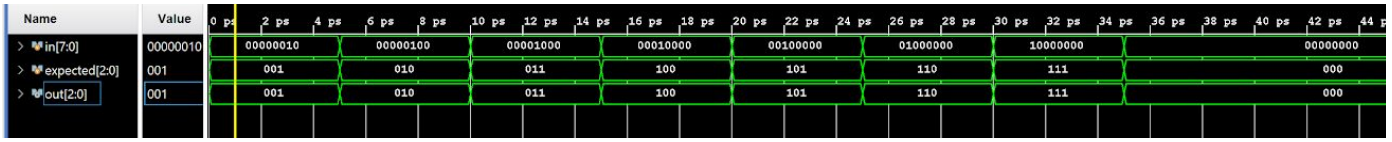
HW 1 Testbench Output

Encoder

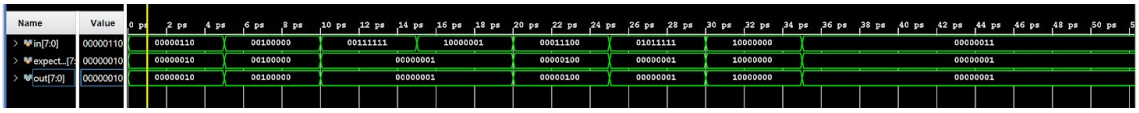
00000010 -> 1, expected 1
00000100 -> 2, expected 2
00001000 -> 3, expected 3
00010000 -> 4, expected 4
00100000 -> 5, expected 5
01000000 -> 6, expected 6
10000000 -> 7, expected 7
00000000 -> 0, expected 0



Inputs and tests were generated with use of a for-loop, which can be seen as in, expected and out are of length 7. In each case of a new input, the expected can be seen to be the same as the output. Each output represents the input (one hot representation) in 3bit binary. We see that as the 1 gets shifted to the left, the out value increases by 1 as expected.

Arbiter

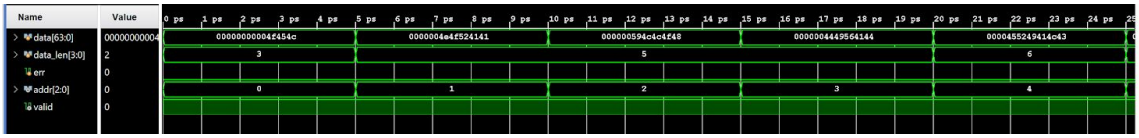
00000110 -> 00000010, expected 00000010 //given case
00100000 -> 00100000, expected 00100000 //given case
00111111 -> 00000001, expected 00000001 //check if multiple 1's in the input, does it pick LSB
10000001 -> 00000001, expected 00000001 //if 1's are on either end of the input, does it pick LSB
00011100 -> 00000100, expected 00000100 //if 1's are in the middle of the input, is output as expected
01011111 -> 00000001, expected 00000001 //if there are gaps between the 1's in the input
10000000 -> 10000000, expected 10000000 //if the MSB is a 1
00000011 -> 00000001, expected 00000001 //if last two bits are 1's, does it pick the LSB



Inputs and tests were generated with use of a for-loop, which can be seen as in, expected and out are of length 7. The expected values match with the output. In the 3rd and 4th input (00111111 and 10000001), you can see that these different inputs result in the same output of 00000001 because it should return the LSB, as expected.

Cam

All tests in test bench passed successfully!



We see that the input data is the username in hexadecimal. It's data length represents the length of the username, and the addr represents its address on the user database table. For example, input 00000000004f454c represents Leo whose name is length 3 and address is 0 as expected.

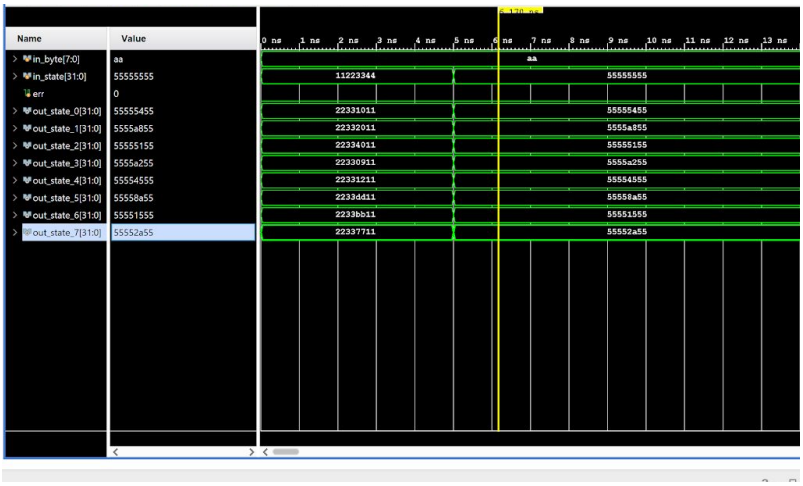
Hash_rom

All tests in test bench passed successfully!

