Digital Circuits Theory - Laboratory						
Academic year	Laboratory exercises on	Mode of studies	Field of studies	Supervisor	Group	Section
2020/2021	Wednesday 15:30 – 17:00	SSI	Informatics	KP	1	3

Report from Exercise No 14

Performed on: 04.11.2020

Exercise Topic: Microprogrammable circuits

Performed by:

Dawid Grobert

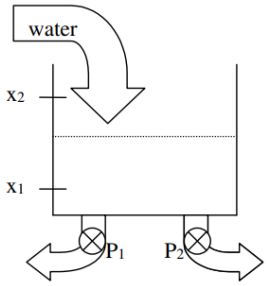
Purpose of the exercises

Microprogrammable circuits allow you to shorten the time of circuit creation as well as reduce the complexity of the circuit, since it is used programmable memory instead of hard wired solution. Therefore, to fully understand the advantages and disadvantages of this solution, the exercises are to familiarize us with more precise knowledge of the memory elements - both from the side of their operation and from the implementation itself and optimization of these implementations.

1 Description of the task

Task 1

Implement as a microprogrammable circuit a system controlling the task of emptying the water container with two pumps. The pumps P_1 and P_2 should be switched on alternately (only one pump can work at a time) when water exceeds the level of the sensor x_2 (i.e. when $x_2 = 1$). Working pump should be switched off when the water lever is below the sensor x_1 (i.e. when $x_1 = 0$). Assume that water level grows when pumps are off, and that it decreases when any pump is working.



For the microprogrammable circuit obtain:

a) Universal Structure

b) Conditional Multiplexer-based Structure

1.1 Universal Structure

1.1.1 Primitive flow map

The task should start, of course, with transforming the verbal description of the program's work into its description using a primitive flow map. Stable states are of course marked with a circle. I also write out possible state reductions.

Present State	X1 X2: 00	01	11	10	P1	P2
1			(S_0)	S_1	1	0
2	S_2			(S_1)	1	0
3	(S_2)			S_3	0	0
4			S_4	(S_3)	0	0
5			(S_4)	S_5	0	1
6	S_6			(S_5)	0	1
7	(S_6)			S_7	0	0
8			S_0	(S_7)	0	0

Next state

Reduction:

$S_0 - S_1$

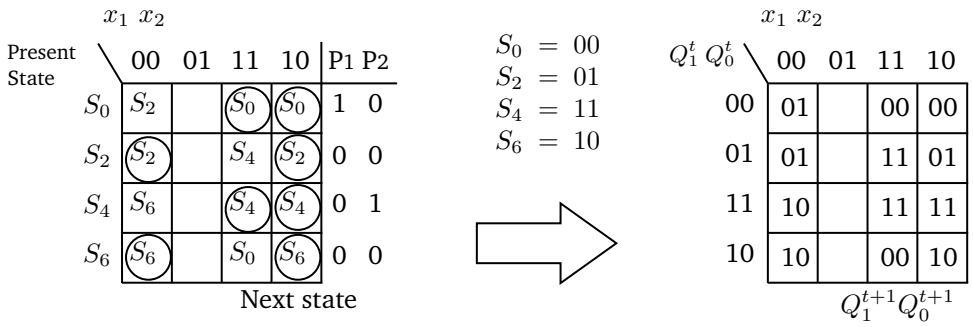
$S_2 - S_3$

$S_4 - S_5$

$S_6 - S_7$

1.1.2 Obtaining the Karnaugh map

I transform the reduced table into a Karnaugh map encoding individual states.



1.1.3 Designing the memory content of the microprogramme

Now, with the help of the obtained Karnaugh map, I transfer its contents to the memory of the micro program.

