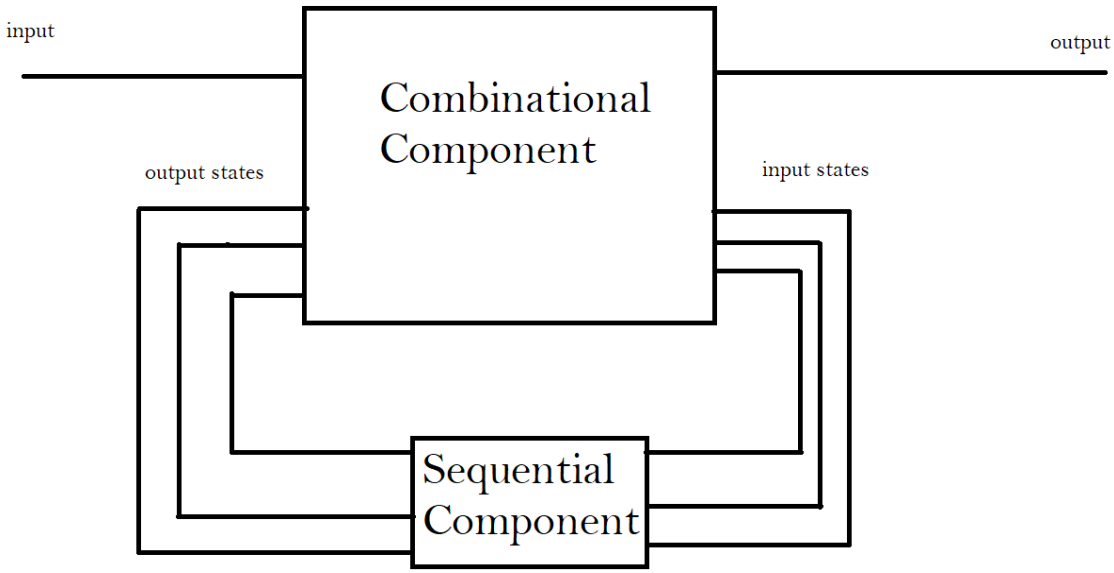
# ECE 5722 Midterm

**By Bruce Huynh**

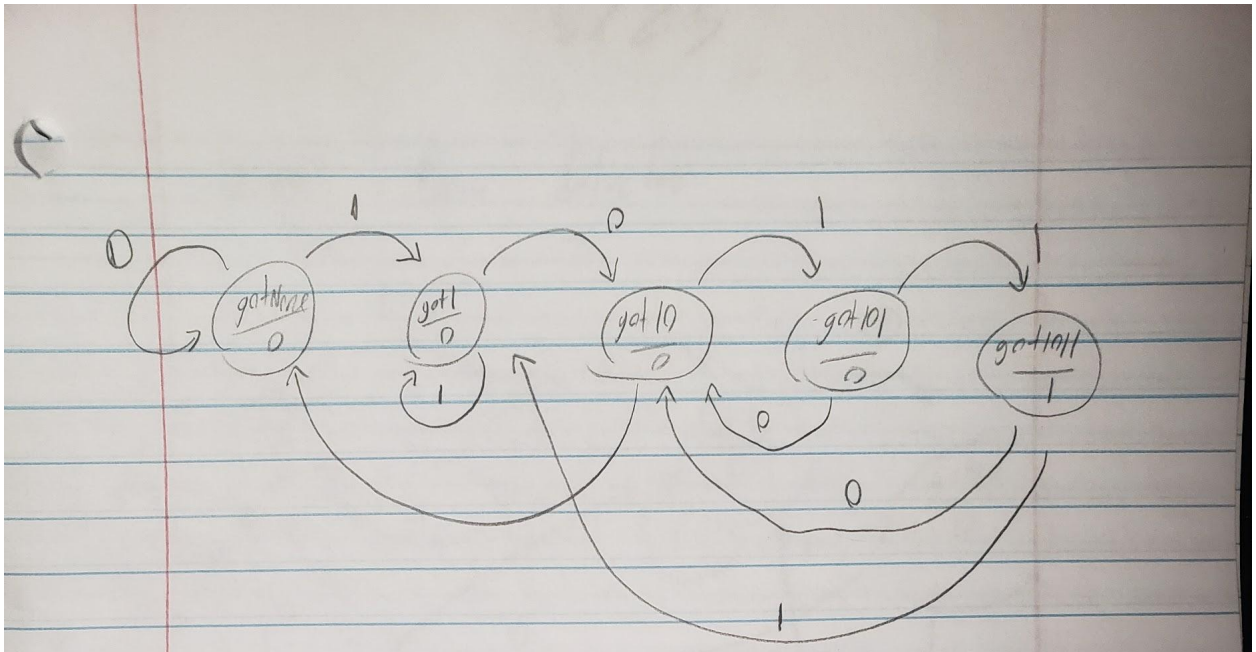
**10/30/2021**

**Problem 1:**

Here is how the