore system, this is CPU 0 talking!
Use switches for performing tasks:
Switch No. |    Task
   0       |    Parallel Port test
   1       |    Hardware mutex
   2       |    Hardware mailbox
   3       |    Hardware counter

Starting parallel port test on CPU 0
Parallel port test on CPU 0 finished

--Performance Counter Report--
Total Time: 48.3473 seconds  (2417363601 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|Startup        |0.000278|    0.00013|           6711|          1|
+---------------+-----+-----------+---------------+-----------+
|Iterations     |0.000111|    0.00005|           2688|        100|
+---------------+-----+-----------+---------------+-----------+
|Sleep          |  100|   48.34689|     2417344394|        100|
+---------------+-----+-----------+---------------+-----------+
```

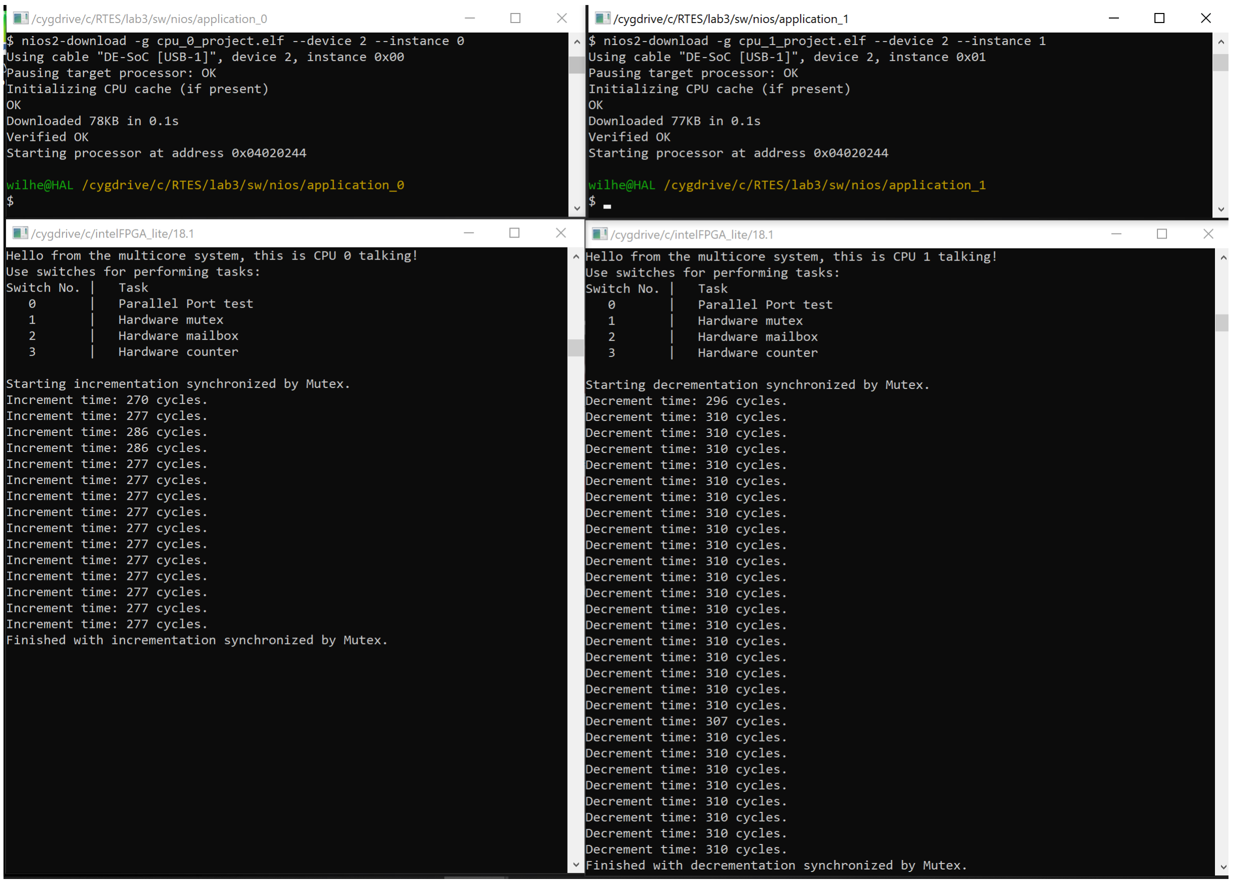
/cygdrive/c/intelFPGA_lite/18.1

```
Hello from the multicore system, this is CPU 1 talking!
Use switches for performing tasks:
Switch No. |    Task
   0       |    Parallel Port test
   1       |    Hardware mutex
   2       |    Hardware mailbox
   3       |    Hardware counter

Starting parallel port test on CPU 1
Parallel port test on CPU 1 finished

--Performance Counter Report--
Total Time: 48.402 seconds  (2420099308 clock-cycles)
+---------------+-----+-----------+---------------+-----------+
| Section       |  %  | Time (sec)|  Time (clocks)|Occurrences|
+---------------+-----+-----------+---------------+-----------+
|Startup        |0.000276|    0.00013|           6671|          1|
+---------------+-----+-----------+---------------+-----------+
|Iterations     |0.000111|    0.00005|           2694|        100|
+---------------+-----+-----------+---------------+-----------+
|Sleep          |  100|   48.40160|     2420080184|        100|
+---------------+-----+-----------+---------------+-----------+
```

Figure 4: Parallel port test

2. Hardware mutex

From this section on, both the processors try to access shared resources. To make sure this all happens in a correct fashion, synchronization primitives need to be used. If they are not used and the operation to modify the value of a shared resource takes multiple cycles (not atomic), the scheduler might switch execution at an unfortunate time and make the program behave in unexpected ways. First, the use of a hardware mutex is examined.
In this test, both processors access a counter with a parallel port. One tries to increment it every 20ms and the other one decrements it every 10ms. A hardware timer is used to measure how long it takes for each processor to gain access to the counter by acquiring the mutex (sometimes one has to wait because the mutex is locked), increment/decrement the counter and finally release the mutex. The results of this test are in figure 5.


/cygdrive/c/RTES/lab3/sw/nios/application_0
/cygdrive/c/RTES/lab3/sw/nios/application_1
/cygdrive/c/intelFPGA_lite/18.1
/cygdrive/c/intelFPGA_lite/18.1

