EE 789 - Algorithmic Design of Digital Systems

Assignment - 1
Shift and Add Multiplier

Name:	**Satvikkumar Patel**
Roll number:	23M1117
Program:	M.tech (EE5 - Electronic Systems)
Department:	Electrical Engineering
Date of Sub:	28th Sep, 2023

Shift and Add Multiplier

Implement an 8-bit multiplier (which multiplies two 8-bit unsigned numbers) using a shift and add multiplier algorithm.
(a) Describe the algorithm using Aa, and write a testbench for the design.
(b) Generate the VHDL and verify it using the GHDL simulator and the C test-bench.
(c) Study the generated VHDL and sketch the structure of the generated hardware (show the modules, their internal structure, and their interactions as you observe them in the VHDL)

Section: a

Explanation:
The Shift and Add Multiplier algorithm is used to multiply two 8-bit unsigned numbers. Here is a step-by-step explanation of the algorithm:

1. Initialize the variables:
- Set the initial state to RST_STATE (0).
- Set the initial value of the temporary product (temp_product) to 0.
- Set the initial value of the temporary multiplicand (temp_multiplicand) to the first input number (a).
- Set the initial value of the counter to 0.

2. Enter the loop:
- Check the current state:
- If the current state is RST_STATE, transition to the LOOP_STATE (1).
- If the current state is LOOP_STATE, check the value of the counter:
If the counter is equal to 8, transition to the DONE_STATE (2).
and If the counter is less than 8, continue in the LOOP_STATE.

3. Update the next state and temporary product:
- Calculate the next state based on the current state and counter value.
- Calculate the next temporary product based on the current state and temporary multiplicand.
- If the current state is LOOP_STATE, check the least significant bit of the temporary multiplicand:
- If it is 0: slice the temp_product to 15-9, shift the bits of the temporary product to the right by 2, and add it with the concatenated '0' to the most significant bit of the second input number (b); and then slice the temp_prodct to 8-1; and save it to next_temp_product.
- If it is 1, shift the bits of the temporary product to the right by 1 and add a 0 as the least significant bit; and save it to next_temp_product.

4. Update the next temporary multiplicand and counter:
- Calculate the next temporary multiplicand based on the current state and temporary multiplicand.
- If the current state is LOOP_STATE, shift the bits of the temporary multiplicand to the right by 1 and add a 0 as the most significant bit.
- Calculate the next counter based on the current state and counter.
- If the current state is LOOP_STATE, increment the counter by 1.

5. Check if the loop should continue:
- Set the continue_flag to true if the counter is less than 9 (indicating that the loop should continue).
- If the continue_flag is true, go back to step 2 and continue the loop.

6. Exit the loop:
- If the counter is equal to 9, transition to the DONE_STATE (2).

7. Output the result:
- The final product is stored in the temporary product variable.

This algorithm uses a shift and add approach to perform multiplication. It iteratively shifts the bits of the multiplicand and adds the corresponding bits of the multiplier to the product. The loop runs for 8 iterations, which is the number of bits in the input numbers. The final product is obtained by shifting the temporary product to the left by 1 in the DONE_STATE.

Test bench

```

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <pthread.h>
#include <pthreadUtils.h>
#include <Pipes.h>
#include <pipeHandler.h>
#ifndef SW
#include "vhdlCStubs.h"
#endif

int main(int argc, char* argv[])
{
  uint8_t a,b,product;

  product = shift_and_add_mul(12,12);
  fprintf(stdout, "shift_and_add_mul(12,12) = %d\n", product);

  return(0);
}
```

