ASSIGNMENT 2 STAGE 7 REPORT

Sreemanti Dey

January 2022

1 Objective

The instructions to be implemented at this stage include short multiply instructions producing 32-bit results and long multiply instructions that produce 64-bit results.

2 Assumptions

VHDL
GHDL
GTKWave for the waveforms

3 Implementation details

I have added support for mul and mul+accumulate operations by making the following changes:

1. I made a new entity-architecture pair called mul-acc.vhd that performs the mul instruction class operations combinationally.

2. I have added a separate pathway for MUL instruction class operations.

I have written the following assembly program files in my program to test all the operations.

1. mul - p1.s

2. mla - p2.s

3. smull,umull - p3.s

4. smulls (checking condition codes) - p4.s

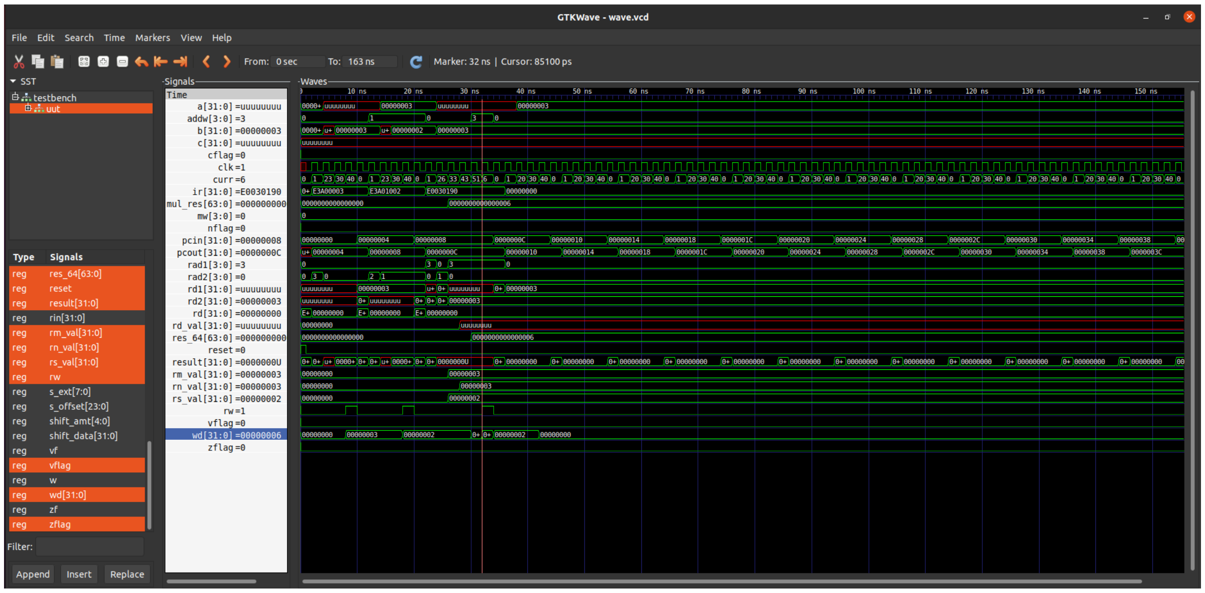
5. smlal,umlal - p5.s

6. other instructions like ldr,str,b - p6.s

4 p1.s - mul

4.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave



```
Time
      a[31:0] =uuuuuuuu
      addw[3:0] =3
      b[31:0] =00000003
      c[31:0] =uuuuuuuu
      cflag =0
      clk =1
      curr =6
      ir[31:0] =E0030190
mul_res[63:0] =000000000
      mw[3:0] =0
      nflag =0
   pcin[31:0] =00000008
  pcout[31:0] =0000000C
   rad1[3:0] =3
   rad2[3:0] =0
    rd1[31:0] =uuuuuuuu
    rd2[31:0] =00000003
     rd[31:0] =00000000
 rd_val[31:0] =uuuuuuuu
 res_64[63:0] =000000000
      reset =0
 result[31:0] =0000000U
 rm_val[31:0] =00000003
 rn_val[31:0] =00000003
 rs_val[31:0] =00000002
         rw =1
      vflag =0
     wd[31:0] =00000006
      zflag =0
```