```
$ nios2-download -g cpu_0_project.elf --device 2 --instance 0
Using cable "DE-SoC [USB-1]", device 2, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 78KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$
```

```
$ nios2-download -g cpu_1_project.elf --device 2 --instance 1
Using cable "DE-SoC [USB-1]", device 2, instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$
```

```
Hello from the multicore system, this is CPU 0 talking!
Use switches for performing tasks:
Switch No. |    Task
   0       |    Parallel Port test
   1       |    Hardware mutex
   2       |    Hardware mailbox
   3       |    Hardware counter

Starting incrementation synchronized by Mutex.
Increment time: 270 cycles.
Increment time: 277 cycles.
Increment time: 286 cycles.
Increment time: 286 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Increment time: 277 cycles.
Finished with incrementation synchronized by Mutex.
```

```
Hello from the multicore system, this is CPU 1 talking!
Use switches for performing tasks:
Switch No. |    Task
   0       |    Parallel Port test
   1       |    Hardware mutex
   2       |    Hardware mailbox
   3       |    Hardware counter

Starting decrementation synchronized by Mutex.
Decrement time: 296 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 307 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Decrement time: 310 cycles.
Finished with decrementation synchronized by Mutex.
```

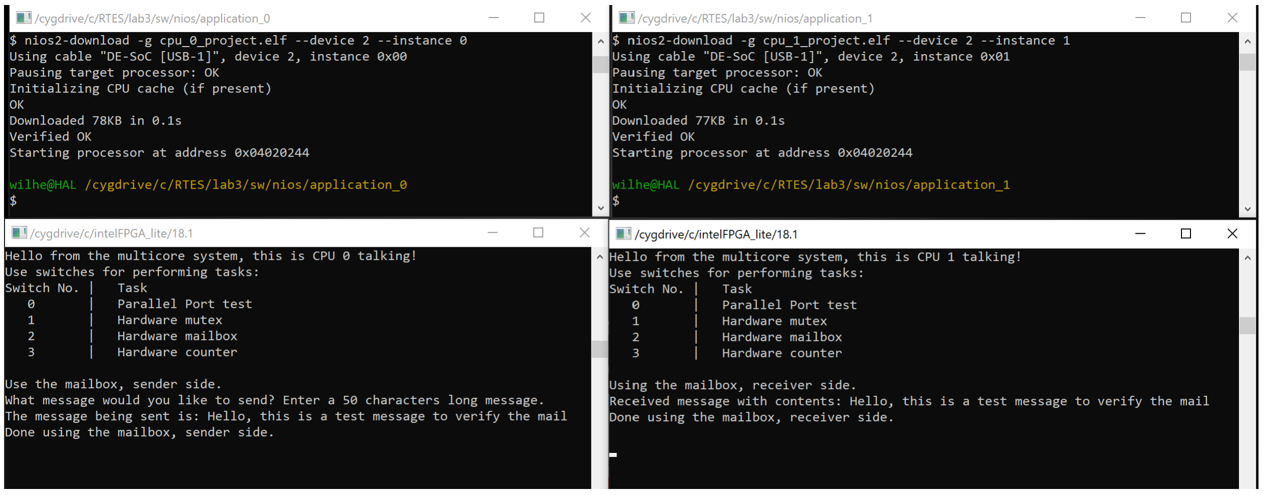
Figure 5: Hardware mutex

When looking at the timings, it is the most interesting to compare these results to the operation on the parallel port tested in section 1.3. There, the same operation was carried out, but not on a shared resource. The difference between the two implementations is thus the overhead introduced by the mutex. For CPU 0, the overhead takes around 8 cycles while for CPU 1 this is 40 cycles. The difference in overhead is due to the decrementing subsystem needing one more line of C code for its operation; storing the decremented counter value, while the incrementing subsystem does not. The base reason for this is that the counter is used for the while loop stopping condition, which would not be verified at the correct time without the mentioned extra code line.

3. Hardware mailbox

The hardware mailbox test is simply an implementation of one subsystem making certain information known to the other, using a mailbox. On the sender side, the user may enter a 50 characters long string, which is stored at a specific address on the shared RAM. Then, the sender side sends the string length and the starting address of where it's stored in the RAM to the mailbox. The receiver side polls the mailbox for messages, and when the message is received, the address where the string is stored is thus known to the receiver subsystem. Finally, the receiver subsystem reads the string from the specified RAM address, and prints it to the terminal.

As the function receive_mail() in the C code of the receiving side subsystem (see Section 5.3) shows, the receiving side has no hard-coded knowledge of either the storage starting address or the string length. These are only collected from the received mail. Therefore, the printout of the same message on both sender and receiver side, see Figure 6, demonstrates that the hardware mailbox is working as intended. For synchronization purposes, this shows that the mailbox may be used for instance when one subsystem has to wait for an arbitrary amount of time for another subsystem to perform its task upon a shared resource such as memory, before the waiting subsystem may access the same resource.


/cygdrive/c/RTES/lab3/sw/nios/application_0
/cygdrive/c/RTES/lab3/sw/nios/application_1
/cygdrive/c/intelFPGA_lite/18.1
/cygdrive/c/intelFPGA_lite/18.1

