

FPGA Serial Mem Tester, Version 1

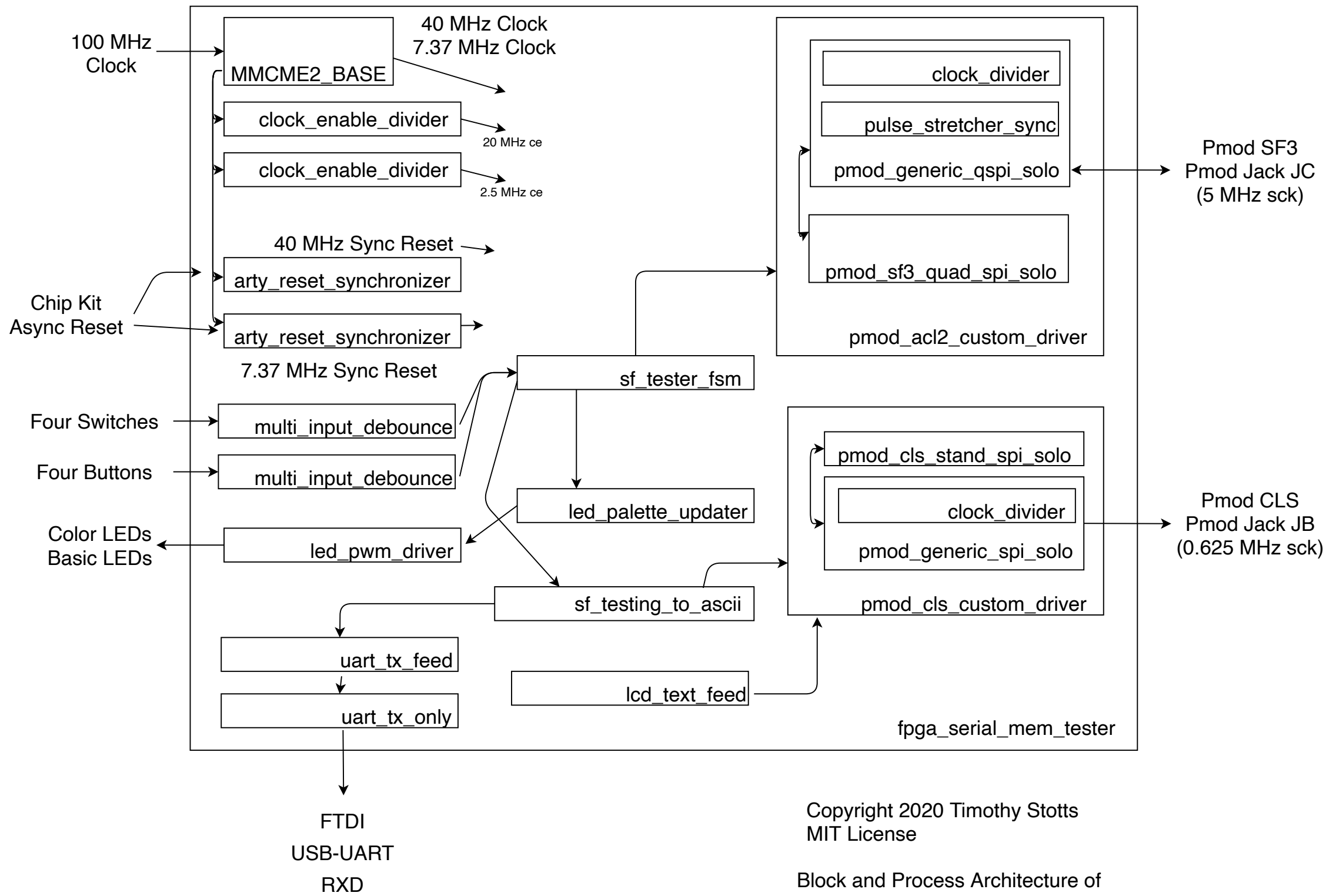
by Timothy Stotts

