ASSIGNMENT 2 STAGE 5 REPORT

Sreemanti Dey

January 2022

1 Objective

This stage enhances the design of the multicycle processor of the last stage with a shifter module and adds appropriate shift/rotate features to DP and DT instructions.

2 Assumptions

VHDL
GHDL
GTKWave for the waveforms

3 Implementation details

I have added support for shift/rotate instructions by making the following changes:

1. I made 3 new entity-architecture pairs - one for left-shift, one for right-shift and one for integrating the two into a Sh-ror.vhd unit.
2. I added a few new states - 2 for shift/rotate in DP instructions with register as shift-operand, 1 for shift/rotate in DP instruction with constant as shift-operand, similar 2 for DT instructions as well.
3. I modified the data path to include shifter output as the input to my ALU.

I have written the following assembly program files in my program to test if shift-rotate works correctly.

1. DP with register as second op
   (a) shift op as immediate - p1.s
   (b) shift op as register - p2.s
2. DP with imm as second op - p3.s

3. DT with register as second op - p4.s
4. DT with imm as second op - p5.s
5. Testing ASR - p6.s
6. checking overall
    (a) p7.s
    (b) p8.s

4 p1.s - DP with register as second op and shift op as immediate

4.1 Simulation results

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

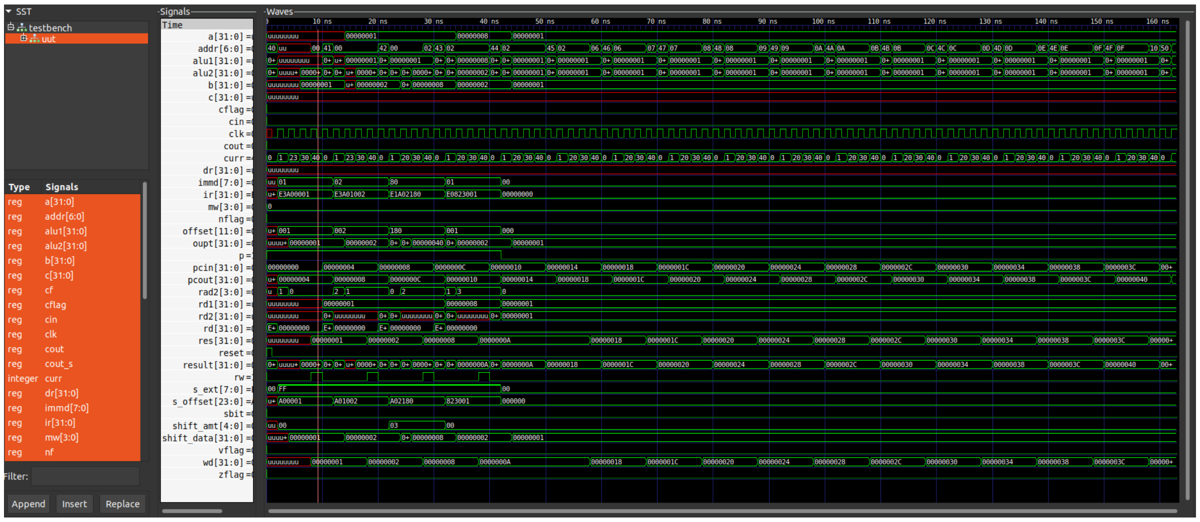


Figure 1: Program 1

We can see that rw is 1 for 4 times, the first time when I have write-data as 1, second time I have it as 2, 3rd time it is 8, 4th time it is 8+2= 10, thus verifying it is working correctly.

