Gebze Technical University

CSE232 – Logic Circuit and Design

#Project 2

Multiplication of two 16 bits binary numbers.

Nevzat Seferoglu

171044024

*Project Description :

You will design a machine that can compute the multiplication of two numbers without using multiplier. Your machine takes two 16-bit numbers and compute the multiplication result of these two numbers. It has two blocks:

1. Control Unit: Has the FSM inside.
2. Datapath: Has all required components like adders, registers, multiplexers and etc.

You will actually implement the following C code with your machine.

```
mult = 0;
while( a > 0 ){
mult = mult + b; a = a – 1;
}
```

Use registers for a, b and mult. Use one adder for addition and subtraction.

BONUS:

If you can also perform negative number multiplications you get extra credit. (25pts)

1. Decide states and draw the state diagram for your FSM controller.
2. Draw datapath.
3. Draw truth table.
4. Derive Boolean expressions from the truth table.
5. Draw the circuit on Logisim.
6. Simulate and see whether it works. If it does not turn back to previous stages and check each carefully.
7. You get low credits if it does not execute in Logisim.
8. Submit your report including all the above stages (from 1 to 5) to the given submission link. Also submit your Logisim .circ file. Please indicate which parts of the project work and which ones do not precisely in your report.
9. It is not a group project. Cheating results in -100.

*Problem Solution Approach :

Design , which is asked for , consisted of three main component. These are Controller , Datapath and main circuit. After all these implementation is done , we can easily test our circuit and view the result of calculation. Before designing all these stuff , we need to design FSM which is going to a first created **finite state machine** of our circuit. I think designing FSM is the most crucial part of the project. Because I made some mistakes while designating conditions on transitions between the states of FSM. It does not affect the main purpose of the project. But if I had noticed before, I could have got 25 more points.

***** I will explain that mistake on the **Mistakes** section in detail.

*Datapath :

A datapath is a collection of functional units such as arithmetic logic units or multipliers that perform data processing operations, registers, and buses. Along with the control unit it composes the central processing unit (CPU). A larger datapath can be made by joining more than one number of datapaths using multiplexer.

*Controller :

The control unit (CU) is a component of a computer's central processing unit (CPU) that directs the operation of the processor. It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor.

*Datapath Designing Approach:

There are some input pins and output pins which are connected to datapath. According to process that is executed by the datapath , some output pins will be true , the others will be false. Controller takes that outputs as an input value for itself.

Jobs that Datapath does :

* Addition using an adder.
* Negation.
* Multiplexer with its input pins which will be controlled by the controller.
* Comparison of two 16 bits binary numbers.
* Embodying memory components at one place.
* Keep some constant

***Controller Designing Approach:**

Controller can be likened to brain of the human. Because , every single step is under the control of that component. These are input pins of datapath and even datapath component itself. We cannot fulfil the datapath without controller. It means like human without brain. Controller takes output of the datapath as an input of itself and exploit these variables to determine next state which are remarked by the FSM. Other than this , it also manages the input of datapath with its input pins.

Jobs that Controller does :

* Determining next state according to table.
* Giving their actual value to datapath input pins according to current state.

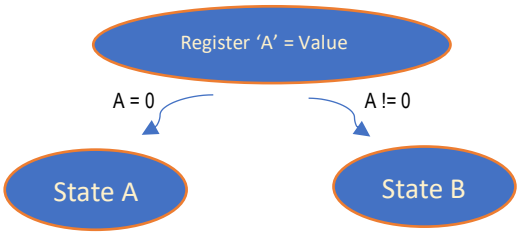
*Approach Result :

Like I have remarked on the top. We need to make meaningful connection between datapath and controller. These connections are just depending on how we design our FSM , other parts of implementation are non-deterministic.

**Mistakes that I have made :

I made a lots of mistakes while designing FSM. I have had a chance to come back and fix the mistake. After I have made sure that final FSM worked correctly , I passed through designing datapath process.