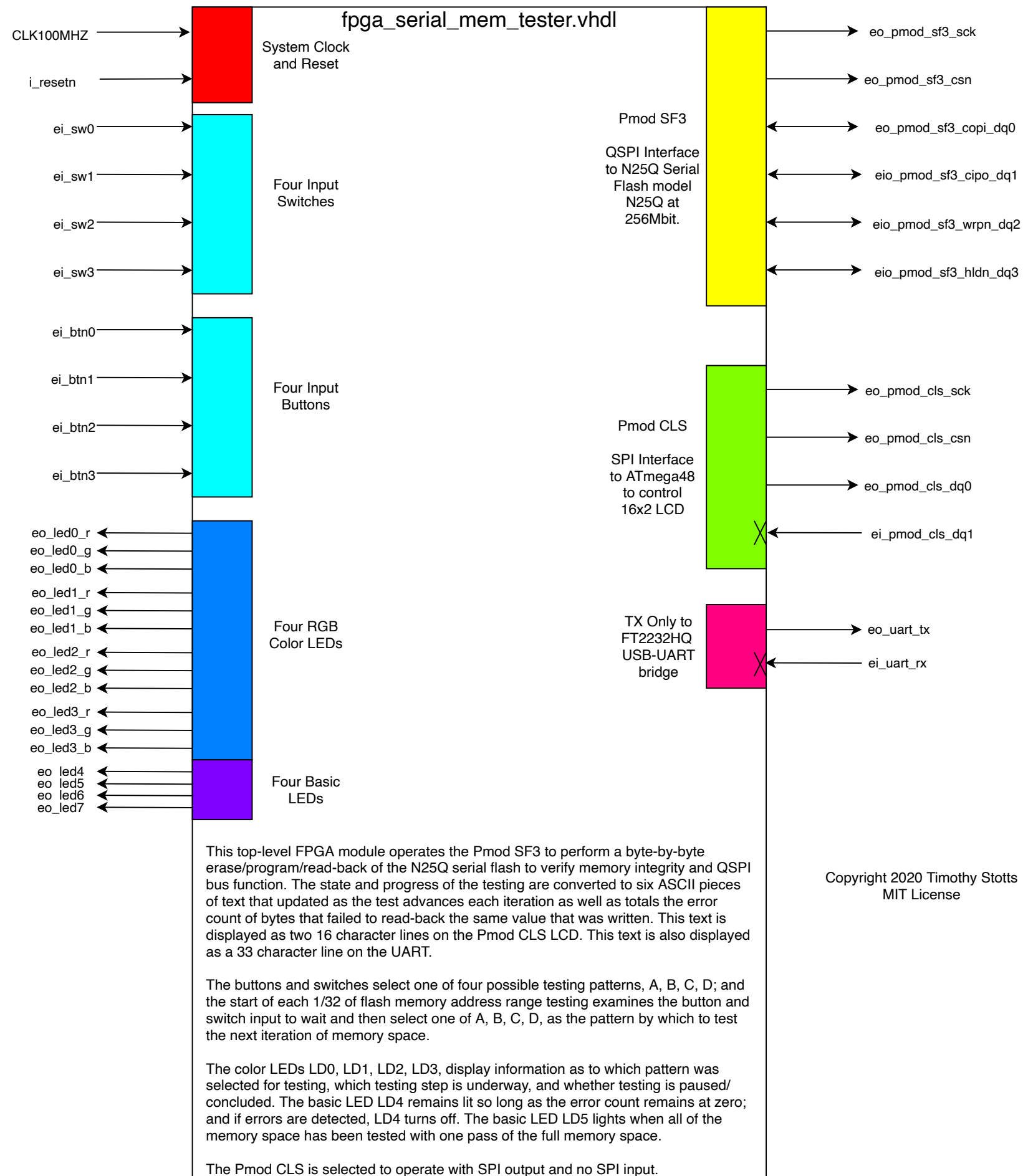
Copyright 2020 Timothy Stotts
MIT License