```
$ nios2-download -g cpu_0_project.elf --device 2 --instance 0
Using cable "DE-SoC [USB-1]", device 2, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 78KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$
```

```
$ nios2-download -g cpu_1_project.elf --device 2 --instance 1
Using cable "DE-SoC [USB-1]", device 2, instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$
```

```
Hello from the multicore system, this is CPU 0 talking!
Use switches for performing tasks:
Switch No. |   Task
   0       |   Parallel Port test
   1       |   Hardware mutex
   2       |   Hardware mailbox
   3       |   Hardware counter

Use the mailbox, sender side.
What message would you like to send? Enter a 50 characters long message.
The message being sent is: Hello, this is a test message to verify the mail
Done using the mailbox, sender side.
```

```
Hello from the multicore system, this is CPU 1 talking!
Use switches for performing tasks:
Switch No. |   Task
   0       |   Parallel Port test
   1       |   Hardware mutex
   2       |   Hardware mailbox
   3       |   Hardware counter

Using the mailbox, receiver side.
Received message with contents: Hello, this is a test message to verify the mail
Done using the mailbox, receiver side.

_
```

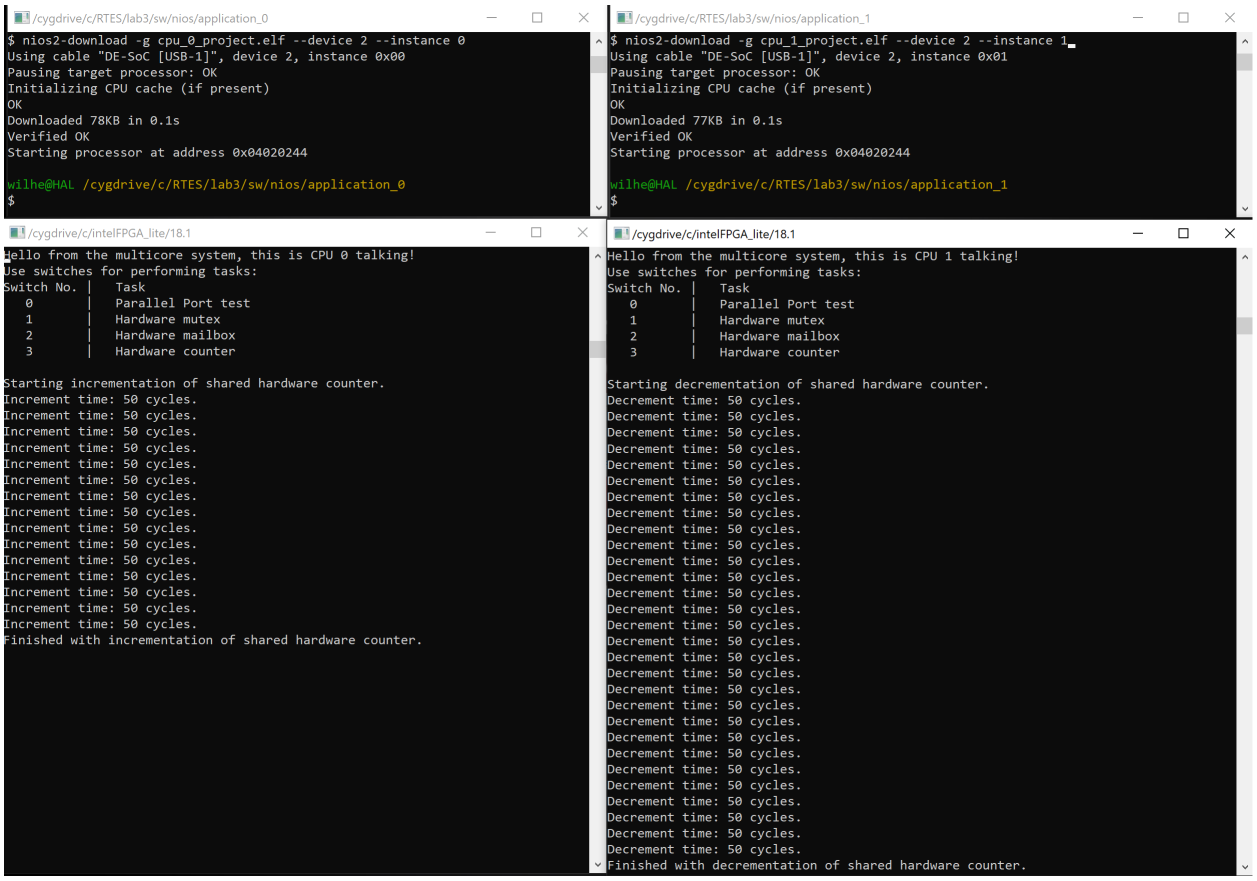
Figure 6: Hardware mailbox

4. Hardware counter

The hardware counter test is very similar to the hardware mutex test, described in Section 2. However, due to a shared instance of the custom counter being used, the action of incrementing or decrementing it is now atomic: the shared Avalon bus prevents both subsystems accessing it in the same clock cycle and the operation only takes one clock cycle. Therefore there is no need for any synchronizing primitive; the subsystems can run in parallel without risking to impede each other's operation. The result, which can be seen in Figure 7, demonstrates a much faster cycle: the improvement is roughly of a factor 5.

Furthermore, the difference in cycle times for the different subsystems that was seen in the hardware mutex test doesn't appear here. This is due to the counter now being a shared hardware resource, while the mutex test had separate software counters for each subsystem, and only a shared parallel port resource. The decrementation cycle now no longer requires an extra instruction.

The lesson here is that synchronization by hardware mutex becomes unnecessary when the shared resource access is atomic as in this case. Also, the use of a hardware mutex can be seen to impose a substantial overhead in terms of time. Put together, this shows that atomic access to shared resources is definitely preferable when possible.



/cygdrive/c/RTES/lab3/sw/nios/application_0