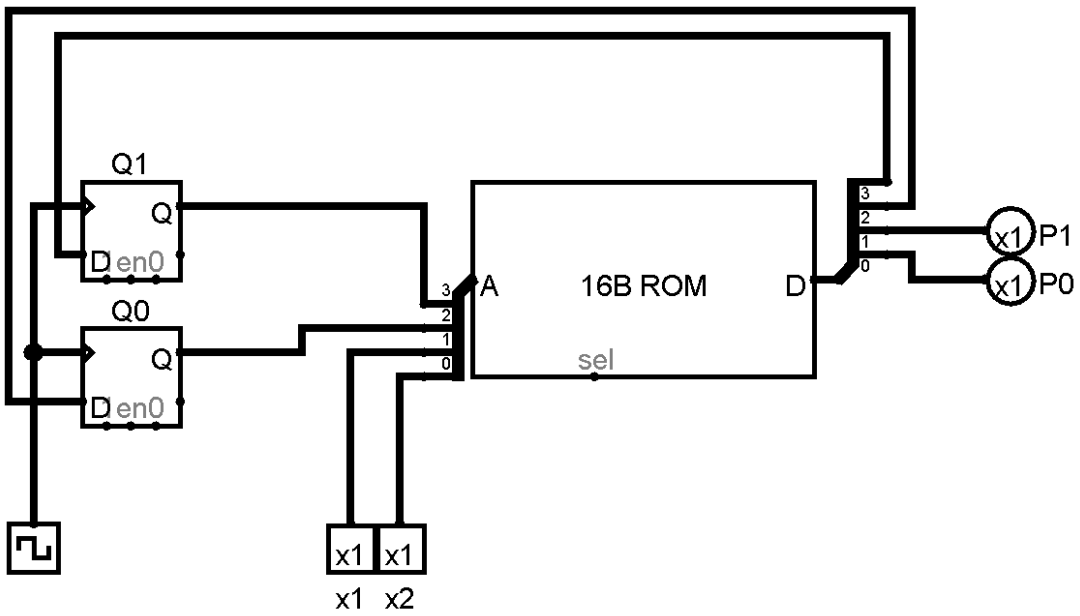
				next state		output	
Q_1^t	Q_0^t	x_1	x_2	Q_1^{t+1}	Q_0^{t+1}	P_1	P_0
A_3	A_2	A_1	A_0	A_3	A_2	A_1	A_0
0	0	0	0	0	1	1	0
0	0	0	1	-	-	-	-
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	-	-	-	-
0	1	1	0	0	1	0	0
0	1	1	1	1	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	-	-	-	-
1	0	1	0	1	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	1	0	0	1
1	1	0	1	-	-	-	-
1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1

Figure 1: Microprogramme memory content

At this stage we can already see that we will need 16-bit memory. Of course, there is such a possibility to optimize it, but this will be dealt with in the section **1.2 Conditional Multiplexer-based Structure**.

1.1.4 Circuit diagram

With the obtained data it is time to start creating the circuit. The ROM has been filled with the values given above (see Fig. 1). In order to create this circuit, 16-bit ROM was needed.



In the Logisim Hex Editor the memory content is written as:

```
6022 404c 8080 90dd
```

1.1.5 Example of how it works

An example of how this digital systems works is moved to the appendix (page 8).

This space has been intentionally left blank.

1.2 Conditional Multiplexer-based Structure

This task starts almost where we ended up before. In order to solve this task, I will use the content of already created microprogramme memory from the previous part of the task (see Fig. 1).

1.2.1 Optimization of microprogam memory.

						next state		output	
	Q_1^t	Q_0^t	x_1	x_2		Q_1^{t+1}	Q_0^{t+1}	P_1	P_0
	A_3	A_2	A_1	A_0		Y_3	Y_2	Y_1	Y_0
S_0	0	0	0	0		0	1	1	0
	0	0	0	1		-	-	-	-
	0	0	1	0		0	0	1	0
	0	0	1	1		0	0	1	0
S_1	0	1	0	0		0	1	0	0
	0	1	0	1		-	-	-	-
	0	1	1	0		0	1	0	0
	0	1	1	1		1	1	0	0
S_2	1	0	0	0		1	0	0	0
	1	0	0	1		-	-	-	-
	1	0	1	0		1	0	0	0
	1	0	1	1		0	0	0	0
S_3	1	1	0	0		1	0	0	1
	1	1	0	1		-	-	-	-
	1	1	1	0		1	1	0	1
	1	1	1	1		1	1	0	1