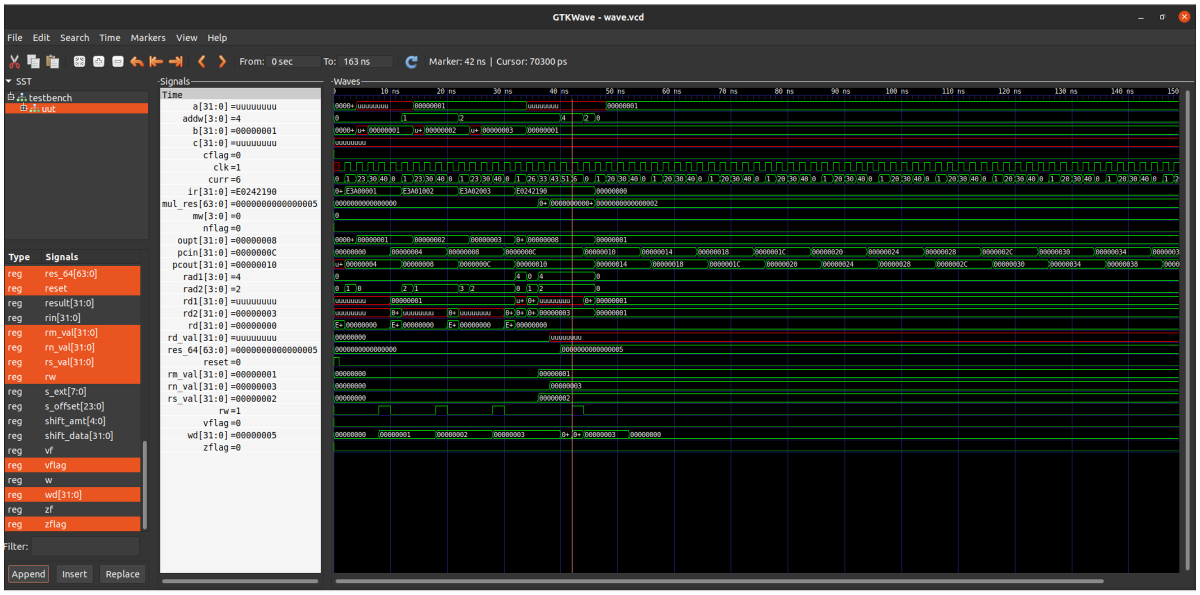
Figure 1: Program 1

Here, we see that we have the final answer as 3x2 = 6, which can also be seen as the value of wd the last time when rw is '1', thus verifying correctness of mul.

5 p2.s - mla

5.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave



```
Type  Signals
reg   res_64[63:0]
reg   reset
reg   result[31:0]
reg   rin[31:0]
reg   rm_val[31:0]
reg   rn_val[31:0]
reg   rs_val[31:0]
reg   rw
reg   s_ext[7:0]
reg   s_offset[23:0]
reg   shift_amt[4:0]
reg   shift_data[31:0]
reg   vf
reg   vflag
reg   w
reg   wd[31:0]
reg   zf
reg   zflag
```

```
Signals
Time
       a[31:0] =uuuuuuuu
    addw[3:0] =4
       b[31:0] =00000001
       c[31:0] =uuuuuuuu
        cflag =0
          clk =1
         curr =6
      ir[31:0] =E0242190
mul_res[63:0] =0000000000000005
      mw[3:0] =0
        nflag =0
    oupt[31:0] =00000008
    pcin[31:0] =0000000C
   pcout[31:0] =00000010
     rad1[3:0] =4
     rad2[3:0] =2
     rd1[31:0] =uuuuuuuu
     rd2[31:0] =00000003
      rd[31:0] =00000000
  rd_val[31:0] =uuuuuuuu
 res_64[63:0] =0000000000000005
        reset =0
  rm_val[31:0] =00000001
  rn_val[31:0] =00000003
  rs_val[31:0] =00000002
           rw =1
        vflag =0
      wd[31:0] =00000005
        zflag =0
```