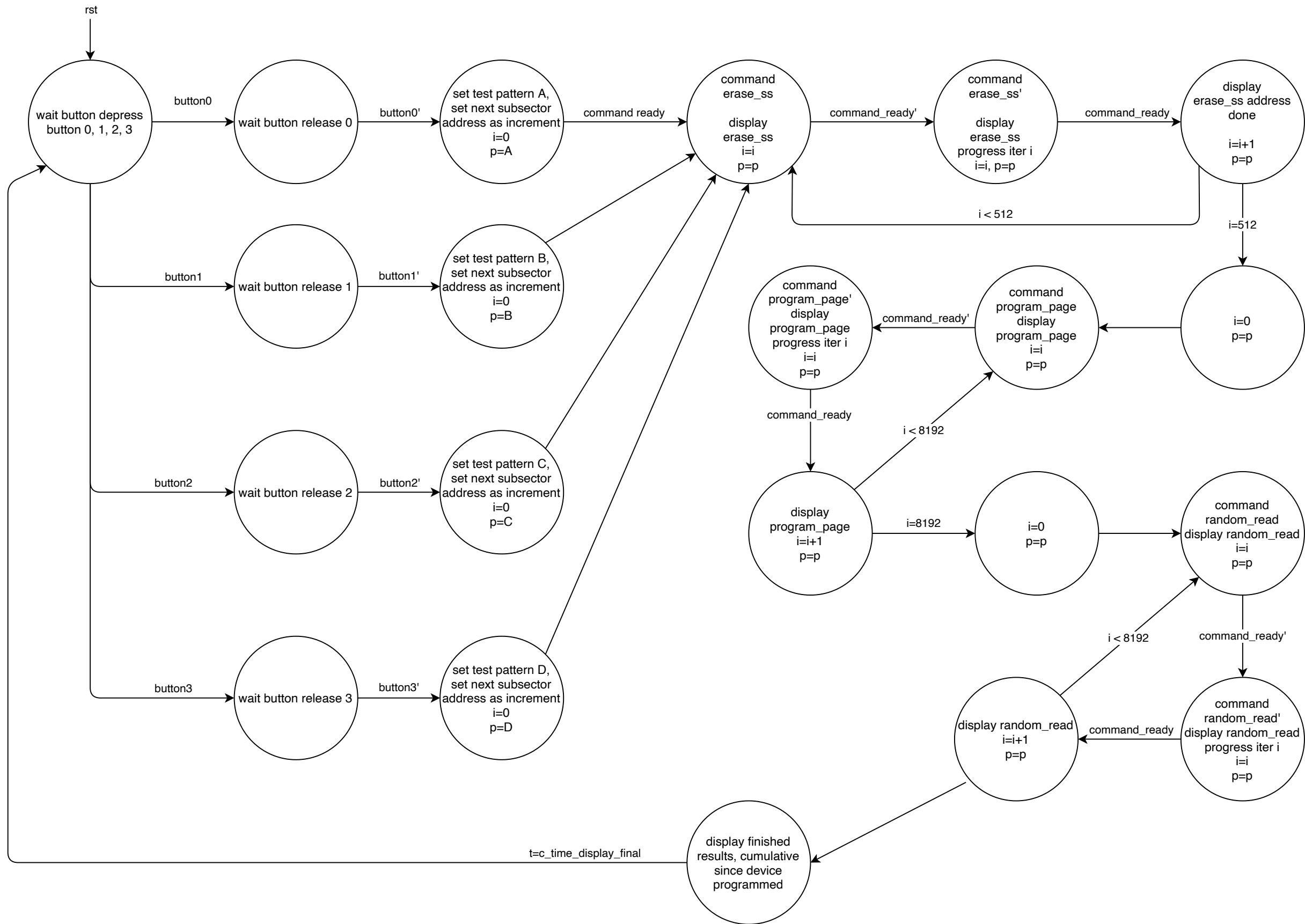
Hosted at:

<https://github.com/timothystotts/fpga-serial-mem-tester-1>

SF-Tester-Design-Diagrams document revision 7A







A top-level tool design to provide a FSM with UI to control pattern write/read testing of PMOD SF3.
With this FSM, there are 32 groups of 32 sectors a piece.

