Concurrency 1

Michael Li - ml1811@ic.ac.uk

1 FSM Design

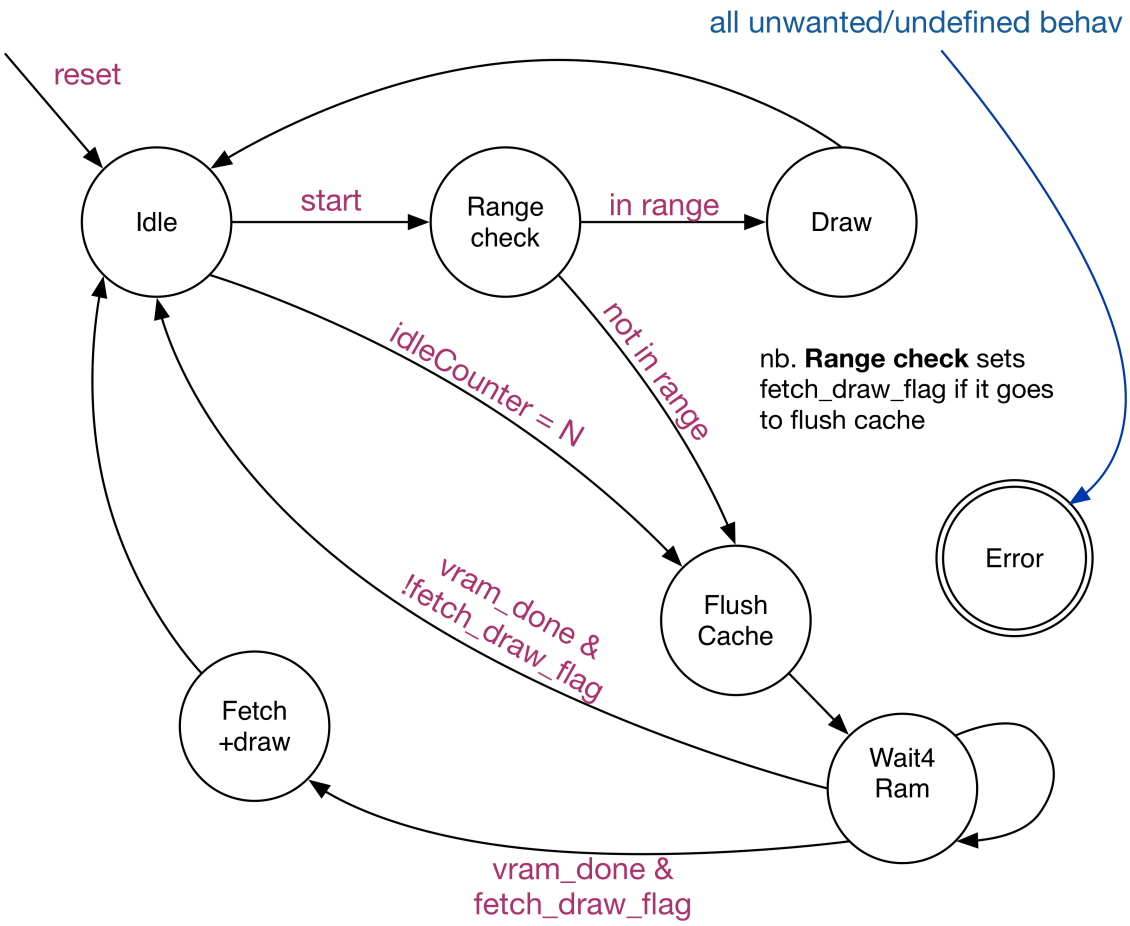

Figure 1: This figure is slightly outdated, an extra state has been added. see shitty drawn diagram in fb chat

The purpose of this block is to accept the various commands from upstream and then write the changes into vram. It is however more efficient to write the entire word into vram in one go, so we will cache all the changes made to the same word and only when we are given a command to draw in a different word do we write out the changes into vram. This block uses the existing px word cache and ram fsm. Ram fsm will need to be modified to take an additional input (from the cache) as well as some combinatorial logic to then merge the changes saved in the cache, with what is in vram. This writing out to ram takes 3 cycles to do and we need to stall the controlling fsm till this is complete.