5 p2.s - DP with register as second op and shift op as register

5.1 Simulation results

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

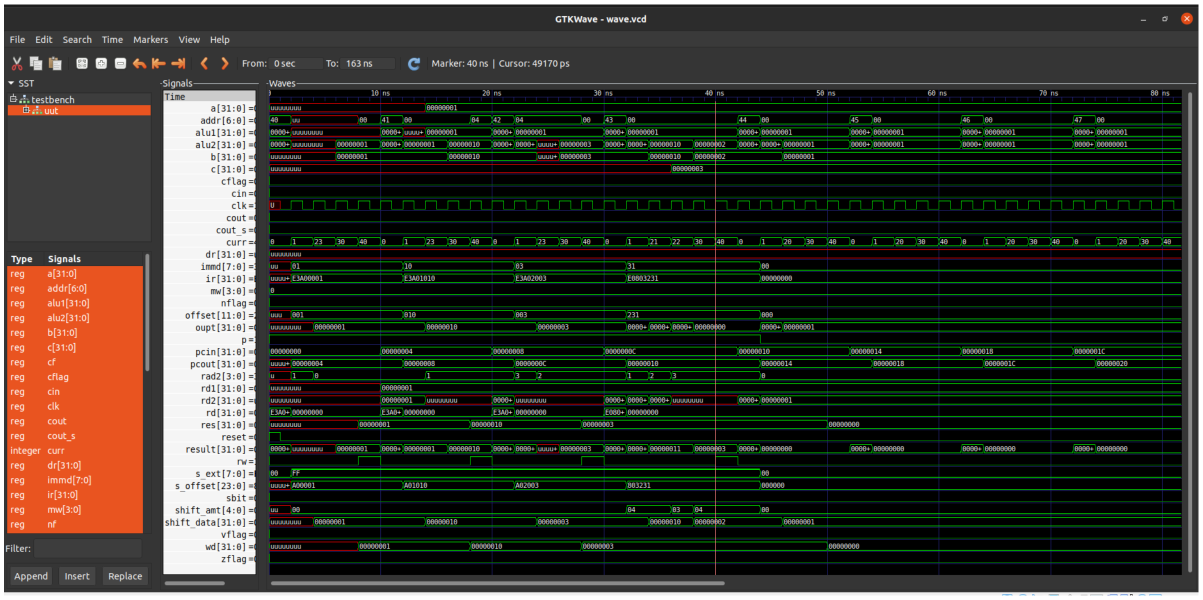


Figure 2: Program 2

We can see that rw becomes 1 for 4 times, and checking the values of write-data we see it is 1, then 16, then 3 and finally it is 1 + 16/8 = 1 + 2 = 3, thus verifying it works correctly.