Aa algorithm

```
$constant RST_STATE: $uint<2>  :=0
$constant LOOP_STATE: $uint<2> :=1
$constant DONE_STATE: $uint<2> :=2

$module [shift_and_add_mul]
    $in (a b: $uint<8>)
    $out (product : $uint<16>) $is
{
  $branchblock[loop]{
      $merge $entry loopback
          $phi current_state := RST_STATE $on $entry next_state $on loopback
          $phi temp_product := $zero<17> $on $entry next_temp_product $on loopback
          $phi temp_multiplicand := a $on $entry next_multiplicand $on loopback
          $phi counter := $zero<4> $on $entry next_counter $on loopback
      $endmerge

      next_state :=
        ($excmux
            (current_state == RST_STATE) LOOP_STATE

            (current_state == LOOP_STATE)
                ($mux (counter == 8) DONE_STATE LOOP_STATE)

            (current_state == DONE_STATE)
                DONE_STATE
        )
      next_temp_product :=
          ($mux (current_state == LOOP_STATE)
            ($mux (temp_multiplicand [] 0)
                ($concat
                    (($concat $zero<2> ($slice temp_product 15 9)) + ($concat $zero<1> b))
                    ($slice temp_product 8 1))
                ($concat $zero<1> ($slice temp_product 16 1)))
            temp_product)

      next_multiplicand := ($mux (current_state == LOOP_STATE)
              ($concat $zero<1> ($slice temp_multiplicand 7 1))
            temp_multiplicand)

      next_counter := ($mux (current_state == LOOP_STATE)
                  (counter + 1) counter)

      continue_flag := (counter < 9)

      $if continue_flag $then
        $place[loopback]
      $endif

  }(temp_product => temp_product)

  product := ($slice temp_product 15 0)
}
```