Refer to project source code for more details of the FSM.
This drawing is a simplified conceptual representation of a more complex machine.

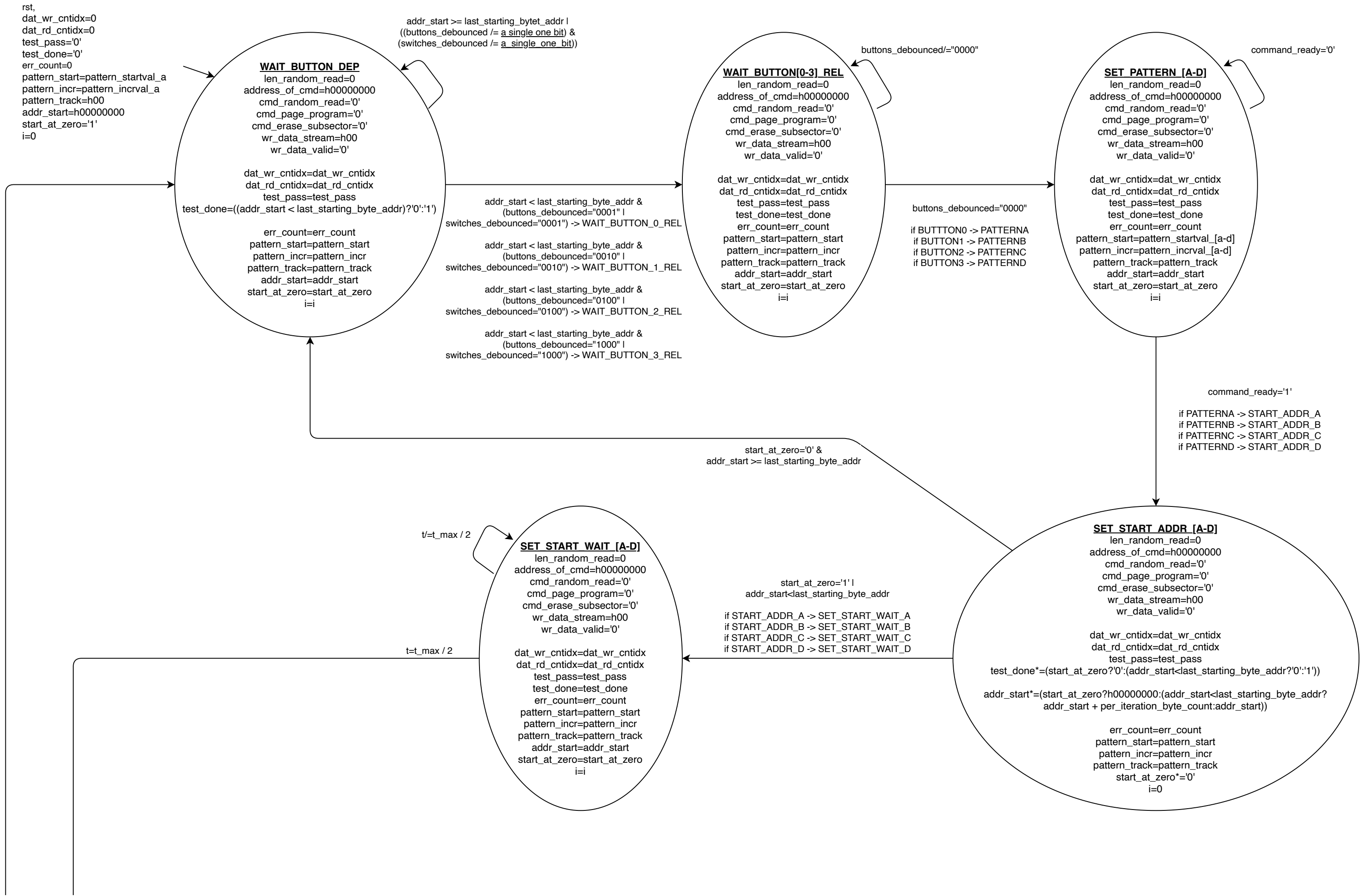
Display 16x2 :

