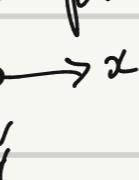
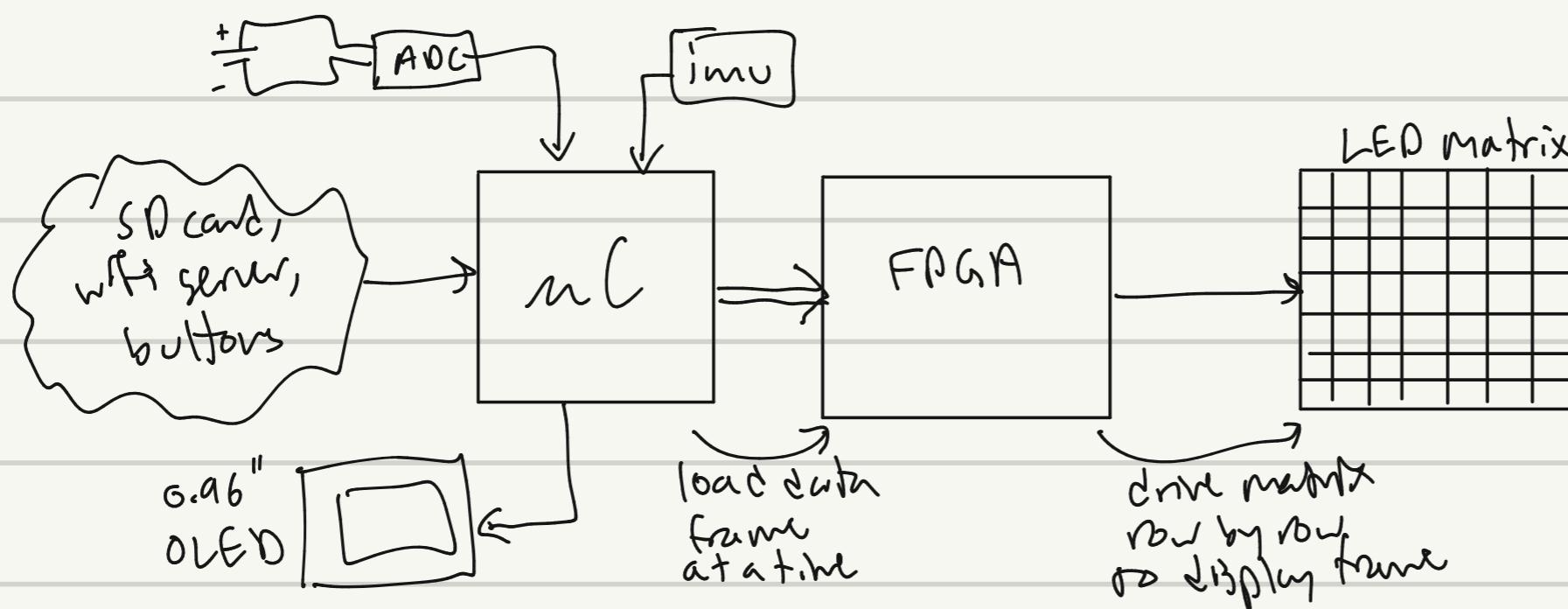


## Fancy Grad Hat Architecture

- There are  $32 \times 32 = 1024$  pixels/LED's.
- Each "frame" sees each pixel as being either ON or OFF
  - each pixel has a 0 or a 1.
  - each "frame" is encoded into a 1024 bit word
    - ↳ each bit corresponds to a pixel in order from left to right, then top to bottom, with origin  ∴ ↗ type path.
- Animations are divided into frames.
- High level data path is as follows:



### Overall Circuit

The mC provides easy access to peripherals over I<sub>2</sub>C (ADC, IMU, OLEd), SPI (SD card), and user interfaces (buttons, wifi, bluetooth etc ways to save, choose, and load frames or animations). When prompted, it loads a frame into the FPGA via a parallel interface (described below), and then commands the matrix to display it.