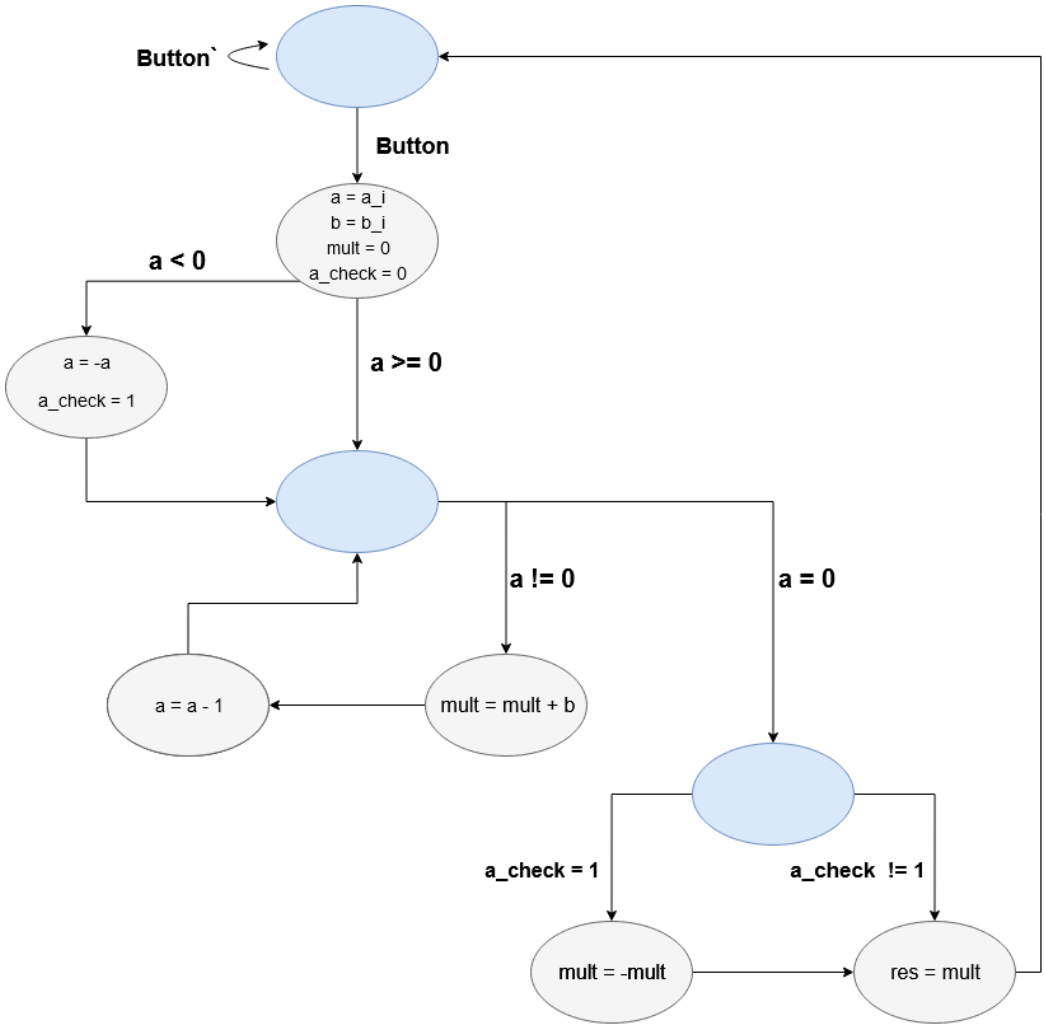
After the part that said above , I have not had no opportunity to come back to fix the problem because I could not even know whether there is an erroneous situation on my interior design of FSM. Although I design my FSM which comes with solution that is also **valid for negative number multiplication** , **things have not gone in correct way** that I have programmed on FSM at the end of the project. After all the things , that is done , such as creating table , simplifying equations that comes from tables , realizing erroneous there were nothing to do, because it was not more fixable. After hours of **debugging**, I determined what was wrong and where it revealed.



There is a loading operation to memory component of the datapath. I supposed that I could make some comparison with that loaded value. But the situation that I missed out is loading to register or kind of memory component occurs at the end of the synchronized clock. Therefore , the next state is determined by the precious value that is already in register and it does not work as I determined on the FSM.

***Note : The solution was adding an extra state which does not anything and let value to be store. I realized that when whole my project has already done. It was too late to change it. That's why I have FSM does not work for negative number but designed for it. Adding a new state change whole table and equation also.**

*FSM (version 1.0 , before designing datapath) :