In this case, the module takes in an address and outputs the stored hash password for that user. For example, we see that the stored hash for address 1 is for Aaron and the stored hash matches the hash in the address table as expected.

Hash_round

All tests in test bench passed successfully!



It takes the input and splits it into different “rounds.”Each out_state corresponds to a different round, where they are then put into a rotator. A seemingly random output that looks different from the initial input is created. For example, out_state_0 represents the 8 LSBs being put into hash_round 0 and then going through a 32-bit wide rotator. The output of this is as expected.

Hasher

0039373931414644,7,dc1a2c9e
454c555253574f43,8,dc2ea8e4
00004e55524e5552,6,355facc3
0000535552414349,6,aaf4adc9
0000454958454958,5,13d41ced
4f4e5247334c3350,8,7ebcf8a8
544c41574e523046,8,f3cddb9b
0000415453315241,6,9948e6be

0000000012153524,1,b9b6a7db
ffffffff8484d609,3,cefcd881
0000000006b97b0d,d,9cee285f
fffffffb2c28465,2,a0431f6f
0000000000f3e301,d,99d4b8ba
000000003b23f176,d,a1d6a92d
0000000076d457ed,c,af08e2c0
000000007cfde9f9,6,6720ec8c
fffffffe2f784c5,a,df4e8378
0000000072aff7e5,7,32a67eac
ffffffff8932d612,f,e89dabf2
00000000793069f2,e,831ef382
fffffffff4007ae8,5,901fd15a
000000002e58495c,d,a38e00c7
ffffffff96ab582d,5,ce3808cc
fffffffb1ef6263,a,4e8a6e60
fffffffc03b2280,0,18dd1f93
00000000557845aa,d,aaa5b3e2
fffffffcb203e96,3,2e9aef03
ffffffff86bc380d,3,6ab93d7f
00000000359fdd6b,5,c660eed2
ffffffff81174a02,e,31235393
000000000effe91d,f,ab6255ed
0000000011844923,a,bf27da4e
fffffffe5730aca,c,c8bd9dbe
000000007968bdf2,a,982436f8
0000000020c4b341,8,5a4c1ca3
000000003c20f378,9,eacafce0
0000000075c50deb,6,69acafe7
00000000634bf9c6,e,30b7a9a9
fffffffde7502bc,a,72bb6135

ffffffff85d79a0b,1,da5a68c8
0000000042f24185,f,4c637be0
ffffffff9dcc603b,a,08c562c0
ffffffffbf23327e,5,2d7c2b71
0000000078d99bf1,9,468f0954
0000000031230762,c,ff426e86
000000004fa1559f,f,3ed3068c
000000007c6da9f8,7,b3b5552a
ffffffffcfc4569f,c,02bac1ca
ffffffffadcbc05b,9,816eefe2
ffffffffa4ae3249,0,99f70bfa
ffffffffebfec0d7,1,b39752b7
000000004b212f96,c,8fa7ada2
ffffffffe12ccec2,8,056e28f2
ffffffffbb825a77,d,cc85fa27
00000000090cdb12,e,c0286788
0000000036e5816d,9,a3944bc6
000000000fd28f1f,3,a83ba99a
0000000042d92f85,8,3e9528ba
000000002dda595b,9,79489d78
ffffffff9ff2ae3f,a,976f683b
000000002c156358,6,f36fd6bd
ffffffffc71a0c8e,c,3d7af8cb
000000007d3599fa,6,e3326cff
0000000039961773,3,196dbe21
ffffffff9799a82f,3,52ad1420
ffffffffafd8565f,4,c12c78e5
000000007bf8fdf7,b,a34d381f
fffffffff3091ae6,a,c4cf43c8
0000000014cfc129,d,f34c1e9b
ffffffffed536cda,5,2fee5f80
ffffffffda8ae2b5,f,c0e67432
000000003cf11979,4,f3d430db
ffffffffe8740cd0,a,52a82ea6
0000000055f6adab,e,2be7adc0
000000006e5daddc,a,40a9bb98
fffffffffedf72fd,3,0aad1f07
000000002b0eed56,e,67d91208
ffffffffb3d97667,a,43680458
000000005b6fb9b6,8,e865d691
000000003cd18779,8,c9469c32
000000004a74bf94,3,590c188c
ffffffff823f2c04,9,1d5e22cf

