Bachelor's Thesis

Engineering in Industrial Technologies

Design of a column decoder for a binary vision sensor implementing run length encoding

Author: Bruno Giménez Umbert

Directed: Dr. Davide Brunelli

Date: December 2014

Università degli Studi di Trento

Universitat Politècnica de Catalunya - Escola Tècnica Superior d'Enginyeria Industrial de Barcelona

Abstract

This work describes the architecture of an asynchronous column decoder for a CMOS binary vision sensor. The decoder compresses and code data, at row-level, before delivering off-chip. The operation is completely asynchronous featuring 160 MB/s data rate.

Acknowledgements

I would like to remember those who have participated in my Bachelor's Thesis. Without their contribution, this project would have never existed.

I appreciate the guidance of Dr. Davide Brunelli, my tutor, for his recommendations about the project and because has given me all the support I needed.

I would like to thank Dr. Massimo Gottardi. His advices, ideas and corrections have been essentials for developing this work. It has been very helpful the fact of disposing his experience and knowledge about the topic.

On the other hand, I would also like to thank Fondazione Bruno Kessler members for providing me the access to the Cadence software for the characterization and simulation of the design.

1. Contents

1.1. List of figures and tables

2. Introduction

A vision sensor is an imager which integrates some image processing directly on-chip. Typically, a vision sensor is targeted to a specific class of application for which it exhibits better performance compared with a standard solution (standard camera + processor). A vision sensor means having a fast parallel image processing, low-power consumption, high-dynamic range and also possibility to implement processing very close to the sensor. However, not every type of image processing can be embedded with the sensor and it is not so beneficial against standard approach.

Many sensor architectures have been proposed in the past exploiting different sensing and processing solutions [1]-[6].

In this work, we refer to a vision sensor which extracts the spatial contrast and binarizes it at pixel-level before to deliver the information [1].

Our target is to study and define how to code such as binary information to be dispatched off-chip in order to minimize the data bandwidth and the power consumption.

Figure 01 shows an example of binary image delivered by a 64x128 pixel vision sensor after extracting and binarizing the spatial contrast:

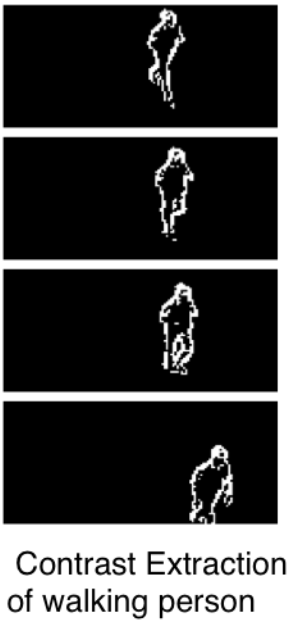

Figure 01 – Sequence of four binary images generated by the vision sensor extracting the spatial contrast in a binary form

There are different ways to read this image with a contrast sensor by sending signals.

2.1. Different techniques

Raster-scan:

There are different ways of delivering a binary image. The most common and straight-forward is a raster-scan technique where pixel values are delivered sequentially, starting from the first pixel of the first row and ending with the last pixel of the last row of the array. This means that, considering this binary image, the values of the following pixels are:

0	0	1	1	1	1	1	0	0	0
0	0	0	0	1	1	1	1	0	0
0	1	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	1	1
0	0	0	1	1	1	1	0	0	0
0	1	1	1	1	0	1	1	0	0
0	0	0	1	1	1	1	0	0	0
0	1	1	1	0	0	1	1	1	1
0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	1	1	1	1	0

Figure 02 – Example of a binary image

As it can be seen, each pixel represents one bit of information (0 or 1). Thus, considering a format of n x m pixels, the total amount of data delivered by the sensor to the processor is n times m.

In this example, the total amount is 10x10 = 100 bits.

Addres-Event Representation:

Another technique for binary data delivering is the so called 'Address-Event Representation' (AER) [2]. Here the pixel readout is completely asynchronous. Each pixel detecting an event sends a request to the arbiter after which it delivers its coordinates (x, y) inside the array.

The basic idea of an asynchronous vision sensor is that the output is in the form of address-events (AEs, encoding the x,y-address of the pixel in the array) that are generated locally by

the pixels. Pixels individually quantize the analog vision signal, usually after local gain control and spatial-temporal redundancy reduction. The output is thus in the form of an address-event representation (AER).

Address-Event Representation method is convenient when the number of pixel detecting an event is low compared to the total number of pixels of the array. This means that there are not so many pixels representing the contrast on the image.

AER consists on deliver the address of each pixel in contrast. This address represents the position of the pixel in the image as the horizontal (x) and vertical (y) coordinates with a binary code. This method is not sequential.

For example, considering an image with a format like 256x256 pixels and C contrast pixels, the total amount of data delivered is 16xC bits (8 bits for x address, 8 bits for y address and C bits for contrast value).

In the example of *Figure 02*, the pixels required are those which have a 1 as a bit (a total of 47). Thus the total amount of data delivered would be 8x47 = 376 bits.

Therefore, is convenient to use AER in case of having an image with not so many pixels in contrast.

Run length encoding:

Another run length encoding technique is the same as the AER but, instead of sending the address with the x and y coordinates it only sends the position in the row (x), row by row. The pulse (or bit) representing the position of the row is not considered now because it means only the data delivered by the sensor.

For example, considering an image with a format like 256x256 pixels and C contrast pixels, the total amount of data delivered is 8xC bits (8 bits for x address and C bits for contrast value).

In the example of *Figure 02*, the total amount of data delivered would be 4x47 = 188 bits.

Therefore, is convenient to use this method in case of having an image with not so many pixels in contrast as well.

2.2. Technique presented in this project

The presented technique adopts run-length data coding to dispatch binary images. The reason for this depends on the fact that contrast images contain very sparse information, which means that delivering data for each pixel can be useless because if these clusters of logical values are so long, it is possible to avoid considering each pixel and obtain the same result, which could mean less time and less power consumption. This is known as a run-length case [7].

Considering that there is an input pattern made of ones (1) and zeros (0) for each row of the binary image which describes the contrast in it (as in *Figure 02*), the idea is to create a reader for this pattern that delivers some signals taking into account that if there is a cluster of consecutives ones or zeros together, it would be better not sending signals for each pixel, but for those which represent a change of logical value.

This means that with this network are delivered the addresses (only the position in the row) of those pixels before a change of value. It is not considered the pixel of the "End of Row" because I mean about the data delivered from the sensor to the processor in order to reduce the power consumption.

The following figure shows what pixels (in red color) are the requested in this case:

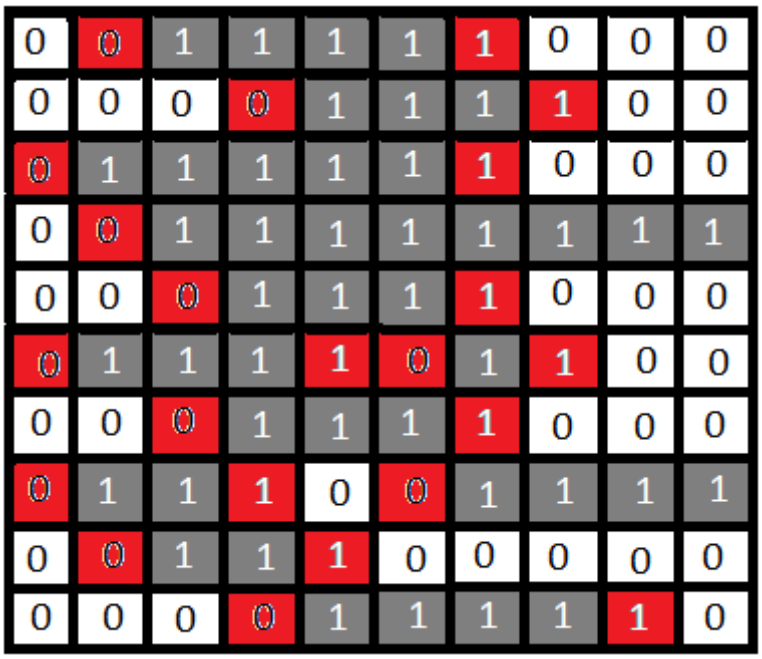

Figure 03 - Pixels (in red color) requested

With the example of the image with a format like 256x256 pixels and C contrast pixels, the total amount of data delivered depends on the distribution of the pixels in contrast.

In the example of *Figure 03*, the number of pixels requested is 22, thus the total amount of data delivered would be 4x22 = 88 bits. Therefore, is convenient to use this method in case of having runs of 0 or 1 in the rows of a contrast image.

The worst case scenario is when an image has each pixel with a different value than the pixel before, this means that all the addresses (except the lasts of the row) would be requested. With the example of the array of 256x256 pixels, in this case the total amount of data delivered would be 255x255x8 = 520200 bits.

2.3. Comparison among different coding techniques

In order to see which can be the limits of each technique, it is useful to make a comparison of them with two kinds of image formats: 256x256 and VGA.

2.3.1. 256x256 pixels format

In case of this format, the total amount of data delivered by the whole bit map would be 256x256 = 65536 bits.

This means that in case of using AER technique, the limit of the number of pixels in contrast would be 4096 in order to be convenient to use this method, which is the 6.25 % of the total of pixels. More than this percentage, it is better to use the first technique.

In case of using the run length encoding technique, the limit of the number of pixels in contrast would be 8192 in order to be convenient to use this method, which represents the 12.5 % of the total of pixels.

Of course, with the network designed in this project it would depend on the distribution of the pixels in contrast.

2.3.2. VGA format

In case of VGA format (640x480 pixels), the total amount of data delivered by the whole bit map would be 640x480 = 307200 bits.

This means that in case of using AER technique, the limit of the number of pixels in contrast would be 16168 in order to be convenient to use this method, which is the 5.26 % of the total of pixels.

In case of using the run length encoding technique, the limit of the number of pixels in contrast would be 34133 in order to be convenient to use this method, which represents the 11.11 % of the total of pixels.

Of course, with the network designed in this project it would depend on the distribution of the pixels in contrast.

2.4. Examples

The first example represents an array of 10x10 pixels with the following distribution of pixels in contrast (in dark color):

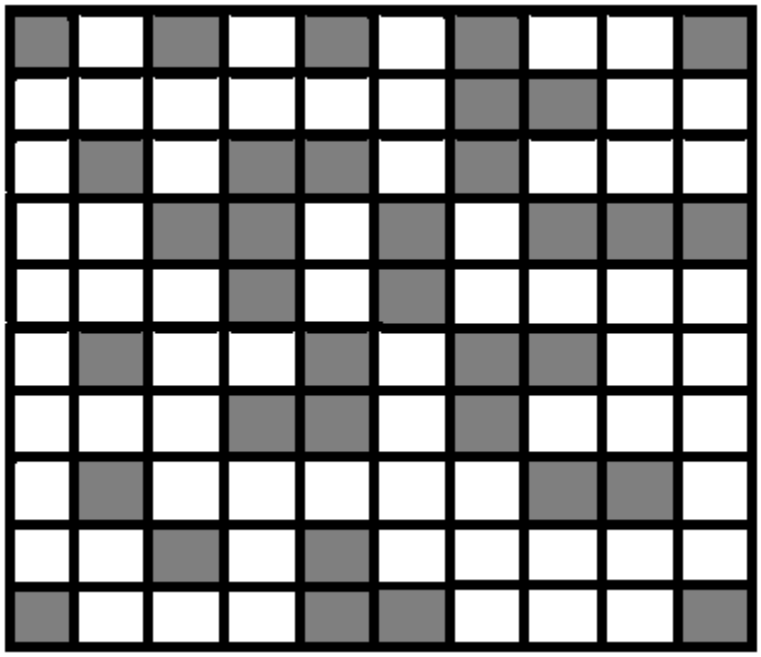

Figure 04 - Example 1

Considering the four techniques, the comparison is the following one:

Techniques	Total number of pixels requested	Total amount of data delivered (in bits)
Delivering the bit map	100	100
AER	35	280
Run length encoding	35	140
Technique presented	47	188

Table 1 - Comparison of the techniques

In this case, the technique presented is not convenient because there are no long groups of consecutives values in the rows.

The second example represents an array of 10x10 pixels with the following distribution of pixels in contrast (in dark color):

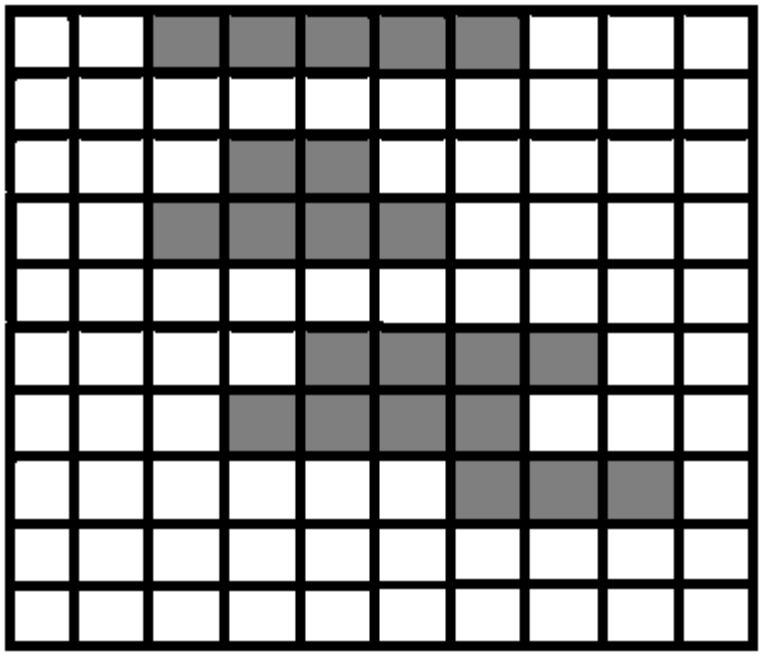

Figure 05 - Example 2

Considering the four techniques, the comparison is the following one:

Techniques	Total number of pixels requested	Total amount of data delivered (in bits)
Delivering the bit map	100	100
AER	22	176
Run length encoding	22	88
Technique presented	12	48

Table 2 - Comparison of the techniques

In this case, the technique presented is the most convenient because there are long groups of consecutives values in the rows.

Therefore, the goal of this project is to design a network with Cadence software that delivers all the addresses of the pixels before a change of logical value in order to reduce the total amount of data delivered and in order to reduce the power consumption.

This network is convenient only in cases of having long groups of consecutives logical values in the rows of the contrast image.

3. First part of the network

3.1. Introduction

The first step is to design a schematic that can take advantage of the signal created by a start signal in order to reduce the time of the reading process and deliver only one signal for each row which can describe exactly the changes between 0 and 1. This signal is made of some pulses -each pulse represents one pixel- which are in high voltage in case of 1 and in low voltage in case of 0 at the input pattern.

If we consider the example in *Figure 02*, the first input pattern (which corresponds to the first row) is “0011111000”, thus the signal delivered should be:

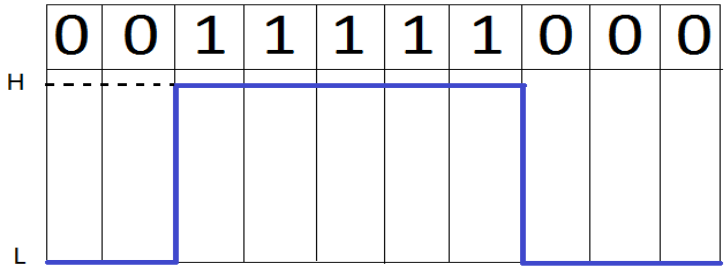

Figure 06 - Signal of a pattern

Therefore, the basic concept is to lower the voltage in case of a 0 or raise the voltage in case of a 1. If there are consecutives 0 or 1, the low or high voltage has to be held until the following change of the value. The way to do this is that each pixel has to deliver an output signal to a common line shared with all the pixels of the row.

3.2. Schematic

In order to have a circuit to do that, cascaded XOR logical gates are used, which are able to detect a disparity between 0 and 1. To hold the value at the common line in case of consecutive zeros or ones, the option chosen is to use a buffer tristate.

The idea is to enable the buffer just the time to give the line the value of the pixel and then pass the control (enable) to the next buffer. This means that we have the same number of buffers than pixels and those are consecutively giving values to the common line (only once each pixel).