6 p3.s - DP with imm as second op

6.1 Simulation results

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

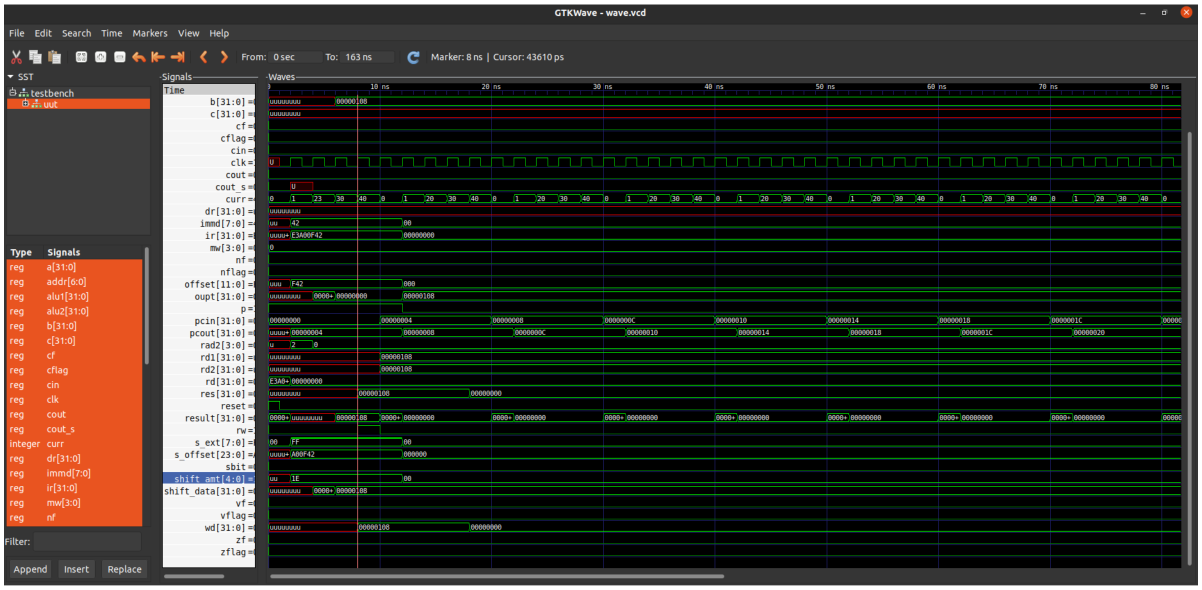


Figure 3: Program 3

Here I have given the constant as 264, we know that constants till 255 can be accommodated by immediate operands without using ror operations, so I have gave 264 here, now we can check wd is 264 at the time when rw is 1 which implies my ror operation is correct as well as the fact that this feature works correctly.

7 p4.s - DT with register as second op

Note: It was mentioned in the pdf that DT with register as second operand will have only constant as shift-operand.