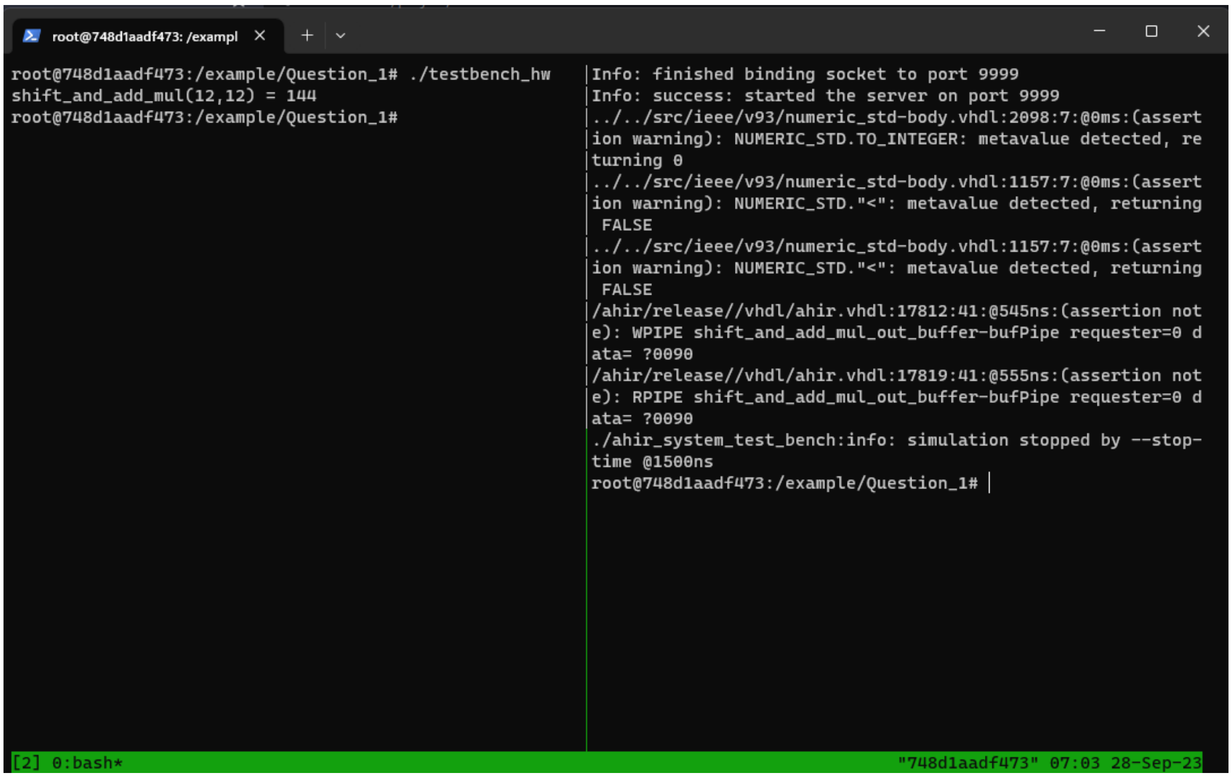
Makefile

```
# build software version of testbench (to check the "desired behaviour")
AHIR_INCLUDE=$(AHIR_RELEASE)/include
AHIR_LIB=$(AHIR_RELEASE)/lib
VHDL_LIB=$(AHIR_RELEASE)/vhdl
VHDL_VHPI_LIB=$(AHIR_RELEASE)/CtestBench/vhdl
FUNCTIONLIB=$(AHIR_RELEASE)/functionLibrary/
SRC=./src
all: HW
TOVC:aalink aa2vc
VC2VHDL: vc2vhdl  vhdlsim
AA2VHDLSIM: aa2vc  vc2vhdl  vhdlsim
TOVHDL:TOVC vc2vhdl


TOPMODULES=-t shift_and_add_mul



# five steps from C to vhdl simulator.
HW: aalink aa2vc  vc2vhdl  vhdltb ghdlsim

AA2VHDL: aa2vc vc2vhdl vhdlsim

# Aa to vC
aalink: $(SRC)/shift_and_add_mul.aa
  AaLinkExtMem $(SRC)/shift_and_add_mul.aa | vcFormat > prog.linked.aa
  AaOpt -B prog.linked.aa | vcFormat > prog.linked.opt.aa

aa2vc: prog.linked.opt.aa
  Aa2VC -O -C prog.linked.opt.aa | vcFormat > prog.vc

# vC to VHDL
vc2vhdl: prog.vc
  vc2vhdl -U -O -v -a -C -e ahir_system -w -s ghdl $(TOPMODULES) -f prog.vc
  vhdlFormat < ahir_system_global_package.unformatted_vhdl >
    ahir_system_global_package.vhdl
  vhdlFormat < ahir_system.unformatted_vhdl > ahir_system.vhdl
  vhdlFormat < ahir_system_test_bench.unformatted_vhdl > ahir_system_test_bench
    .vhdl