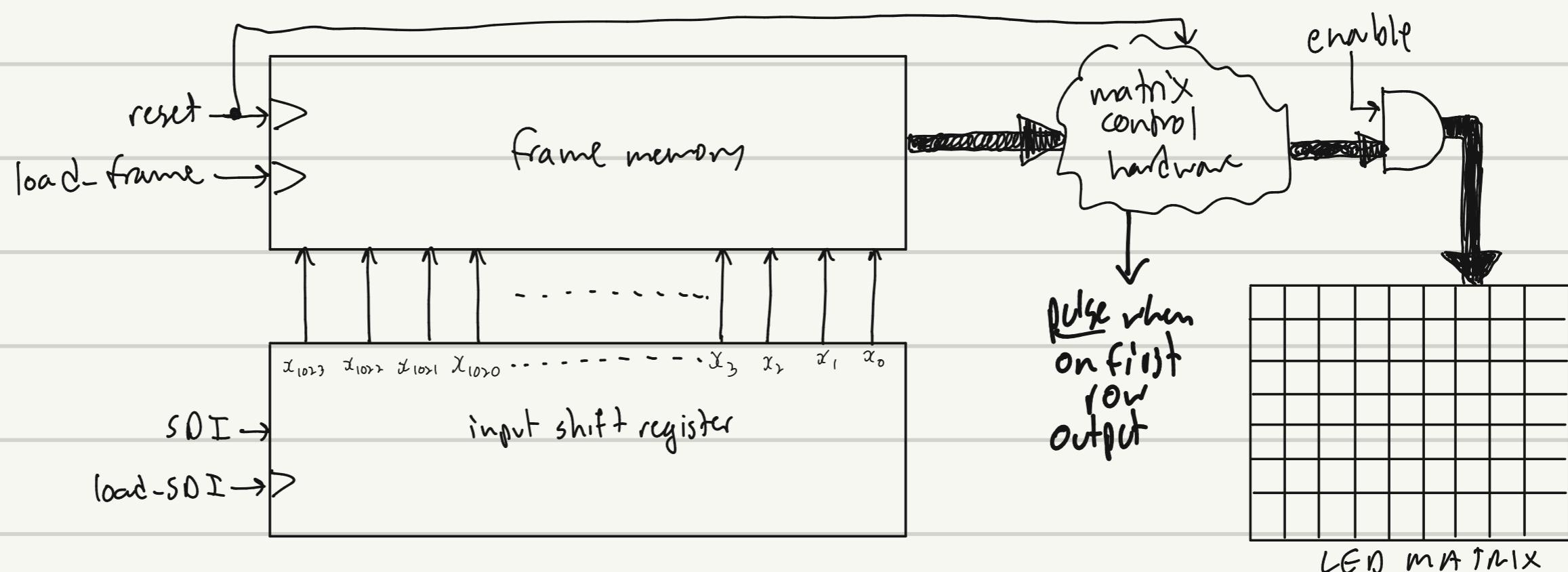
external cos reading  
volt voltage → vdd voltage & 3.3v for mC  
volt voltage → for cool potential features  
for cool potential features  
to display batt voltage  
& other stuff things  
to store frames

The purpose of the FPGA is to take the "frame" data provided to it by the mC and drive the matrix accordingly with appropriate timing. It does this by exploiting the "Persistence of Vision" (POV) effect: It lights up specific led's row by row from top to bottom repeatedly & very quickly, to give the illusion of a still image. We will only go one row at a time per 'top to bottom' sweep for simplicity.

ML to FPLA parallel interface

4 wires are used for this interface:

- ① reset → clears FPGA's frame memory. [positive edge triggered.]
  - ② enable → 1: matrix driven based on current FPGA frame memory  
0: entire matrix off
  - ③ load-SDI → loads value on SDI input into input shift register [positive edge triggered]
  - ④ serial data in (SDI) → like a FIFO shift register, loads the bit into the MSB of the FPGA's input shift register & shifts the word to the right, on every positive Load-SDI edge.  
Since we have 1024 pixels, this input shift register is 1024 bits long.



Using `DB`, the ml can store & fetch frame data however it wants to - whatever is convenient!

I was thinking of storing memory in a feat file: each frame would be represented in a  $32 \times 32$  grid of '\*'s (for 1=LED on) & '.'s (for 0=LED off), which would give the user a de facto way of previewing

what their frame would look like in a binary  $32 \times 32$  grid. This would then get parsed by the ml when needed into a 1024 bit serial stream to load into the fpga appropriately. Append 16 bits for how many milliseconds to hold image.