In order to enable the buffers some pulses of voltage are used, thus the enable input of each buffer will be active only for a short period of time.

3.2.1. Creation of the pulses

To create pulses (change of voltage value during a while) is necessary a carry-in signal which changes its value of voltage through a rising edge, for example. For this reason, the input pattern and the carry-in signal are combined in a XOR gate.

The change of value of the carry-in affects all the values on the cascaded XOR, so all of these values change too. Therefore, to take profit of these changes of values the way is to create consecutive pulses at the output of the cascaded XOR with a delay on each end of the gate.

The way to create this kind of pulses is the following one:

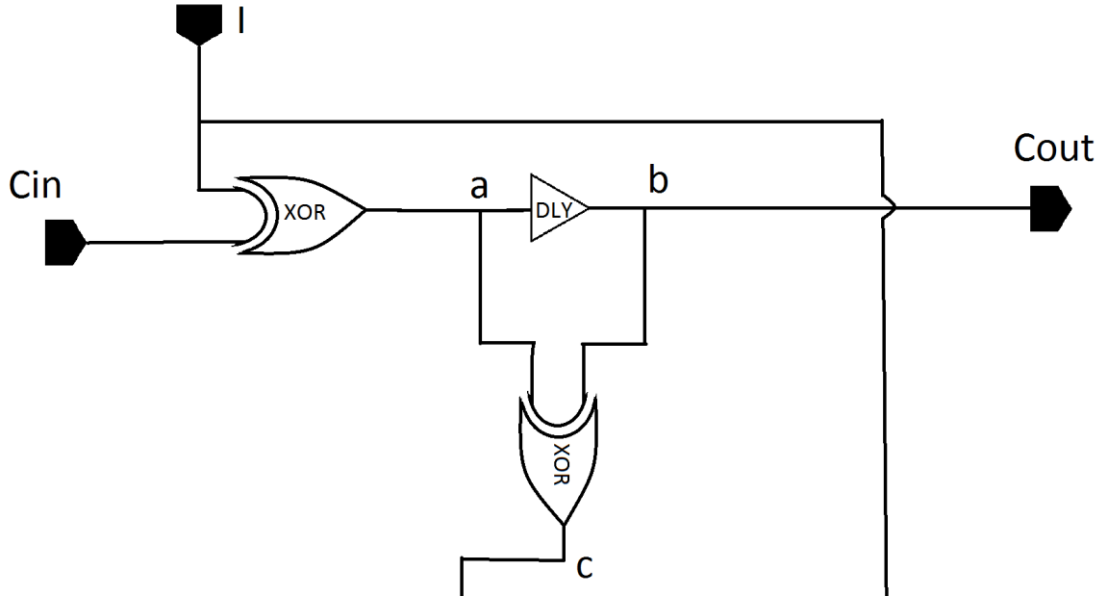

Figure 07 - Circuit for create pulses

When the value of the carry-in signal (Cin) changes from low (L) to high (H), it produces a change of value in “a” and in “b”. The value of “b” is the same as the value of “a” but shifted a short period of time: this delay (a few ns) is what creates the pulse. During this period of time, “a” has a different value than “b” and for this reason the XOR gate makes a pulse in “c” with the width of the short while. Then, the signal on “b” is the one that corresponds to the carry-out signal (Cout) which is going to be the Cin of the next cell (considering that there is one cell per pixel) and the pulse created is the time while the buffer is enabled.

The following figure shows the creation of a pulse with these signals (considering that I = L):

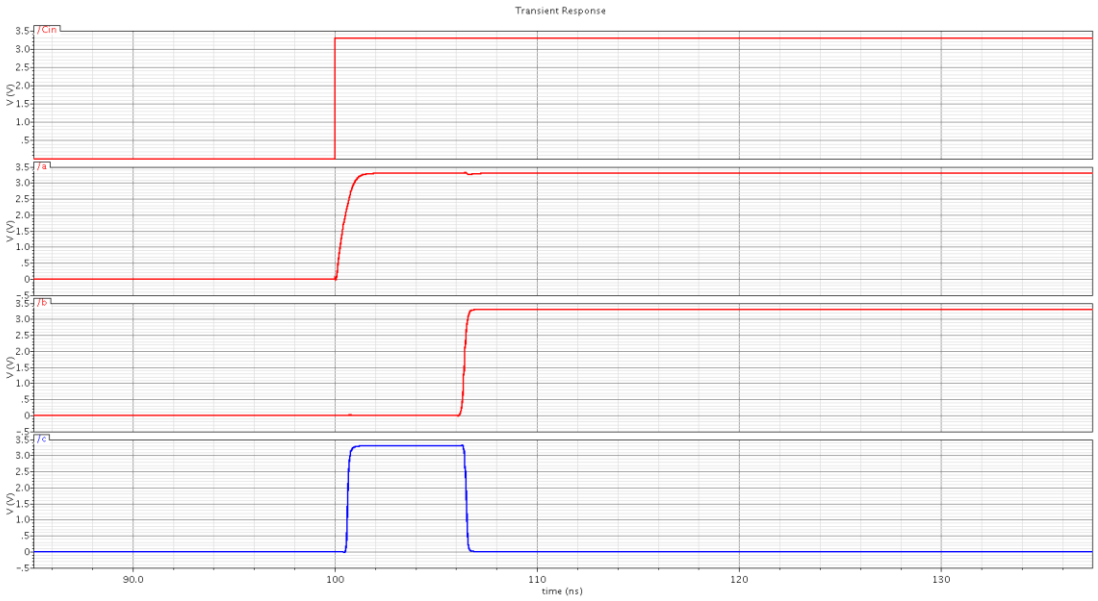

Figure 08 – Timing diagram of a pulse generation

In Figure 08 the first signal corresponds to the Cin, which represents a step. When this change of value occurs, “a” changes its value too and after a delay period of time, “b” has its rising edge. Therefore, the signal in “c” is a pulse with a width of this period of time.

3.2.2. Using buffers

With this procedure the first pixel has a pulse and, when the width of the pulse finishes (the time while the buffer of the pixel is enabled), the following cell creates another pulse with the same width and thus the following buffer takes the control of the common line. Therefore, the time of the process is really fast.

During the time while the pulse of a pixel is high, the buffer has to write the value at the line. In order to give the value of the pattern to the buffer, it is easy to see that is possible to provide the same value that is delivered by the pixel through the input of this buffer.

The following figure shows an example of the procedure of this network:

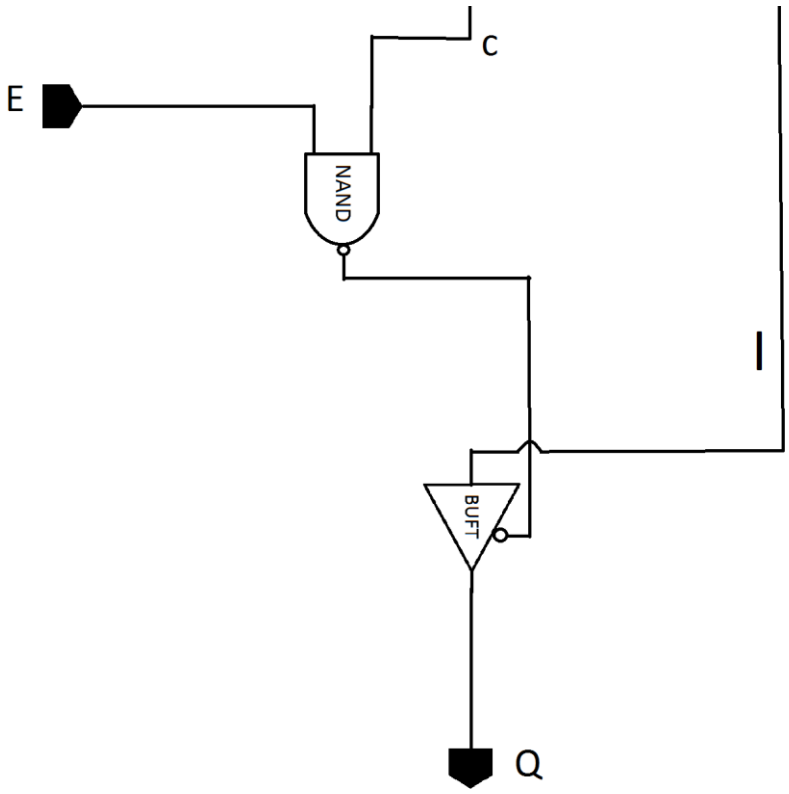

Figure 09 – Schematic that shows how to enable the buffer

The signal on “c” corresponds to the pulse that enables the driver and the value (high or low) from the pattern (I) is written through the input of the buffer to the common line (Q).

It is easy to see that the pulse in “c” is not going directly to the enable input of the driver, but is one of the inputs of a NAND gate. The other input (E) is the same signal as the Cin of the first pixel at the very begging of the process has. This is because a pulse in “c” is created every time the first Cin (or E) changes its value: from low to high (rising edge) and from high to low (falling edge). This means that when E goes from high to low (falling edge), the same pulse is created and then we have two times the same output signal in each period of E, which is a waste of power and it has no sense to use the same signal so many times.

The following figure shows that two pulses are created per period, but also shows how the buffer writes the value to the common line during for the while one time per period (considering that I = H):

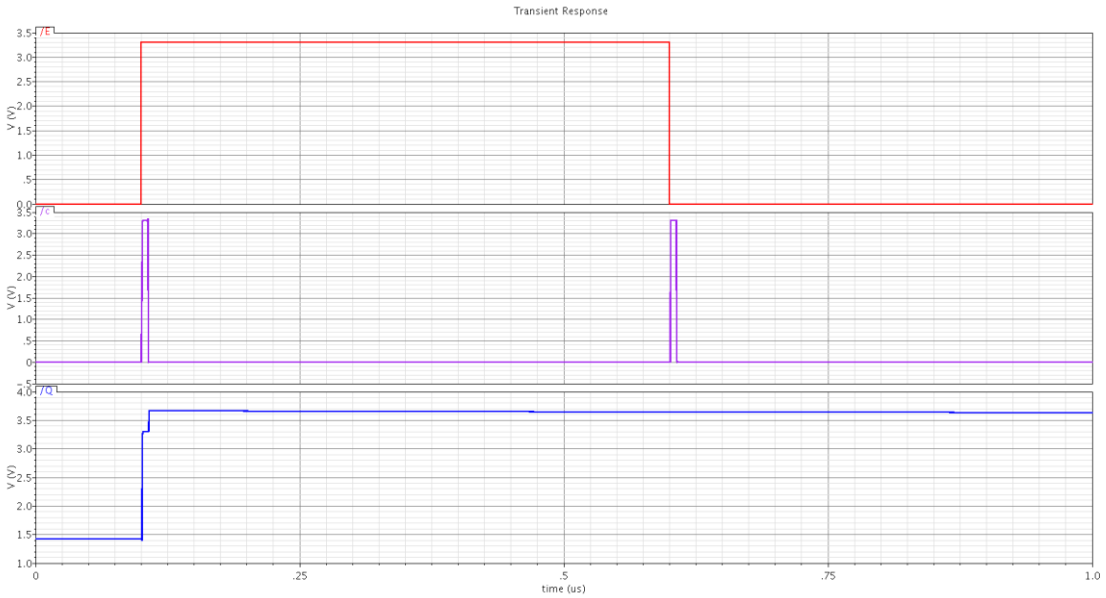

Figure 10 - Signal on c (pulses generated at each detected edge) and on Q. Simulation result of 16 pixels with 10 consecutive ones generating two edges (dark to bright and bright to dark)

As we can see, the high voltage is written at the line and held until the end, because there is no more than one input in this example and because the network doesn't use the second pulse in a period. The following figure shows the complete schematic for one pixel:

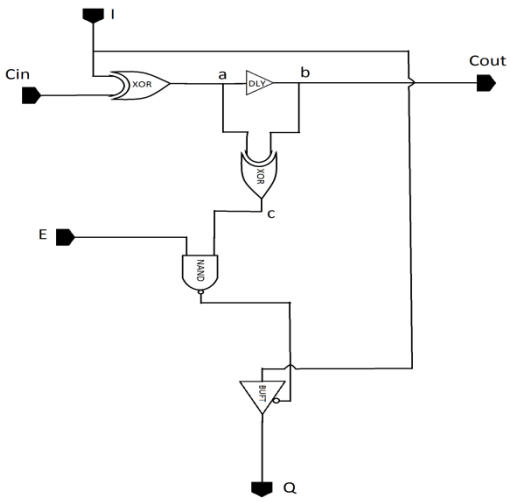

Figure 11 - Schematic for one pixel

3.3. Creation of the blocks

Once the schematic for each pixel is done, the next step is to make an array of 256 drivers (a good number for this kind of technology) and simulate it.

The block that corresponds to the last schematic shown in an array of 256 pixels is the following one:

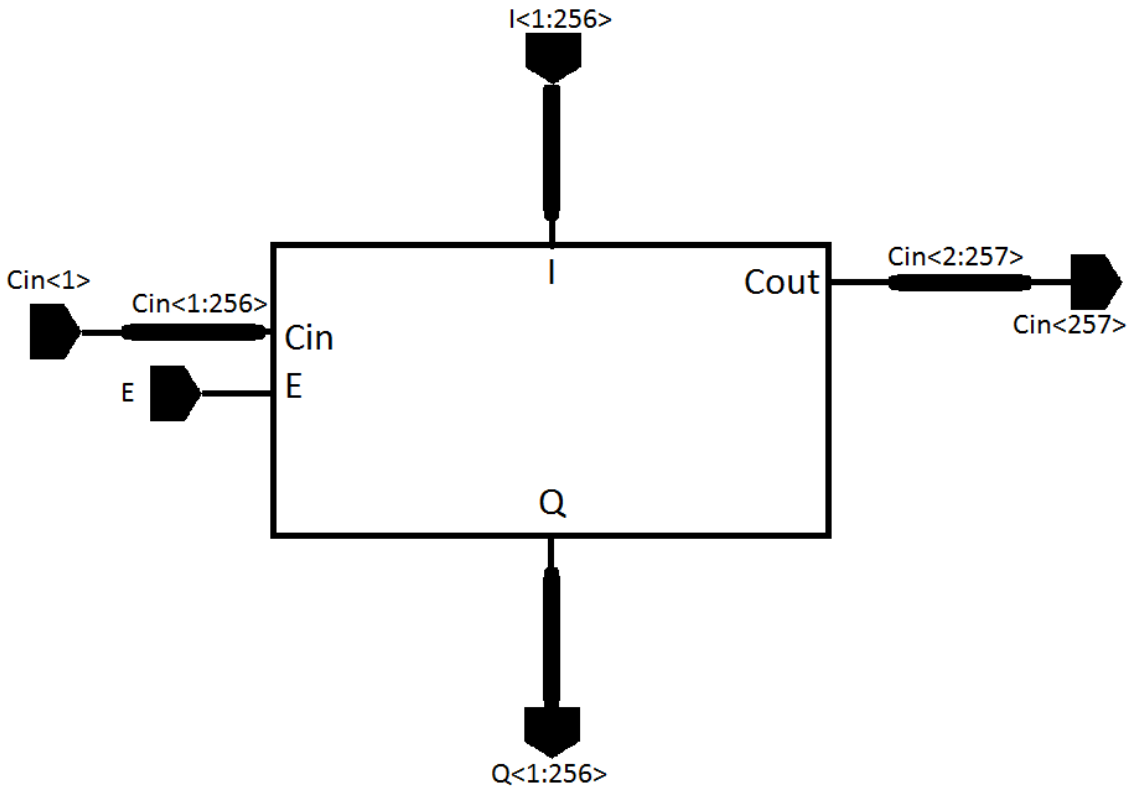

Figure 12 - First block created

This block has 5 different pins. The one which corresponds to the input pattern called I (in each case is a 1 or a 0), another input for the carry-in signal (note that is only for the first Cin, because the others are the Cout of the previous cells), the third input is the E signal (the same as the first Cin) and two outputs: the carry-out signal Cout (note that is only for the last Cout which has no utility, because the others are the Cin signals from the second to the last one) and the output signal which feeds the common line (Q).

After having the array, it is also necessary to create the block for the simulation.