# build testbench and ghdl executable
# note the use of libVhpi in building the testbench.
vhdltb: ahir_system.vhdl ahir_system_test_bench.vhdl $(SRC)/testbench.c
    vhdlCStubs.h vhdlCStubs.c
  gcc -c vhdlCStubs.c  -I$(SRC) -I./ -I$(AHIR_INCLUDE)
  gcc -c $(SRC)/testbench.c -I$(AHIR_INCLUDE) -I$(SRC) -I./
  gcc -o testbench_hw testbench.o vhdlCStubs.o  -L$(AHIR_LIB) -
    lSocketLibPipeHandler -lpthread

ghdlsim: ahir_system.vhdl ahir_system_test_bench.vhdl $(SRC)/testbench.c
    vhdlCStubs.h vhdlCStubs.c
  ghdl --clean
  ghdl --remove
  ghdl -i --work=GhdlLink  $(VHDL_LIB)/GhdlLink.vhdl
  ghdl -i --work=aHiR_ieee_proposed  $(VHDL_LIB)/aHiR_ieee_proposed.vhdl
  ghdl -i --work=ahir  $(VHDL_LIB)/ahir.vhdl
  ghdl -i --work=work ahir_system_global_package.vhdl
  ghdl -i --work=work ahir_system.vhdl
  ghdl -i --work=work ahir_system_test_bench.vhdl
  ghdl -m --work=work -Wl,-L$(AHIR_LIB) -Wl,-lVhpi ahir_system_test_bench

clean:
  rm -rf *.o* *.cf *.*vhdl vhdlCStubs.* *.vcd in_data* out_data* testbench_sw
   testbench_hw ahir_system_test_bench vhpi.log *.aa *.vc *.lso xst *.ngc *
   _xmsgs *.xrpt pipeHandler.log *.srp *.ghw *.dot

PHONY: all clean
```

Section: b

The test bench is setting $a = (12)_{dec}$ and $b = (12)_{dec}$, and the algorithm generates the output $product = (144)_{dec}$ as expected.