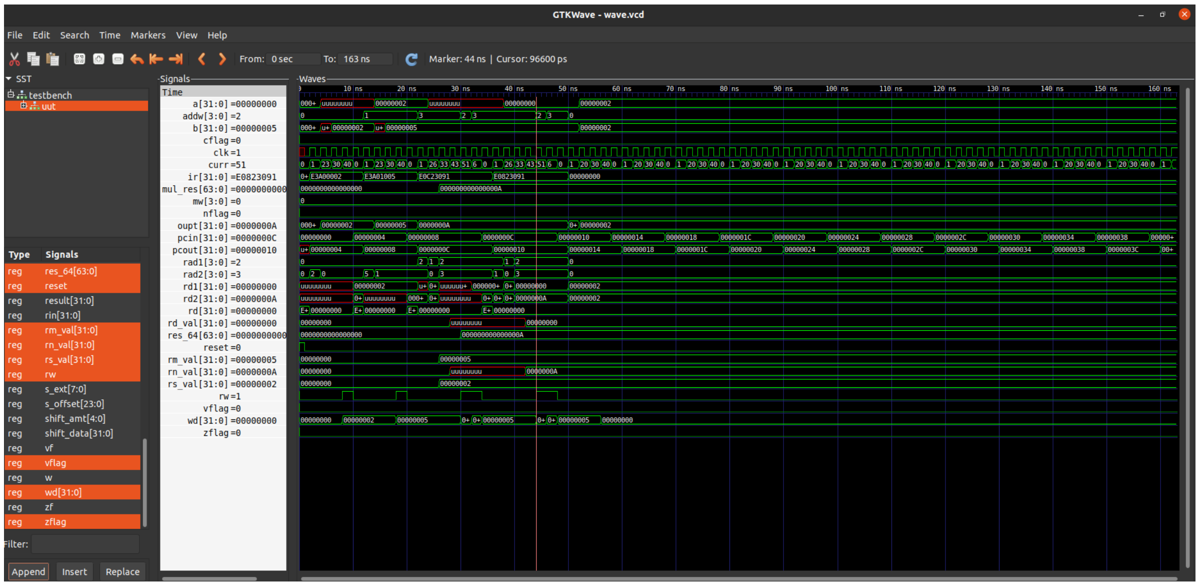
Figure 2: Program 1

Here, we have the final answer as 2x1+3 = 5, which can be seen as the value of wd when we have rw as 1 for the last time, thus this verifies that my code for mla is correct.

6 p3.s - smull,umull

6.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave

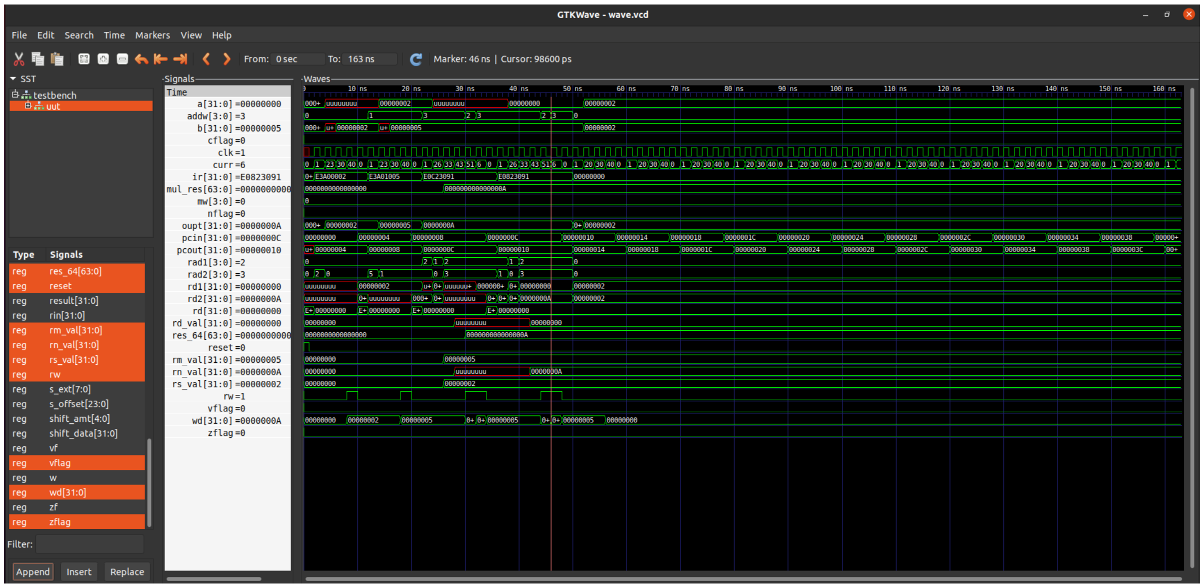


```
GTKWave - wave.vcd
From: 0 sec   To: 163 ns   Marker: 44 ns | Cursor: 96600 ps

       a[31:0] =00000000
     addw[3:0] =2
       b[31:0] =00000005
         cflag =0
           clk =1
          curr =51
      ir[31:0] =E0823091
 mul_res[63:0] =0000000000
       mw[3:0] =0
         nflag =0
    oupt[31:0] =0000000A
    pcin[31:0] =0000000C
   pcout[31:0] =00000010
     rad1[3:0] =2
     rad2[3:0] =3
     rd1[31:0] =00000000
     rd2[31:0] =0000000A
      rd[31:0] =00000000
  rd_val[31:0] =00000000
  res_64[63:0] =0000000000
         reset =0
  rm_val[31:0] =00000005
  rn_val[31:0] =0000000A
  rs_val[31:0] =00000002
            rw =1
         vflag =0
      wd[31:0] =00000000
         zflag =0
```

Type	Signals
reg	res_64[63:0]
reg	reset
reg	result[31:0]
reg	rin[31:0]
reg	rm_val[31:0]
reg	rn_val[31:0]
reg	rs_val[31:0]
reg	rw
reg	s_ext[7:0]
reg	s_offset[23:0]
reg	shift_amt[4:0]
reg	shift_data[31:0]
reg	vf
reg	vflag
reg	w
reg	wd[31:0]
reg	zf
reg	zflag