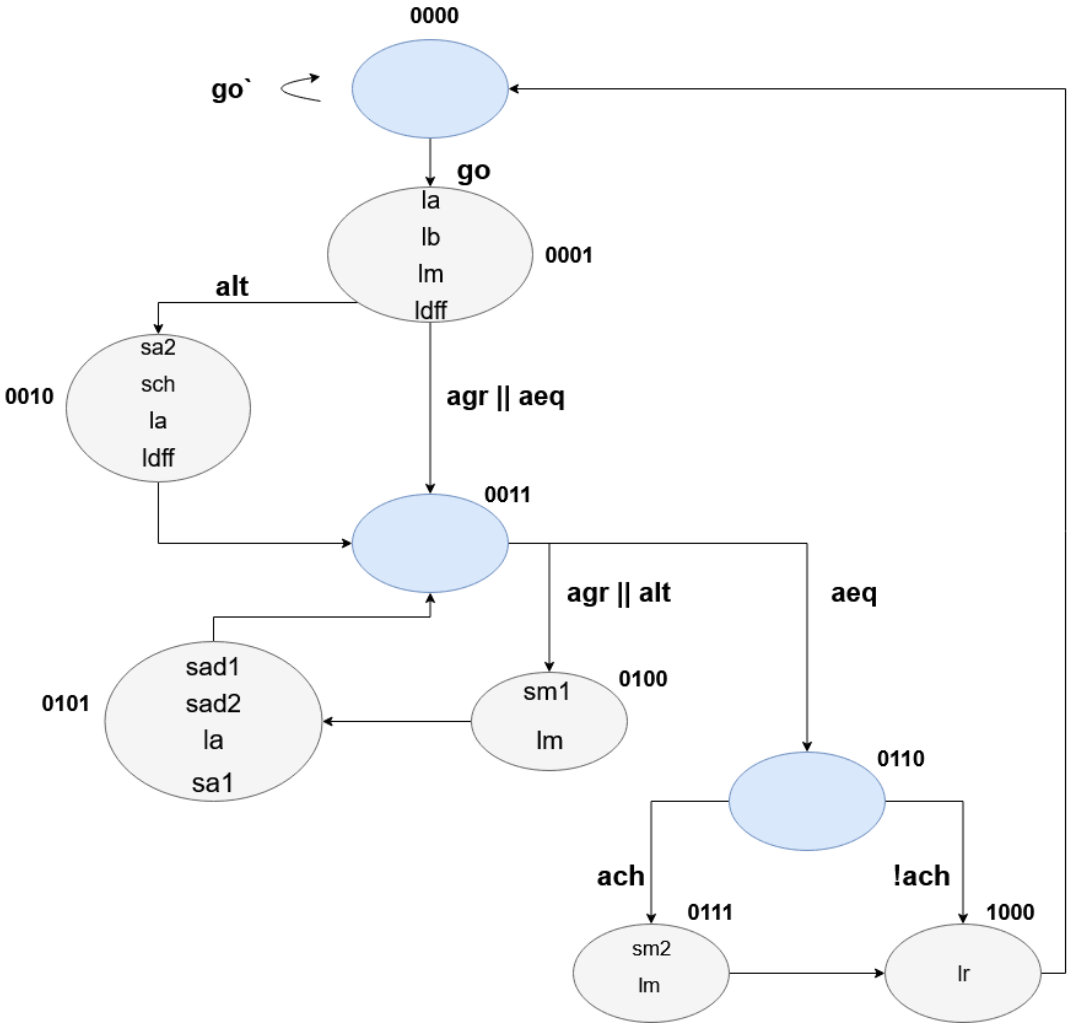
*Explanation:

In the first version of the FSM , there are nine different states. It maybe reduces to eight states, but I used an extra register for keeping the result. As you see , there are several states to handle the negative binary number but as I mentioned above , I have no chance to return and resolve the mistake that I also explained in the **Mistakes** section. As it's demonstrated , there are different variables to keep some data related to our program.

Data Explanation :

* **a** represents input that will be stored in datapath , **a_in** represents input that is taken from the user.
* **b** represents input that will be stored in datapath , **b_in** represents input that is taken from the user.
* **a_check** would be used for sign bit of given as an input which is **a.**
* **mult** is used for keeping calculation updated.
* **res** represents result of the calculation.

*FSM (version 2.0 , after designing datapath) :



*Explanation:

In the second version of FSM , datapath was designed and there were some input pins and output pins for controller. **I assumed that input pins , that I did not demonstrate on the state , has zero value for each state**. Every **single code** , that is in the state on FSM , has a unique meaning for controller. These are the input pins of the datapath and output pins of the controller. Similar meaning is valid for **transaction code**. Each transaction code is taken by the controller as an input to determine next state of process. They are also an outputs of datapath.

Input pins explanation of datapath :

* **la** enable bit of register of **a**.
* **lb** enable bit of register of **b**.
* **lm** enable bit of register of **mult**..
* **ldff** D-Flipflop enable bit..

* **sa1** unique multiplexer control bit of register **a**.
* **sa2** unique multiplexer control bit of register **b**.
* **sch** D-Flipflop multiplexer selection bit.
* **sm1** unique multiplexer selection bit of register **mult**.
* **sm2** unique multiplexer selection bit of register **mult**.
* **sad1** unique multiplexer selection bit for adder to control more than inputs in a different time.
* **sad2** unique multiplexer selection bit for adder to control more than inputs in a different time.
* **lr** result register enable bit.

