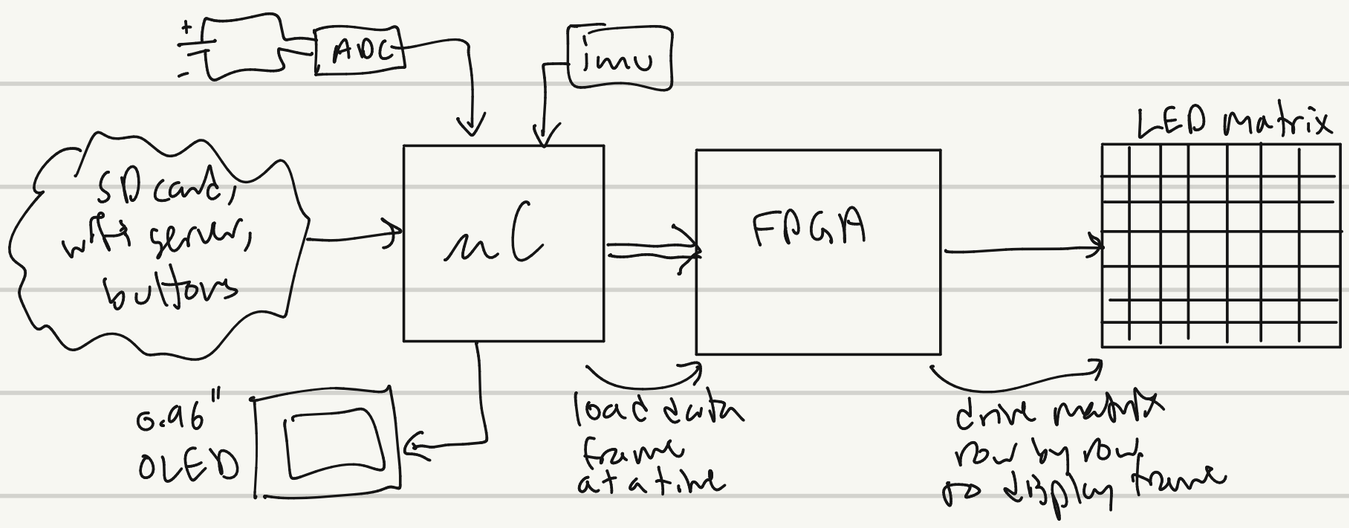
June 10 2021

Fancy Grad Mat 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 $0 \rightarrow x$, $0 \downarrow y$ ∴ ≋ type path.
- Animations are divided into frames.
- High level data path is as follows:



Overall Gist

external cos reading batt voltage > vdd voltage of 3.3v for mC
for cool potential features
to display batt voltage & other useful things
to store frames

The mC provides easy access to peripherals over i2c (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.

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.

uC to FPGA parallel interface

4 wires are used for this interface:

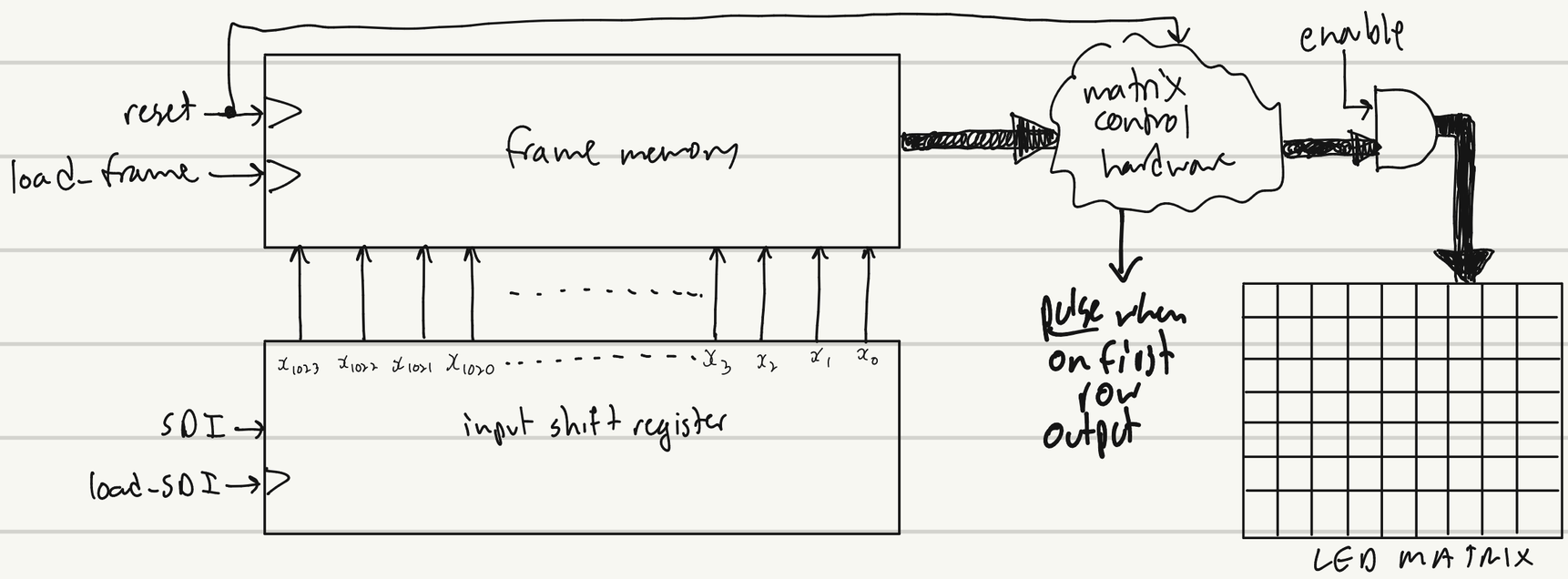
① reset → clears FPGA's frame memory. [positive edge triggered.]