7.1 Simulation results

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

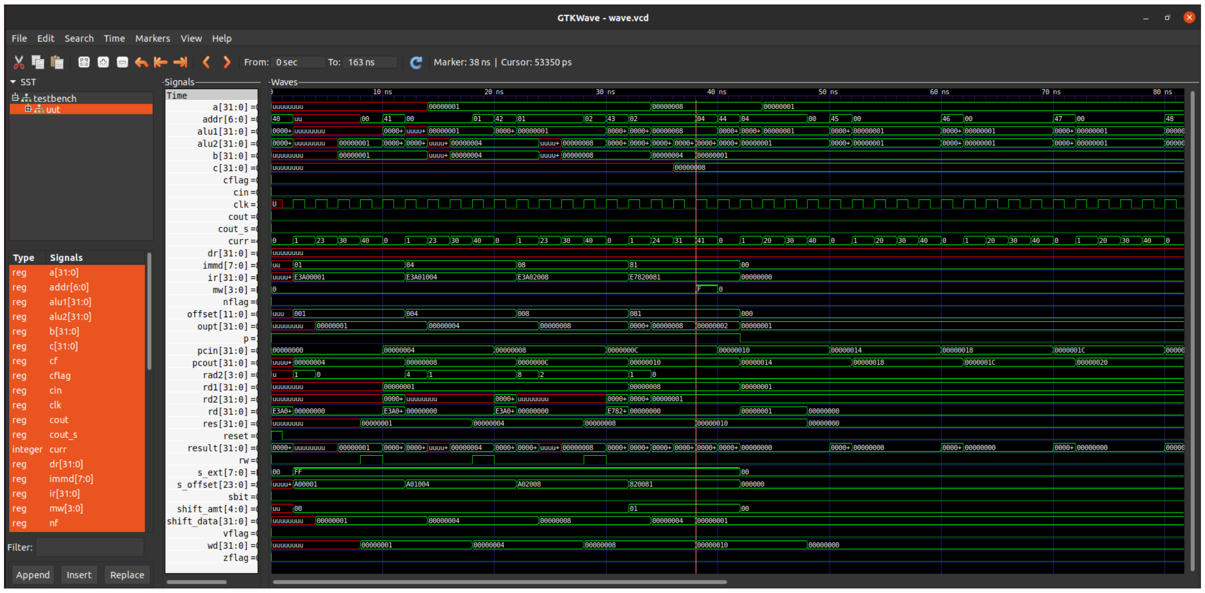


Figure 4: Program 4

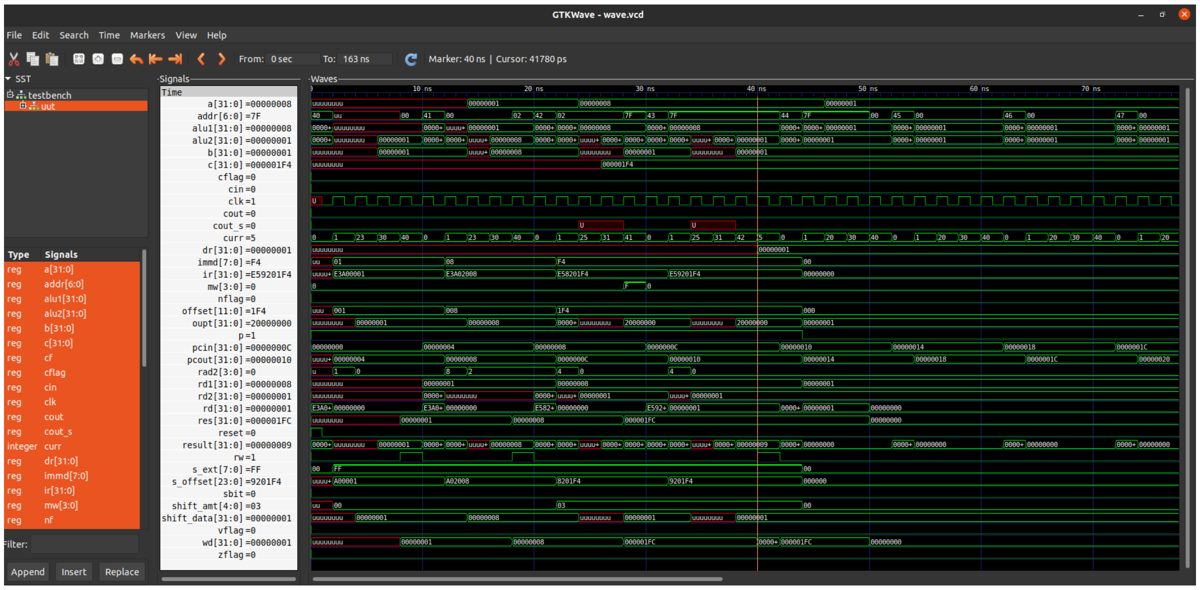
Here we can check that the final address is stored in the register RES, and it is 16, which is correct since I did r2(8)+ r1(4), with r1 having LSL as 1 so we have 8+4*2 = 16, and the data to be written is 1 which is in B, thus verifying the correctness of my code.

8 p5.s - DT with imm as second op

Here we don't have any shift/rotate operations.

8.1 Simulation results

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



```
Time
      a[31:0] =00000008
     addr[6:0] =7F
    alu1[31:0] =00000008
    alu2[31:0] =00000001
       b[31:0] =00000001
       c[31:0] =000001F4
         cflag =0
           cin =0
           clk =1
          cout =0
        cout_s =0
          curr =5
      dr[31:0] =00000001
     immd[7:0] =F4
      ir[31:0] =E59201F4
       mw[3:0] =0
         nflag =0
  offset[11:0] =1F4
    oupt[31:0] =20000000
             p =1
    pcin[31:0] =0000000C
   pcout[31:0] =00000010
     rad2[3:0] =0
     rd1[31:0] =00000008
     rd2[31:0] =00000001
      rd[31:0] =00000001
     res[31:0] =000001FC
         reset =0
  result[31:0] =00000009
            rw =1
    s_ext[7:0] =FF
s_offset[23:0] =9201F4
          sbit =0
shift_amt[4:0] =03
shift_data[31:0] =00000001
         vflag =0
      wd[31:0] =00000001
         zflag =0
```

Type	Signals
reg	a[31:0]
reg	addr[6:0]
reg	alu1[31:0]
reg	alu2[31:0]
reg	b[31:0]
reg	c[31:0]
reg	cf
reg	cflag
reg	cin
reg	clk
reg	cout
reg	cout_s
integer	curr
reg	dr[31:0]
reg	immd[7:0]
reg	ir[31:0]
reg	mw[3:0]
reg	nf

Figure 5: Program 5

Here, we can see that wd = 1 when rw = 1 the last time, thus imm operation with str and ldr works correctly.