The following figure shows this block with the transistor, buffer and capacitor needed for simulating it:

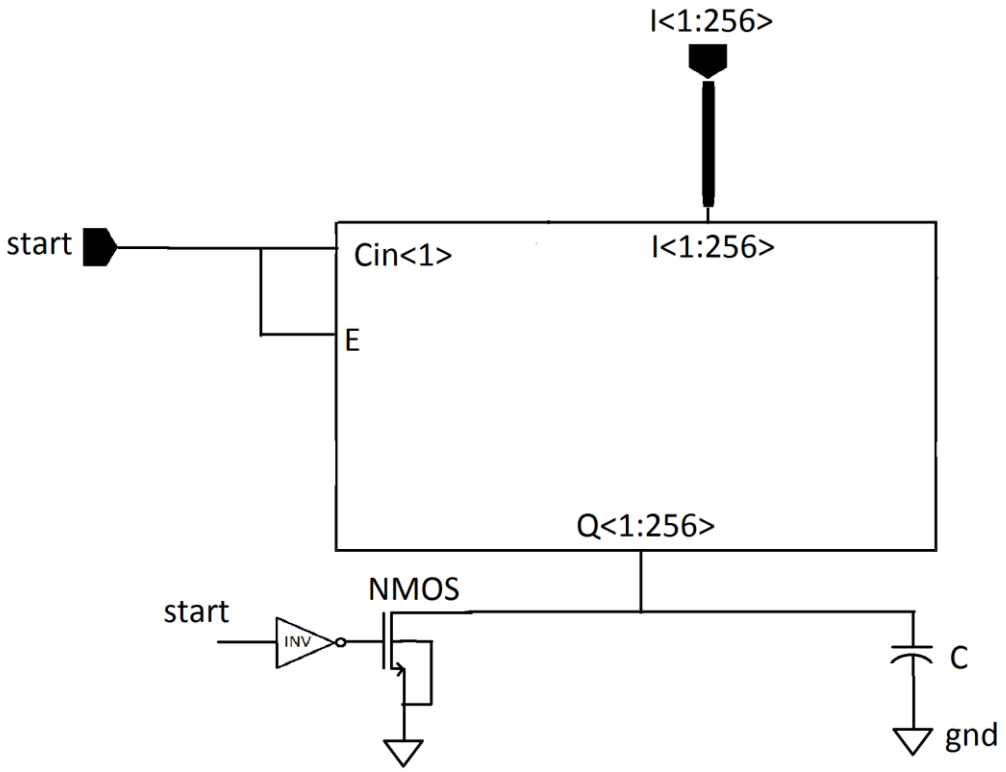

Figure 13 – Last block ready for the followings simulations of the chapter

As it can be seen, the block has the input pattern of the 256 pixels in a row, the input which corresponds to the called “start” signal which is the same for the first Cin and for E (the rest of Cin are driven inside the block) and the output for the common line.

The inputs have the symbol of a pin because are the ones that I have to provide a signal for the simulation.

The common line has a transistor NMOS and a capacitor connected together and to the ground (gnd). The transistor is because at the very beginning of the simulation I have to set the line to the low level in order to have clearer plotted results. It exists an option for the simulation in Cadence that allows setting the line to a logical value with the initial conditions, but when the start signal starts another period the line has the last value it was written. Therefore, with the transistor the line is always set to zero at the beginning of a new period of time.

The following figure shows a simulation with the same signal replicated two times and with the line set to zero (considering an input pattern as 1100011110):

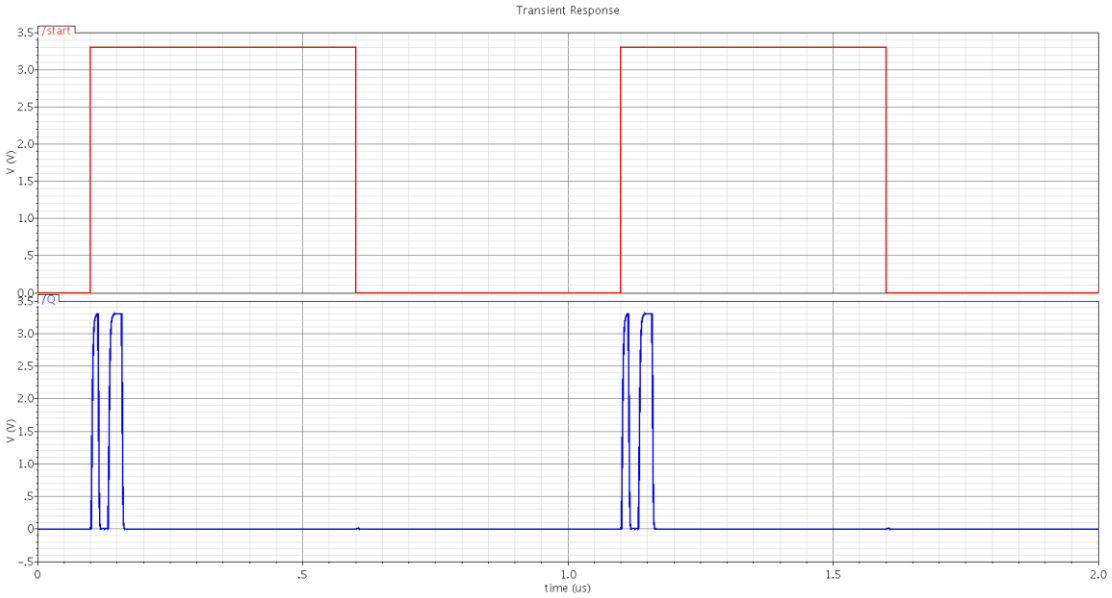

Figure 14 – This simulation represents the signal written on the output line and set to L in each start of period

As we can see, for each period of the start signal there are two pulses (the second one has the double of width than the first one) and only one signal is written, with the line set to the low level at the beginning of the period.

The capacitor is in order to hold better the signal at the common line. The value of this capacitance (2 pF) is chosen approximately at the moment, because the final network has not been designed yet at all. The value of the real capacitance is estimated and used at the appendix.

The final cascaded blocks can be represented as the following figure:

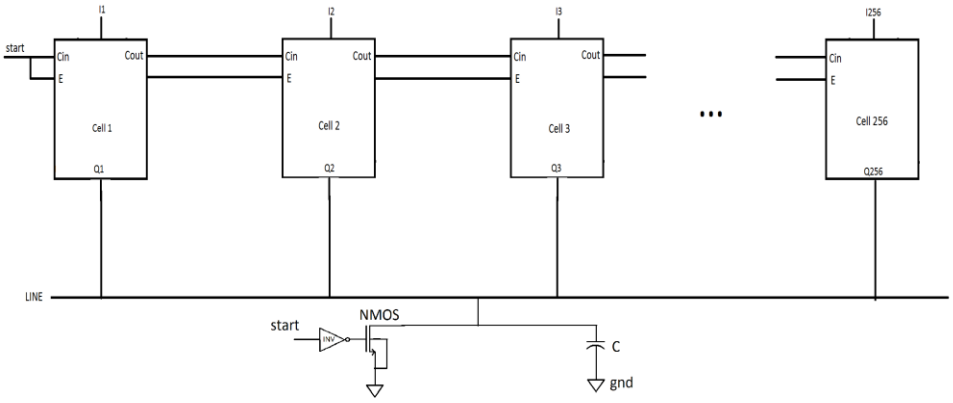

Figure 15 – Schematic representing 256 cascaded blocks which implement the column decoder. Inputs of the decoder are I1-I256. The decoder is triggered by START after which the coded output is delivered on 8bit LINE bus asynchronously

3.4. Simulations

In order to verify if the blocks and the schematic are well-designed, they have been simulated in different ways. The first simulations are about an array of 10 pixels with different input patterns just to see if it works as it is expected. A buffer has been included at the end of the line in order to have clearer waveforms.

Input pattern: 1100010101

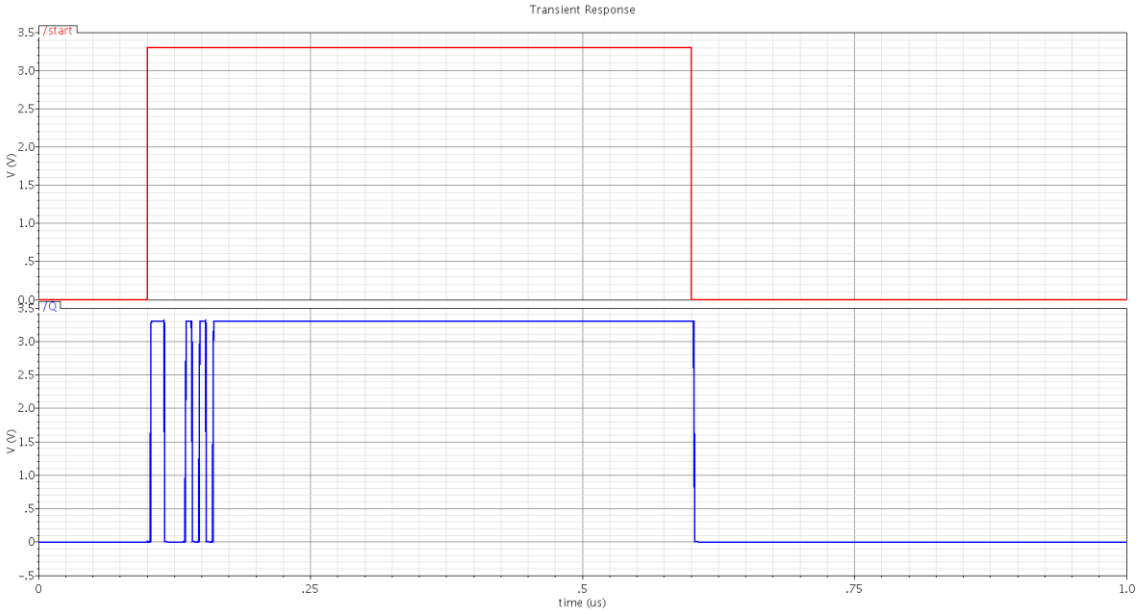

Figure 16 – Simulation results of the decoder loaded with a 10 pixel binary pattern (1100010101) and triggered by START. The Q signal is the expected one (the output of the block).

As we can see, the plotted result is the expected one. The high voltage is held until the end of the period because the last value at the pattern is 1. If the last value was a 0, the value held would be a 0. In order to check it, the following simulation represents this example.

Input pattern: 1100010100

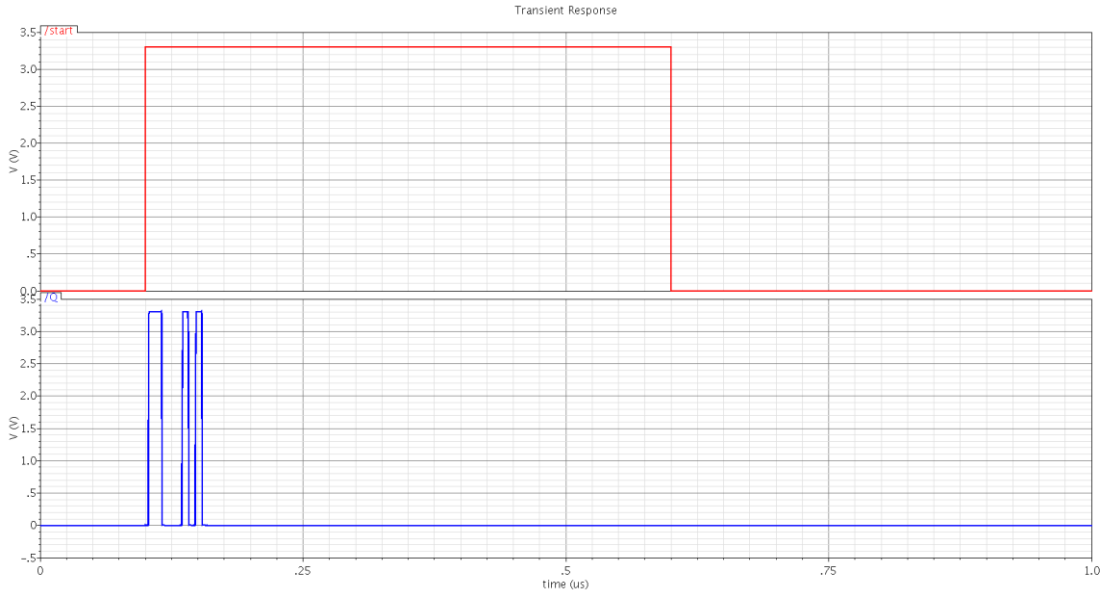

Figure 17 - Simulation results of the decoder loaded with a 10 pixel binary pattern (1100010100) and triggered by START. The Q signal is the expected one (the output of the block).

Now is it possible to see the held value as a 0 with a similar pattern as the last one.

Input pattern: 0111111110

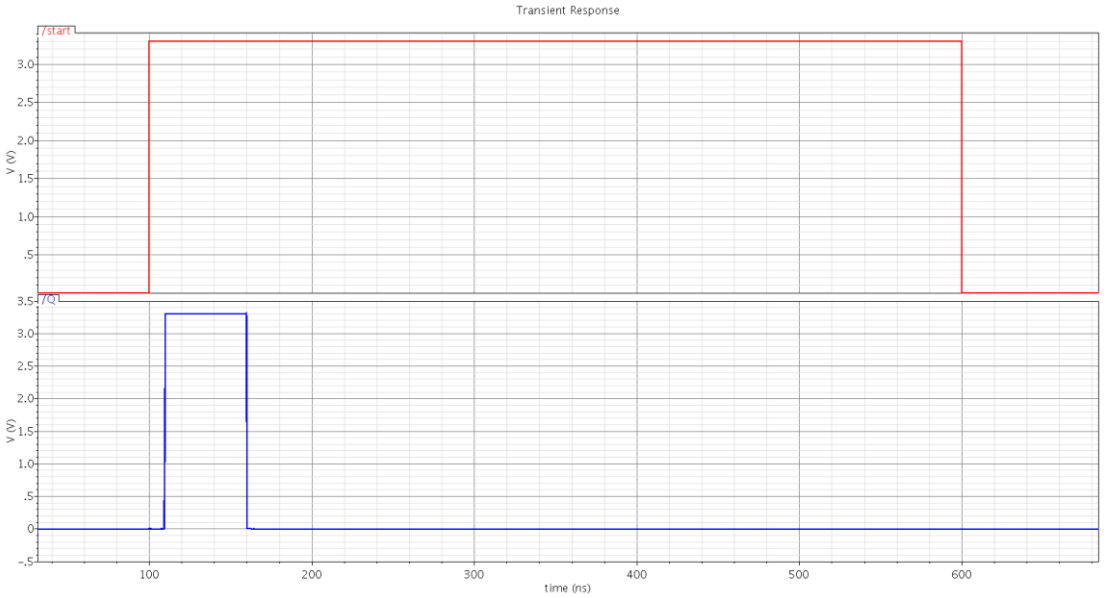

Figure 18 - Simulation results of the decoder loaded with a 10 pixel binary pattern (0111111110) and triggered by START. The Q signal is the expected one (the output of the block).

With this pattern is it possible to see only a pulse per period with a width of 8 pulses at the beginning of the network and the first value set to 0.

Input pattern: 0000110000

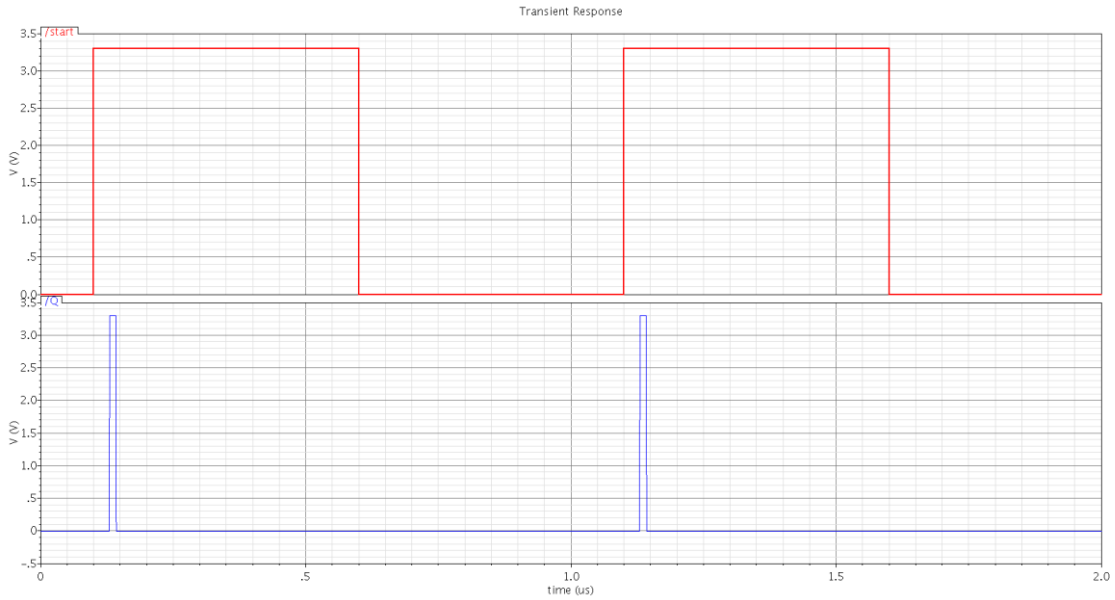

Figure 19 - Simulation results of the decoder loaded with a 10 pixel binary pattern (0000110000) and triggered by START. The Q signal is the expected one (the output of the block).