combinational and sequential components are separated:

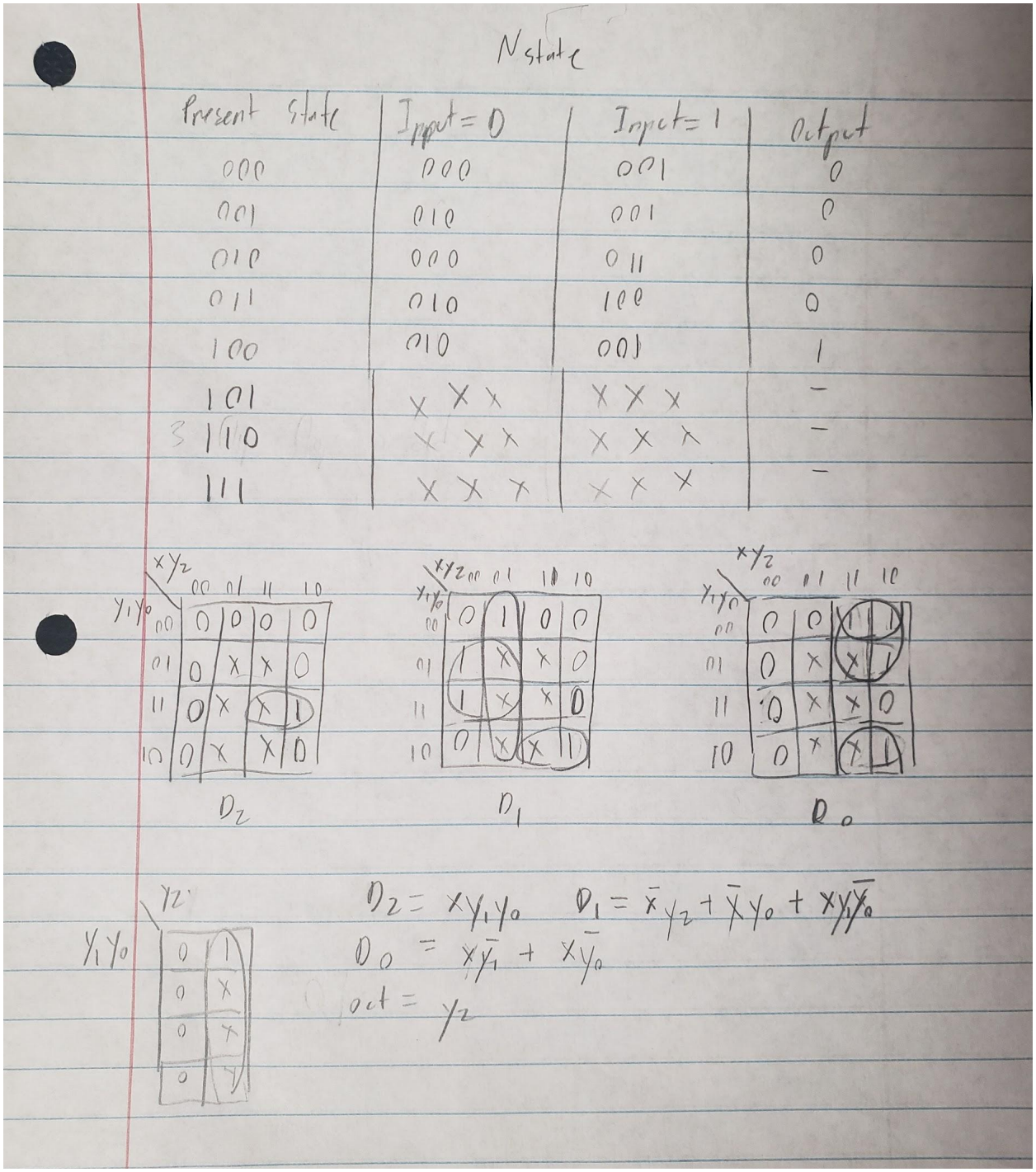


First, I made a state diagram and state table in to get all the variables:




| Present State | Input | Next State | Output |
|---|---|---|---|
| 000 | 0 | 000 | 0 |
| 000 | 1 | 001 | 0 |
| 001 | 0 | 010 | 0 |
| 001 | 1 | 001 | 0 |
| 010 | 0 | 000 | 0 |
| 010 | 1 | 011 | 0 |
| 011 | 0 | 010 | 0 |
| 011 | 1 | 100 | 0 |
| 100 | 0 | 010 | 1 |
| 100 | 1 | 001 | 1 |

Afterwards, I turned the state table into a K-map and then I simplified the expressions of each state:



Next State

| Present State | Input = 0 | Input = 1 | Output |
|---|---|---|---|
| 000 | 000 | 001 | 0 |
| 001 | 010 | 001 | 0 |
| 010 | 000 | 011 | 0 |
| 011 | 010 | 100 | 0 |
| 100 | 010 | 001 | 1 |
| 101 | x x x | x x x | – |
| 110 | x x x | x x x | – |
| 111 | x x x | x x x | – |

$D_2$ $D_1$ $D_0$

$D_2 = xy_1y_0$ $D_1 = \bar{x}y_2 + \bar{x}y_0 + xy\bar{y}_0$

$D_0 = x\bar{y}_1 + x\bar{y}_0$

$out = y_2$

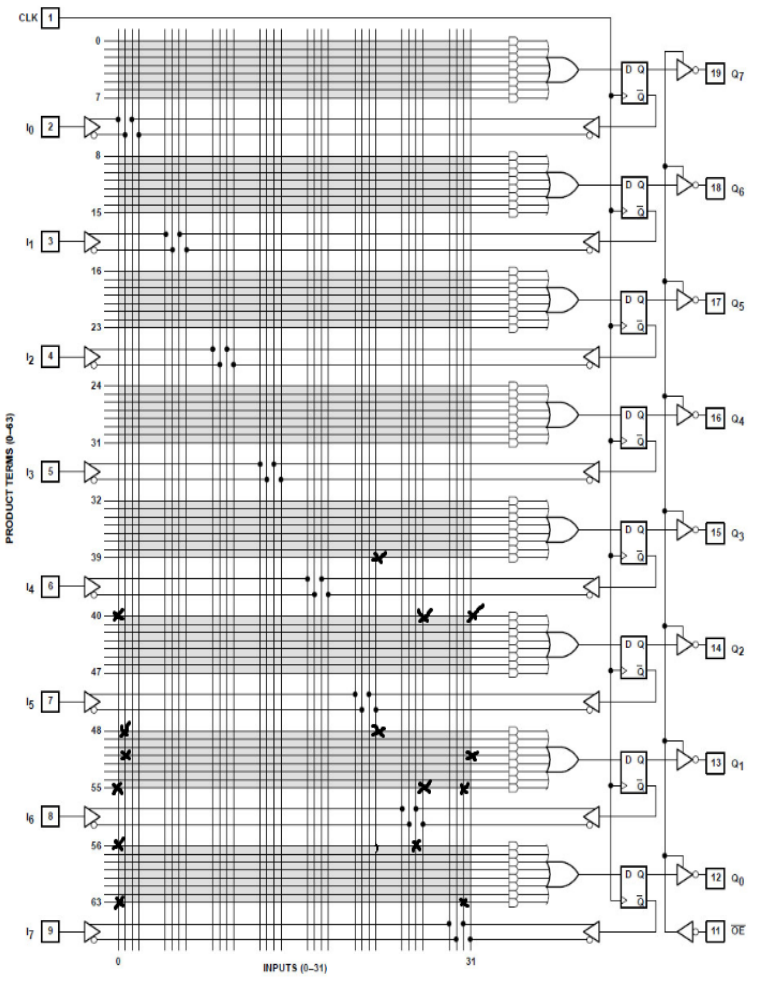
Using the above expressions, I integrated this into the PAL:

out = $Q_2$

$D_2 = xQ_1Q_0$

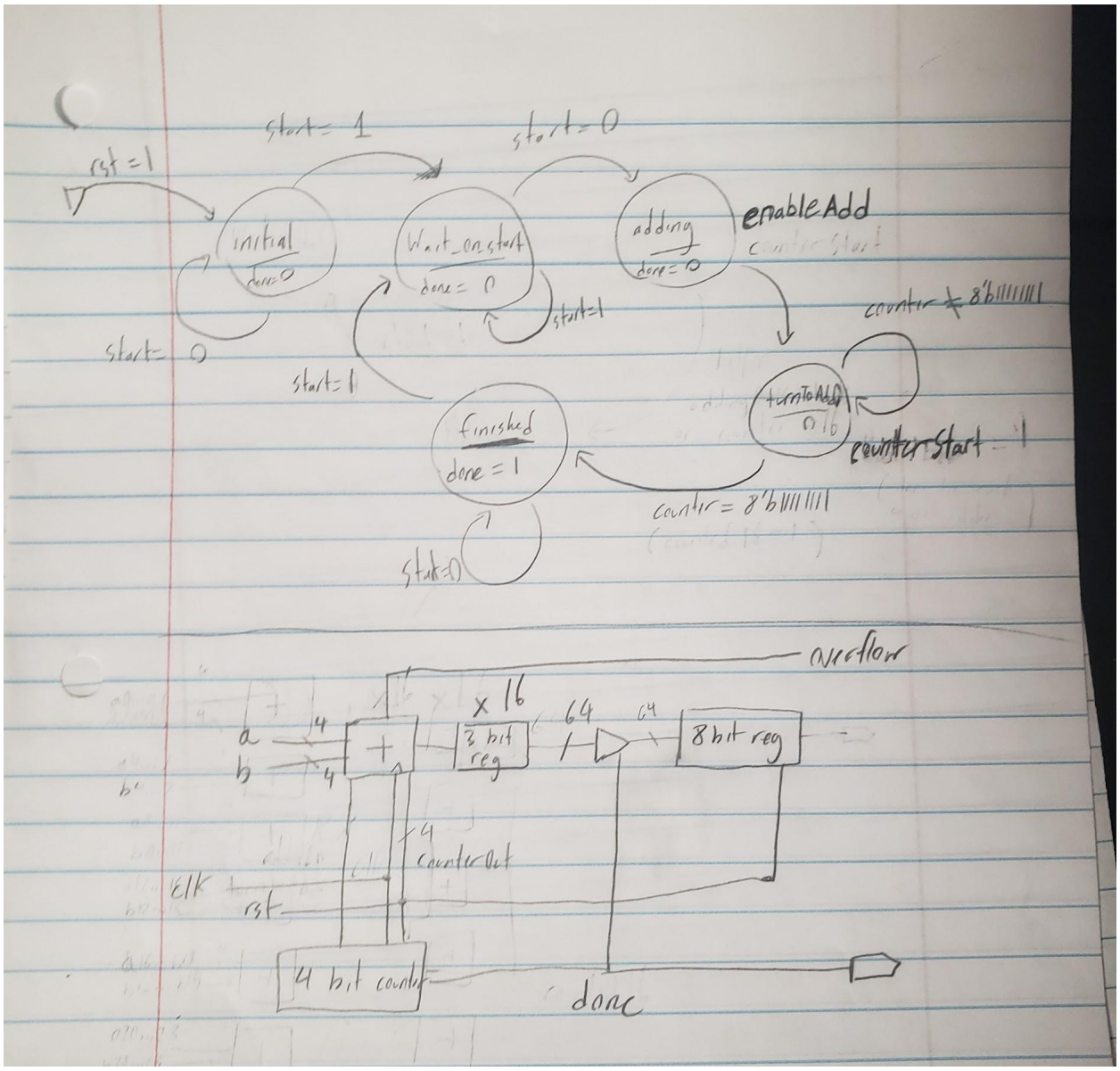
$D_1 = \sim xQ_2 + \sim xQ_0 + xQ_1\sim Q_0$

$D_0 = x\sim Q_1 + x\sim Q_2$



**Problem 2:**

Here is the state diagram and datapath for the serial adder circuit:




The code for the datapath and controller will be provided in a .zip file.