SF3 PA hADDRESS_
ERS ERR 67108864

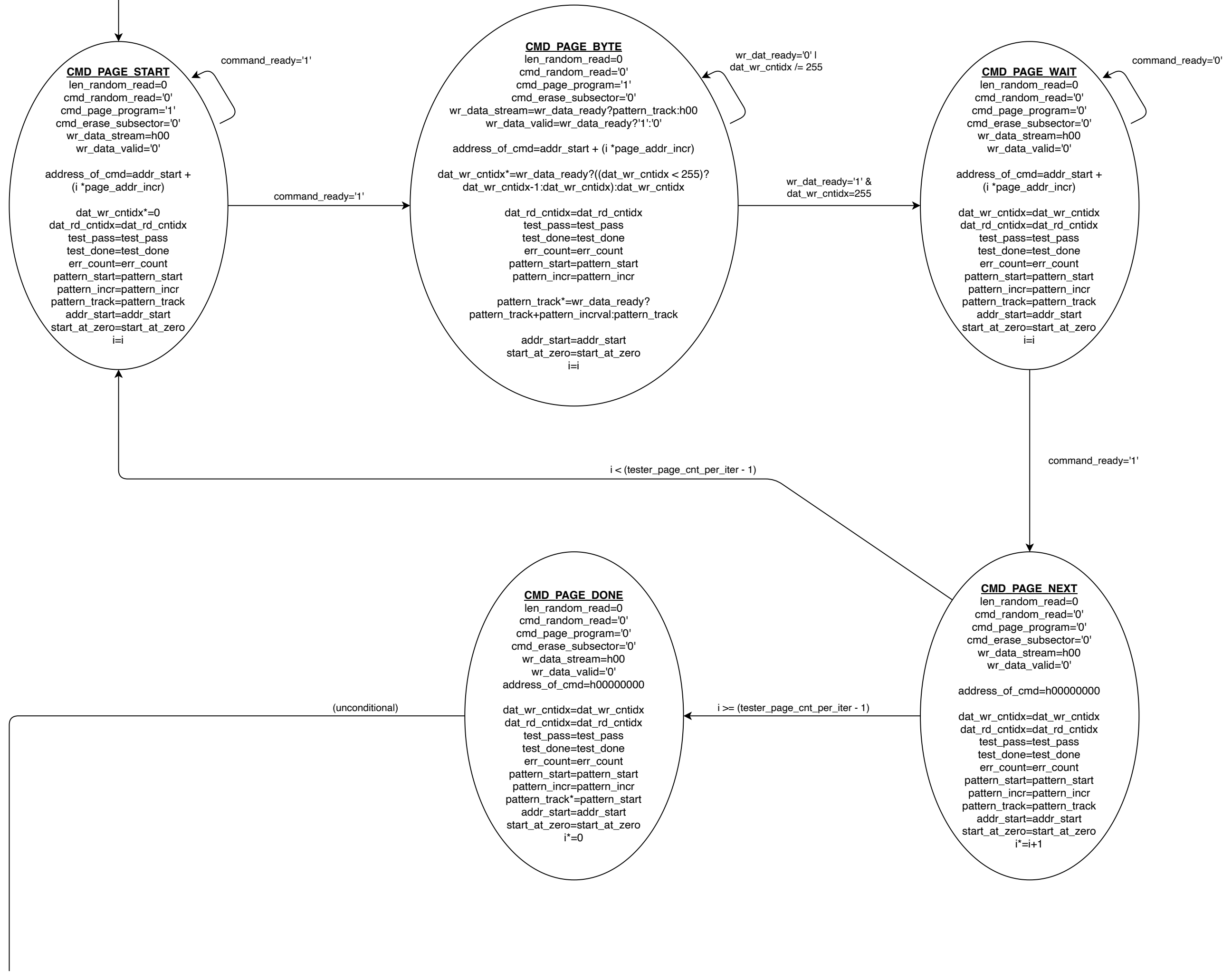
SF3 is SF3
PA = P* or PA or PB or PC or PD
hADDRESS = h00100000
ERS = GO_ or ERS or PRO or TST or END
ERR is ERR
67108864 is decimal in range 00000000 to 67108864