Figure 3: Program 3a



```
GTKWave - wave.vcd
From: 0 sec   To: 163 ns   Marker: 46 ns | Cursor: 98600 ps

       a[31:0] =00000000
     addw[3:0] =3
       b[31:0] =00000005
         cflag =0
           clk =1
          curr =6
      ir[31:0] =E0823091
 mul_res[63:0] =0000000000
       mw[3:0] =0
         nflag =0
    oupt[31:0] =0000000A
    pcin[31:0] =0000000C
   pcout[31:0] =00000010
     rad1[3:0] =2
     rad2[3:0] =3
     rd1[31:0] =00000000
     rd2[31:0] =0000000A
      rd[31:0] =00000000
  rd_val[31:0] =00000000
  res_64[63:0] =0000000000
         reset =0
  rm_val[31:0] =00000005
  rn_val[31:0] =0000000A
  rs_val[31:0] =00000002
            rw =1
         vflag =0
      wd[31:0] =0000000A
         zflag =0
```

Type	Signals
reg	res_64[63:0]
reg	reset
reg	result[31:0]
reg	rin[31:0]
reg	rm_val[31:0]
reg	rn_val[31:0]
reg	rs_val[31:0]
reg	rw
reg	s_ext[7:0]
reg	s_offset[23:0]
reg	shift_amt[4:0]
reg	shift_data[31:0]
reg	vf
reg	vflag
reg	w
reg	wd[31:0]
reg	zf
reg	zflag

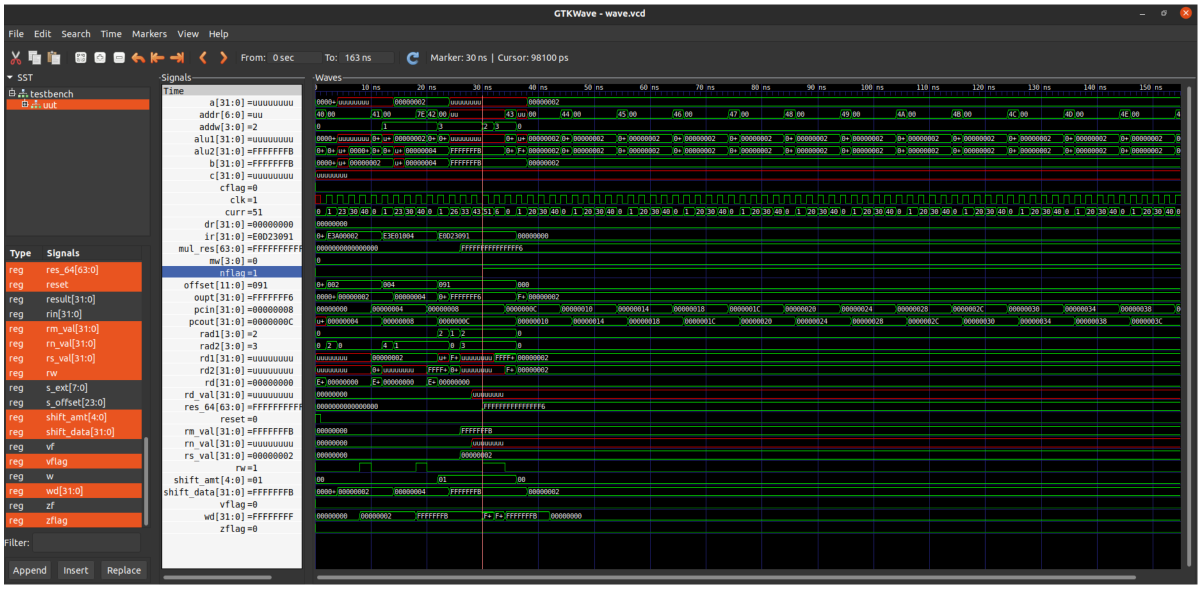
Figure 4: Program 3b

Here, we have the final answer as 10, which is a 64-bit number thus spread over 2 registers, the first screenshot shows the leftmost 32 bits(0) stored in one register and the second screenshot shows the rightmost 32 bits(10) stored in another register, thus verifying my code is correct.

7 p4.s - smulls(checking condition codes)

7.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave



```
Time
         a[31:0] =uuuuuuuu
       addr[6:0] =uu
       addw[3:0] =2
       alu1[31:0] =uuuuuuuu
       alu2[31:0] =FFFFFFFB
         b[31:0] =FFFFFFFB
         c[31:0] =uuuuuuuu
           cflag =0
             clk =1
            curr =51
        dr[31:0] =00000000
        ir[31:0] =E0D23091
  mul_res[63:0] =FFFFFFFFFF
         mw[3:0] =0
           nflag =1
    offset[11:0] =091
      oupt[31:0] =FFFFFFF6
      pcin[31:0] =00000008
     pcout[31:0] =0000000C
       rad1[3:0] =2
       rad2[3:0] =3
       rd1[31:0] =uuuuuuuu
       rd2[31:0] =uuuuuuuu
        rd[31:0] =00000000
    rd_val[31:0] =uuuuuuuu
   res_64[63:0] =FFFFFFFFFF
           reset =0
    rm_val[31:0] =FFFFFFFB
    rn_val[31:0] =uuuuuuuu
    rs_val[31:0] =00000002
              rw =1
   shift_amt[4:0] =01
shift_data[31:0] =FFFFFFFB
           vflag =0
        wd[31:0] =FFFFFFFF
           zflag =0
```