Output pins explanation of datapath :

* **go** button that will controlled by the user.
* **alt** "being less than" bit of comparison of **a**.
* **aeq "**equality" bit of comparison of **a**.
* **agr** "being greater than" bit of comparison of **a**.
* **ach** output sign bit of D-Flipflop.

---

*Boolean Equation Tables :

*Controller Input Table :

Present State				Input					Next State			
P3	P2	P1	P0	alt	aeq	agr	ach	go	N3	N2	N1	N0
0	0	0	0	X	X	X	X	0	0	0	0	0
0	0	0	0	X	X	X	X	1	0	0	0	1
0	0	0	1	1	0	0	X	X	0	0	1	0
0	0	0	1	0	*	*	X	X	0	0	1	1
0	0	1	0	X	X	X	X	X	0	0	1	1
0	0	1	1	*	0	*	X	X	0	1	0	0
0	0	1	1	0	1	0	X	X	0	1	1	0
0	1	0	0	X	X	X	X	X	0	1	0	1
0	1	0	1	X	X	X	X	X	0	0	1	1
0	1	1	0	X	X	X	1	X	0	1	1	1
0	1	1	0	X	X	X	0	X	1	0	0	0
0	1	1	1	X	X	X	X	X	1	0	0	0
1	0	0	0	X	X	X	X	X	0	0	0	0

Meaning of 'X' : It means that we can put either put 0 or 1 as a value.
Meaning of '*' : It means it must be zero if there is another value which is 1 and marked with "*" at the same row. Otherwise it must be 1 if there is another value which is 0 and marked with "*" at the same row.

***These two sign reduce the burden of simplification process.**
***Equations and simplifications are demonstrated below.**

Simplification of next states :

N3 = P3` P2 P1 P0` (ach)` + P3` P2 P1 P0
= P3` P2 P1 ( P0`(ach)` + P0)

N2 = P3` P2` P1 P0(alt + agr)(aeq)` + P3` P2` P1 P0 ( alt )`(agr)`(aeq) + P3` P2 P1` P0` + P3` P2 P1 P0`(ach)
= P3` P2` P1 P0 ((alt + agr) (aeq)` + (alt)`(agr)`(aeq) ) + P3` P2 P0`( P1` + P1(ach))

N1 = P3` P2` P1` P0 (alt) (aeq)`(agr)` + P3` P2` P1` P0`(alt)`(aeq + agr) + P3` P2` P1 P0` + P3` P2` P1 P0(alt)`(aeq)(agr)` + P3` P2 P1` P0 + P3` P2 P1 P0` (ach)
= P3` P2` P1` P0 ((alt)( aeq)`(agr)` + (alt)`(aeq + agr)) + P3` P2` P1(P0` + P0(alt)`(aeq)(agr)`) + P3`P2(P1`P0+P1P0`+ (ach))

N0 = P3` P2` P1` P0` (go) + P3` P2` P1` P0(alt)`(aeq + agr) + P3`P2`P1 P0` + P3` P2 P1` P0` + P3` P2 P1` P0 + P3` P2 P1 P0`(ach)
= P3` P2` P1`( P0`(go) + P0(alt)`(aeq +agr)) + P3`P0`( P1 XOR P2 ) + P3`P2(P1`P0 + P1P0`(ach))

*Controller Output Table :

Present State				Output											
P3	P2	P1	P0	Ia	sa1	sa2	Ib	Im	sm1	sm2	sad1	sad2	sch	Idff	Ir
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	1	1	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Simplification of output :

Ia = P3` P2` P1` P0 + P3` P2` P1 P0` + P3` P2 P1` + P3` P2 P1` P0
= P3` P2` (P1 XOR P0) + P3` P2 P1` P0

sa1 = P3` P2 P1` P0
sa2 = P3` P2` P1 P0`
Ib = P3` P2` P1` P0

Im = P3` P2` P1` P0 + P3` P2 P1` P0` + P3` P2 P1
= P3` P1` (P0 XOR P2) + P3` P2 P1 P0

sm1 = P3` P2 P1` P0
sm2 = P3` P2 P1 P0

sad1 = P3` P2 P1` P0
sad2 = P3` P2 P1` P0
sch = P3` P2` P1 P0`

Idff = P3` P2` P1` P0 + P3` P2` P1 P0`
= P3` P2` (P0 XOR P1)
Ir = P3 P2` P1` P0`

***All simplification done by hand without using any tool.**

*Running Command and Result :

There some constraints about the program. These are listed below.

***Overflow** : There is no overflow handling , it works for 16 bits binary numbers.

***Negative Numbers** : Although I designed circuit also work for negative numbers. But that does not work because of the mistake that I make in designing FSM process. I explain why this has happened on the Mistakes section of this document.