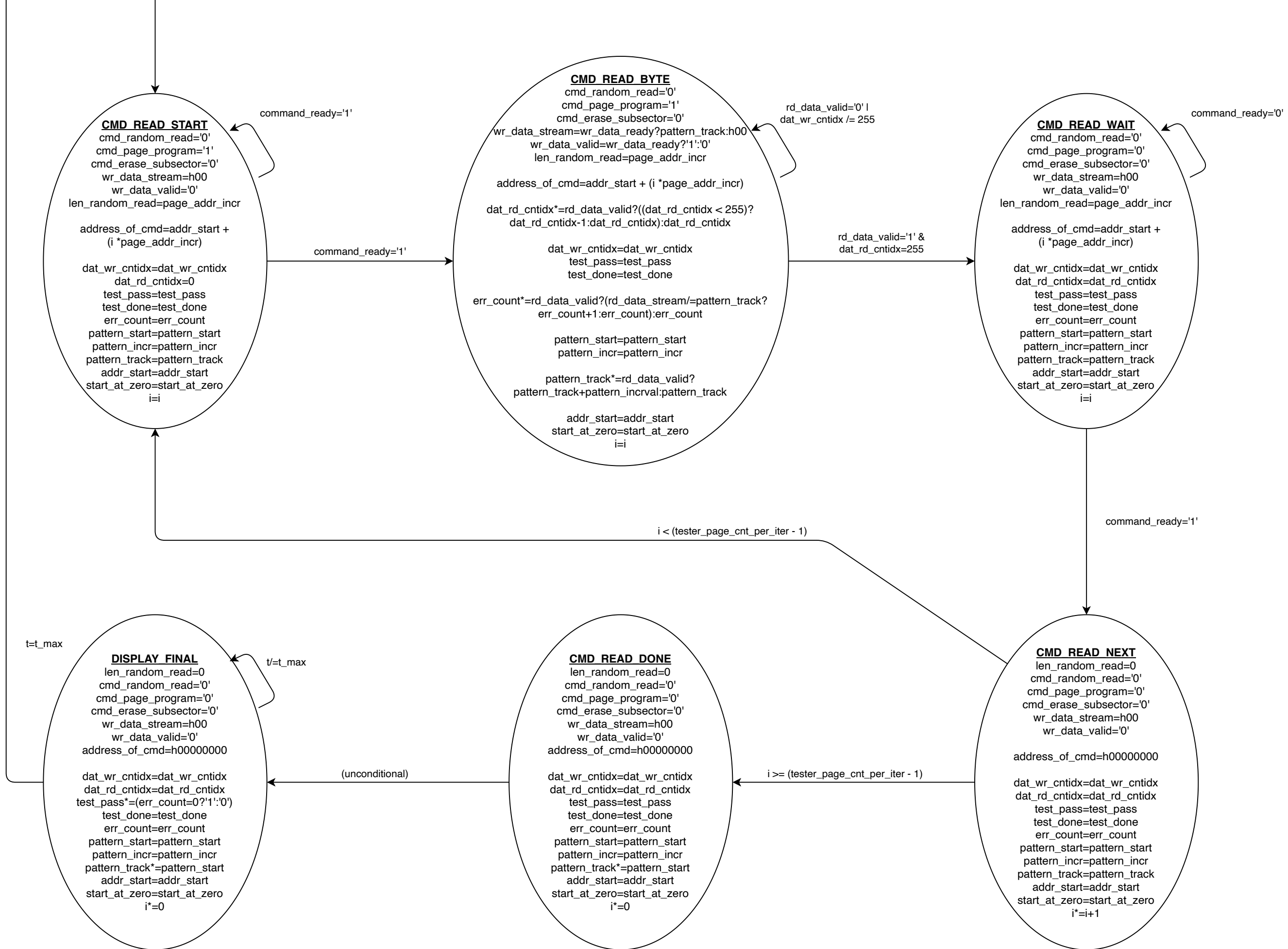
Copyright 2020 Timothy Stotts