Figure 2: Microprogramme memory content before optimization

The states were marked and assigned in green. Outputs of individual states were marked in red. Orange indicates variables depending on specific states Y_3, and Y_2. As always one variable is fully dependent on one of the two possible settings of Y_3, and Y_2 (using don't care as the value of the second state) we can shorten the memory content using an internal state variable. In other words – in each state the change of state depends on only one variable (see Fig. 3).

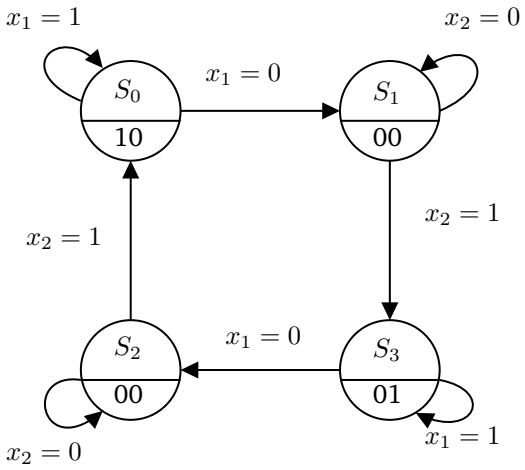


Figure 3: State diagram showing the dependence of states on individual variables

Using the above mentioned information, we are preparing new memory content for the micro-programme.

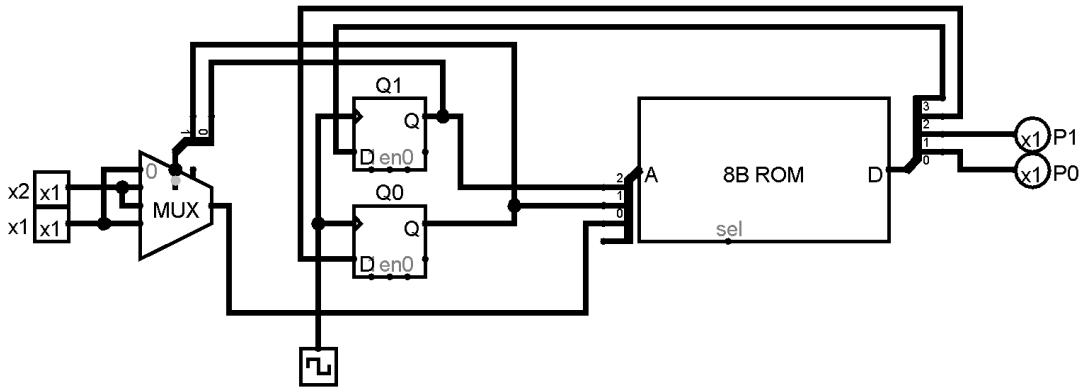
			next state		output	
Q_1^t	Q_0^t	w^t	Q_1^{t+1}	Q_0^{t+1}	P_1	P_0
A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0 (x_1)	0	1	1	0
0	0	1	0	0	1	0
0	1	0 (x_2)	0	1	0	0
0	1	1	1	1	0	0
1	0	0 (x_2)	1	0	0	0
1	0	1	0	0	0	0
1	1	0 (x_1)	1	0	0	1
1	1	1	1	1	0	1

Figure 4: New, optimised micro program memory content

We will use Multiplexer to work as an internal state variable. It is also worth noting that the memory has decreased by half and we can now use 8-bit memory.

1.2.2 Circuit diagram

It could be said that our circuit does not differ much, although reducing memory by half can be considered a concrete change.