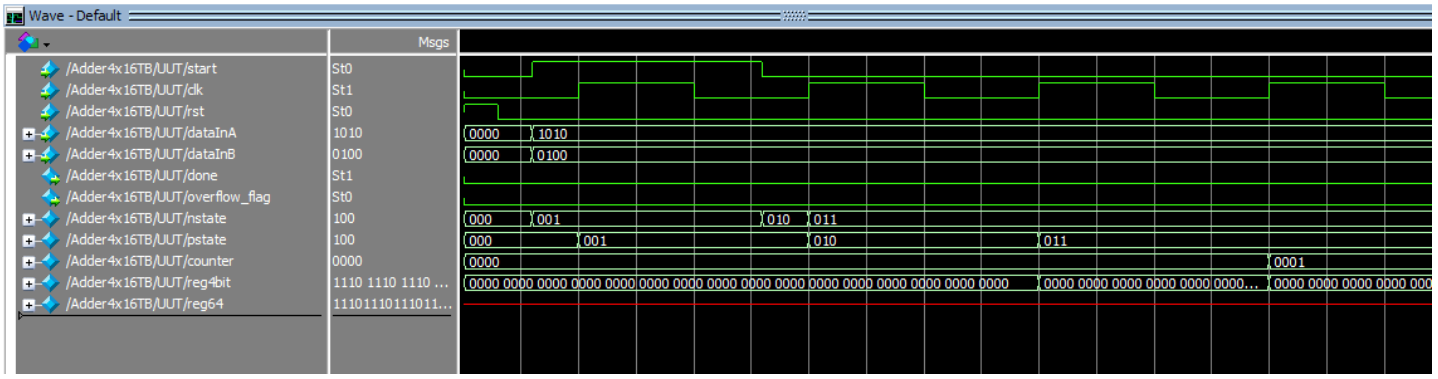
**Testbench:**

For the testbench, we are using four different scenarios.

**Scenario 1:**

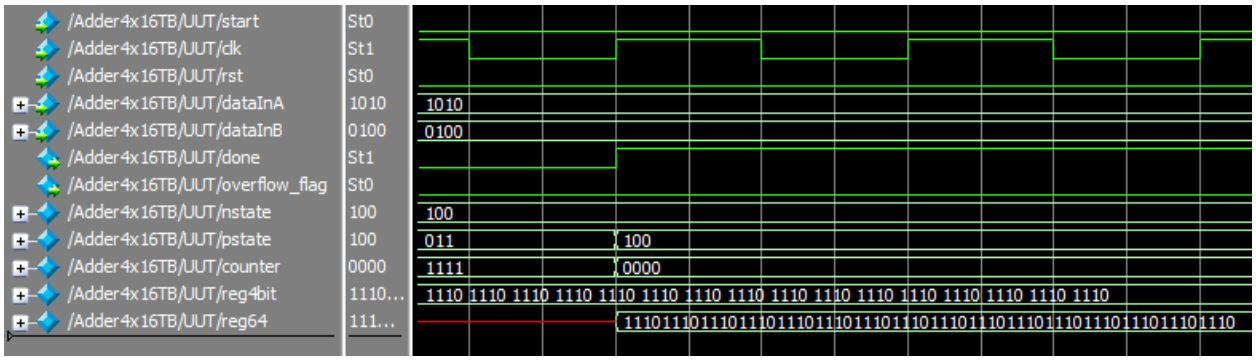
The first scenario is simply adding two non-changing values the entire time (0xAAAAAAAAAAAAAAAA and 0x4444444444444444):

Our expected value is 4919131752989213764 + 12297829382473034410 which results in 1.7216961e+19.




This waveform shows that the counter starts incrementing once the start pulse has been recognized and another pulse which is used to switch states.


Looking at this part, we see that for each 4 bit register, the sum gets added every clock pulse and every tick of the counter. The 64 bit register does not get added until all 16 4-bit counters are filled.




Once the counter counts to 16, it combines the 16 registers to the single 64 bit register. Then the done signal is issued as well. We get a value of 0xEEEEEEEEEEEEEEEE which is 17216961135462248174 which is equal to our expected value of 1.7216961e+19.