We could then add extra functionality by providing a number before each  $32 \times 32$  frame in this grid that would tell the uC how many continual times it would like to load the frame into the FPLA.

Then at the end of the text file we could say if we wanted the ml to loop back to the top or stop at the end.

This would allow us to play either repeating or non-repeating motivations to our desire!

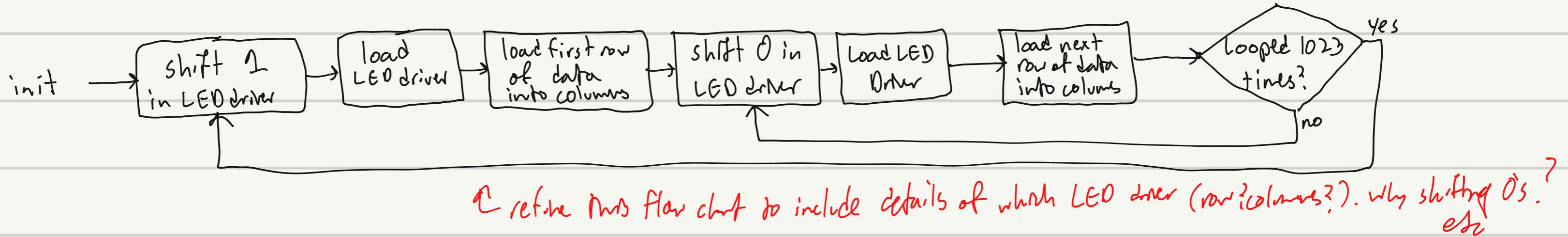
## FPGA to Matrix Control Hardware

This is a bit more complicated.

There is a dedicated wire to each column of the matrix, but the rows are controlled by a shift register LED driver IC that has the benefit of providing easy cheap & compact current limiting functionality.

So the order of operations would be as follows:

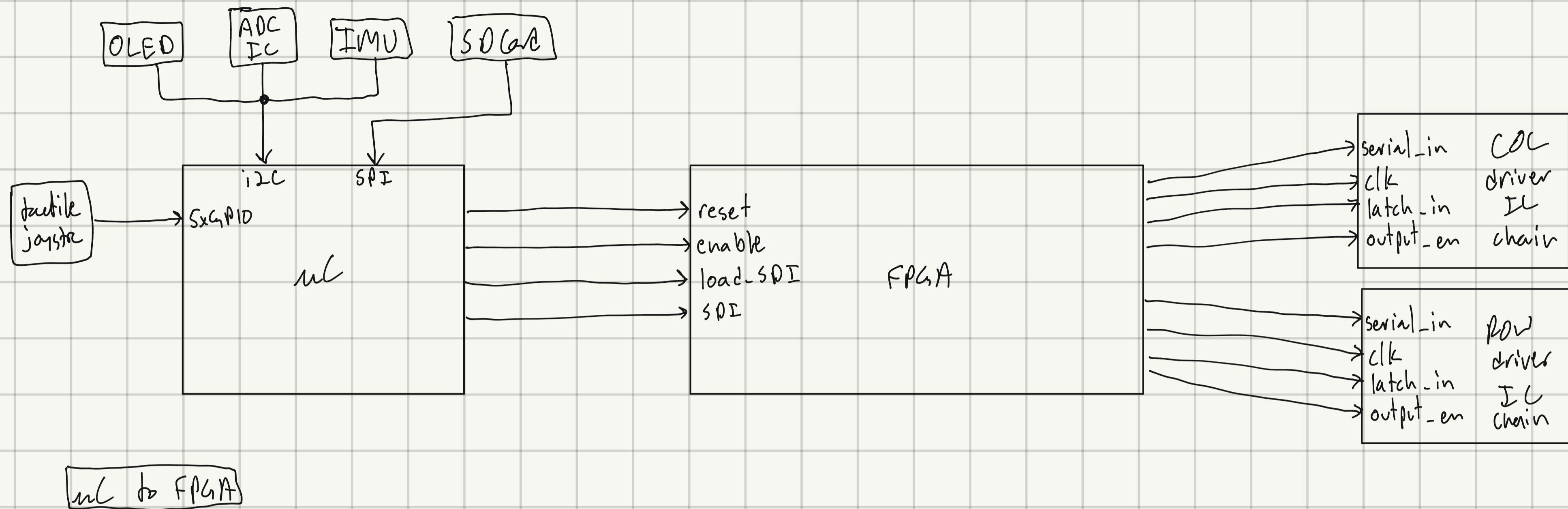
The circuitry indiscriminately & always reads from frame memory from power up or reset:



A more detailed state machine for each of the signals in this part of the FPGA system

will be given later. This state machine would be driven by the FPGA's own internal clock, which for our chosen FPGA board (<sup>Arduino</sup><sub>ATMEL</sub><sup>16000</sup>) is 48MHz. The switching speed cycling of the PMOS's controlling the matrix columns + the LED Driver IC's max clock speed will determine if we can run at 50MHz or if we will have to divide this clock frequency - it is around 20MHz.

# Overall Block Diagram & More Control Details



mc to FPGA

image: on request of new image, enable held low, reset is pulsed, all 1040 bits are pushed into the FPGA via SDI on load\_SDI positive edges, then enable is then pushed back high.

first 16 is how many milliseconds to hold image. doesn't matter cos looped, so set to max to save energy.  
remaining 1024 represent image

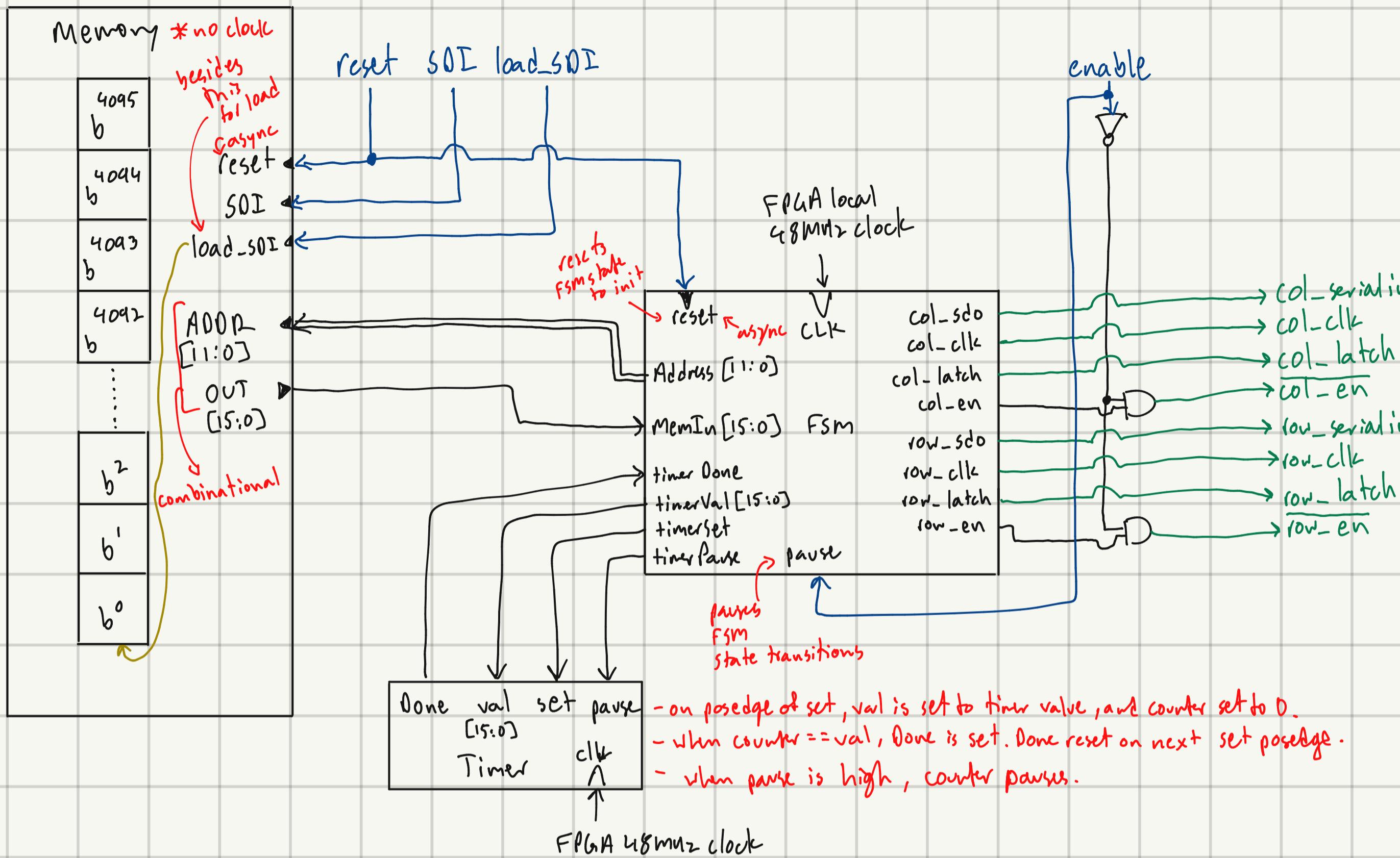
video: same as image, except a multiple of 1040 bits are pushed, representing each frame,