In the Logisim Hex Editor the memory content is written as:

```
624c 809d
```

1.2.3 Example of how it works

An example of how this digital systems works is moved to the appendix (page 10).

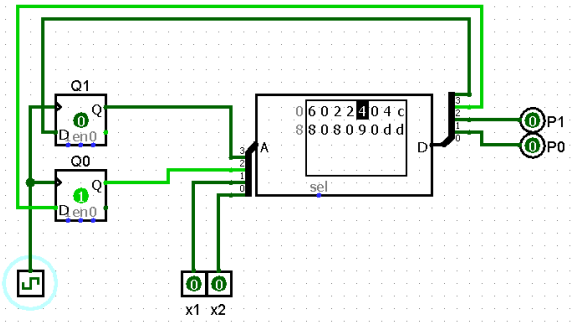
2 Conclusions

I have definitely learned to look more closely at the created content of microprogrammable memory. At first it seemed to me that it is impossible to shorten the memory, because for some reason I treated "don't cares" as an unchangeable part of memory. I forgot that I can write anything there that will allow me to shorten the memory – which was done later.

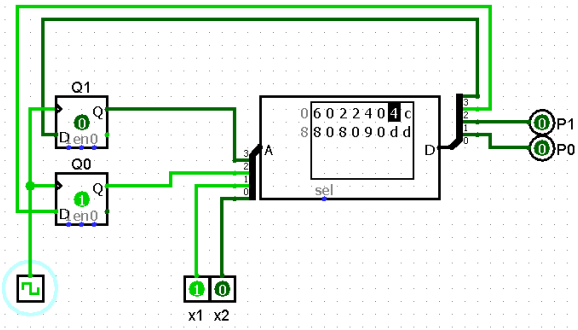
Appendix
An example of how the digital circuits works

1 Universal structure

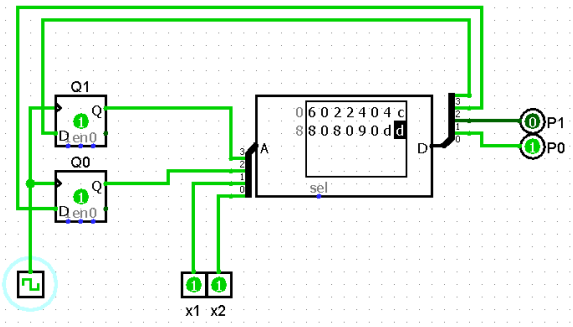
An example of how a digital system built in the form of Universal Structure (page 2) works.