With this pattern is it possible to see only a pulse per period with a width of 2 pulses at the beginning of the network in two periods of the start signal.

Then an array of 100 pixels has been simulated with the input pattern with the same value (high) just to see if the value of the capacitance chosen is correct for the following simulations until the estimation of this value:

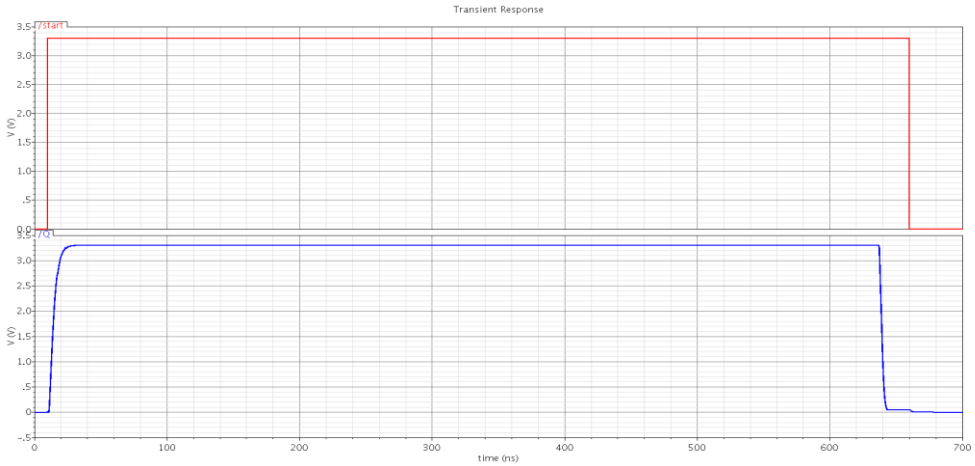

Figure 20 - Simulation results of the decoder loaded with a 100 pixel binary pattern (1111…) and triggered by START. The Q signal is the expected one (the output of the block).

As we can see, the value is well-held and the capacitor chosen is correct for the moment.

And finally, the last simulation is an array of 10 pixels with the following input pattern: 1010101010, which is the worst case scenario because it means that the maximum amount of data has to be sent. This is in order to know the frequency of the signal at the output line, which I have needed later:

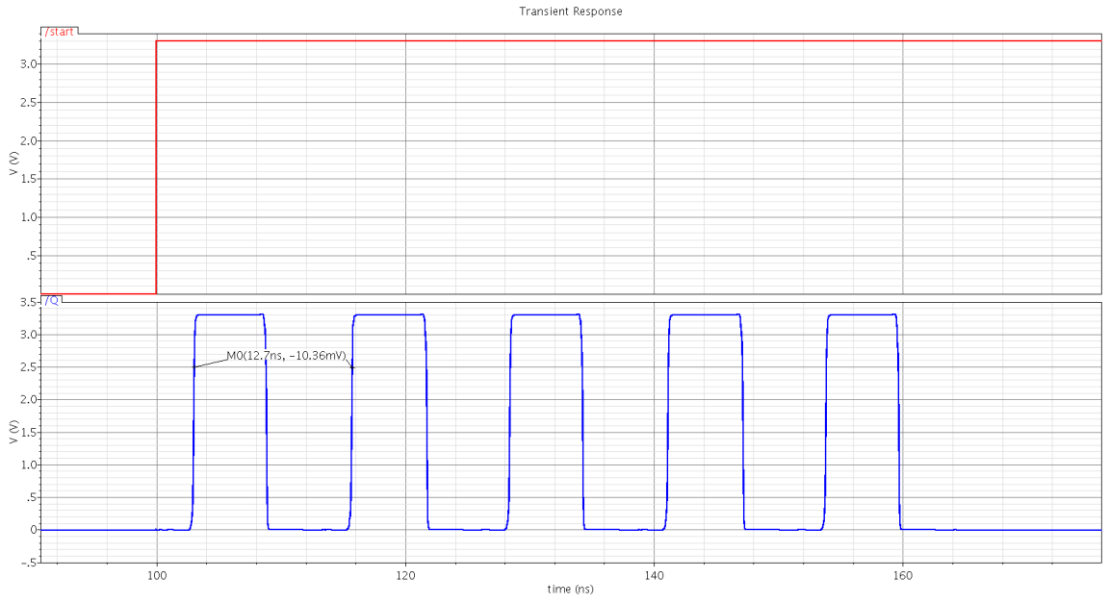

Figure 21 - Simulation results of the decoder loaded with a 10 pixel binary pattern (1010101010) and triggered by START. The Q signal is the expected one (the output of the block)

As we can see, the period of the signal is about 12.7 ns (a frequency about 79 MHz).

4. Description of the completed design

4.1. Introduction

After having the design of the network that will deliver the signal in each case, it is also necessary to have a kind of reference in order to know where we are at the moment of reading that signal. This means that it is necessary to know which pixels are those that are in high voltage and those which are in low voltage, because it is not useful to have a signal with the information of the binary contrast without knowing to which pixels correspond each part of the signal.

For example, in case of having an array of 10 pixels and the following signal:

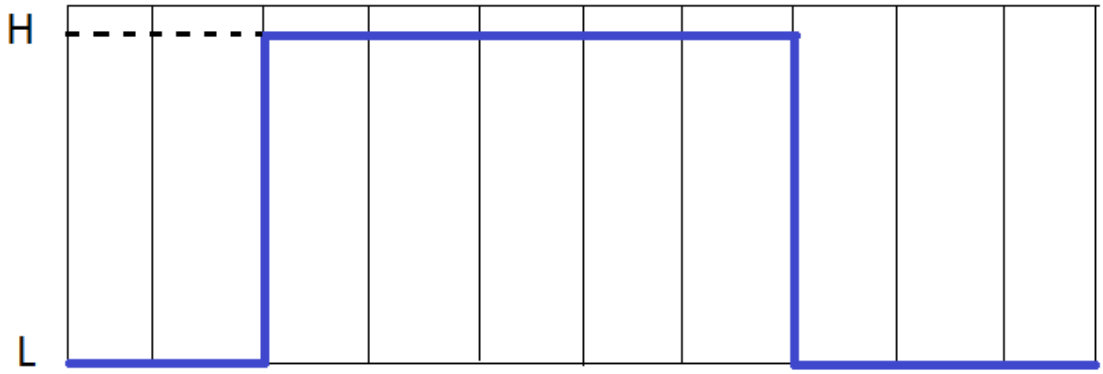

Figure 22 - Signal of an array of 10 pixels

It is possible to have an idea of which pixels are related to the low and to the high voltage, but it is not possible to know it exactly.

Therefore, in order to avoid delivering a lot of data, the new concept is to include a new block at the network designed that can allow us to know the address (the pixel in we are) only when a change between 0 and 1 or 1 and 0 occurs. This means that when a change is detected, at the same time -with a certain little delay- it should be possible to see in another line the address where this change has happened.

The idea is to deliver the address of the pixel before a change of logical value, supposing that the first value of the row will always be zero. Thus, if all the pixels before a change are known, it is not necessary to deliver all the addresses of each pixel -which could mean a waste of power- and the signal at the output can be recognized perfectly. The following figure represents the last example:

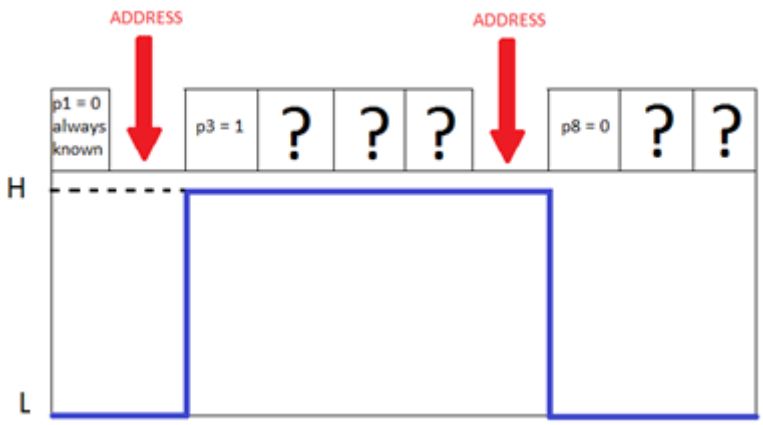

Figure 23 – Output signal delivering the pixel addresses at which an edge is detected by the decoder

In this example, the addresses delivered are the pixel number 2 and the pixel number 7. This means that (considering the first pixel always equal to 0) from the pixel number 1 to the pixel number 2 the binary signal is 0; from the pixel number 3 to the pixel number 7 is 1; and from the pixel number 8 to the end is 0 again.

The way to do it is to use a bus and deliver the address as a binary code. The bus should have 8 lines because it is an 8 bit-code if I want to use the array of 256 cells (the last address possible is 11111111 which correspond to the pixel number 255).

Thus, in this example the addresses sent are 00000010 (pixel number 2) and 00000111 (pixel number 7).

4.2. Second schematic

This new schematic doesn't modify the last one designed for delivering the output signal. In fact, the first network is used in order to complete this new one. The basic concept of delivering the output signal is the same as is used before.

As it can be seen, the main block has the same structure:

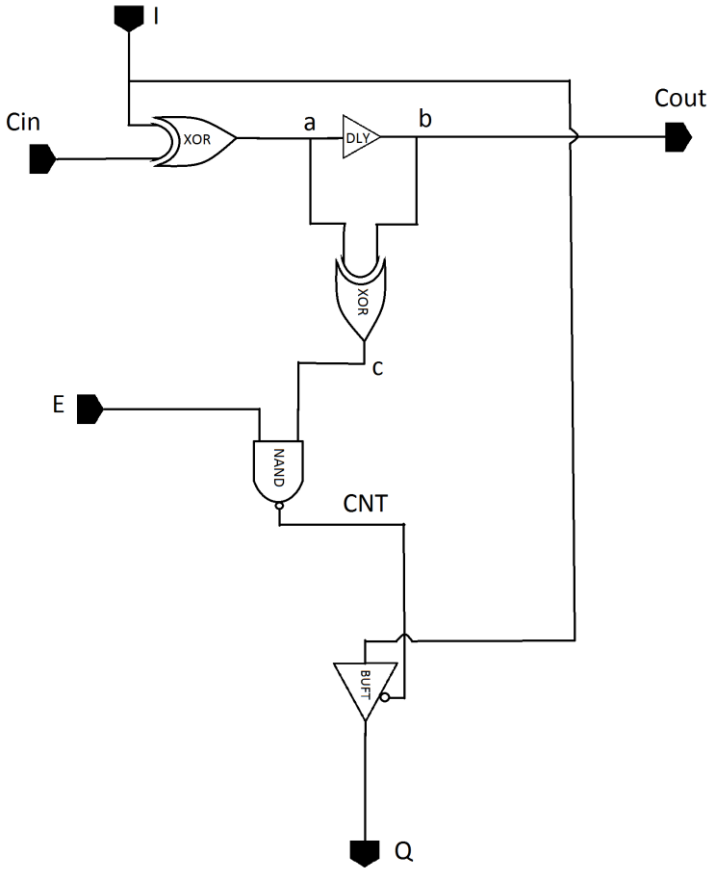

Figure 24 - First schematic

This schematic has a new point of consideration: CNT. This point represents the pulse which enables the buffer during a short period of time in order to be able to write the value to the output line. CNT is used for the enabling or disabling of the other buffers for the bus lines, as it can be seen on the following chapters.

The following figure shows the cascaded schematics which is used for writing the signal:

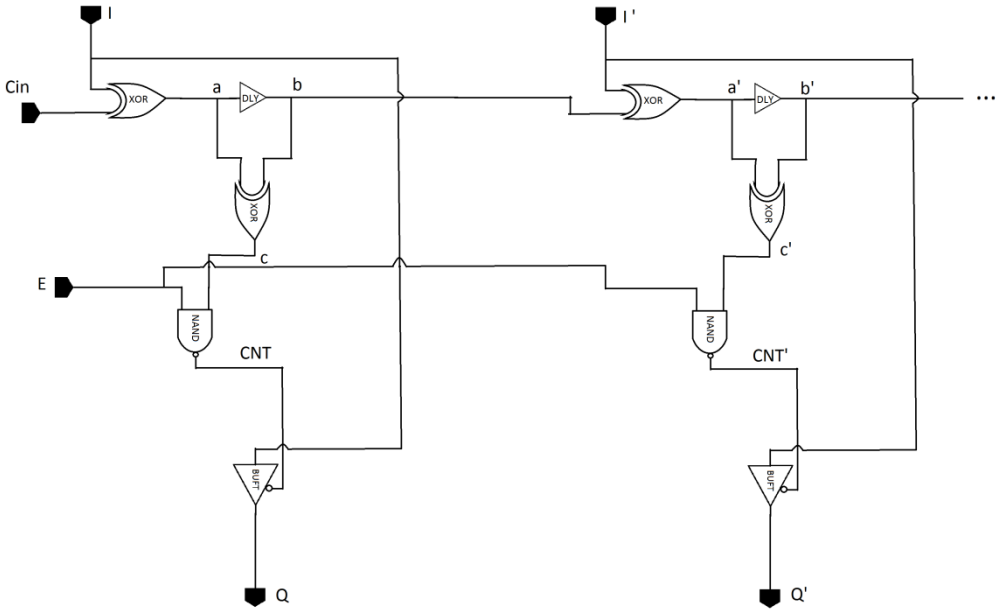

Figure 25 – Cascaded blocks

As we can see, the only carry-in signal which is inputted is the first one, because from the second cell there is a propagation of this signal. The other inputs are the E signal (the same as the first Cin) and the input pattern (I I' …). The output (Q Q' …) is the one which writes the signal to the line.

The following figure shows an example of how the signal is written by this network (considering I = H and I' = L):

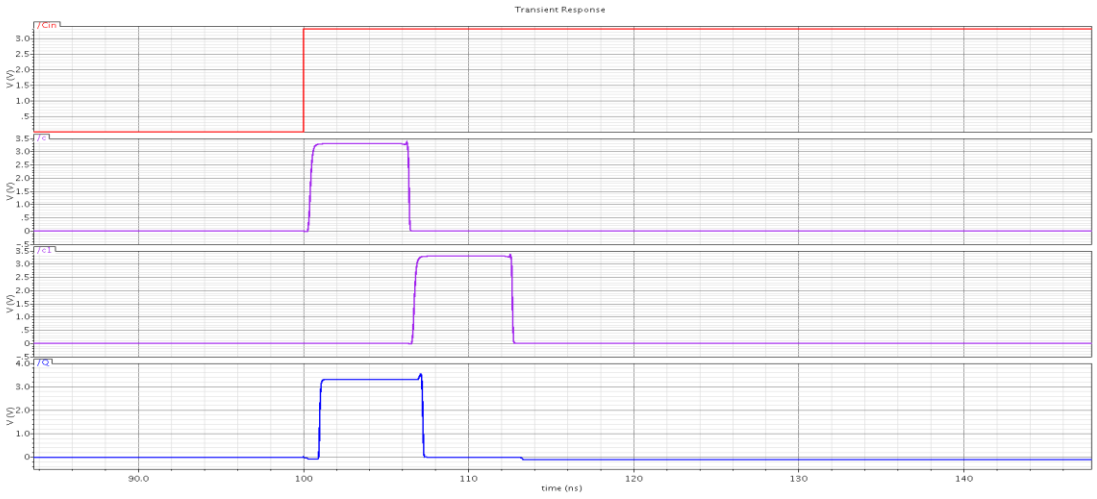

Figure 26 - Signal written in the common line

As we can see, the line writes the two values (H and L) consecutively.