FSM fires control signals denoted _trig, any flags are denoted _ and are registered such that they persist across state changes. Processes corrseponding to the trigger signals are all conditionally executed based on the respective trigger signal being asserted by enclosing the entire body in a giant IF.

2 FSM Operation

FSM has states.
`s_error` : trap state to help debugging. can only leave with a reset command

`s_idle` : reset initialises FSM to this state.
IF idleCounter = N, goto `s_flush`
Otherwise stay in same state untill startcmd.
`s_checkrange` : checks to see if target pixel is in current loaded word
if out of range goto `s_flush`
otherwise decode and goto either draw or clear.

`s_draw` : check for move or draw and signal releveant process to start
drawdone at end lets it transition
is it necessary?
probably an idea as draw will always complete in one cycle, that we can have more logic that reads in a new sample while we are in draw, as we know it will finish in one cycle, by the time we could fetch and decode the next instruction, some kinda of pipelining?

`s_clear` :to be implemented, currently goes to idle

`s_flush` :triggers flush process with `flush_trig`
next transition based on which state it came from if vram done writing
stored in the registered fetchdraw flag which is high if it came from rangecheck, instead of idle goes to fetchdraw if flag is high, otherwise idle.
otherwise goes to waitram state
`s_fetch_draw` : triggers the fetchdraw process with control signal fetchdraw.

`s_waitram` : hacky fix, coudlnt figure out the control logic to persist. If I just loop inside `s_flush`, i'll keep asserting the flush_trig signal, and keep running that process, which I Dont wanna do, I just wanna wait till the vram write is done.

RESET will force the FSM back to the idle state on the next clock edge, default state transition is to error to help debugging. All intentional transitions should be explicitly specified.

3 Processes

3.1 state_transition

Combinatorial transition matrix for fsm. Computes the next state to transition to and asserts control signals to fire processes.When not in this state, dbb_delaycmd is high.
Driven outputs: next_state, idle_counter_trig, draw_trig, move_trig, flush_trig, fetch_draw_trig, dbb_delaycmd

3.2 fsm_clocked_process

Clocked part of fsm for transitions and registered flags.
Driven outputs: rcb_finish, fetch_draw_flag

3.3 idle_counter

Handles the counter to store the number of idle cycles elapsed.
Driven outputs:idle_counter.

3.4 current_word_register

Stores the current word address loaded into cache, so we can check if the new command is in this range.
Driven outputs: curr_vram_word.

3.5 draw_px

Does the draw single pixel, if in range. Sets the address for pixel cache and enables single write by setting pw =1, wenall=0 and pixopin to the correct value based on the draw command.
Driven outputs: pxcache_pxopin, pxcache_pixnum, pxcache_wen_all, pxcache_pw.

3.6 fetch_draw

Does draw and clears out the cache. The duplicated draw colour decoding could be optimised perhaps with a full decode stage that will store all the relevant flags and commands in a set of registers could be more efficient or at least prevent code duplciation? Can never be in both draw state and fetch_draw at the same time, so ok to drive same signals?? OK NO This is bad, I need to multiplex these I guess. In progress
Driven outputs: pxcache_pxopin, pxcache_pixnum, pxcache_wen_all, pxcache_pw.

3.7 flush_cache

Writes out the changes in the pixel cache to vram. This process triggers the ram_fsm entity to begin the interfacing. I will need to modify ram_fsm to take an addition input, connecting the output of px_cache to this new input of ram_fsm. Add a new combinatorial process that will apply the changes from cache into what was read from memory by combining vdout with store_ram. If the value in cache is a explicit colour, I write that to vram, otherwise I either invert or leave the same. If the if_same signal is asserted, then I have no need to write to vram as there have been no changes made in the cache.

Driven outputs: ram_addr, ram_start.

4 Functions

4.1 getRamWord

This function takes in an x and y coordinate from dbb_bus and computes the corresponding word address containing that pixel specified returning it as an 8 bit vector. This value is then used to check if it will be a ”cache hit” and also passed to the ramfsm.

4.2 getRamBit

This function takes in an x and y coordinate from dbb_bus and computes the corresponding bit number in the word that is then used to address the pixel cache.

5 Other notes

rcb_finish asserted inside clocked fsm proc

There is scope for potential parallelisation of commands. Seeing as it takes 3 clock cycles to complete a standard draw due to the state transitions, we could pipeline the instruction decode in some way to take advantage of that as long as we're not in the cache flush stage.