000000006dcb69db,d,b6e4e9bf
000000006cb0b7d9,d,b2adfca2
ffffffffbb45e276,a,64cc7740
000000005b172db6,5,6f4cccac
ffffffffa3071a46,4,c692c79c
000000007bd261f7,9,f2a3d244
ffffffffda6ebab4,8,46101108
00000000147cd928,d,f97588db
ffffffffe3c530c7,e,7804e491
ffffffff8477e408,c,908db630
fffffffffea7a6fd,9,f694b5d5
ffffffff8e37901c,6,041f2024
ffffffffed3408da,d,c71a2f73
00000000334ea766,0,cfbe1723
ffffffffb9f50473,a,0d68b3c0
000000002f3ab35e,a,1a3caa40
000000006a8e05d5,a,6e68aaf4
ffffffffdcf000b9,7,f0de5f5f
000000004b273796,0,a9c0af77
0000000013259f26,6,5dce6e09
000000003e99837d,c,8899c23a
0000000043615786,8,72ab33c8
000000003f5a9b7e,b,fbc6120e
ffffffffe7c3b6cf,9,ab6b64f0
fffffffffd28e4fa,1,5d4c1a68
000000000b940917,1,d8178093
0000000043779186,0,0cc8bf67
000000007a8c59f5,5,4ae05b4c
ffffffff949a8a29,1,431b9599
ffffffffe2e574c5,8,a3d6f1ff
0000000025b27b4b,3,9aaeae6a
fffffffff622e6ec,a,98441378
000000002758d14e,8,d2cb4d98
00000000549efda9,1,75e2c929
00000000070bb90e,6,830408a8
ffffffffcfd6c09f,a,8de8d430
00000000155a1d2a,d,cd96cbe3
000000004f75ff9e,8,06bb985b
ffffffffbccfa879,8,6994528f
00000000652345ca,3,8ff4e7e8
0000000035a0c96b,7,6997b2bb
000000005b0bddb6,a,aadc4bc0
000000006216abc4,9,4e48db4c

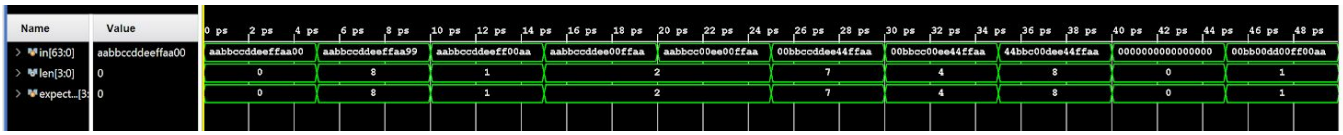
00000000492fd392,4,71ca2fe9
000000003fbb3b7f,6,7028d827
000000007d6df5fa,2,0db4a5ac
0000000019452132,d,cdb52665
00000000424fcd84,4,ba56c530
000000006543cfca,9,5fcc9eb5
ffffffffd095a8a1,e,cacb75e0
fffffffffd8b6afb,b,d85e52bd
ffffffff78290ef,9,ce50ea02
000000001b60e536,5,87e494fc
ffffffffc7e8568f,b,05f6c92c
000000004465e788,e,8c0b044f
000000004df3819b,2,2dbb9edf
000000001444df28,d,6b4f8a54
0000000025b75f4b,2,ee493ad3
ffffffff8f1cf61e,d,c0483ea5
000000007679fdec,8,934da9fb
0000000068ae1bd1,6,2c205f8d
ffffffffa0c02441,b,d8703b78
000000006c44f9d8,3,921618bd
ffffffffab196256,b,220dbe15
fffffffff166fae2,4,bd106a31
0000000039ac0373,8,2599ab49
00000000093e4d12,8,a48ab37f
ffffffff9c811239,5,da5e5785
ffffffffd0c5dca1,b,4b715ce4
0000000040905d81,8,09b8484c
0000000013b55527,1,189c4609
ffffffff8f8c6e1f,4,9e04cc7e
000000002c2d2358,6,bec5b978
000000000a6e9314,2,74e32568
ffffffffcb227096,1,94359397
000000002ac2d555,d,45f5a842
00000000158b2b2b,5,a573f364
0000000056b403ad,7,f259aac3
000000004249ff84,7,0604887b
fffffffff3d7a6e7,9,43c578e6
ffffffffa4da5649,b,519429b5
0000000064ba0fc9,1,d6aa801f
ffffffffd0bc5ea1,a,e0bae0a2
000000007d2a45fa,5,24772fbb
0000000041a10583,c,03c7c13f
ffffffffb9461472,e,d5accabe