The network of the rest of the design for one pixel is simple too, as it can be seen in the following figure:

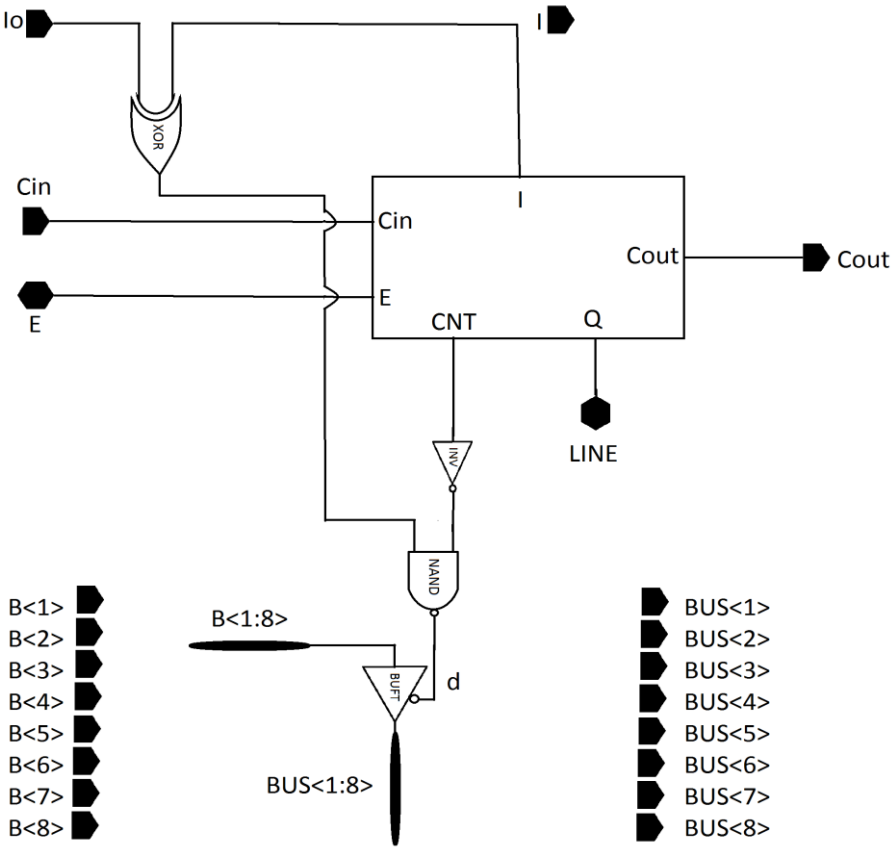

Figure 27 – Complete column decoder consisting of 256 cascaded cells. Each cell looks for a binary transition on consecutive pixels and delivers a bit for the required address

It consists on considering two inputs from the pattern: the before one (Io) and the current one (I). The block which is in the figure corresponds to the first schematic designed and it is easy to see that the output Q is the one which writes the signal to the common line (LINE).

The CNT pulse is firstly inverted and then is combined with the result of the end of the XOR gate. This XOR gate detects a disparity between two inputs of the pattern: this means that if the current input I has a different value than the input before Io, there is a change of value and then an address has to be sent (the address of the pixel before).

We can also see other 8 buffers tristate which are connected to the bus lines and are those which write the addresses. If there is a change of value ($Io \neq I$) then the XOR gate delivers a 1 which is combined with the CNT pulse with a NAND gate. This resultant pulse enables the buffer to write the address to the bus during the time which corresponds to the width of this pulse. If there is no change of value ($Io = I$) then the XOR gate delivers a 0 and so there is no pulse at the enable input of the buffer, thus the buffer is disabled and no address is written (the bus keeps floating).

The 8 inputs from B<1> to B<8> are the addresses I have to write in order to be sent in case of disparity between two inputs of the pattern, and the 8 outputs from BUS<1> to BUS<8> are the outputs (addresses) that go to the bus. It has been used an 8-bit code, so the bus has 8 bit-lines and this is the reason why are needed 8 drivers (one driver for each bit-line) for each pixel.

4.3. Creation of the blocks

The block of the last schematic has two other levels: one created in order to use the pins and the highest one. The first mentioned is shown in the following figure:

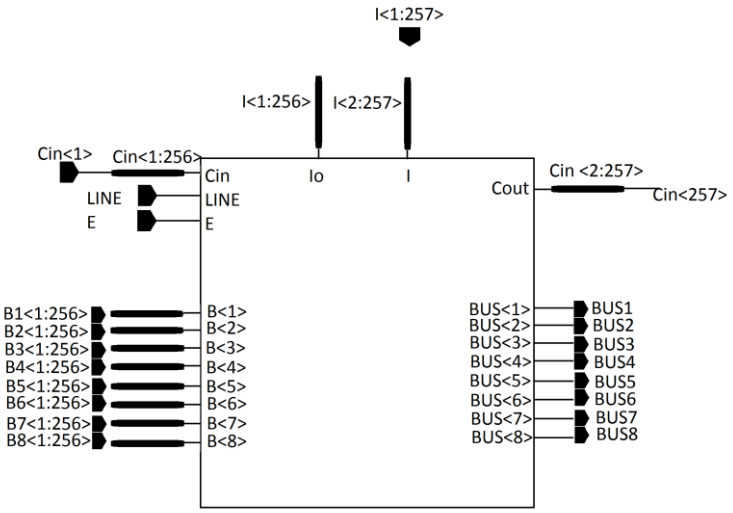

Figure 28 - First block

As we can see, two buses have been created for the input pattern: one for the previous input of the cell and one for the current input. This is the reason why the current pattern has 257 inputs instead of having 256, in an example of 256 cells (the 257th input has no effect on the simulations).

There is also the carry-in signal (connected with the 256 cells) and is only necessary to enter the pulse signal on the first carry-in input cell. The enable signal is connected with the start signal as well.

We have to note that I include 8 buses in order to write the addresses. Each bus has 256 inputs which correspond to the 256 pixels. For example, if we have to write the address of the 113th pixel, the binary code is 01110001 and so the inputs would be:

B1<113> = 1 B4<113> = 0 B7<113> = 1

B2<113> = 0 B5<113> = 1 B8<113> = 0

B3<113> = 0 B6<113> = 1

We also have to note that there are 8 lines for the bus that will be read for the addresses.

Once the block has been implemented, it has been created the other one which corresponds to the highest level and is the one used for the simulations. The following figure shows it with the environment:

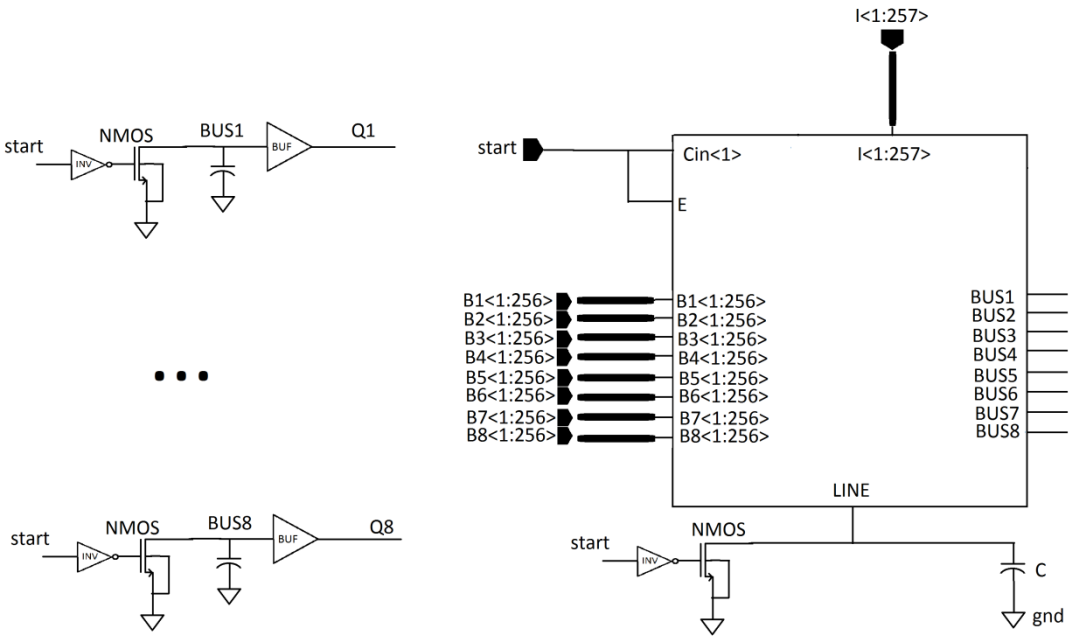

Figure 29 - Block for the simulations

As we can see, the first Cin and the E input are connected together by the start signal. On the common line and in all the bus lines there are included a transistor in order to set each line to 0 at the start of each period of time. It is also included a capacitor of 2 pF of capacitance at the common line and at each line of the address bus in order to not allow the lines discharging during the process. The real value needed is going to be exactly calculated at the appendix, but these 2 pF are enough for the following simulations, as I demonstrate before.

It is also included a buffer (BUF) at the bus lines in order to have better and clearer waveforms at the output.

Therefore, the cascaded blocks are like in the following figure with their respective transistors, capacitors and buffers.

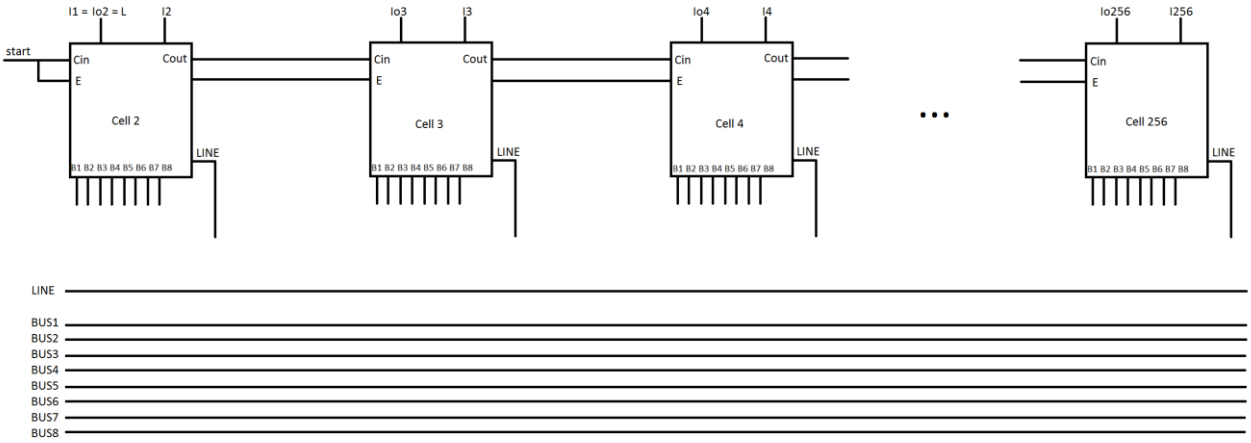

Figure 30 - Cascaded blocks

4.4. Simulations

Once all the blocks with the environment are determined, it has been proceeded to simulate them. First of all, to check the functionality of the buffer BUF, two simulations have been made: one without the buffer and another one with it. The two following images show the comparative of the results at the output with a row of 10 pixels with the pattern 1110101010, as an example:

Image without the buffer:

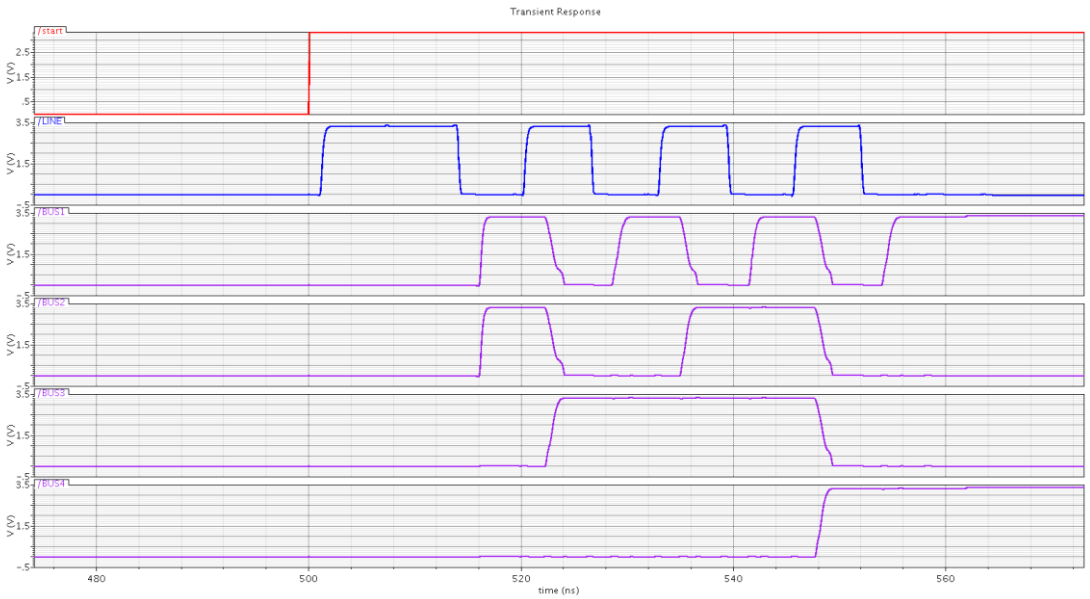

Figure 31 - Simulation without the buffer

Image with the buffer:

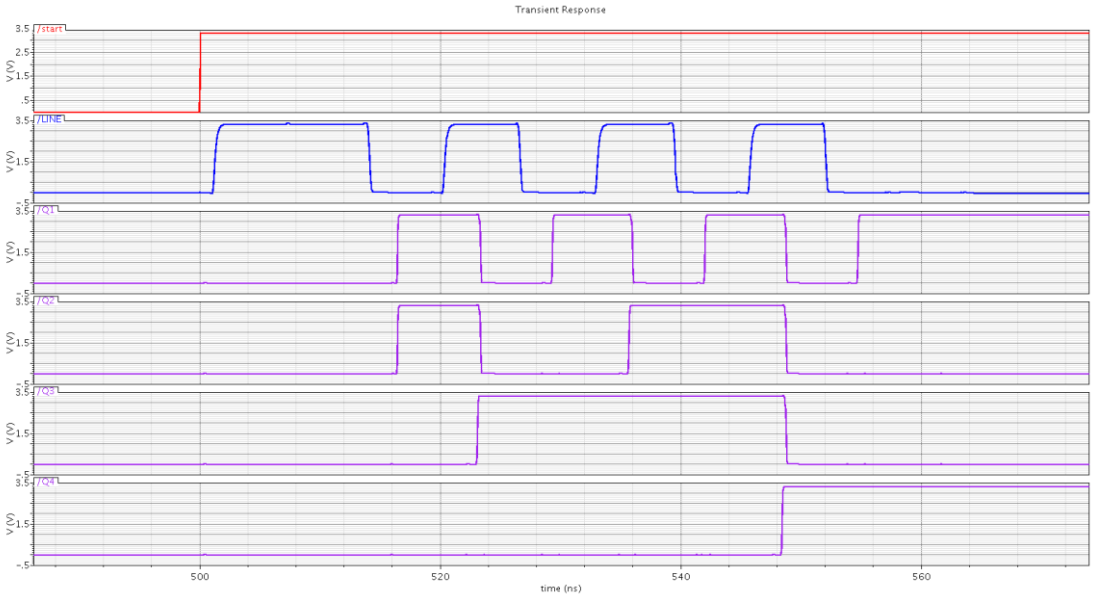

Figure 32 - Simulation with the buffer

As we can see, the second image (the one which corresponds to the waveforms after the buffer) is clearer than the first (without the buffer). In order to check if the signals delivered are the good ones is easy to see that the signal has 4 pulses corresponding to the high voltage values from the pattern. The addresses that should be delivered are the corresponding to the pixels number 3, 4, 5, 6, 7, 8, and 9 which are the following ones: 00000011, 00000100, 00000101, 00000110, 00000111, 00001000, 00001001. This means that, considering that the first value of the row is known (equal to 1 in this case), from the 1st to the 3rd pixel the signal is 1 and from the 4th to the 10th the binary value is changing in each pixel.