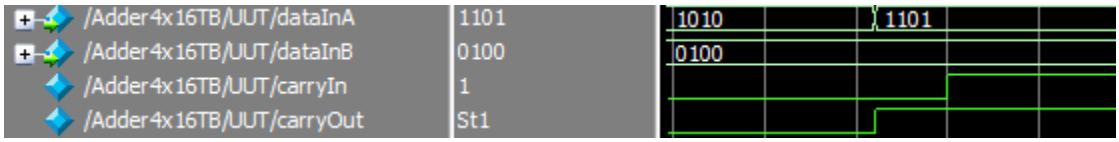
**Scenario 2:**

Everything is mostly the same as scenario 1 except that we change the values during the operation of the circuit: We are adding 0x333333333333AAAA and 0x8888888884444444 (9838263505906844740 + 3689348814741940906) for an expected result of 1.3527612e+19.

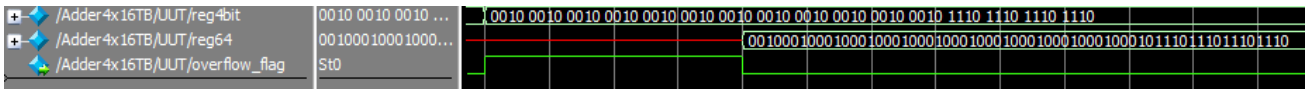

Here are the waveforms switching. In the end we get a sum result of 0xBBBBBBBBB777EEEE, or 13527612320648785646 in decimal which matches our expected value.

**Scenario 3:**

Again, everything is mostly the same except that we are going to be adding values with carry outs. This time we are adding 0xDDDDDDDDDDDDAAAA and 0x4444444444444444 (17216961135462230698 + 4919131752989213764) for an expected result of 2.090631e+19. The reason for the carryout is because we are adding '1101' and '0100' which would lead to overflow.



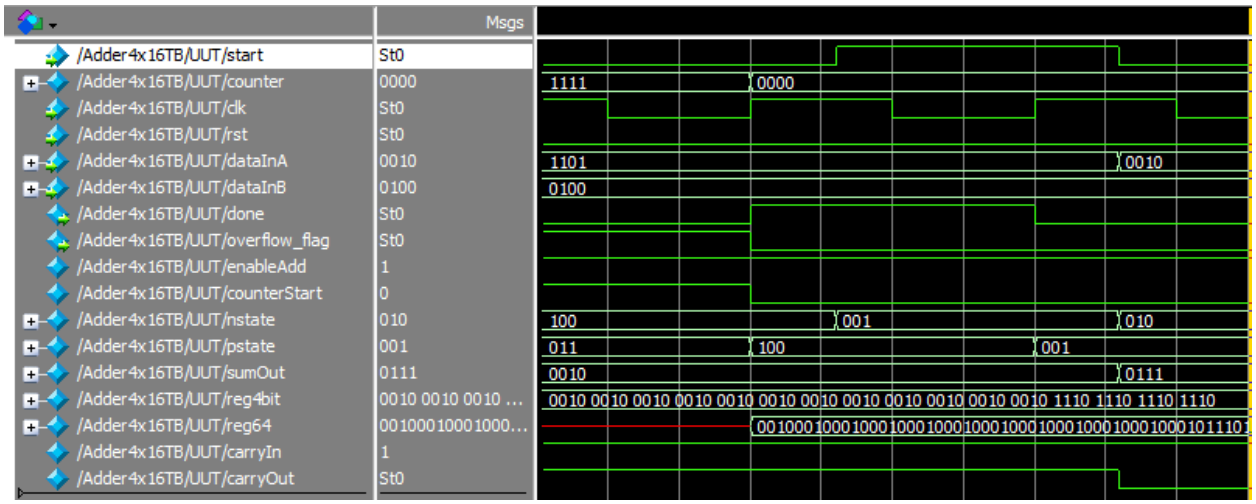

When the values get too large, it overflows but we have a carry in / carry out to mitigate this. These carry ins and carry outs are connected between every register. In the end we end up with this:



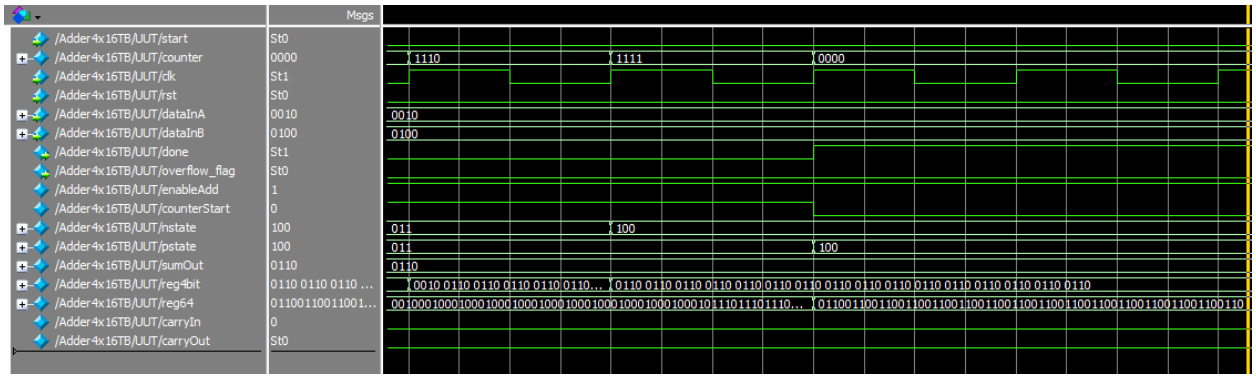

The output isn't correct as the 4 MSB have overflowed. Luckily, I added an overflow flag to let the engineer know. If we take the overflowed bit in account, the result we get is 0x1222222222222EEEE, otherwise known as 20906309950204210926, which is the same as the expected value.

**Scenario 4:**

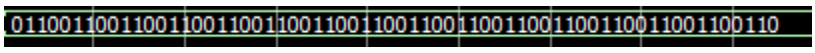
Scenario 4 is almost identical to scenario 3 but we test what would happen to the circuit if it receives another start pulse. After the second start pulse, it adds together 0x2222222222222222 and 0x4444444444444444 which should result in 0x6666666666666666




After some time has passed, the start signal turns on the circuit again. The counter starts the count. Without a reset, the 64 bit register stays at the same value. Each 4 bit register gets replaced with every clock cycle.




When the timer counts to 16, the new 64 bit register value is put in. Everything behaves as it should. The result is 0x6666666666666666 as expected:



**Problem 3:**

**Part A:**

ISR1:

```
clw
wpl
mil r0 01
mih r0 00 - increment
mil r1 01
mih r1 41 -- command
mil r2 02
mih r2 41 -- address
mil r3 03
mih r3 41 -- count
wph
mil r4 00
mih r4 55 -- buffer location
mil r5 01
mih r5 00 -- increment
mil r6 FF
mih r6 55 -- max
L1 add r4 r5 -- increment buffer location by 1
wpl
add r3 r0 -- increment count register by 1
wph
cmp r6 r4
brc L3 -- branch if equal
jpr L1
L3
lda r2 r0 -- load address into register
```

add r1 r0 -- add to bit 0 of command register as a "go" flag

**Part B:**

ISR2 C code:

```
reg1;
dataArray[256];
int main(){
  int MAX = 0;
  for (int i = 0; i < 256; i = i + 1){
    if (dataArray[i] > MAX){
      MAX = dataArray[i];
    }
  }
  reg1 = MAX;
  return MAX;
};
```

ISR2 ESAYEH code:

```
wph
mil r4 01
mih r4 00 -- increment
mil r5 00
mih r5 00
mil r6 FF
mih r6 00 -- max (for loop)
mil r7 00
mih r7 00 -- count
wpl
mil r0 01
mih r0 00 -- increment
```

```
mil r1 00
mih r1 00 -- eSAYEH R1 location
mil r2 00
mih r2 09 -- storage location
mil r3 00
mih r3 00 -- max holder
L1 cmp r3 r| -- see if r2 is larger
brc L2 -- branch to L2 if it is
add r2 r0 -- increment memory address
wph
add r7 r4
cmp r6 r7
brc L3
jpr L1
L2 mvr r3 r2 -- move r2 into r3 as new max
jpr L1 -- go back to L1 after setting new max
L3 sta r1 r4 -- store max into r1 after the loop is done
```