ffffffffb455f268,f,fd6e067a
0000000043460d86,0,45bc9fe9
000000001c719738,0,7379af7c
ffffffff94097628,6,8f130c96
ffffffffe2bf1ac5,0,adf66292
000000003a625f74,9,7d6bbca2
ffffffffd86a6ab0,c,4e6342ea
000000001521932a,2,db258868
000000000aec3515,1,544aca8f
000000000be29d17,3,b2306548
0000000064b5e3c9,6,c6cdc8ed
000000001297cb25,1,8de26d07
ffffffffc69da28d,a,679af6b3
0000000003d62707,c,8f70e91c
00000000060a5d0c,1,7aa00142
ffffffff9de17c3b,6,4a13e334
ffffffffbdfc2f7,e,7b91ab84
ffffffffae78585c,5,30282ebf
ffffffff902a3a20,0,bb5daee8
0000000039600972,4,34d02991
000000006e8af5dd,d,c2b8d726
0000000025b0994b,9,08ba8881
ffffffffcf63da9e,d,d142d996
ffffffffbde0d27b,f,34d13aee
ffffffff81c39a03,3,51701bc0
000000000e92431d,1,aad5e8e3
0000000022119f44,5,84ae90cb
fffffffff01d34e0,d,d9ebfd7c
00000000297a1552,8,7c0b9ecd
0000000046dcb78d,2,9d080a43
000000004219e784,6,043a58bb
ffffffffc63a928c,0,77509f68

The input is a sample password which is then changed by the hasher to be a seemingly random output represented by the var out.

Length_finder

// given tests
aabbccddeeffaa00 : len = 0000, expected len = 0000
aabbccddeeffaa99 : len = 1000, expected len = 1000
aabbccddeeff00aa : len = 0001, expected len = 0001
aabbccddee00ffaa : len = 0010, expected len = 0010
aabbcc00ee00ffaa : len = 0010, expected len = 0010
00bbccddee44ffaa : len = 0111, expected len = 0111
00bbcc00ee44ffaa : len = 0100, expected len = 0100
44bbc00dee44ffaa : len = 1000, expected len = 1000
0000000000000000 : len = 0000, expected len = 0000
// new tests
// lots of 0 bytes added
00bb00dd00ff00aa : len = 0001, expected len = 0001 // 0 bytes added throughout, especially at the most significant byte
bb00dd0000eeffaa : len = 0011, expected len = 0011 // 0 bytes sprinkled throughout to ensure proper count
bb00dd0000eeff00 : len = 0000, expected len = 0000 // 0 bytes sprinkled throughout, especially at the least significant byte which should be counted first
bb00dd0000000000 : len = 0000, expected len = 0000 // the least significant byte is 0, therefore it should ignore the rest and be 0

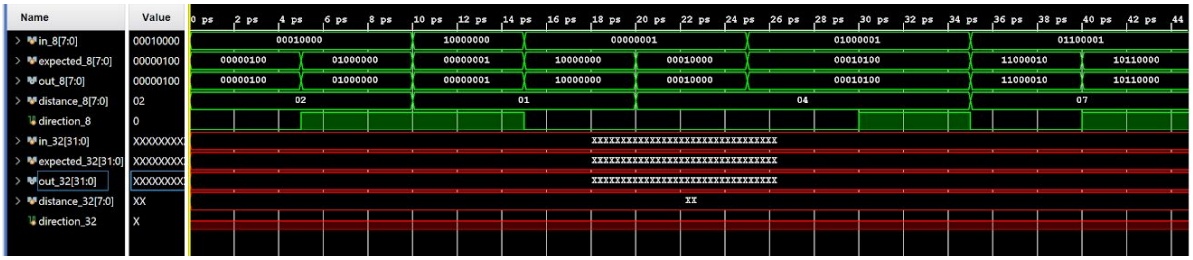


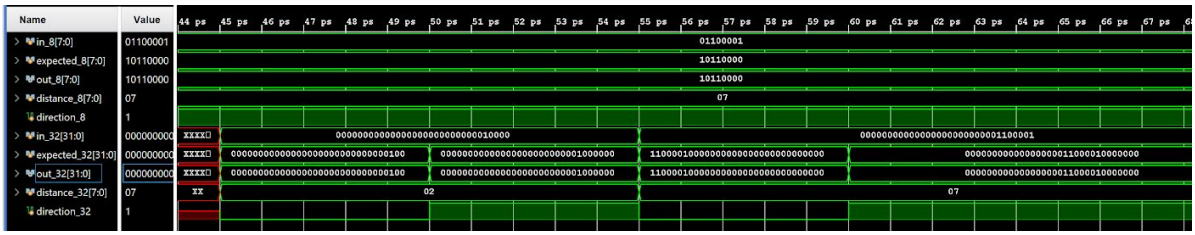
in represents the input, the len found starts counting from the LSB so if it starts with a 00 byte from the right hand side, then disregarding the rest of the bytes, it will output 0 as the length. In other cases where it does not start with a 00 byte from the right side then it will count bytes until it hits a 00 byte. For example, in aabbccddeeffaa00, the length is 0 as expected. If we move the 00 to be aabbccddeeff00aa, then the length is 1 as expected. If there are no 00, then the length is 8 as expected.