Thus the signals written are correct (there are no included the other buses because their value is 0 in this case).

It is also easy to see that the period of propagation time is also about 12.7 ns (frequency about 79 MHz).

5. Simulations of the final network and results

After knowing that is better to use the buffer BUF and knowing the value of the capacitance needed, the next step is to simulate the blocks with different input patterns (considering a row of 10 pixels to simplify the time of the simulations) in order to see if the circuit works as it was expected to:

Input pattern 010101010101:

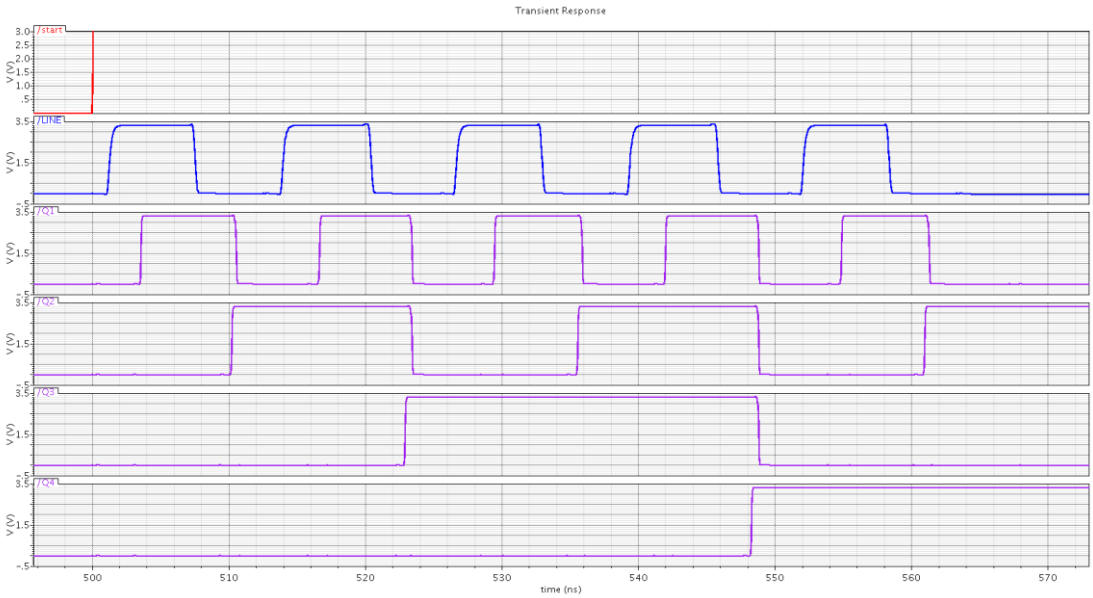

Figure 33 - Simulation 1

With this input pattern, the addresses that should be delivered are the corresponding from the 1st pixel to the 9th which are the following ones: 00000001, 00000010, 00000011, 00000100, 00000101, 00000110, 00000111, 00001000 and 00001001. This means that, considering that the first value of the row is 0, from the 1st to the last pixel the value is changing in each pixel.

The time from the first clock signal until the last address written is about 61 ns.

As we can see, the signal at the common line and the addresses delivered are the expected.

Input pattern 0111000111:

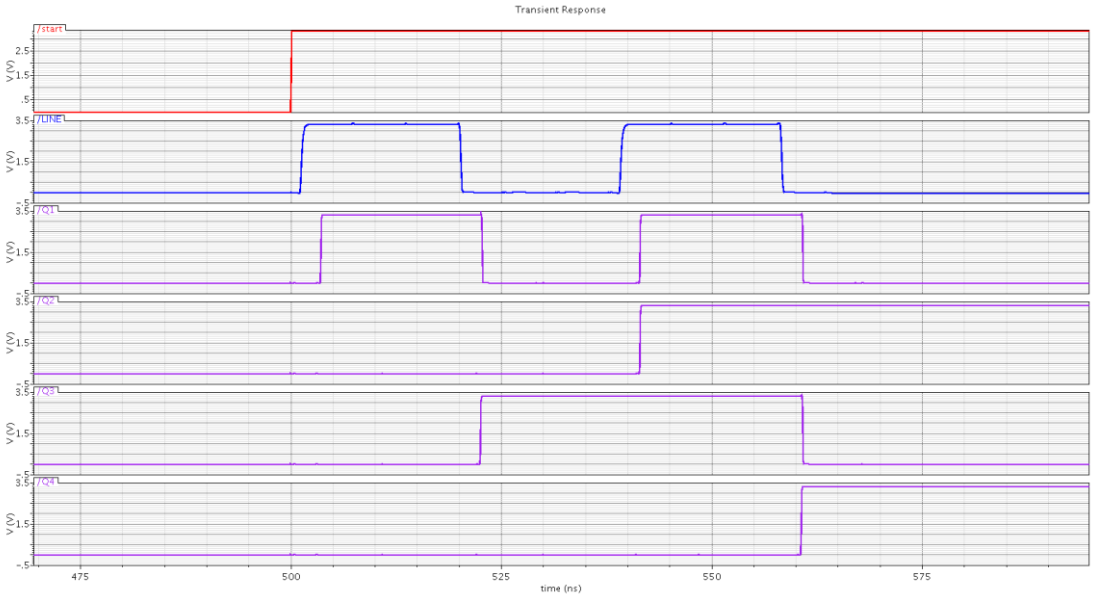

Figure 34 - Simulation 2

With this input pattern, the addresses that should be delivered are the corresponding for the pixel number 1, 4 and 7 which are the following ones: 00000001, 00000100 and 00000111. This means that, considering that the first value of the row is 0, from the 2nd to the 4th pixel the value is 1, from the 5th to the 7th is 0 and from the 8th to the last one is 1 again.

The time from the first clock signal until the last address written is about 42 ns.

As we can see, the signal at the common line and the addresses delivered are the expected.

Input pattern 0000110000:

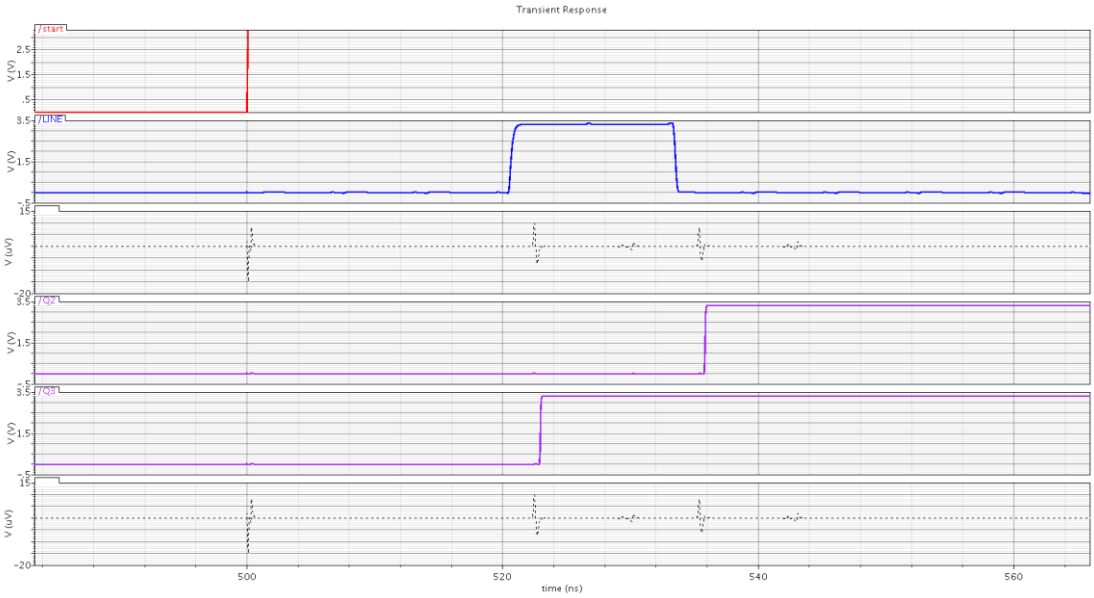

Figure 35 - Simulation 3

With this input pattern, the addresses that should be delivered are the corresponding for the pixel number 4 and 6 which are the following ones: 00000100 and 00000110. This means that, considering that the first value of the row is 0, from the 1st to the 4th pixel the value is 0, from the 5th to the 6th is 1 and from the 7th to the last one is 0 again.

The time from the first clock signal until the last address written is about 42 ns.

As we can see, the signal at the common line and the addresses delivered are the expected. The signals on the buses number 1 and 4 (Q1 and Q4 at the plotted results) are not printed because haven't received any value.

Input pattern 0110100110:

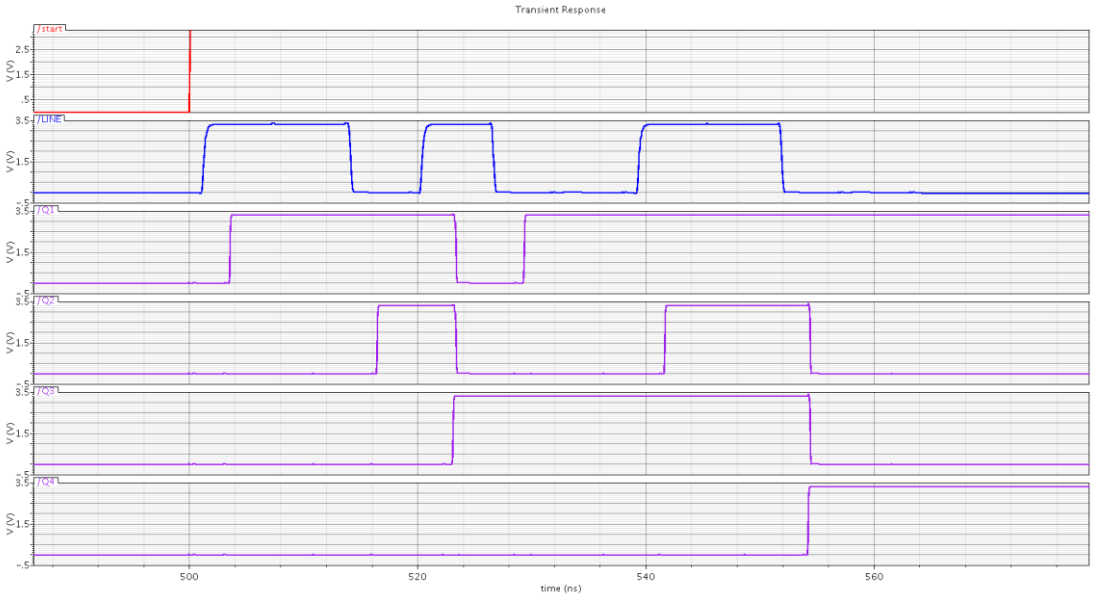

Figure 36 - Simulation 4

With this input pattern, the addresses that should be delivered are the corresponding for the pixel number 1, 3, 4, 5, 7 and 9 which are the following ones: 00000001, 00000011, 00000100, 00000101, 00000111 and 00001001 (the addresses are held if there is no change of value, as it can be seen). This means that, considering that the first value of the row is 0, the value of the row is the same as the input pattern.

The time from the first clock signal until the last address written is about 61 ns.

As we can see, the signal at the common line and the addresses delivered are the expected.

Input pattern 0010011000:

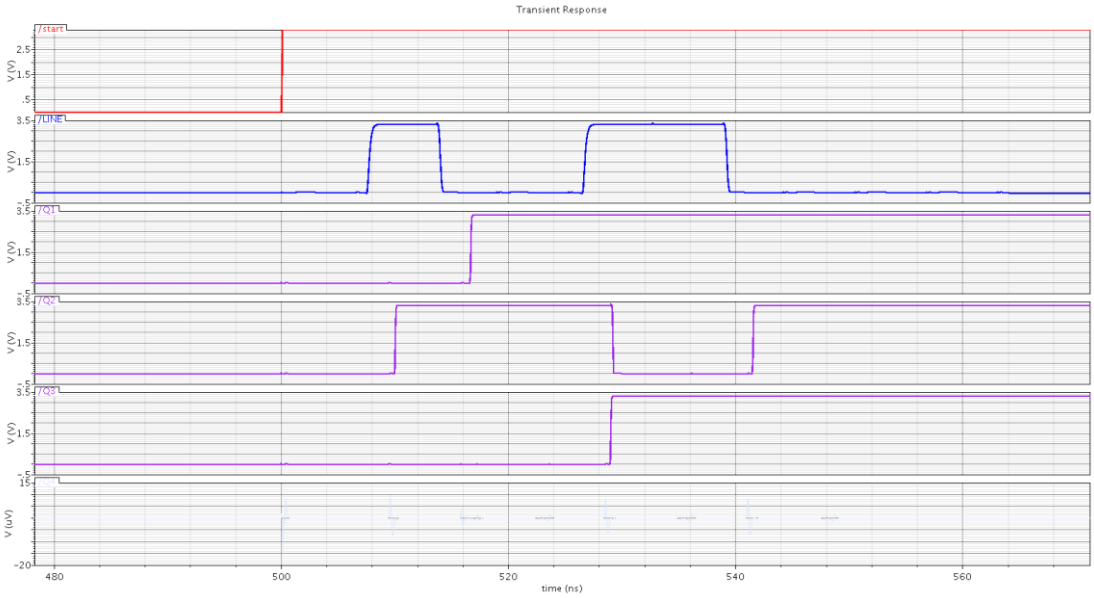

Figure 37 - Simulation 5

With this input pattern, the addresses that should be delivered are the corresponding for the pixel number 2, 3, 5 and 7 which are the following ones: 00000010, 00000011, 00000101 and 00000111. This means that, considering that the first value of the row is 0, from the 1st to the 2nd pixel the value is 0, the 3rd value is 1, from the 4th to the 5th is 0, from the 6th to the 7th is 1 and from the 8th to the last one is 0 again.

The time from the first clock signal until the last address written is about 46 ns.

As we can see, the signal at the common line and the addresses delivered are the expected.

After obtaining the results of the simulation, the following table shows a comparison between these patterns:

Input pattern	Number of requested pixels (over 10)	Time of response of the addresses
0101010101	9	61 ns
0111000111	3	42 ns
0000110000	2	42 ns
0110100110	6	61 ns
0010011100	4	46 ns

Table 3 - Comparison of the results

Of course, the time of response of the addresses doesn't depend on the number of pixels requested but on the first and last values of the pattern. If the last values are the same, thus the bus doesn't change its value at the end of the process and for this reason it is not necessary to read it until the end.

6. Conclusions

This work presents the design of a CMOS Asynchronous Column Decoder for a binary vision sensor which delivers data using run-length encoding in order to minimize the output bandwidth and the power consumption as well. Different circuit solutions have been investigated and simulated aimed at meeting the sensor characteristics. Intensive simulations have been carried out using CADENCE design tools.

The network designed writes the signal that represents the input pattern of a row of a binary contrast image in a line shared by all the cells in each rising edge of the start signal.

The network designed delivers the addresses only of the pixels before a change of logical value in order to reduce the total amount of data delivered by the sensor.

The system is asynchronous. The propagation time per cell is about 6.3 ns. The time of response of the output signal and of the addresses depend on the clusters of logical values at the beginning and final of the input pattern.

This technique is only convenient to use in case of having large runs of 0 or 1 in a row of a binary contrast image. In this case, the data delivered by the sensor is low in comparison with the other techniques presented (delivering bit map, AER and AER by row) and the power consumption can be reduced.

A binary contrast image with not so many pixels but dispersed individually cannot be a good case of using this network, because an individual pixel with a different value from the others around it means delivering two addresses per this kind of pixel.

The network designed works as it was expected at the beginning of the project.

7. References

[1] M. GOTTARDI, N. MASSARI, S. A. JAWED, *A 100µW 128x64 Pixels Contrast-Based Asynchronous Binary Sensor for Sensor Network Applications*. IEEE Journal of Solid State Circuits, vol. 44, no. 5, May 2009. p. 1582–1592

[2] PATRICK LICHTSTEINER, CHRISTOPH POSCH, TONI DELBRUCK. *A 128 x 128 120 dB µs Latency Asynchronous Temporal Contrast Vision Sensor*. Vol. 43, NO. 2, February 2008.