```
$ nios2-download -g cpu_0_project.elf --device 2 --instance 0
Using cable "DE-SoC [USB-1]", device 2, instance 0x00
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 78KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_0
$
```

/cygdrive/c/RTES/lab3/sw/nios/application_1

```
$ nios2-download -g cpu_1_project.elf --device 2 --instance 1
Using cable "DE-SoC [USB-1]", device 2, instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 77KB in 0.1s
Verified OK
Starting processor at address 0x04020244

wilhe@HAL /cygdrive/c/RTES/lab3/sw/nios/application_1
$
```

/cygdrive/c/intelFPGA_lite/18.1

```
Hello from the multicore system, this is CPU 0 talking!
Use switches for performing tasks:
Switch No. |   Task
   0       |   Parallel Port test
   1       |   Hardware mutex
   2       |   Hardware mailbox
   3       |   Hardware counter

Starting incrementation of shared hardware counter.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Increment time: 50 cycles.
Finished with incrementation of shared hardware counter.
```

/cygdrive/c/intelFPGA_lite/18.1

```
Hello from the multicore system, this is CPU 1 talking!
Use switches for performing tasks:
Switch No. |   Task
   0       |   Parallel Port test
   1       |   Hardware mutex
   2       |   Hardware mailbox
   3       |   Hardware counter

Starting decrementation of shared hardware counter.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Decrement time: 50 cycles.
Finished with decrementation of shared hardware counter.
```

Figure 7: Hardware counter

5. Annex 1: Source code

5.1 Custom counter

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Register map:
-- 000 = Counter[31... 0]    Counter value (read access)
--          Decrement by the amount in WriteData once (write access)
-- 001 = Rz         Reset the counter (write access)
-- 010 = Start         Start counting (write access)
-- 011 = Stop         Stop counting (write access)
-- 100 = Command[7... 0]    General command (RW access)
-- 101 = Status[7... 0]      General status (RW access)
-- 110 = Target[31... 0]    Count target value (RW access)
-- 111 = IncrVal[31... 0]    Increment by the amount in WriteData once (write access)

entity Counter is
  PORT(
    Clk       :  IN    std_logic;
    nReset     :  IN    std_logic;
    -- Avalon Slave interface signals
    Address     :  IN    std_logic_vector(2 downto 0);
    Read       :  IN    std_logic;
    ReadData     :  OUT  std_logic_vector(31 downto 0);
    Write       :  IN    std_logic;
    WriteData   :  IN   std_logic_vector(31 downto 0);
    -- Interrupt Request signal
    IRQ       :  OUT  std_logic
  );
end Counter;

architecture comp of Counter is
  -- Counter
  signal   iCounter   :  unsigned(31 downto 0);
  -- Enable count
  signal  iEn     :  std_logic;
  -- Reset count
  signal  iRz     :  std_logic;
  -- Has reached end
  signal  iEOT     :  std_logic;
  -- Clear has reached end
  signal  iClrEOT   :  std_logic;
  -- Interrupts enabled
  signal  iIRQEn   :  std_logic;
  -- Count target value
  signal  iTarget   :  std_logic_vector(31 downto 0);
  -- Increment/Decrement write flag
  signal  iIncr     :   std_logic;
  signal  iDecr     :  std_logic;
```

```vhdl
signal  iIncrVal  :  std_logic_vector(31 downto 0);

begin
-- Counter process
-- Drives iCounter
Counter_process:
process(Clk)
begin
  if rising_edge(Clk) then
    if iRz = '1' then
      -- Reset counter to zero
      iCounter <= (others => '0');
    elsif iEn = '1' then
      if iEOT = '0' then
        -- Increment counter by 1
        iCounter <= iCounter + 1;
      end if;
    elsif iIncr = '1' then
        -- Increment count
        iCounter <= iCounter + unsigned(iIncrVal);
    elsif iDecr = '1' then
        -- Decrement count
        iCounter <= iCounter - unsigned(iIncrVal);
    end if;
  end if;
end process Counter_process;

-- Read process
-- Drives no signals
Read_process:
process(Clk)
begin
  if rising_edge(Clk) then
    -- Default value

    -- TODO: should the default value be std_logic_vector(iCounter)
    -- to facilitate that
    -- "The counter value must be readable at all times -> transfer
     --   the counter value at the start of the read cycle"

    ReadData <= (others => '0');
    -- Read cycle
    if Read = '1' then
      case Address(2 downto 0) is
        when "000" =>
          -- Read the counter register value
          ReadData <= std_logic_vector(iCounter);
        when "100" =>
          -- Read the command register value (which is only
          -- the iIRQEn value)
          ReadData(0) <= iIRQEn;
        when "101" =>
          -- Read the status register value (iEOT and iEn)
          ReadData(0) <= iEOT;
          ReadData(1) <= iEn;
        when "110" =>
          -- Read the count target value
          ReadData <= std_logic_vector(iTarget);
        when others => null;
      end case;
    end if;
  end if;
end process Read_process;

-- Write process (also handles asynchronous reset)
-- Drives iEn, iRz, iIRQEn and iClrEOT
Write_process:
process(Clk, nReset)
begin
  -- Asynchronous reset: counting disabled, counter is reset, interrupts disabled
  if nReset = '0' then
    iEn <= '0';
    iRz <= '1';
    iIncr <= '0';
    iDecr <= '0';
    iIRQEn <= '0';
    iClrEOT <= '1';
  elsif rising_edge(Clk) then
    -- Default values: don't reset or clear
    iRz <= '0';
    iClrEOT <= '0';
    iIncr <= '0';
    iDecr <= '0';
    -- Write cycle
    if Write = '1' then
      case Address(2 downto 0) is
        -- Commands: Writing to them just means to perform the
        -- command associated with the register, so WriteData
        -- is not considered.
        when "001" =>
          -- Rz: reset counter command
          iRz <= '1';
          iClrEOT <= '1';
        when "010" =>
          -- Start: enable counting command
          iEn <= '1';
        when "011" =>
          -- Stop: disable counting command
          iEn <= '0';
        -- Registers: WriteData is considered
        when "100" =>
          -- Command register
          iIRQEn <= WriteData(0);
        when "101" =>
          -- Status register
          iClrEOT <= WriteData(0);
          when "110" =>
            -- Target value register
            iTarget <= WriteData;
            iClrEOT <= '1';
          when "111" =>
            iIncrVal <= WriteData;
            iIncr <= '1';
          when "000" =>
            iIncrVal <= WriteData;
            iDecr <= '1';
          when others => null;
        end case;
      end if;
    end if;
  end process Write_process;

  -- Interrupt process. End Of Time (iEOT) is activated when iCounter
  -- is at the value in iTarget
  -- Drives iEOT
  Interrupt_process:
  process(Clk)
  begin
    if rising_edge(Clk) then
      if std_logic_vector(iCounter) = iTarget then
        -- Flag End Of Time when counter = target
        iEOT <= '1';
      end if;
      if iClrEOT = '1' then
        -- Cleared by writing 1 to status(0)
        iEOT <= '0';
      end if;
    end if;
  end process Interrupt_process;

  -- IRQ activation. No process required since it's only one signal
  -- conditionally activated
  IRQ <= '1' when (iEOT = '1' and iIRQEn = '1' and iEN = '1') else '0';

end comp;
```