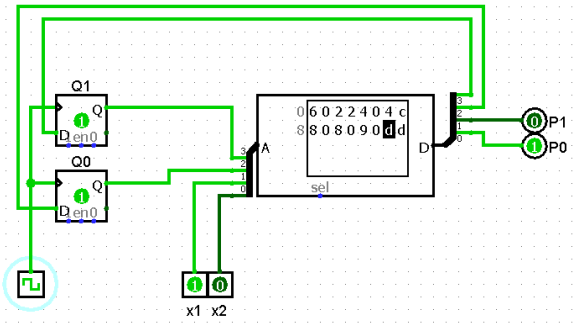
Step 1



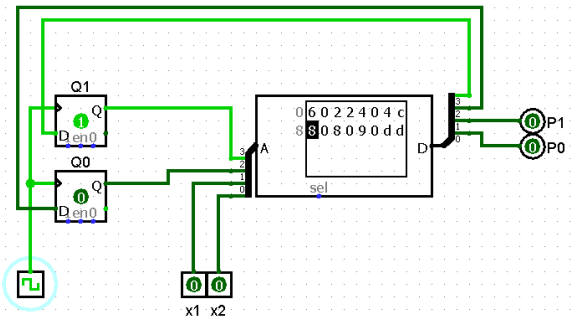
Step 2



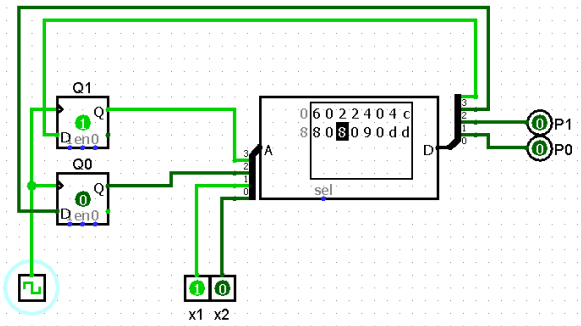
Step 3



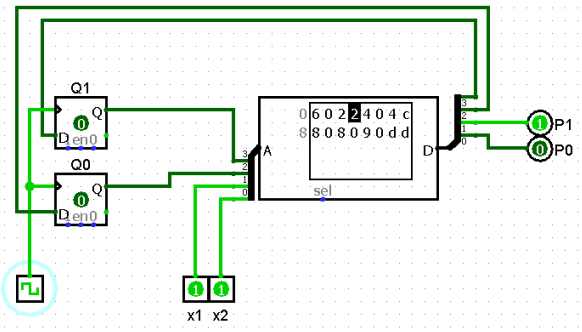
Step 4



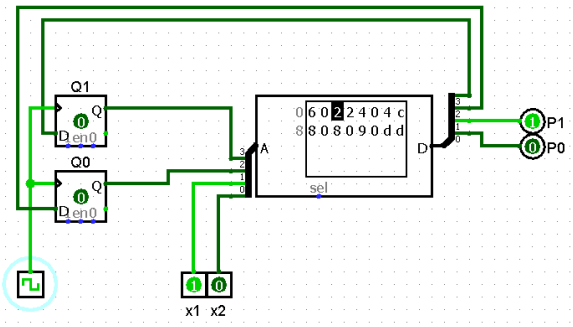
Step 5



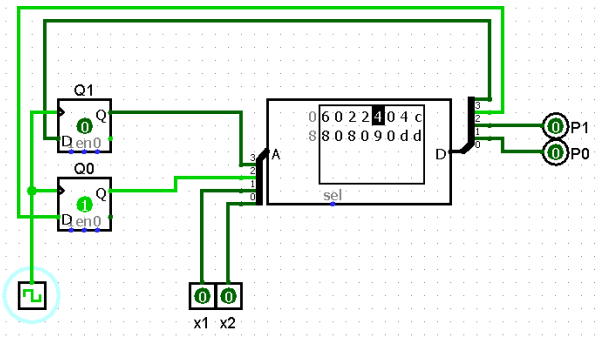
Step 6



Step 7



Step 8