FPGA to Row & Col Drivers

Controlled by finite state machine. FPGA memory organized in 16 bit word sizes. Since each frame is 1040 bits, each frame takes up 65 words.

Our memory will contain  $2^{12} = 4096$  words == 63.0154 frames,  $\therefore 0.0154$  frames (16 bits i.e. 1 word wasted).

Blue for external Inputs Green for external outputs



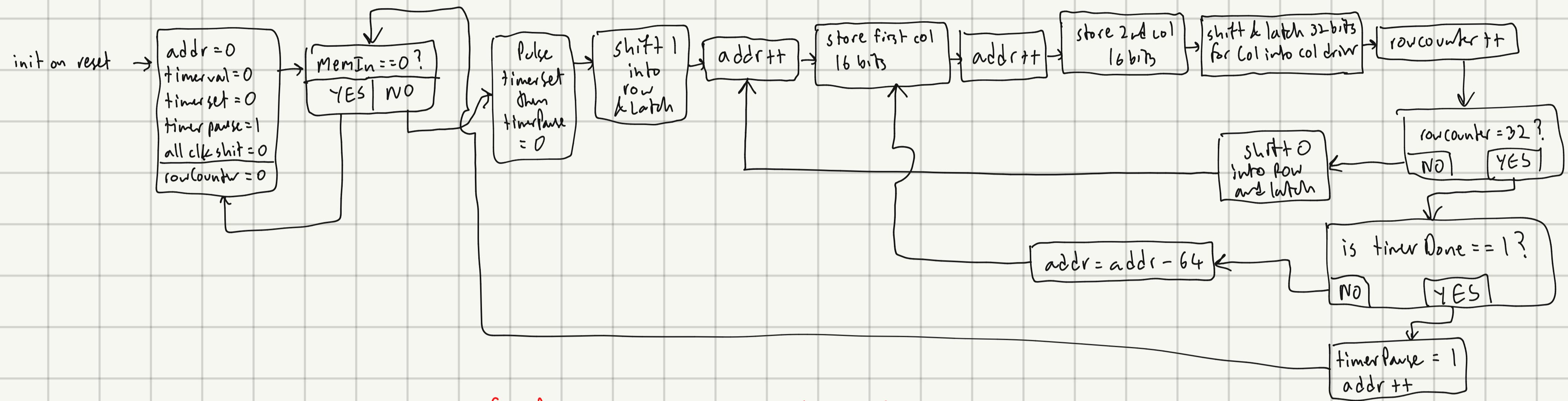
remember memory arrangement!:

- 208,000 bits, in 200 sets of 1040 bits.
- Each set of 1040 bits contains info for each frame.  
 First 16 bits is hold time, remaining 1024 is pixel data, starting at top left corner and shaking down left to right row by row.
- If hold time bits == 0, Then that frame is ignored &

<FSM flowchart on next page>

# FSM control flow chart

Registers for keeping track of stuff: [11:0] addr → for memory access  
[5:0] rowcounter → to keep track of rows



- \* if at any point page input is high,  
don't increment to next state, & other page == high.
- \* each of these blocks are high level descriptions of states. maybe just give the clock lol, asynchronous.  
& back for Area but who cares.