5.2 CPU_0 C program

```
/*
 *--------------------------------- THIS IS CPU 0 -------------------------------------------
 * Main code for the cpu 0 subsystem
 */

#include <stdio.h>
#include "system.h"
#include "io.h"
#include <stdint.h>
#include <unistd.h>
#include <string.h>
```

```c
#include <altera_avalon_mutex.h>
#include "altera_avalon_mailbox_simple.h"
#include <altera_avalon_performance_counter.h>

/*
 *---------------------------------- THIS IS CPU 0 ------------------------------------------------
 */

// To monitor terminal in powershell:
// nios2-terminal --device 2 --instance 0

// To download code to board
// nios2-download -g cpu_0_project.elf --device 2 --instance 0

#define ArbVal 0xffffffff

// Custom counter address offsets
#define CustomCounterValue 0  // For _reading_ access, address 0 is the current counter value
#define CustomCounterDecr 0    // For _writing_ access, address 0 is the command to Decrement
#define CustomCounterReset 4
#define CustomCounterStart  8
#define CustomCounterStop  12
#define CustomCounterCommand  16
#define CustomCounterStatus  20
#define CustomCounterTarget 24
#define CustomCounterIncr 28

// PIO definitions
#define PIO_Data  0
#define PIO_IRQEN  4*2
#define PIO_IRQFLAG  4*3
// Variable for recording choices made by interrupt
volatile int choice;

// Message max length
#define MSG_MAX    50

void part1()
{
  printf("Hello from the multicore system, this is CPU %d talking! \n", NIOS2_CPU_ID_VALUE);
}

// Access parallel port 0 connected to LEDs 3, 2 & 1
// and increment counter 0 every mscound milliseconds
// for iterations amount of times.
void pptest(int msdelay, uint32_t iterations)
{
  // Start overall performance counter
  PERF_RESET(PERFORMANCE_COUNTER_0_BASE);
  PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
  // Start performance counter, recording the setup portion (1)
  PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 1);
  printf("Starting parallel port test on CPU %d\n", NIOS2_CPU_ID_VALUE);
  uint32_t itercount = 0x03;
  uint8_t curr_lamp = 0x03;
  // Reset custom counter
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterReset, ArbVal);
  // Set target value to be larger than iterations to be sure we never reach it (avoid trigger
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterTarget, iterations + 1);
  // Stop performance counter for the setup portion (1)
  PERF_END(PERFORMANCE_COUNTER_0_BASE, 1);
  while(itercount < iterations)
  {
    // Start performance counter, recording the iterative work (2)
    PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 2);
    // Access parallel port to toggle LED
    IOWR_8DIRECT(PARALLEL_PORT_0_BASE, 0, curr_lamp);
    // Increment counter
    IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterIncr, 0x01);
    // Make sure to toggle LEDs correctly for 3 available
    if(curr_lamp < 0x07)
    {
      curr_lamp = curr_lamp + 0x01;
    }
    else
    {
      curr_lamp = 0x00;
    }
    // Stop performance counter for the iterative work (2)
    PERF_END(PERFORMANCE_COUNTER_0_BASE, 2);
    // Start performance counter, recording the wait (3)
    PERF_BEGIN(PERFORMANCE_COUNTER_0_BASE, 3);
    // Wait msdelay ms
    usleep(1000 * msdelay);
    // Stop performance counter for the wait (3)
    PERF_END(PERFORMANCE_COUNTER_0_BASE, 3);
    // Get count value
    itercount = IORD_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterValue);
  }
  printf("Parallel port test on CPU %d finished\n\n", NIOS2_CPU_ID_VALUE);
  // Stop overall performance counter and print results
  PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
  perf_print_formatted_report(PERFORMANCE_COUNTER_0_BASE, alt_get_cpu_freq(), 3, "Startup", "I
  printf("\n");
}

void hwmutex(uint8_t startvalue)
{
  // Initialize counter and timer variables
  uint8_t counter = 0x1;
  uint32_t timer_start = 0x0;
  uint32_t timer_stop = 0x0;
  // Fetch mutex address
  alt_mutex_dev* common_mutex = altera_avalon_mutex_open(MUTEX_0_NAME);
  // Setup start value
  altera_avalon_mutex_lock(common_mutex, 1);
  IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, startvalue);
  altera_avalon_mutex_unlock(common_mutex);
  // Setup and start custom counter
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterReset, ArbVal);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterCommand, 0x0);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterTarget, 0xffffffff);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterStart, ArbVal);
  printf("Starting incrementation synchronized by Mutex.\n");
  while(counter > 0x0)
  {
    // Increment every 20 ms
    usleep(20000);
    // Read start value
    timer_start = IORD_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterValue);
    altera_avalon_mutex_lock(common_mutex, 1);
    counter = IORD_8DIRECT(PARALLEL_PORT_2_BASE, 0);
    IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, counter + 1);