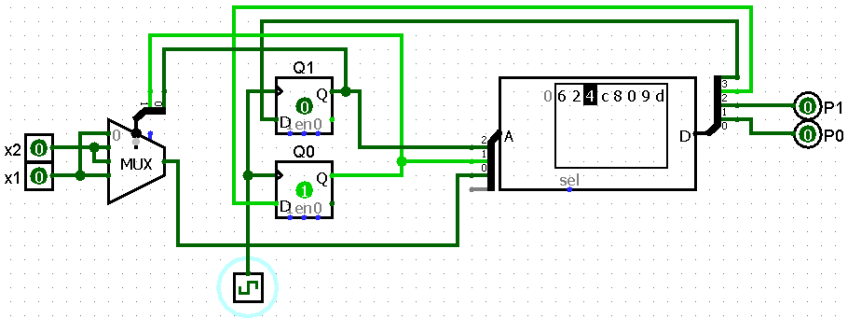


Step 9

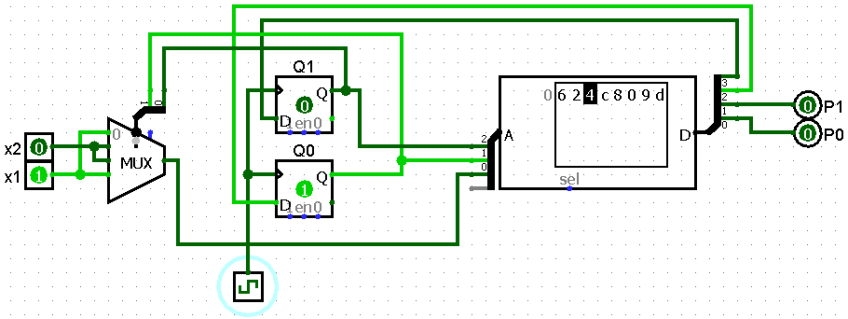
This space has been intentionally left blank.

2 Conditional Multiplexer-based Structure

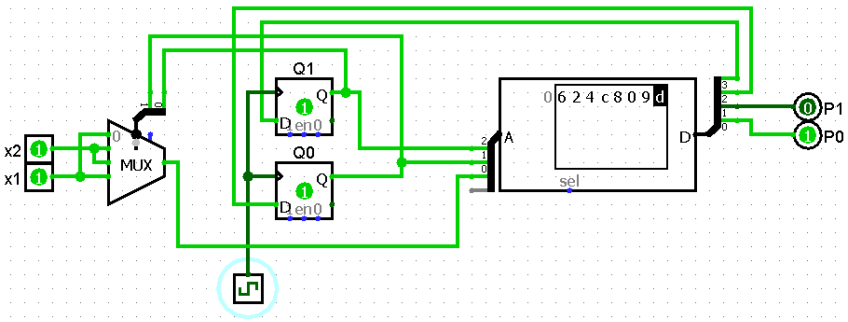
An example of how a digital system built in the form of Conditional Multiplexer-based Structure (page 5) works.