GHDL Simulation

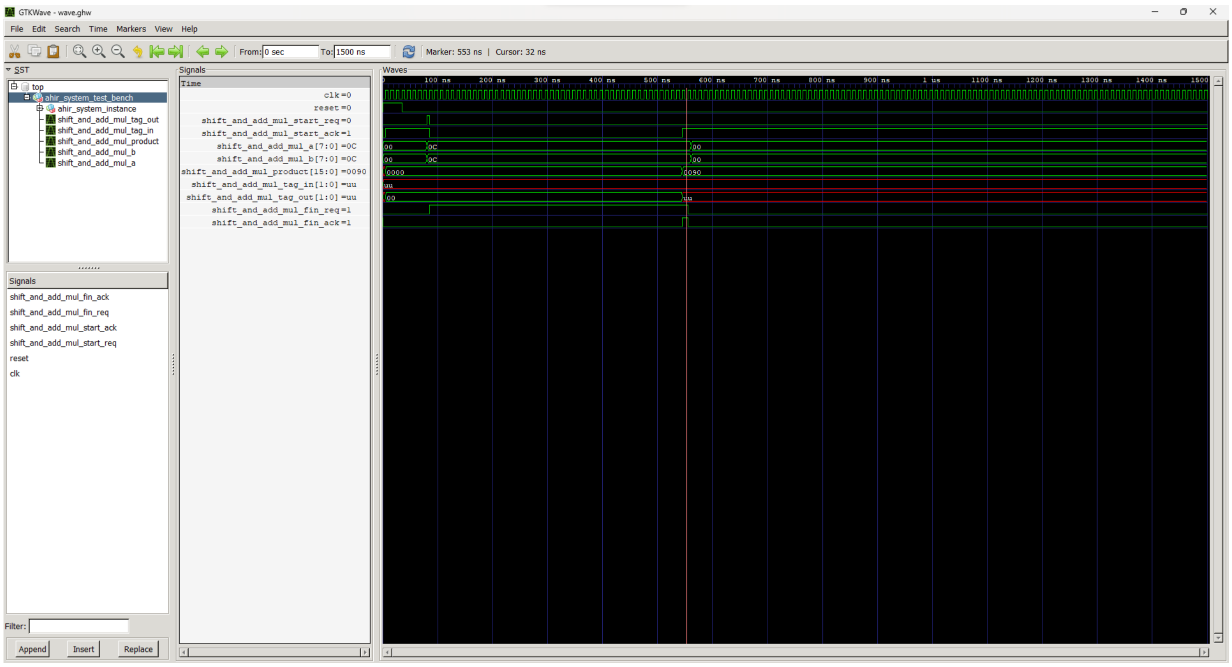


```
root@748d1aadf473:/example/Question_1# ./testbench_hw
shift_and_add_mul(12,12) = 144
root@748d1aadf473:/example/Question_1#
```

```
Info: finished binding socket to port 9999
Info: success: started the server on port 9999
../../src/ieee/v93/numeric_std-body.vhdl:2098:7:@0ms:(assert
ion warning): NUMERIC_STD.TO_INTEGER: metavalue detected, re
turning 0
../../src/ieee/v93/numeric_std-body.vhdl:1157:7:@0ms:(assert
ion warning): NUMERIC_STD."<": metavalue detected, returning
 FALSE
../../src/ieee/v93/numeric_std-body.vhdl:1157:7:@0ms:(assert
ion warning): NUMERIC_STD."<": metavalue detected, returning
 FALSE
/ahir/release//vhdl/ahir.vhdl:17812:41:@545ns:(assertion not
e): WPIPE shift_and_add_mul_out_buffer-bufPipe requester=0 d
ata= ?0090
/ahir/release//vhdl/ahir.vhdl:17819:41:@555ns:(assertion not
e): RPIPE shift_and_add_mul_out_buffer-bufPipe requester=0 d
ata= ?0090
./ahir_system_test_bench:info: simulation stopped by --stop-
time @1500ns
root@748d1aadf473:/example/Question_1#
```

```
[2] 0:bash*                                    "748d1aadf473" 07:03 28-Sep-23
```