[3] J. CHOI, S. PARK, J. CHO, E. YOON. *A 3.4 µW CMOS image sensor with embedded feature extraction algorithm for motion-triggered object-of-interest imaging*. IEEE International Solid-State Circuits Conference Digest of Technical Papers, February 2013. p. 478-479

[4] N. COTTINI, M. GOTTARDI, N. MASSARI, R. PASSERONE, Z. SMILANSKY. *A 32 µW 42 gops/W 64 x 64 Pixels Vision Sensor with Dynamic Background Subtraction for Scene Interpretation*. Int. Symposium in Low Power Electronics and Design, July 30 – August 1 2012. p. 315-320

[5] G. KIM, M. BARANGI, Z. FOO, N. PINCKNEY, S. BANG, D. BLAAUW, D. SYLVESTER. *A 467 nW CMOS visual motion sensor with temporal 19 averaging and pixel aggregation*. Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International, February 2013. p. 480-481

[6] R. S. – G. J. COSTAS-SANTOS, T. SERRANO-GOTARREDONA, B. LINARES-BARRANCO. *A Spatial Contrast Retina with on-chip calibration for Neuromorphic Spike-Based AER vision systems*.

[7] BERTHOLD KLAUS PAUL HORN. *Robot Vision*. Cambridge, Massachusetts, 1986. p. 58-61

[8] AUSTRIAMICROSYSTEMS. *ENG – 182*. p. 20-23, p. 51

8. Additional bibliography

ALEJANDRO LINARES-BARRANCO, GABRIEL JIMENEZ-MORENO, BERNABÉ LINARES-BARRANCO, ANTÓN CIVIT-BALCELLS. *On Algorithmic Rate-Coded AER Generation*. Vol. 17, NO. 3, May 2006.

9. Appendix - Estimation of the capacitance

9.1. Introduction

The bus lines need a capacitor in order to hold the signal during the process when is needed. Rather than use a random capacitance, simulate the circuit and then see if it works or not, it would be better to estimate the capacitance needed at each line depending on the number of cells used in the circuit. For example, if it is known the capacitance that is needed for one cell (for one driver), is it possible to know the capacitance needed for n cells.

9.2. Calculation

To calculate it, it is necessary the length and the width of the two MOSFET transistors that are connected with the line of the bus. These two transistors (GT_MP5W and GT_MN9W in the library of Cadence) are shown in the following figure, which represents the schematic of the buffers tristate used for the design (BUFT2 or BUFT15 in the library of Cadence):

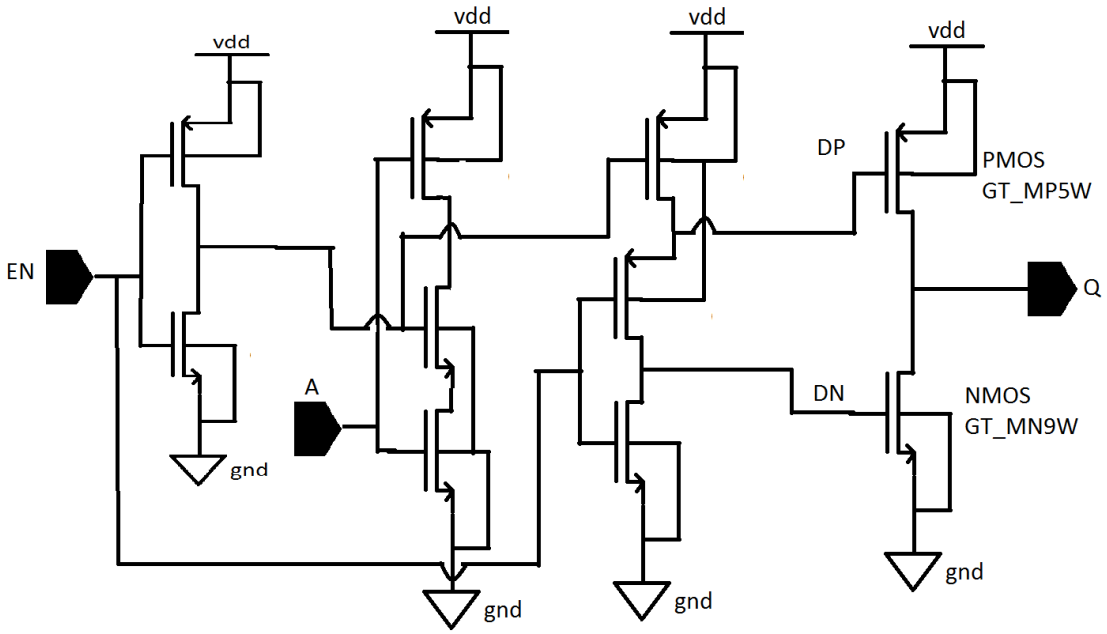

Figure 38 - Schematic of a buffer tristate in Cadence

For the buffers have been chosen the smallest and the strongest one in the library, in order to compare both and see what is better: having a smaller capacitance but with more bad waveforms or having a bigger capacitance but having clearer waveforms.

The parameters of these transistors are shown in the following figure:

Figure 39 - Length and width of the transistors

As we can see, the length and width of the two MOSFET are (in case of BUFT15):

PMOS:

L = 350 nm W = 24 µm

NMOS:

L = 350 nm W = 15 µm

And in case of BUFT2 are:

PMOS:

L = 350 nm W = 3.2 µm

NMOS:

L = 350 nm W = 2 µm

In order to calculate the capacitance for the capacitor, the process has been separated in two steps: calculate the capacitance between the gate and the drain and calculate the junction capacitance.

9.2.1. Capacitance between gate and drain:

The way to do that is to use the different coefficients [8], such as the following ones:

Pass/Fail Parameters				
Parameter	Min	Typical	Max	Unit
POLY1-DIFF (gate oxide)				
POLY-DIFF area	4.26	4.54	4.86	fF/µm^2

Table 4 - Coefficients

Now it is just to make the product between the area (WxL) and the "typical" coefficient in fF/ µm^2.

9.2.2. Junction capacitance:

To estimate the capacitance of the junction it is used the following equation [8]:

$$C = \frac{W \cdot L \cdot CJ}{(1+\frac{V}{PB})^{MJ}} + \frac{2 \cdot (W+L) \cdot CJSW}{(1+\frac{V}{PB})^{MJSW}}$$

C is the junction capacitance, W is the width of the transistor, L is the length of the transistor, V is the bias voltage, CJ is the junction capacitance per drawn area, CJSW is the junction capacitance per drawn perimeter, PB is the junction potential, MJ is the area junction grading coefficient and MJSW is the sidewall junction grading coefficient.

The rest of the parameters are determined such as the following ones [8]:

NDIFF - PWELL		
Parameter	Typical	Unit
CJN	0.84	$fF/\mu m^2$
MJN	0.34	-
PBN	0.69	V
CJSWN	0.25	fF/µm
MJSWN	0.23	-
PDIFF - NWELL		
Parameter	Typical	Unit
CJP	1.36	$fF/\mu m^2$
MJP	0.54	-
PBP	1.02	V
CJSWP	0.35	fF/µm
MJSWP	0.46	-

Table 5 – Parameters

9.3. Results

Capacitance needed for BUFT15:

After estimate each capacitance, the total capacitance needed in the line for a driver (the sum of the capacitance between gate and drain and the capacitance of the junction) has been found. The total number is C = 83.5 fF, which is a bit high. But of course, with this driver the waveforms are clearer.

Capacitance needed for BUFT2:

The total number is C = 11.4 fF, but with this driver the waveforms are less clear.

These capacitances mean that for an array of 10 pixels, for example, it should be used a capacitor with about 1 pF of capacitance (in case of using the BUFT15) or a capacitor with about 120 fF of capacitance (in case of using the BUFT2).

9.4. Simulation of the drivers

In order to know if the estimation of the capacitance value is correct, a simulation with the buffers used for the calculi has been made. To know it, one of the drivers connected has been used as in the following figure:

Figure 40 - Buffer simulated

As we can see, the input of the driver is connected to the ground. This is because before providing a voltage to disable the buffer and make it in tristate we want to set the line to zero. This means that while the driver is enabled (low voltage) it is writing the zero value to the line, and while is disabled (high voltage) is in tristate and then keeps the line as the same value.

The thing is that while the buffer is in tristate, the line is on zero value and just a few nanoseconds after being disabled, a current (1 pA, for example) is provided in order to increase the voltage of the line. The following figure shows the signal (in V) delivered by the two sources (current and voltage):

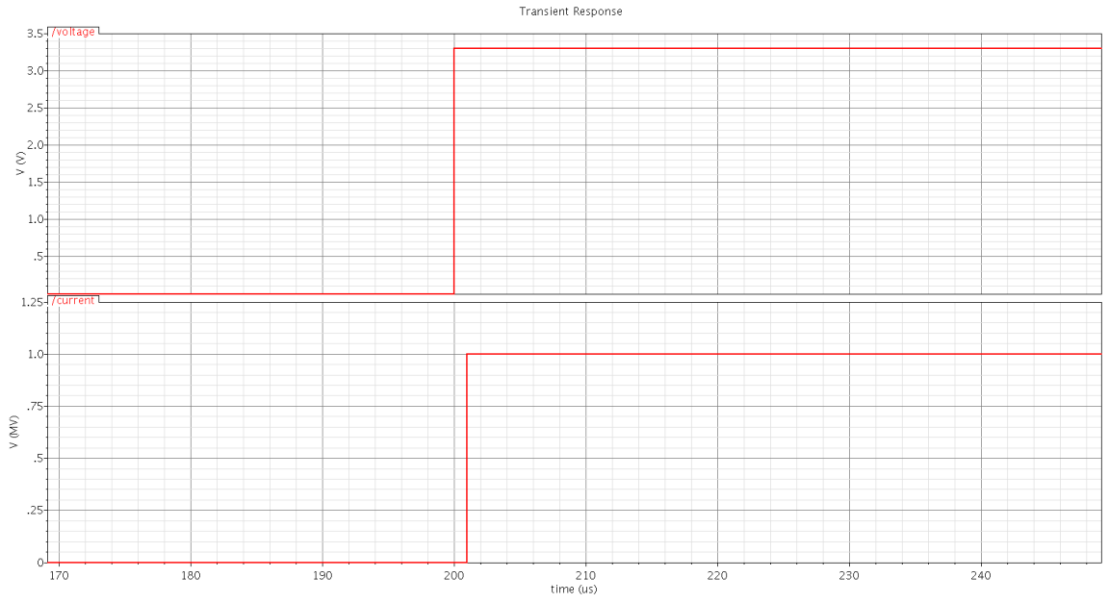

Figure 41 - Voltages entered

While the voltage of the line is increasing, it is writing a curve from zero to the last value of voltage. If the two values of this curve and their respective value of the time could be known, we would be able to calculate its slope: $\frac{\Delta V}{\Delta t}$. If we have the slope, it is possible to know the capacitance of the output of the driver, which should be more or less the same as the one has been estimated. The way to know it is the following equation:

$$I = C \cdot \frac{dV}{dt}$$

So, the capacitance is:

$$C = I \cdot \frac{\Delta t}{\Delta V}$$

Two drivers have been simulated (BUFT2 and BUFT15) and the plotted results are the following shown:

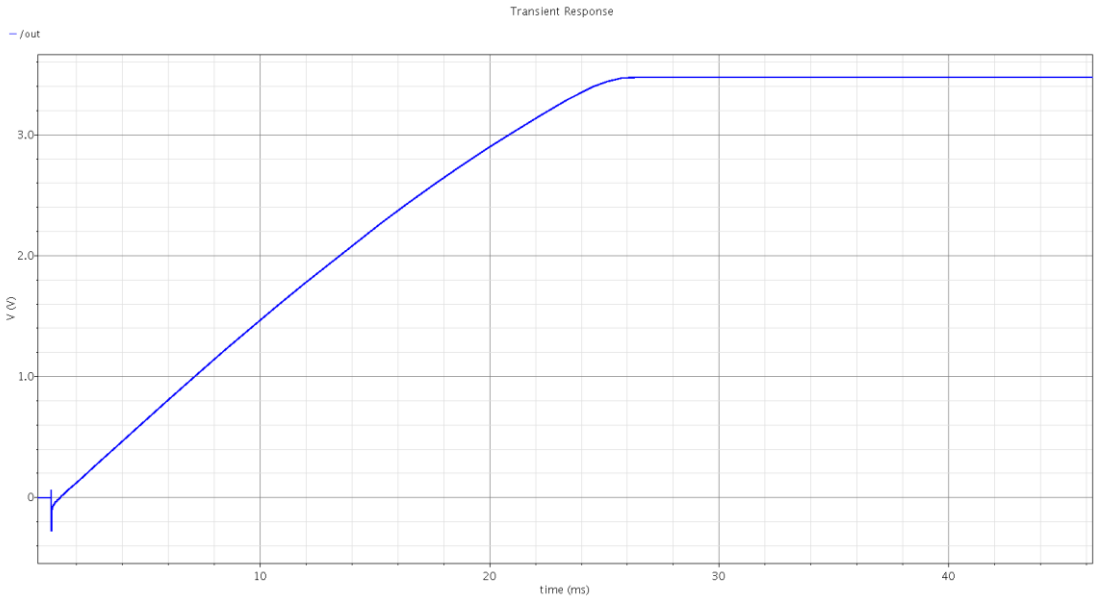

Figure 42 - Curve with BUFT2

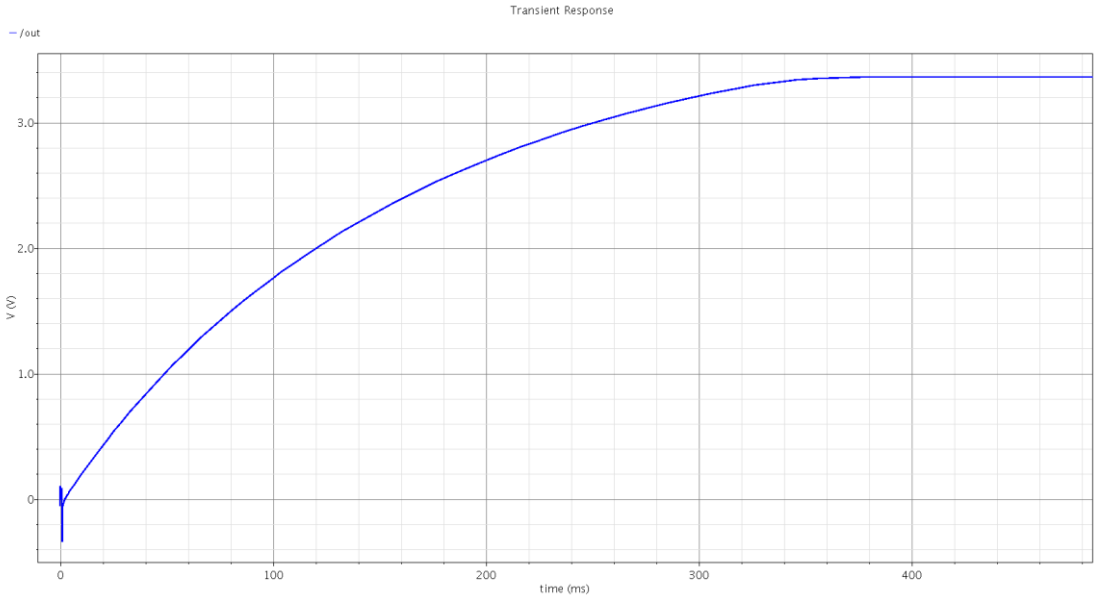

Figure 43 - Curve with BUFT15

With the last equation, it have been calculated the capacitances of the two buffers. The values chosen are the first instant of time when the voltage starts to increase and the instant of time when the value of the voltage is on the 63% of the last value.

The capacitances calculated are:

BUFT2

Capacitance calculated: C = 11.64 fF

BUFT15

Capacitance calculated: C = 84.29 fF

Therefore, the comparison is the following one:

	BUFT2	BUFT15
Capacitance estimated	11.39 fF	83.53 fF
Capacitance calculated	**11.64 fF**	**84.29 fF**

Table 6 - Capacitances estimated and calculated

As we can see, the two values are really similar and thus we know that the two capacitances of the drivers are well estimated.