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.

⑤ ~~get~~ load_frame → loads data from input shift register into frame memory, which is where the data that actually gets displayed on the matrix lives. [positive edge triggered]



Using this, the uC can store & fetch frame data however it wants to - whatever is convenient!

I was thinking of storing memory in a text file: each frame would be represented in a 32×32 grid of '*'s (for 1 = LED on) & '.'s (for 0 = LED off), which would give the user a defacto way of previewing what their frame would look like in a binary 32×32 grid. This would then get parsed by the uC when needed into a 1024 bit serial streams to load into the fpga appropriately. Append 16bits for however milliseconds to hold image.

We could then add extra functionality by providing a number before each 32×32 frame in this grid that would tell the uC how many continual time's it would like to load the frame into the FPGA.

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

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

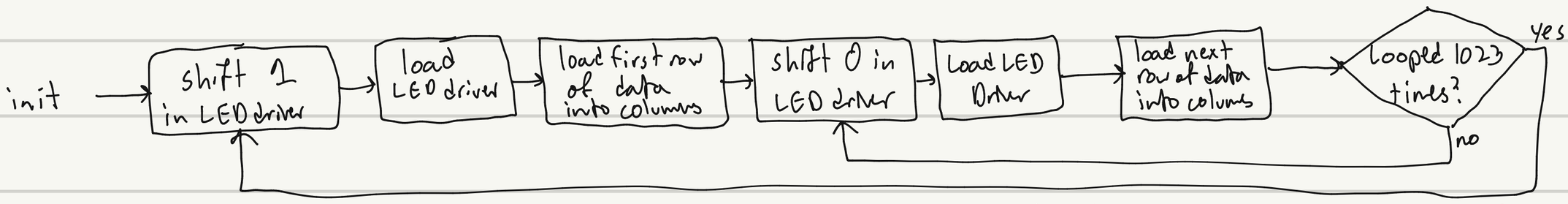
FPGA to Matrix Control Hardware

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:

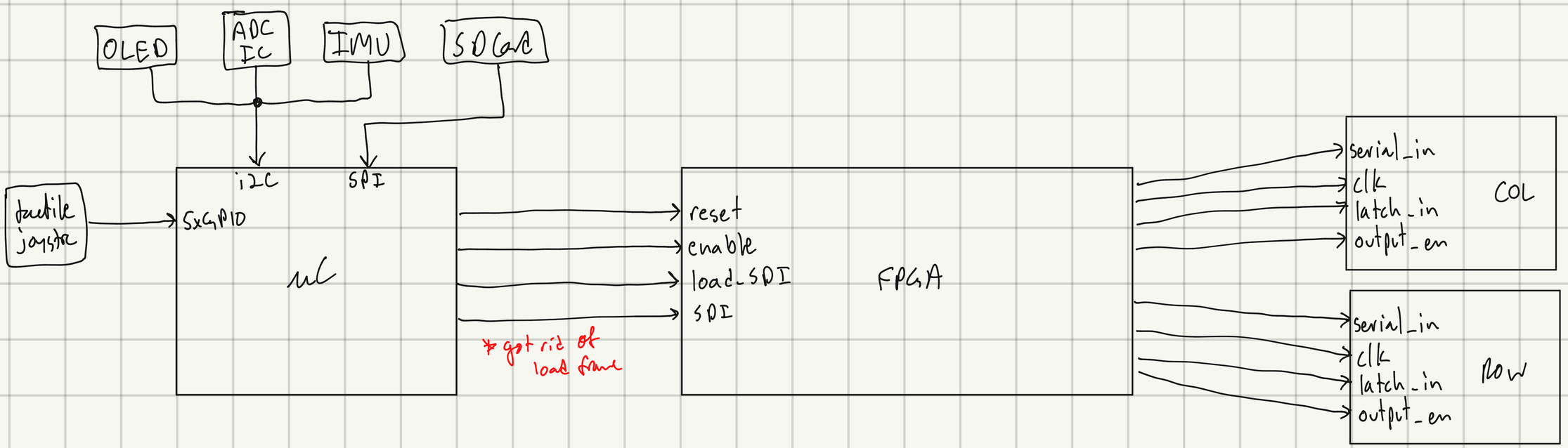
The circuitry indiscriminately & always reads from frame memory from power up or reset:



↑ refine this flow chart to include details of which LED driver (row? column?). why shifting 0's? esp

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 (Arduino MKR Vidor 4000) is 48MHz. The switching speed ceiling 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 Nets



uC 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 as looped, so set to max to save energy. remaining 1024 represent image. If 0 then it means end of memory & will stop reading further & loop back.)

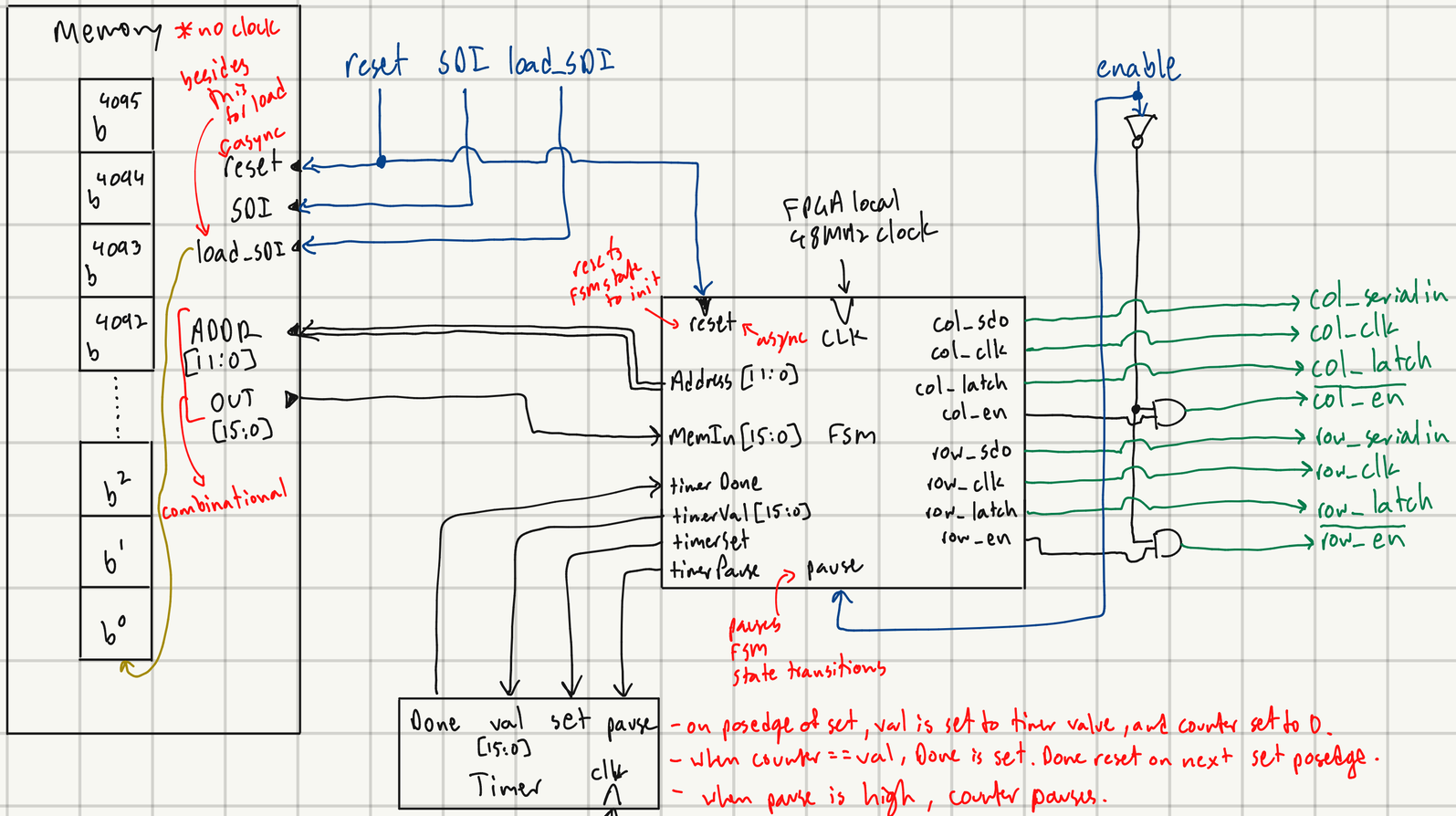
video : Same as image, except a multiple of 1040 bits are pushed, representing each frame,