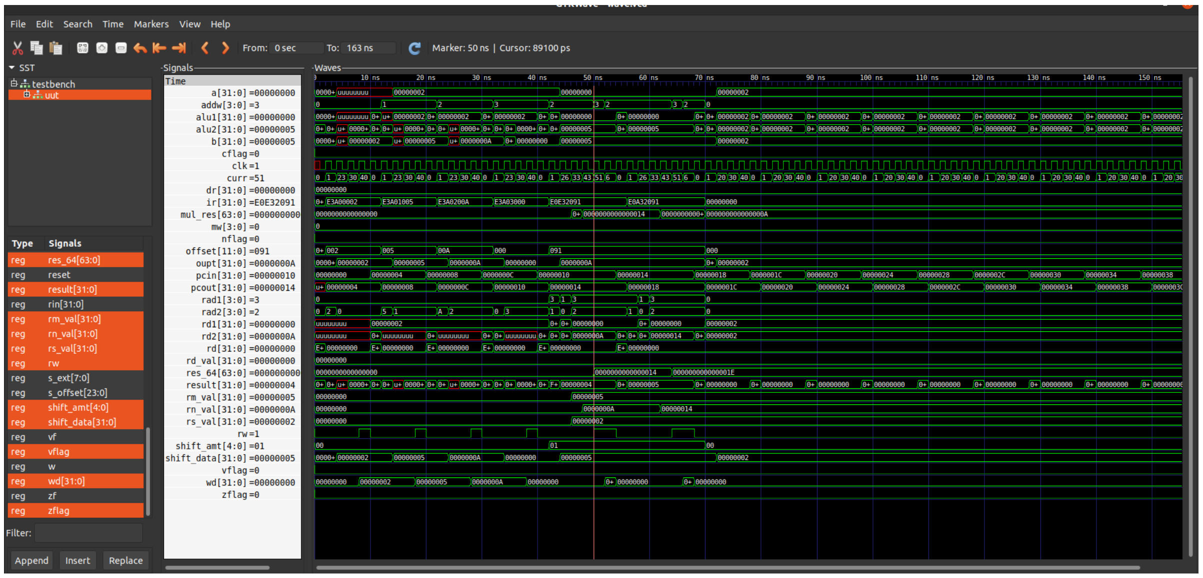
Figure 5: Program 4

Here, we see that NFlag gets set and Zflag remains '0' since we have SBit as '1' and the final value is -10, thus verifying the code for condition codes.

8 p5.s - smlal, umlal

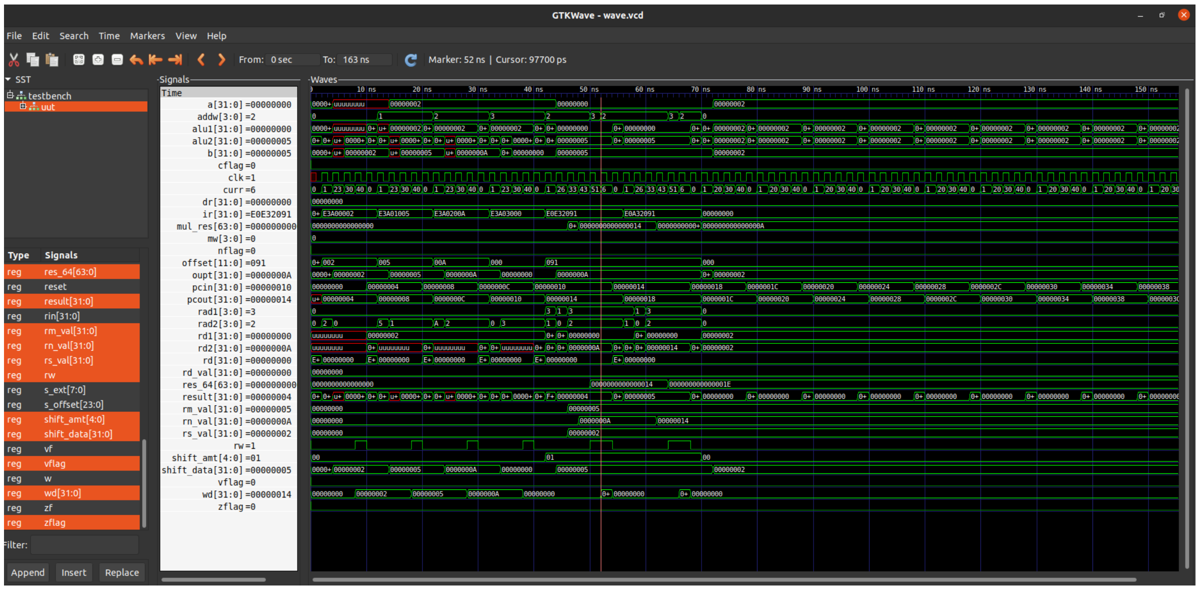
8.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave



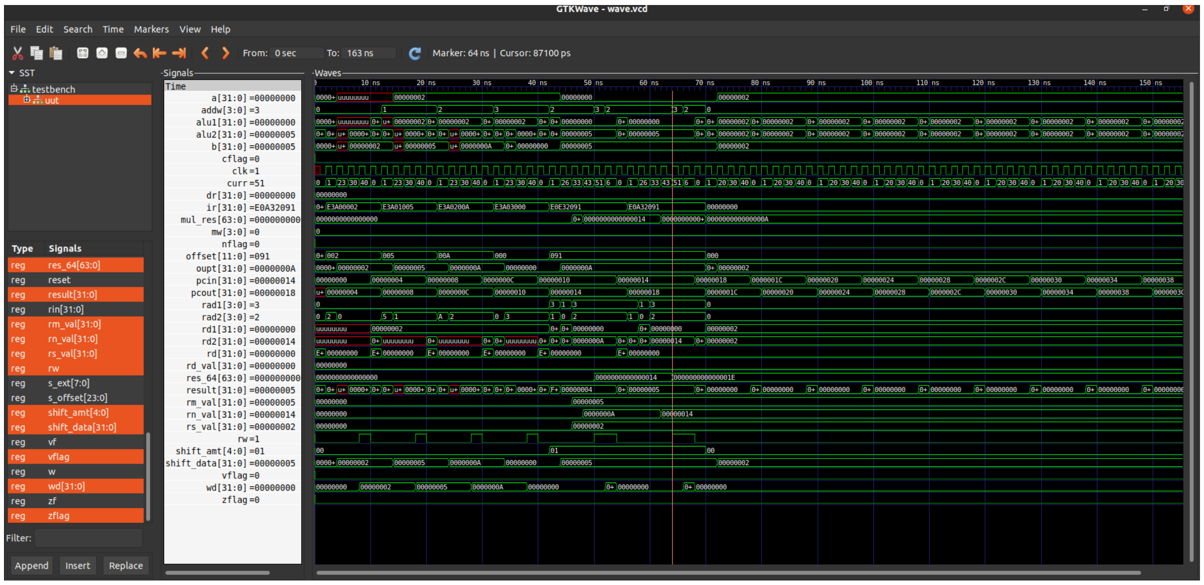
```
a[31:0] =00000000
addw[3:0] =3
alu1[31:0] =00000000
alu2[31:0] =00000005
b[31:0] =00000005
cflag =0
clk =1
curr =51
dr[31:0] =00000000
ir[31:0] =E0E32091
mul_res[63:0] =000000000
mw[3:0] =0
nflag =0
offset[11:0] =091
oupt[31:0] =0000000A
pcin[31:0] =00000010
pcout[31:0] =00000014
rad1[3:0] =3
rad2[3:0] =2
rd1[31:0] =00000000
rd2[31:0] =0000000A
rd[31:0] =00000000
rd_val[31:0] =00000000
res_64[63:0] =000000000
result[31:0] =00000004
rm_val[31:0] =00000005
rn_val[31:0] =0000000A
rs_val[31:0] =00000002
rw =1
shift_amt[4:0] =01
shift_data[31:0] =00000005
vflag =0
wd[31:0] =00000000
zflag =0
```

Figure 6: Program 5a



```
a[31:0] =00000000
addw[3:0] =2
alu1[31:0] =00000000
alu2[31:0] =00000005
b[31:0] =00000005
cflag =0
clk =1
curr =6
dr[31:0] =00000000
ir[31:0] =E0E32091
mul_res[63:0] =000000000
mw[3:0] =0
nflag =0
offset[11:0] =091
oupt[31:0] =0000000A
pcin[31:0] =00000010
pcout[31:0] =00000014
rad1[3:0] =3
rad2[3:0] =2
rd1[31:0] =00000000
rd2[31:0] =0000000A
rd[31:0] =00000000
rd_val[31:0] =00000000
res_64[63:0] =000000000
result[31:0] =00000004
rm_val[31:0] =00000005
rn_val[31:0] =0000000A
rs_val[31:0] =00000002
rw =1
shift_amt[4:0] =01
shift_data[31:0] =00000005
vflag =0
wd[31:0] =00000014
zflag =0
```