    altera_avalon_mutex_unlock(common_mutex);
    timer_stop = IORD_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterValue);
    printf("Increment time: %ld cycles.\n", (timer_stop - timer_start));

  }
  // Make sure the LED is turned off when it's all done
  altera_avalon_mutex_lock(common_mutex, 1);
  IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, 0);
  altera_avalon_mutex_unlock(common_mutex);
  printf("Finished with incrementation synchronized by Mutex.\n\n");
}

void hwcounter()
{
  // Initialize counter and timer variables
  uint8_t LED_Counter = 0x0f;
  uint32_t timer_start = 0x0;
  uint32_t timer_stop = 0x0;
  // Setup start value to LEDs
  IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, LED_Counter);
  // Setup custom counter 2, which can be accessed by both CPUs
  IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterReset, ArbVal);
  IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterCommand, 0x0);
  IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterTarget, 0xffffffff);
  IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterIncr, LED_Counter);
  // Setup and start custom counter 0, used for measuring how long access takes
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterReset, ArbVal);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterCommand, 0x0);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterTarget, 0xffffffff);
  IOWR_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterStart, ArbVal);
  printf("Starting incrementation of shared hardware counter.\n");
  while(LED_Counter > 0x0)
  {
    // Increment every 20 ms
    usleep(20000);
    // Read start value
    timer_start = IORD_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterValue);

    // Increment counter 2
    IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterIncr, 0x01);
    // Read counter 2 value
    LED_Counter = (uint8_t)(IORD_32DIRECT(CUSTOM_COUNTER_2_BASE, 0) & 0xFF);


    // Display new value on LED
    IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, LED_Counter);

    timer_stop = IORD_32DIRECT(CUSTOM_COUNTER_0_BASE, CustomCounterValue);
    printf("Increment time: %ld cycles.\n", (timer_stop - timer_start));
    if (LED_Counter == 0x01)
    {
      break;
    }

  }
  // Make sure the LED is turned off when it's all done
  IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, 0);
  printf("Finished with incrementation of shared hardware counter.\n\n");
}

void send_callback(void* report, int status)
{
  if(!status)
  {
    printf("Sending completed.\n");
  }
  else
  {
    printf("Sending error.\n");
  }
}

void send_mail()
{
  printf("Use the mailbox, sender side.\n");
  // Load mailbox
  altera_avalon_mailbox_dev* mailbox = altera_avalon_mailbox_open("/dev/mailbox_simple_0", sen
  // Create mail "envelope"
  alt_u32 mail[2] = {0x0, 0x0};
  // Create pointer and message
  char *msg_ptr = SDRAM_CONTROLLER_2_BASE;
  char message[MSG_MAX];
  // Ask user for message
  printf("What message would you like to send? Enter a %i characters long message.\n", MSG_MAX
  fgets(message, sizeof(message), stdin);


  printf("The message being sent is: ");
  for(int i = 0; i < MSG_MAX; i++)
  {
    if(message[i] == '/')
    {
      break;
    }
    *(msg_ptr + i) = message[i];
    mail[0] = i + 1;
    printf("%c", message[i]);
  }
  printf("\n");
  mail[1] = (alt_u32) msg_ptr;
  alt_dcache_flush_all();
  // Send message
  altera_avalon_mailbox_send(mailbox, mail, 0, POLL);
  // Close mailbox
  altera_avalon_mailbox_close(mailbox);
  printf("Done using the mailbox, sender side.\n\n");
}

void choose_task(int task)
{
  if(task == 1)
  {
    // 3.1 Parallel port test, 100 iterations, 50 ms each
    uint32_t iters = 0x64;
    int delaytime = 50;
    pptest(delaytime, iters);
  }
  else if(task == 2)
  {
    // Manipulation 2: hardware mutex
    hwmutex(0x0f);
  }
  else if(task == 4)
  {
    // Manipulation 3: hardware mailbox
    send_mail();
  }
  else if(task == 8)
  {
    // Manipulation 4: hardware counter
    hwcounter();
  }
}

void isr_buttons(void* context)
{
  uint8_t pinvals = IORD_8DIRECT(PIO_2_BASE, PIO_IRQFLAG);
  choice = (int)pinvals;
  // Clear the interrupt flag
  IOWR_8DIRECT(PIO_2_BASE, PIO_IRQFLAG, pinvals);
}
```

```
int main()
{
  // Setup interrupts on input pins
  IOWR_8DIRECT(PIO_2_BASE, PIO_IRQEN, 0xff);
  alt_ic_isr_register(PIO_2_IRQ_INTERRUPT_CONTROLLER_ID, PIO_2_IRQ, isr_buttons, NULL, NULL);
  // Print which CPU it is
  choose_task(1);
  printf("Use switches for performing tasks:\nSwitch No. |   Task\n   0       |   Parallel Por
  // Wait for buttons
  while(1)
  {
    // Poll choice once every millisecond
    usleep(1000);
    if(choice != 0x0)
    {
      choose_task(choice);
      choice = 0x0;
    }
  };
  return 0;
}

/*
 *--------------------------------- THIS IS CPU 0 ---------------------------------------------
 */
```