9 p6.s - Testing ASR

9.1 Simulation results

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

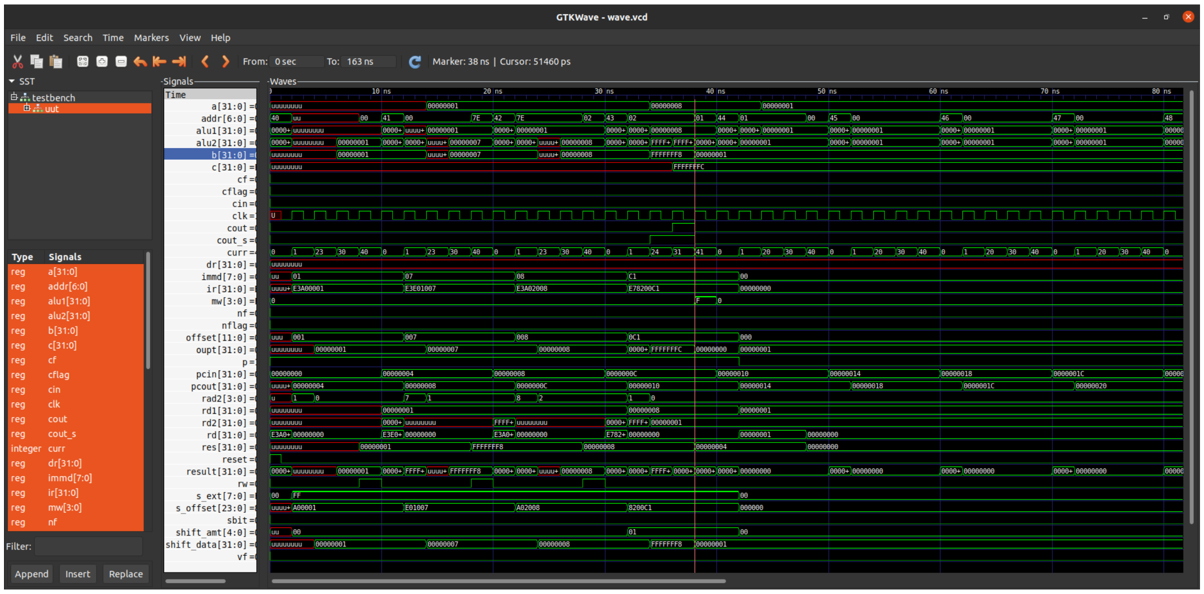


Figure 6: Program 6

Here, I have the final address in RES which is 8-8/2 = 4, which is verified by the epwave, hence verifies that my code is correct.