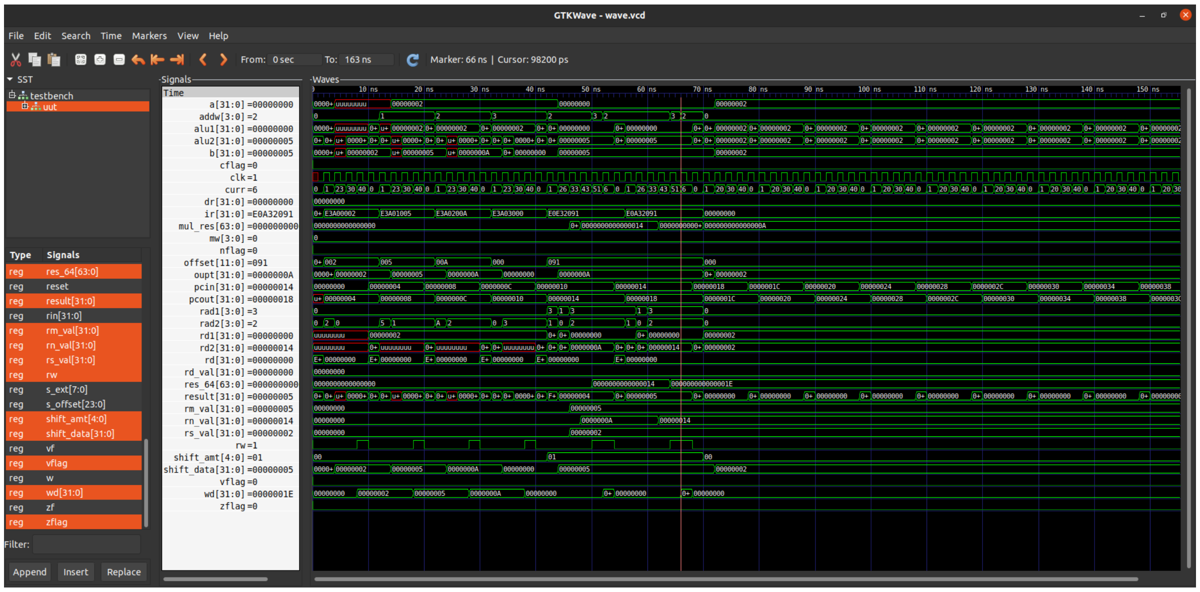
Figure 7: Program 5b



```
Type  Signals
reg   res_64[63:0]
reg   reset
reg   result[31:0]
reg   rin[31:0]
reg   rm_val[31:0]
reg   rn_val[31:0]
reg   rs_val[31:0]
reg   rw
reg   s_ext[7:0]
reg   s_offset[23:0]
reg   shift_amt[4:0]
reg   shift_data[31:0]
reg   vf
reg   vflag
reg   w
reg   wd[31:0]
reg   zf
reg   zflag
```

```
Time
         a[31:0] =00000000
        addw[3:0] =3
       alu1[31:0] =00000000
       alu2[31:0] =00000005
          b[31:0] =00000005
            cflag =0
              clk =1
             curr =51
         dr[31:0] =00000000
         ir[31:0] =E0A32091
   mul_res[63:0] =000000000
         mw[3:0] =0
            nflag =0
     offset[11:0] =091
       oupt[31:0] =0000000A
       pcin[31:0] =00000014
      pcout[31:0] =00000018
        rad1[3:0] =3
        rad2[3:0] =2
        rd1[31:0] =00000000
        rd2[31:0] =00000014
         rd[31:0] =00000000
     rd_val[31:0] =00000000
    res_64[63:0] =000000000
     result[31:0] =00000005
     rm_val[31:0] =00000005
     rn_val[31:0] =00000014
     rs_val[31:0] =00000002
               rw =1
   shift_amt[4:0] =01
 shift_data[31:0] =00000005
            vflag =0
         wd[31:0] =00000000
            zflag =0
```

Figure 8: Program 5c



```
Type  Signals
reg   res_64[63:0]
reg   reset
reg   result[31:0]
reg   rin[31:0]
reg   rm_val[31:0]
reg   rn_val[31:0]
reg   rs_val[31:0]
reg   rw
reg   s_ext[7:0]
reg   s_offset[23:0]
reg   shift_amt[4:0]
reg   shift_data[31:0]
reg   vf
reg   vflag
reg   w
reg   wd[31:0]
reg   zf
reg   zflag
```

```
Time
         a[31:0] =00000000
        addw[3:0] =2
       alu1[31:0] =00000000
       alu2[31:0] =00000005
          b[31:0] =00000005
            cflag =0
              clk =1
             curr =6
         dr[31:0] =00000000
         ir[31:0] =E0A32091
   mul_res[63:0] =000000000
         mw[3:0] =0
            nflag =0
     offset[11:0] =091
       oupt[31:0] =0000000A
       pcin[31:0] =00000014
      pcout[31:0] =00000018
        rad1[3:0] =3
        rad2[3:0] =2
        rd1[31:0] =00000000
        rd2[31:0] =00000014
         rd[31:0] =00000000
     rd_val[31:0] =00000000
    res_64[63:0] =000000000
     result[31:0] =00000005
     rm_val[31:0] =00000005
     rn_val[31:0] =00000014
     rs_val[31:0] =00000002
               rw =1
   shift_amt[4:0] =01
 shift_data[31:0] =00000005
            vflag =0
         wd[31:0] =0000001E
            zflag =0
```