5.3 CPU_1 C program

```
/*
 *--------------------------------- THIS IS CPU 1 ---------------------------------------------
 * Main code for the cpu 1 subsystem
 */

#include <stdio.h>
#include "system.h"
#include "io.h"
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <altera_avalon_mutex.h>
#include "altera_avalon_mailbox_simple.h"
#include <altera_avalon_performance_counter.h>

/*
 *--------------------------------- THIS IS CPU 1 ---------------------------------------------
 */
// REMEMBER TO CHANGE THE CPU ID VALUE IN SYSTEM.H AFTER GENERATING BSP

// To monitor terminal in powershell:
// nios2-terminal --device 2 --instance 1

// To download code to board
// nios2-download -g cpu_1_project.elf --device 2 --instance 1
```

```
#define ArbVal 0xffffffff

// Custom counter address offsets
#define CustomCounterValue 0  // For _reading_ access, address 0 is the current counter value
#define CustomCounterDecr 0    // For _writing_ access, address 0 is the command to Decrement
#define CustomCounterReset 4
#define CustomCounterStart  8
#define CustomCounterStop  12
#define CustomCounterCommand  16
#define CustomCounterStatus  20
#define CustomCounterTarget 24
#define CustomCounterIncr 28

// PIO definitions
#define PIO_Data  0
#define PIO_IRQEN  4*2
#define PIO_IRQFLAG  4*3
// Variable for recording choices made by interrupt
volatile int choice;

void part1()
{
  printf("Hello from the multicore system, this is CPU %d talking! \n", NIOS2_CPU_ID_VALUE);
}

// Access parallel port 1 connected to LEDs 6, 5 & 4
// and increment counter 1 every mscound milliseconds
// for iterations amount of times.
void pptest(int msdelay, uint32_t iterations)
{
  // Start overall performance counter
  PERF_RESET(PERFORMANCE_COUNTER_1_BASE);
  PERF_START_MEASURING(PERFORMANCE_COUNTER_1_BASE);
  // Start performance counter, recording the setup portion (1)
  PERF_BEGIN(PERFORMANCE_COUNTER_1_BASE, 1);
  printf("Starting parallel port test on CPU %d\n", NIOS2_CPU_ID_VALUE);
  uint32_t itercount = 0x00;
  uint8_t curr_lamp = 0x00;
  // Reset custom counter
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterReset, ArbVal);
  // Set target value to be larger than iterations to be sure we never reach it (avoid trigger
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterTarget, iterations + 1);
  // Stop performance counter for the setup portion (1)
  PERF_END(PERFORMANCE_COUNTER_1_BASE, 1);
  while(itercount < iterations)
  {
    // Start performance counter, recording the iterative work (2)
    PERF_BEGIN(PERFORMANCE_COUNTER_1_BASE, 2);
    // Access parallel port to toggle LED
    IOWR_8DIRECT(PARALLEL_PORT_1_BASE, 0, curr_lamp);
    // Increment counter
    IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterIncr, 0x01);
    // Make sure to toggle LEDs correctly for 3 available
    if(curr_lamp < 0x07)
    {
      curr_lamp = curr_lamp + 0x01;
    }
    else
    {
      curr_lamp = 0x00;
    }
    // Stop performance counter for the iterative work (2)
    PERF_END(PERFORMANCE_COUNTER_1_BASE, 2);
    // Start performance counter, recording the wait (3)
    PERF_BEGIN(PERFORMANCE_COUNTER_1_BASE, 3);
    // Wait msdelay ms
    usleep(1000 * msdelay);
    // Stop performance counter for the wait (3)
    PERF_END(PERFORMANCE_COUNTER_1_BASE, 3);
    // Get count value
    itercount = IORD_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterValue);
  }
  printf("Parallel port test on CPU %d finished\n\n", NIOS2_CPU_ID_VALUE);
  // Stop overall performance counter and print results
  PERF_STOP_MEASURING(PERFORMANCE_COUNTER_1_BASE);
  perf_print_formatted_report(PERFORMANCE_COUNTER_1_BASE, alt_get_cpu_freq(), 3, "Startup", "I
  printf("\n");
}

void hwmutex()
{
  // Initialize counter and timer variables
  uint8_t counter = 0x1;
  uint32_t timer_start = 0x0;
  uint32_t timer_stop = 0x0;
  // Fetch mutex address
  alt_mutex_dev* common_mutex = altera_avalon_mutex_open(MUTEX_0_NAME);
  // Wait for cpu_0 to finish setting up
  while(!altera_avalon_mutex_first_lock(common_mutex)){};
  // Setup and start custom counter
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterReset, ArbVal);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterCommand, 0x0);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterTarget, 0xffffffff);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterStart, ArbVal);