Waveform

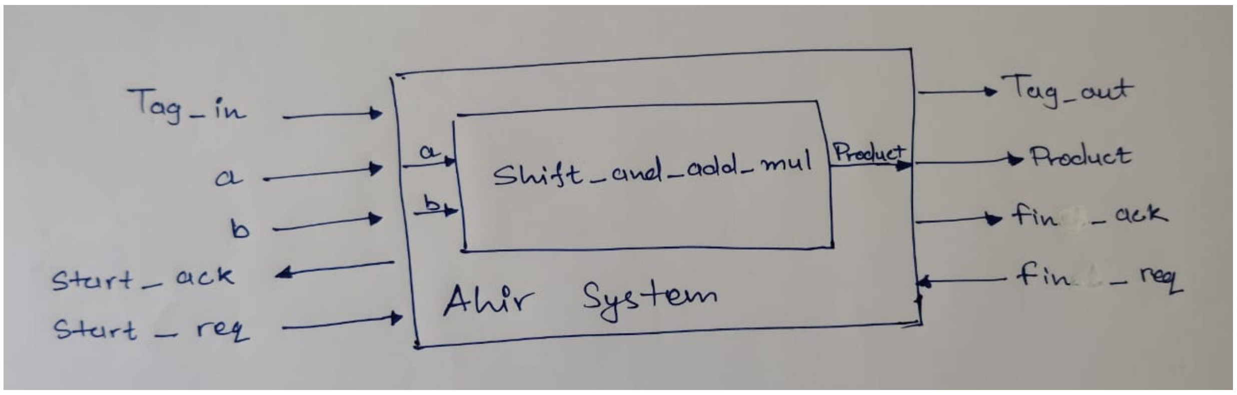


```
Time
                          clk =0
                        reset =0
     shift_and_add_mul_start_req =0
     shift_and_add_mul_start_ack =1
         shift_and_add_mul_a[7:0] =0C
         shift_and_add_mul_b[7:0] =0C
shift_and_add_mul_product[15:0] =0090
    shift_and_add_mul_tag_in[1:0] =uu
   shift_and_add_mul_tag_out[1:0] =uu
       shift_and_add_mul_fin_req =1
       shift_and_add_mul_fin_ack =1
```

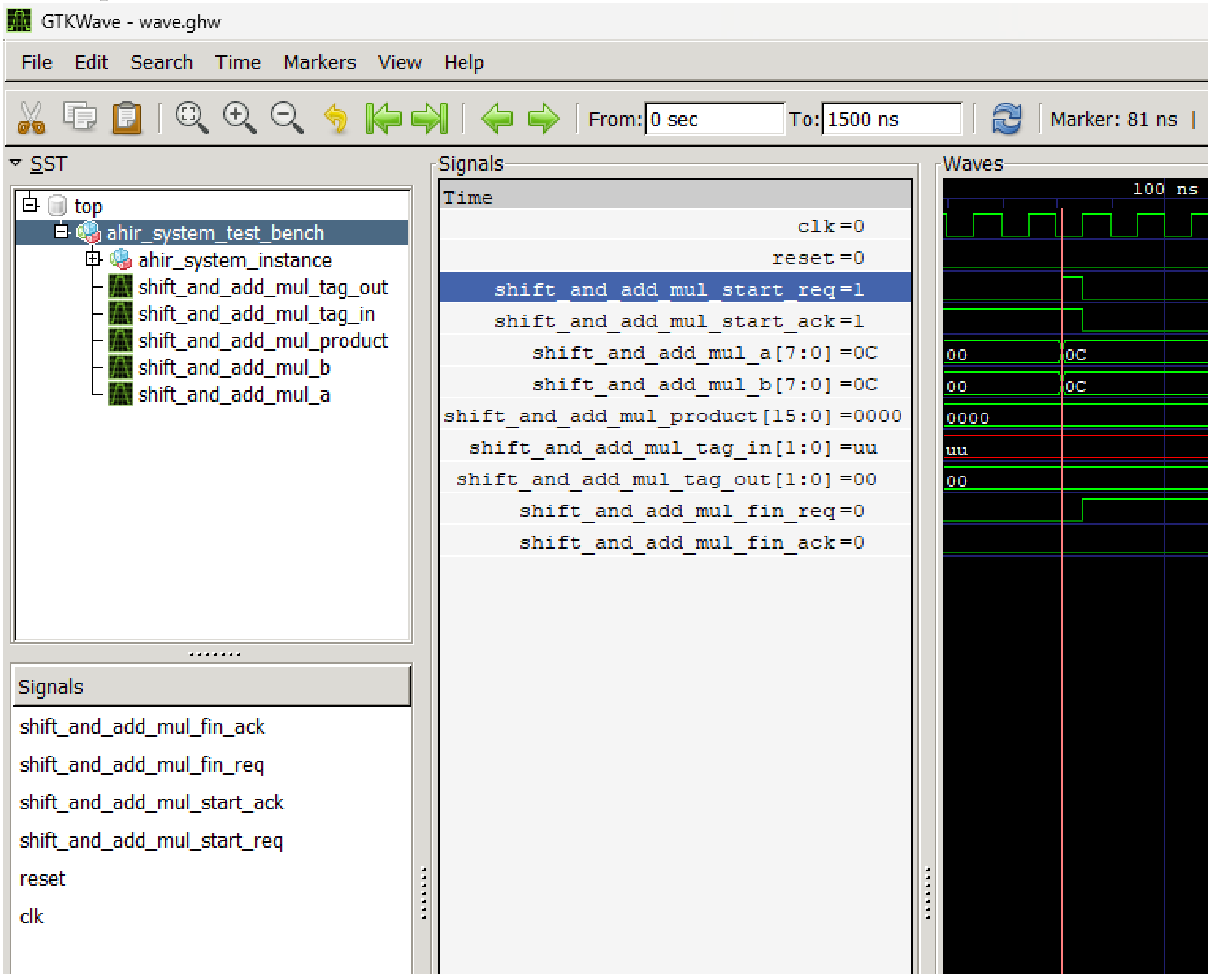
GTKWave - wave.ghw
File Edit Search Time Markers View Help
From: 0 sec To: 1500 ns
Marker: 553 ns | Cursor: 32 ns
SST
top
ahir_system_test_bench
ahir_system_instance
shift_and_add_mul_tag_out
shift_and_add_mul_tag_in
shift_and_add_mul_product
shift_and_add_mul_b
shift_and_add_mul_a
Signals
shift_and_add_mul_fin_ack
shift_and_add_mul_fin_req
shift_and_add_mul_start_ack
shift_and_add_mul_start_req
reset
clk
Filter:
Append
Insert
Replace
Waves