10 p7.s - Testing other instructions

10.1 Simulation results

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

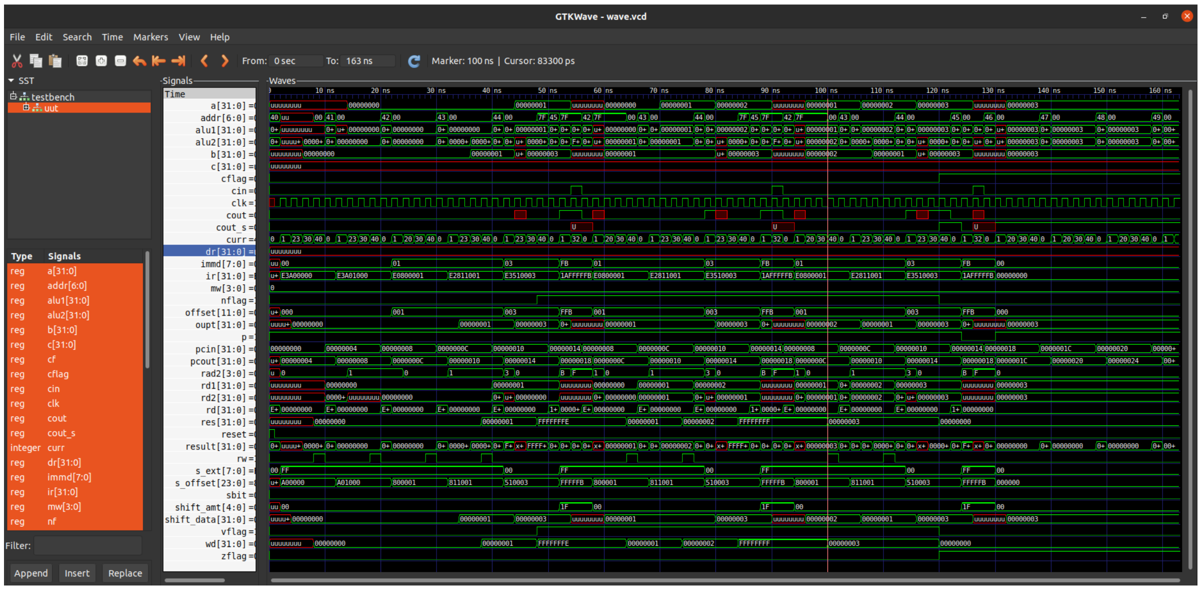


Figure 7: Program 7

This program checks if branch and cmp instructions work properly on adding shift/rotate operations. We can follow the wd values when rw is '1' and we see that finally it is 3, which is the correct value before we come out of the loop, hence verifying my code is correct.

11 p8.s - Testing other instructions

11.1 Simulation results

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

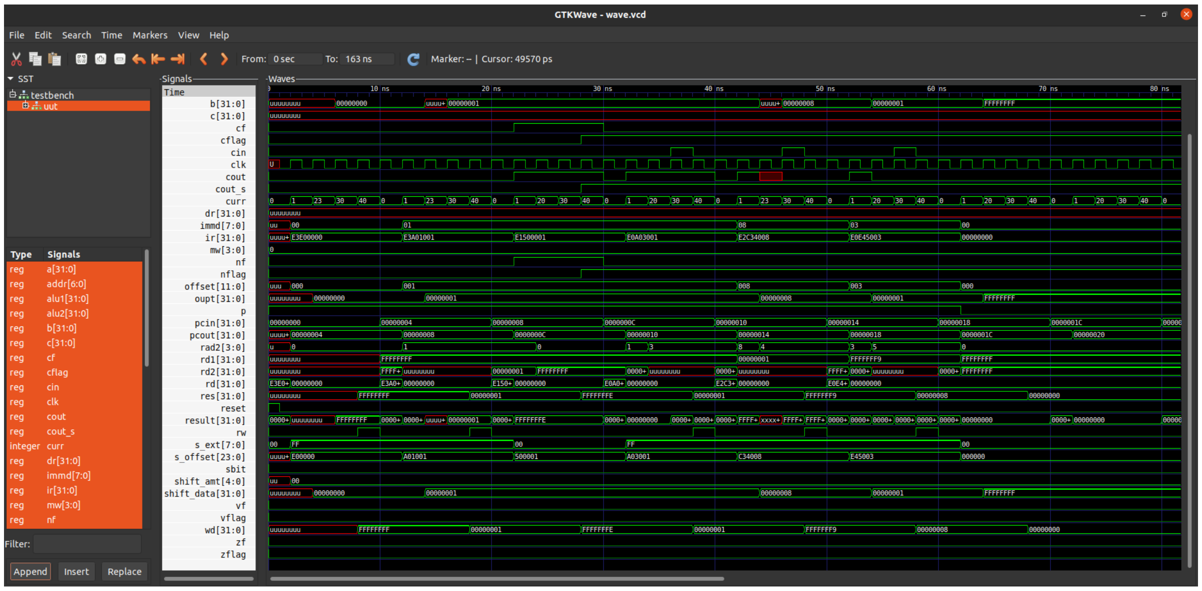


Figure 8: Program 8

This program checks if my flags get set properly on adding shift/rotate operations. Here, we can see that carry flag gets set when I do cmp 1, -1 since it is a subtraction, hence we get a carry-out and I use this carry out in adc, sbc and rsc instructions, and thus I get 1, -7 and 8 as the answers which are seen in the wd signal when rw is 1.