  printf("Starting decrementation synchronized by Mutex.\n");
  while(counter > 0x0)
  {
    // Decrement every 10 ms
    usleep(10000);
    timer_start = IORD_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterValue);
    altera_avalon_mutex_lock(common_mutex, 2);
    counter = IORD_8DIRECT(PARALLEL_PORT_2_BASE, 0);
    counter = counter - 1;
    IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, counter);
```

```c
    altera_avalon_mutex_unlock(common_mutex);
    timer_stop = IORD_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterValue);
    printf("Decrement time: %ld cycles.\n", (timer_stop - timer_start));
  }
  printf("Finished with decrementation synchronized by Mutex.\n\n");
}

void hwcounter()
{
  // Initialize counter and timer variables
  uint8_t LED_Counter = 0x0f;
  uint32_t timer_start = 0x0;
  uint32_t timer_stop = 0x0;
  // Setup and start custom counter 1, used for measuring how long access takes
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterReset, ArbVal);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterCommand, 0x0);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterTarget, 0xffffffff);
  IOWR_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterStart, ArbVal);
  // Wait for the other processor to finish setting up custom counter 2
  uint8_t ready = (uint8_t)(IORD_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterValue) & 0xff);
  while(ready != LED_Counter)
  {
    ready = (uint8_t)(IORD_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterValue) & 0xff);
  }
  printf("Starting decrementation of shared hardware counter.\n");
  while(LED_Counter > 0x0)
  {
    // Decrement every 10 ms
    usleep(10000);
    // Read start value
    timer_start = IORD_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterValue);

    // Decrement counter 2
    IOWR_32DIRECT(CUSTOM_COUNTER_2_BASE, CustomCounterDecr, 0x01);
    // Read counter 2 value
    LED_Counter = (uint8_t)(IORD_32DIRECT(CUSTOM_COUNTER_2_BASE, 0) & 0xFF);

    // Display new value on LED
    IOWR_8DIRECT(PARALLEL_PORT_2_BASE, 0, LED_Counter);

    timer_stop = IORD_32DIRECT(CUSTOM_COUNTER_1_BASE, CustomCounterValue);
    printf("Decrement time: %ld cycles.\n", (timer_stop - timer_start));

  }
  printf("Finished with decrementation of shared hardware counter.\n\n");
}

void receive_callback(void* message)
{
  if(message != NULL)
  {
    printf("Receiving completed.\n");
  }
  else
  {
    printf("Receiving error.\n");
  }
}

void receive_mail()
{
  printf("Using the mailbox, receiver side.\n");
  // Load mailbox
  altera_avalon_mailbox_dev* mailbox = altera_avalon_mailbox_open("/dev/mailbox_simple_0", NUL
  // Message storage array
  alt_u32 mail[2];
  altera_avalon_mailbox_retrieve_poll(mailbox, mail, 0);
  alt_dcache_flush_all();
  char *msg_ptr = (void*) mail[1];
  printf("Received message with contents: ");
  for(int i = 0; i < mail[0]; i++)
  {
    printf("%c", *(msg_ptr + i));
  }
  altera_avalon_mailbox_close(mailbox);
  printf("\nDone using the mailbox, receiver side.\n\n");
}

void choose_task(int task)
{
  if(task == 1)
  {
    // 3.1 Parallel port test, 100 iterations, 50 ms each
    uint32_t iters = 0x64;
    int delaytime = 50;
    pptest(delaytime, iters);
  }
  else if(task == 2)
  {
    // Manipulation 2: hardware mutex
    hwmutex();
  }
  else if(task == 4)
  {
    // Manipulation 3: hardware mailbox
    send_mail();
  }
  else if(task == 8)
  {
    // Manipulation 4: hardware counter
    hwcounter();
  }
}

void isr_buttons(void* context)
{
  uint8_t pinvals = IORD_8DIRECT(PIO_2_BASE, PIO_IRQFLAG);
  choice = (int)pinvals;
  // Clear the interrupt flag
  IOWR_8DIRECT(PIO_2_BASE, PIO_IRQFLAG, pinvals);
}

int main()
{
  // Setup interrupts on input pins
  IOWR_8DIRECT(PIO_2_BASE, PIO_IRQEN, 0xff);
  alt_ic_isr_register(PIO_2_IRQ_INTERRUPT_CONTROLLER_ID, PIO_2_IRQ, isr_buttons, NULL, NULL);
  // Print which CPU it is
  choose_task(1);
  printf("Use switches for performing tasks:\nSwitch No. |   Task\n   0       |   Parallel Por
  // Wait for buttons
  while(1)
  {
    // Poll choice once every millisecond
    usleep(1000);
    if(choice != 0x0)
    {
      choose_task(choice);
      choice = 0x0;
    }
  };
  return 0;
}

/*
 *---------------------------------- THIS IS CPU 1 --------------------------------------------
 */
```