Rotator

// 8 bit
// basic shift of two
00010000 -> 00000100, expected 00000100 // right shift
00010000 -> 01000000, expected 01000000 // left shift
// new cases we added from this point on
// edge case
10000000 -> 00000001, expected 00000001 // left shift
00000001 -> 10000000, expected 10000000 // right shift
// wrap-around, distance of 4
00000001 -> 00010000, expected 00010000 // right shift
// two 1s, wrap-around, distance of 4
01000001 -> 00010100, expected 00010100 // right shift
01000001 -> 00010100, expected 00010100 // left shift
// three 1s, distance of 7
01100001 -> 11000010, expected 11000010 // right shift
01100001 -> 10110000, expected 10110000 // left shift

// 32 bit
// basic shift of 2
00000000000000000000000000010000 -> 00000000000000000000000000000100, expected
00000000000000000000000000000100 // right shift
00000000000000000000000000010000 -> 00000000000000000000000001000000, expected
00000000000000000000000001000000 // left shift
// three 1s, shift of 7
00000000000000000000000001100001 -> 11000010000000000000000000000000, expected
11000010000000000000000000000000 // right shift, wraps around
00000000000000000000000001100001 -> 00000000000000000011000010000000, expected
00000000000000000011000010000000 // left shift

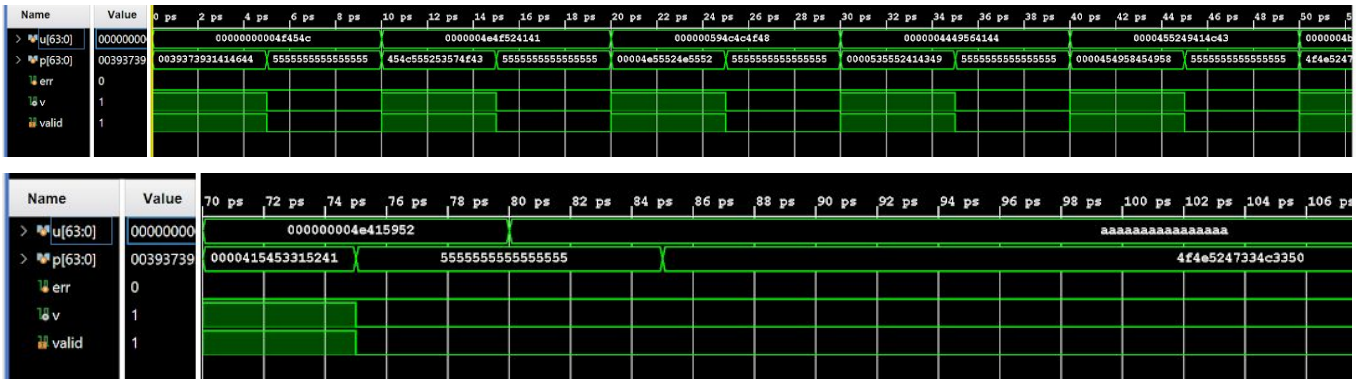




Anything with _8 following the var name is for 8 bit inputs, _32 is for 32 bit input. Distance is how many times the bits should be moved over, and direction indicates whether it should be a right or left shift. You can see that in the first picture anything that is for 32 bit related inputs is red with XXXs as the values for it had not been assigned yet. You can see the 8bit cases iterate through with the 32 cases as red until the 32 cases are assigned, then the 8 bit cases remain constant on the last input it was fed.

Verifier

All tests in test bench passed successfully!



U is the username entered, p is the password entered. In the last case, we have a invalid username and a valid password. Because the username is invalid, the address given out is 0 and the respective stored password hash is outputed as hash_out. 7ebcf8a8 is consistent with the stored password hash for Leo who is at address 0. The inputted password is passed through the hasher we get pass_rom_out. The hash given matches the hash for Frank. Since hash_out and pass_rom_out do not match (meaning the stored hash password for the given username does not match the hash for the password), then we get that valid = 0, as expected.