Figure 9: Program 5d

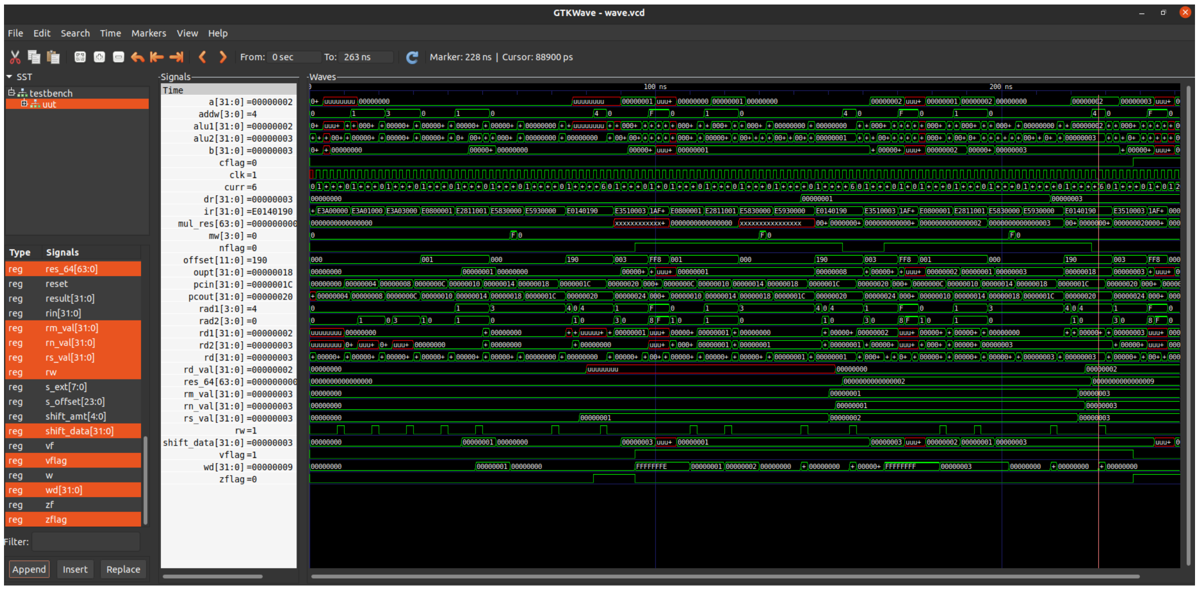
Here, we have the final answer for smlal as 5x2 + 10 = 10 + 10 = 20 , that is spread across 2 registers (as we can see in 5a and 5b) and then I add another

10 when executing umlal so now the final answer is 30, which is also spread over 2 registers (as we can see in 5c and 5d), thus verifying my code is correct.

9 p6.s - other instructions ldr,str,b

9.1 Simulation results

Here is a picture of the simulation results I have achieved by EPWave



```
Type  Signals
reg   res_64[63:0]
reg   reset
reg   result[31:0]
reg   rin[31:0]
reg   rm_val[31:0]
reg   rn_val[31:0]
reg   rs_val[31:0]
reg   rw
reg   s_ext[7:0]
reg   s_offset[23:0]
reg   shift_amt[4:0]
reg   shift_data[31:0]
reg   vf
reg   vflag
reg   w
reg   wd[31:0]
reg   zf
reg   zflag
```

```
Time
           a[31:0] =00000002
          addw[3:0] =4
        alu1[31:0] =00000002
        alu2[31:0] =00000003
           b[31:0] =00000003
             cflag =0
               clk =1
              curr =6
          dr[31:0] =00000003
          ir[31:0] =E0140190
   mul_res[63:0] =000000000
           mw[3:0] =0
             nflag =0
     offset[11:0] =190
        oupt[31:0] =00000018
        pcin[31:0] =0000001C
       pcout[31:0] =00000020
         rad1[3:0] =4
         rad2[3:0] =0
         rd1[31:0] =00000002
         rd2[31:0] =00000003
          rd[31:0] =00000003
      rd_val[31:0] =00000002
    res_64[63:0] =000000000
      rm_val[31:0] =00000003
      rn_val[31:0] =00000003
      rs_val[31:0] =00000003
                rw =1
shift_data[31:0] =00000003
             vflag =1
          wd[31:0] =00000009
             zflag =0
```

Figure 10: Program 6

Here, I have checked the overall compatibility of adding mul instructions with other instructions. We can see that the final values of r0 and r1 are 3 and 3 respectively, and thus using muls gives 9 (3 x 3), which we can see as the value of wd, hence this verifies my code.