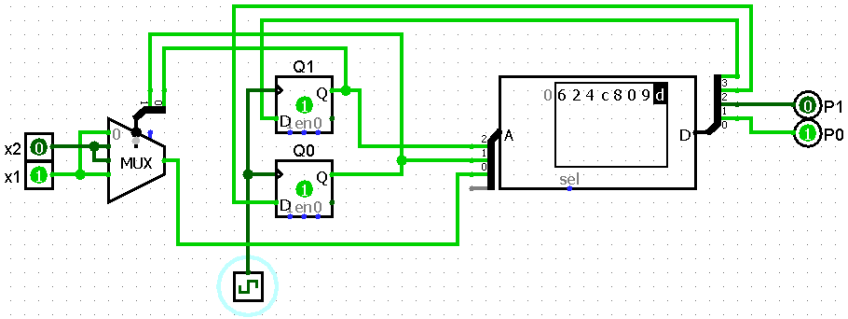
Step 1



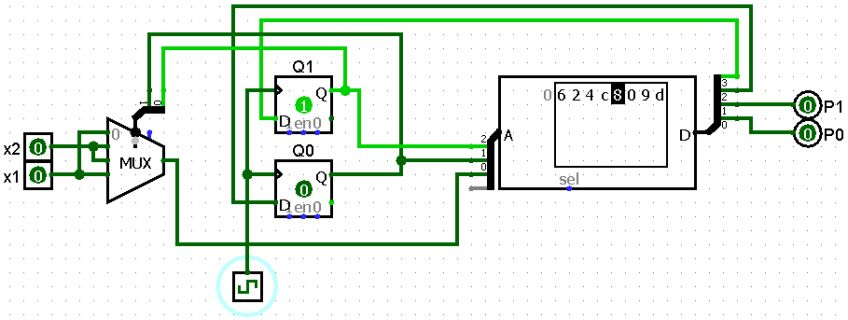
Step 2



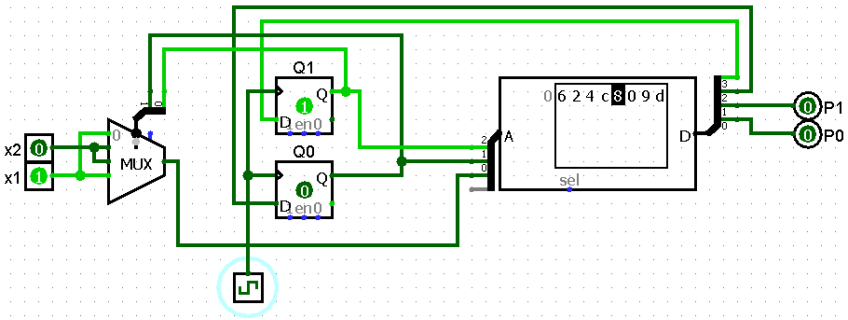
Step 3



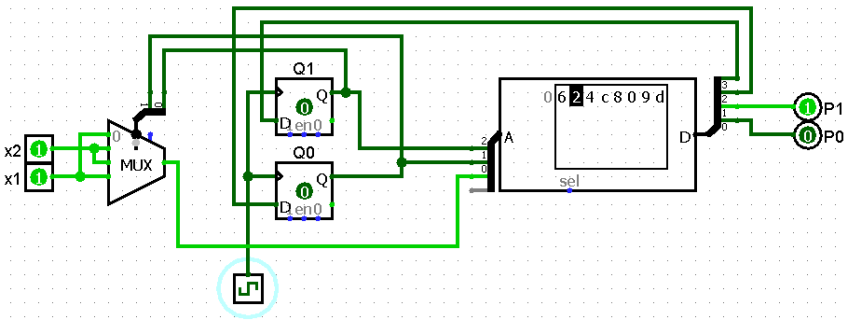
Step 4



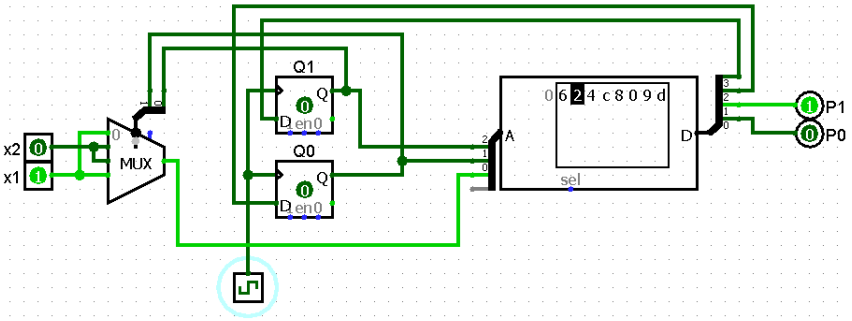
Step 5



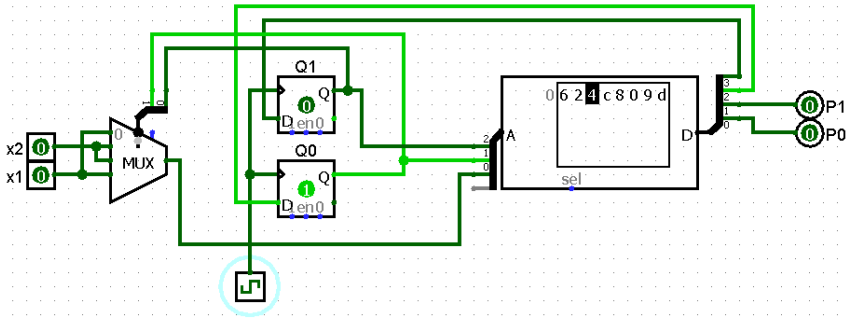
Step 6



Step 7


Q1
Q
D en
Q0
Q
D en
x2
x1
MUX
0 6 2 4 c 8 0 9 d
A
sel
D
P1
P0

Step 8


Q1
Q
D en
Q0
Q
D en
x2
x1
MUX
0 6 2 4 c 8 0 9 d
A
sel
D
P1
P0

Step 9