Section: c

While generating the VHDL codes, the Ahir System generates start_ack, start_req, fin_ack, fin_req, tag_in and tag_out signals.

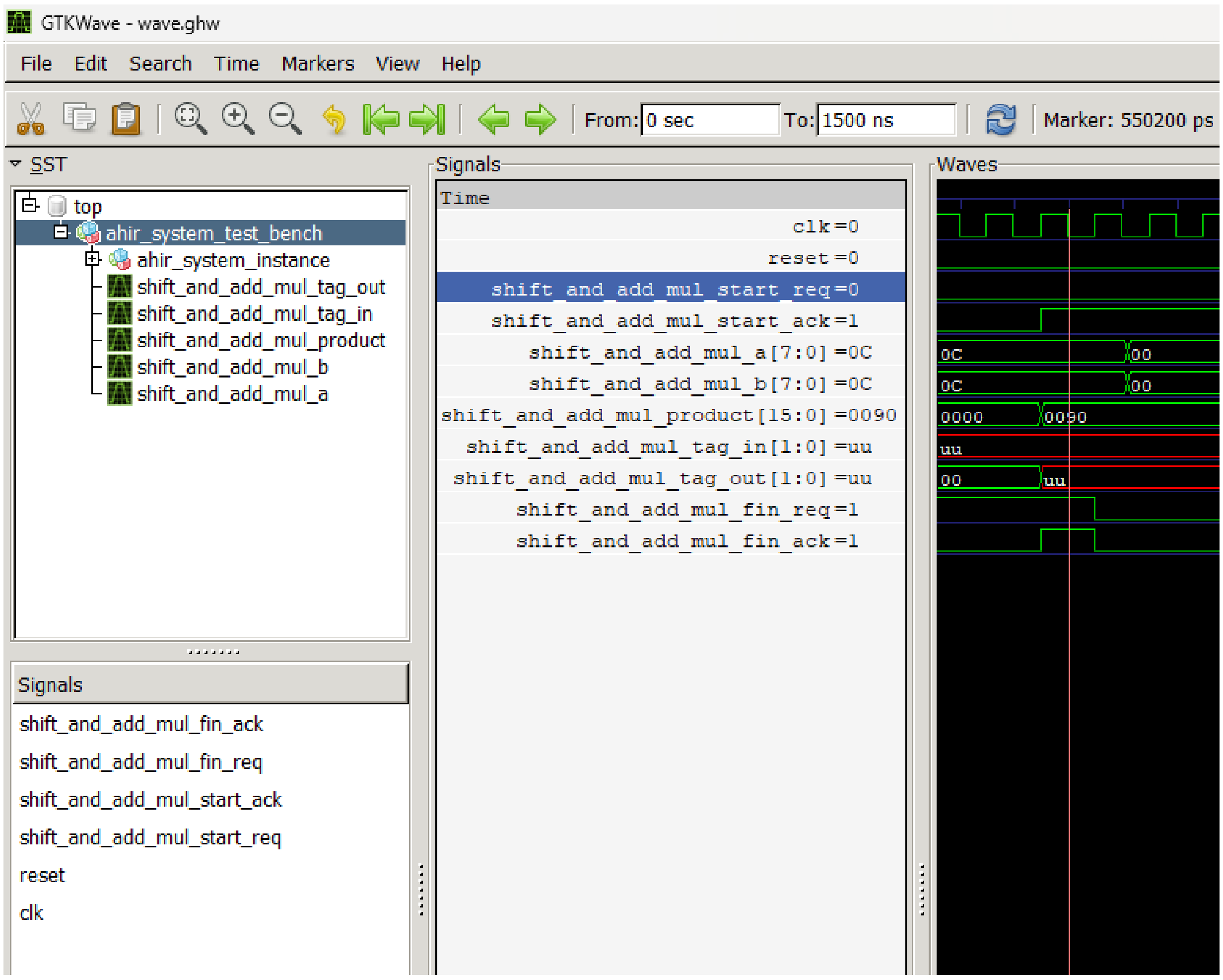


It can be seen that, when start_ack and start_req are 1, the system takes the input.


GTKWave - wave.ghw
File Edit Search Time Markers View Help
From: 0 sec
To: 1500 ns
Marker: 81 ns
SST
top
ahir_system_test_bench
ahir_system_instance
shift_and_add_mul_tag_out
shift_and_add_mul_tag_in
shift_and_add_mul_product
shift_and_add_mul_b
shift_and_add_mul_a
Signals
shift_and_add_mul_fin_ack
shift_and_add_mul_fin_req
shift_and_add_mul_start_ack
shift_and_add_mul_start_req
reset
clk
Waves
100 ns

```
Time
clk=0
reset=0
shift_and_add_mul_start_req=1
shift_and_add_mul_start_ack=1
shift_and_add_mul_a[7:0]=0C
shift_and_add_mul_b[7:0]=0C
shift_and_add_mul_product[15:0]=0000
shift_and_add_mul_tag_in[1:0]=uu
shift_and_add_mul_tag_out[1:0]=00
shift_and_add_mul_fin_req=0
shift_and_add_mul_fin_ack=0
```

It can be seen that, When fin_ack and fin_req are 1, the system gives the output.


GTKWave - wave.ghw

From: 0 sec To: 1500 ns Marker: 550200 ps

SST
```
top
  ahir_system_test_bench
    ahir_system_instance
    shift_and_add_mul_tag_out
    shift_and_add_mul_tag_in
    shift_and_add_mul_product
    shift_and_add_mul_b
    shift_and_add_mul_a
```

Signals
```
shift_and_add_mul_fin_ack
shift_and_add_mul_fin_req
shift_and_add_mul_start_ack
shift_and_add_mul_start_req
reset
clk
```

Signals
```
Time
                               clk =0
                             reset =0
     shift_and_add_mul_start_req =0
     shift_and_add_mul_start_ack =1
       shift_and_add_mul_a[7:0] =0C
       shift_and_add_mul_b[7:0] =0C
shift_and_add_mul_product[15:0] =0090
   shift_and_add_mul_tag_in[1:0] =uu
  shift_and_add_mul_tag_out[1:0] =uu
       shift_and_add_mul_fin_req =1
       shift_and_add_mul_fin_ack =1
```