9.4.1. Simulation of the array of 100 buffers

Now it is important to see if the capacitance calculated changes accordingly with the number of drivers. In this case, some things for this simulation have been changed respecting with the last ones. First of all, the *enable* gates have been set at the high voltage (vdd = 3.3 V) and the line has been set to zero with the initial conditions: this means that it is not necessary to create a pulse voltage. It has been also created an array of 99 drivers (BUFT15 is the chosen because of its waveforms) using only one symbol of them, as we can see in the following figure:

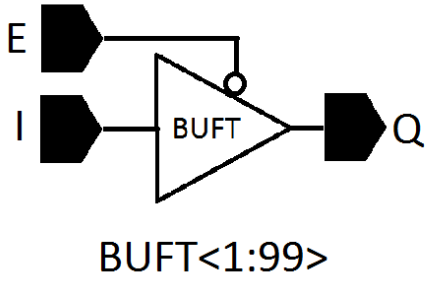

Figure 44 - 99 BUFT15

In this way, the final schematic is the following one:

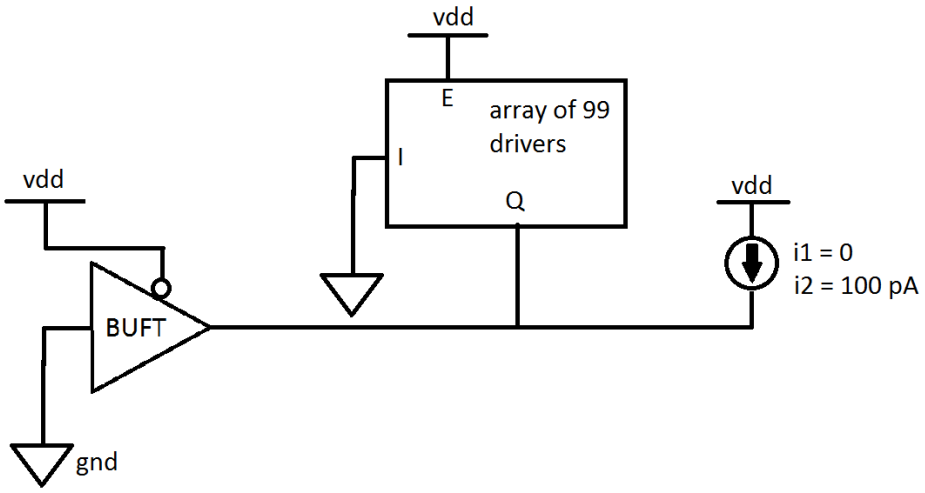

Figure 45 - Schematic of 100 BUFT15

In order to know if the capacitance changes accordingly to the number of drivers, firstly the circuit has been simulated with the only driver connected to the line with a current of 1 pA, and the rising time which corresponds to the time at the 63% of the reached voltage has been calculated. As the reached voltage is 3.3 V, the 63% is about 2 V; this means that it is necessary to know the time between the start of the charging current and the time reached at 2 V. The schematic is the following one:

Figure 46 - Schematic of 1 BUFT15

The result of this first simulation is:

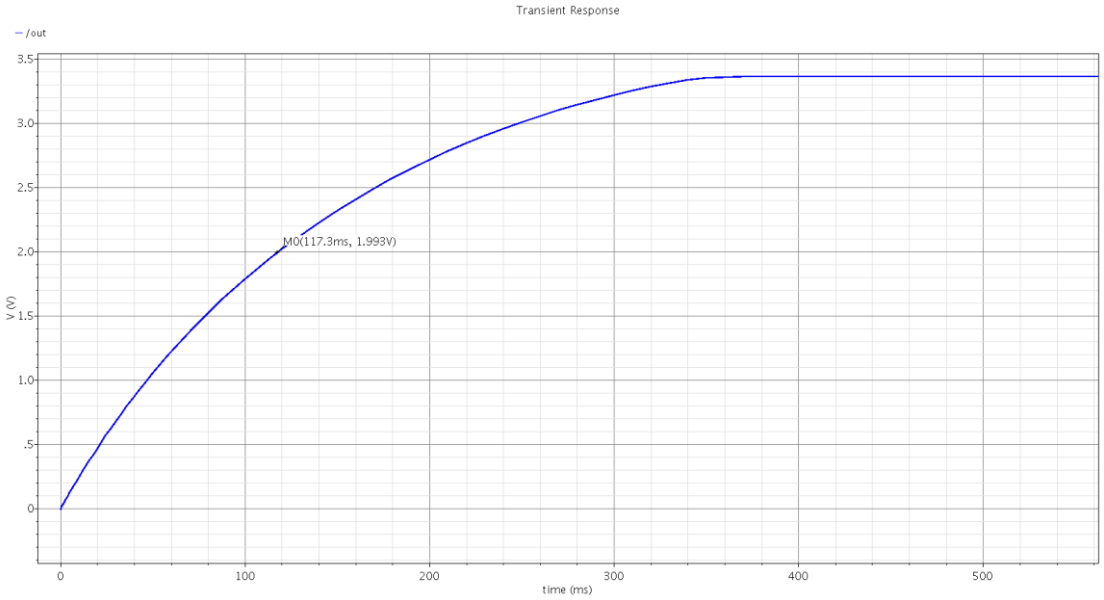

Figure 47 - 1 BUFT15 simulated

As we can see, the rising time is about 117 ms.

To make the simulation with the array of 100 drivers, the same process has been used: calculate the rising time at the 63% of the reached voltage. In order to simplify the calculation, the value of the current has been changed to 100 pA; this means that the rising time should be the same as the last simulation, because the capacitance and the current are 100 times more than the last ones. The following figure shows the plotted voltage of this simulation:

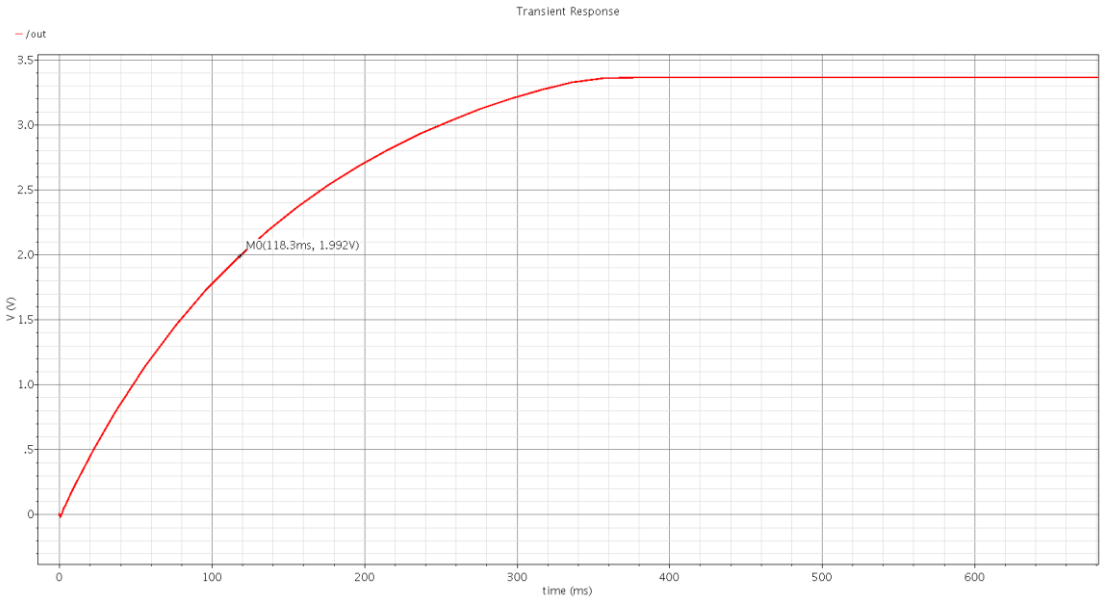

Figure 48 - 100 BUFT15 simulated

Thus the rising time at 2 V is about 118 ms, approximately the same value of the simulation with only 1 driver.

9.4.2. Simulation of the array of 256 buffers

An array of 256 drivers has been simulated and the rising time is about 116 ms, as we can see in the following figure:

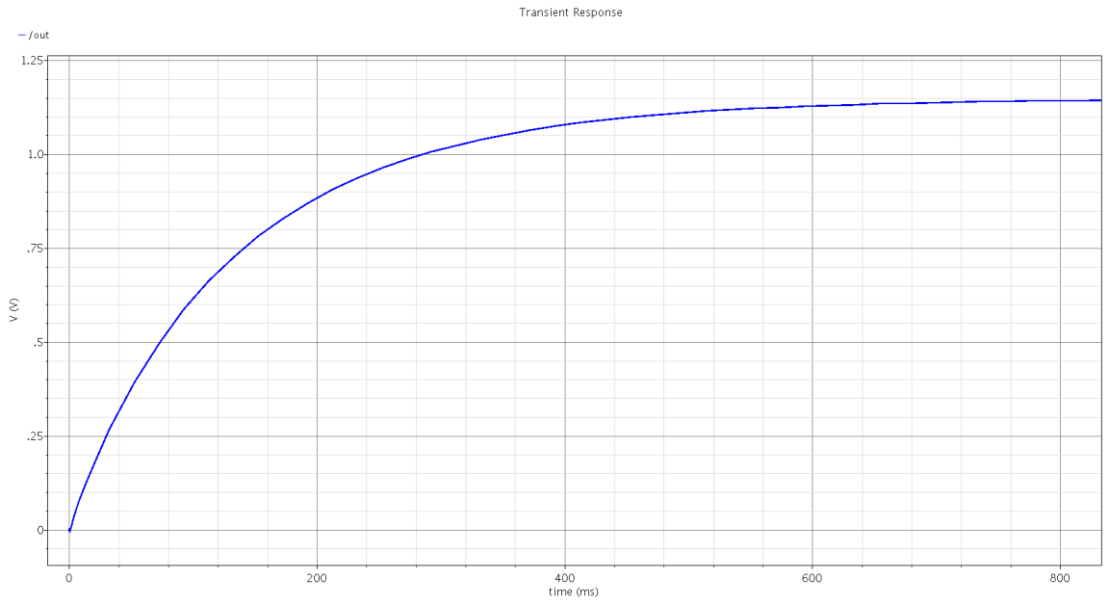

Figure 49 - 256 BUFT15 simulated

Another way to measure the capacitance is to know that if with a current of 256 pA it takes 116 ms to reach the 2 V (63% of 3.3 V), then the capacitance of the 256 drivers is:

$$C = 256\ pA \cdot 166\ ms / 2\ V = 21248\ fF \sim 84 \text{ (capacitance of 1 driver) x 256 (total drivers)}$$

These results mean that the capacitance changes accordingly to the number of drivers used.

After having checked the correct behavior of the capacitance's changes, a simulation has been made just to see if the output signal is too much affected by all of these buffers. In order to do that, some changes have been made to the schematic, as we can see on the following figure:

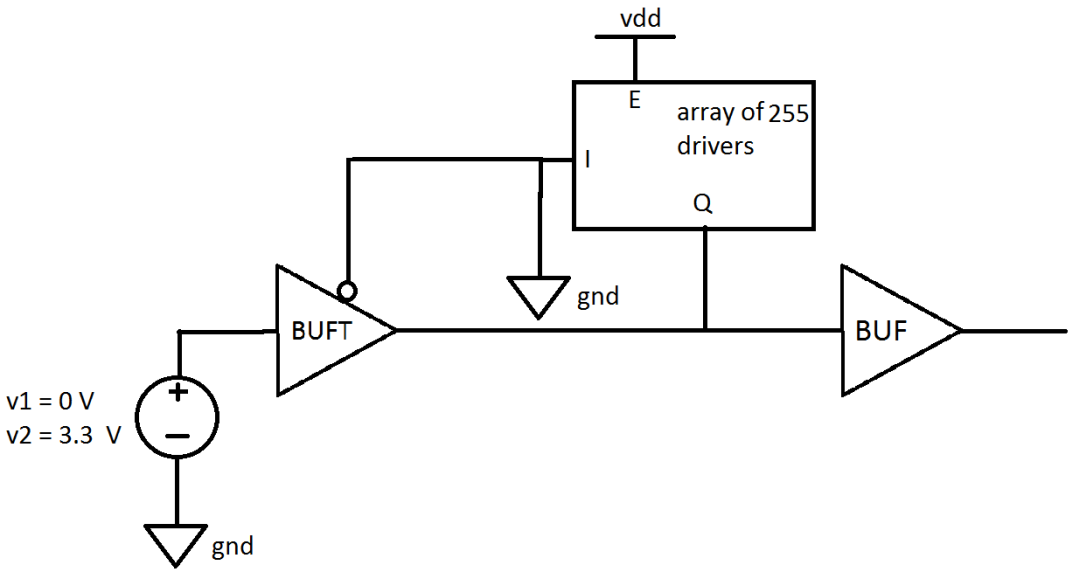

Figure 50 - New simulation with buffers

As we can see, now we have the array of 255 buffers disabled (in tristate) and the other buffer enabled. The input of this enabled buffer is a signal pulse with a period of 13 ns. This is because the period of the network designed in case of having 10101010… (worst case scenario) at the input pattern is about 12.7 ns (a frequency about 79 MHz). It is easy to note that it is not necessary the current source now and a buffer is also included (BUF4 in the library of Cadence) at the output line in order to obtain clearer waveforms.

After simulating this circuit, the following signals have been obtained:

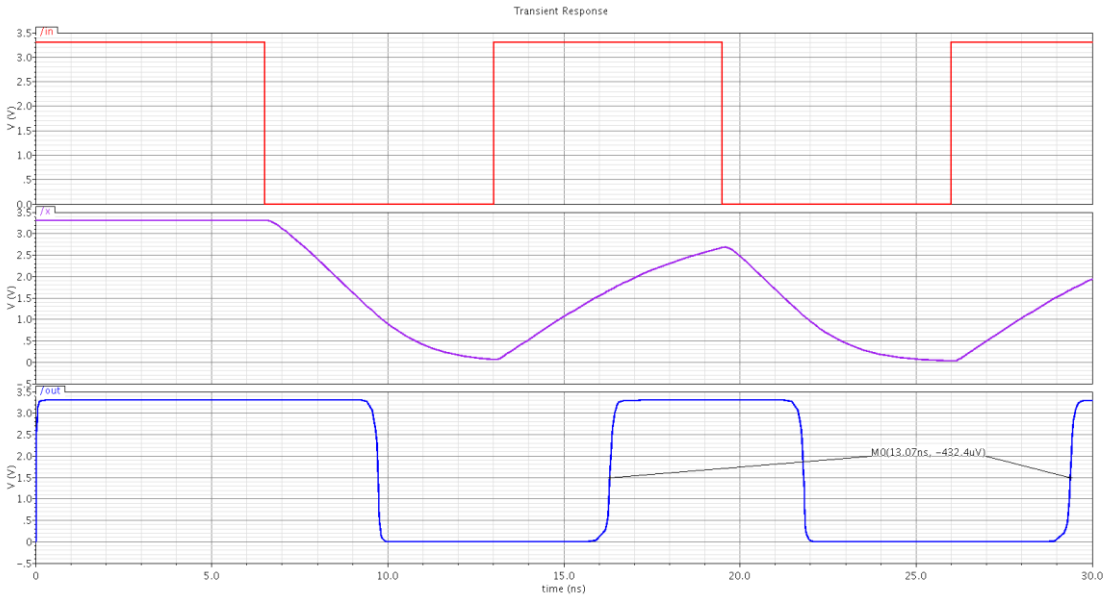

Figure 51 - Signals obtained

As we can see, the first signal (in) corresponds to the pulse at the input of the enabled buffer (with a period of 13 ns or a frequency about of 77 MHz).

The second signal (x) is the signal between the buffer at the output signal (BUF4) and the output of the array of 256 buffers. This signal is really smooth and this is the reason why this BUF4 is needed.

The third signal (out) is the output signal after the BUF4 and it is easy to see that the waveform is clearer than the second signal. It can also be seen that the pulses are shifted respecting to the pulse at the input, this is because the line has a large capacitance (the strongest buffers BUFT15 have been used for the simulation) but it doesn't matter because the period -as we can see- is about 13 ns as well.