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, ∴ 0.0154 frames (16 bits i.e. 1 word wasted).

Blue for external Inputs Green for external outputs



remember memory arrangement!:

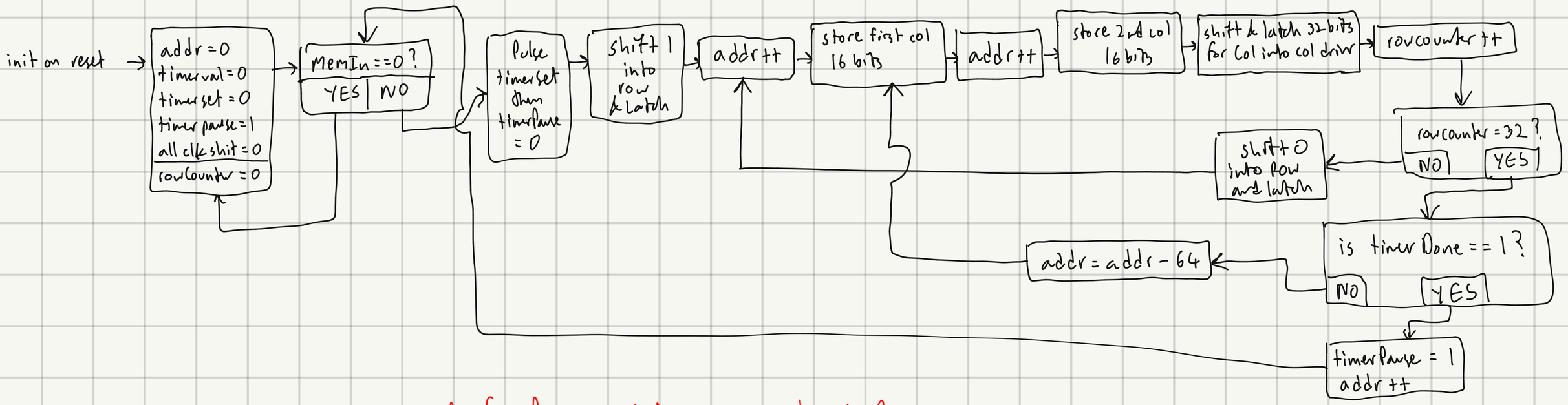
- 208,000 bits, in 200 sets of 1040 bits.
- Each set of 1040 bits contains info for each frame.
  First 16 bits is hold time, remaining 1024 is pixel data, starting at top left corner and snaking down left to right row by row.
- If hold time bits == 0, then that frame is ignored &

<FSM flow chart on next page>

FSM control flow chart

registers for keeping track of shift : [11:0] addr → for memory access

[5:0] rowCounter → to keep track of rows



* if at any point pause input is high, don't increment to next state, & timer Pause == high.

* each of these blocks are high level descriptions of states. maybe just gate the clock to 1, asynchronous. & bad for area but who cares.