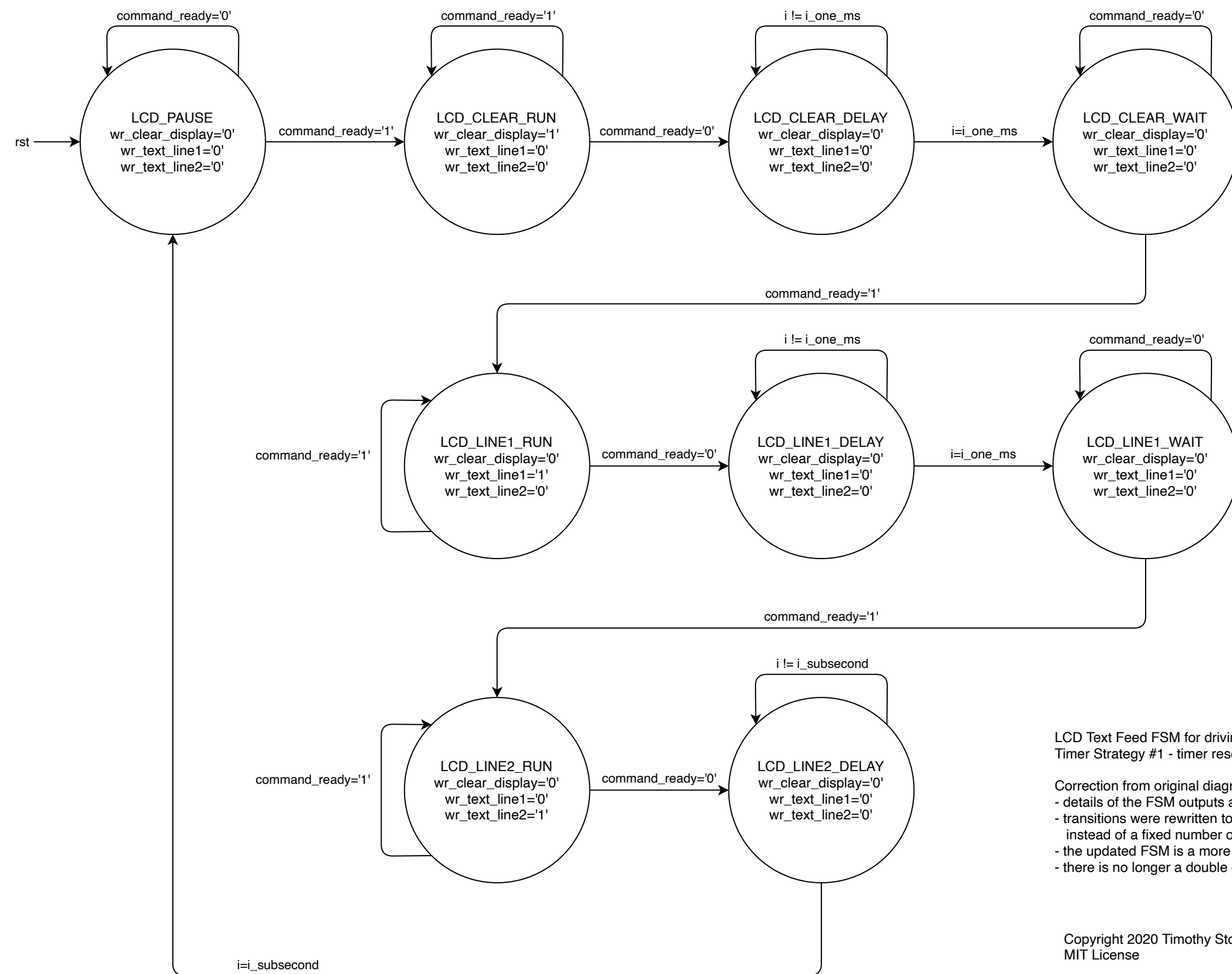
MIT License. Refer to LICENSE
file included with this software.







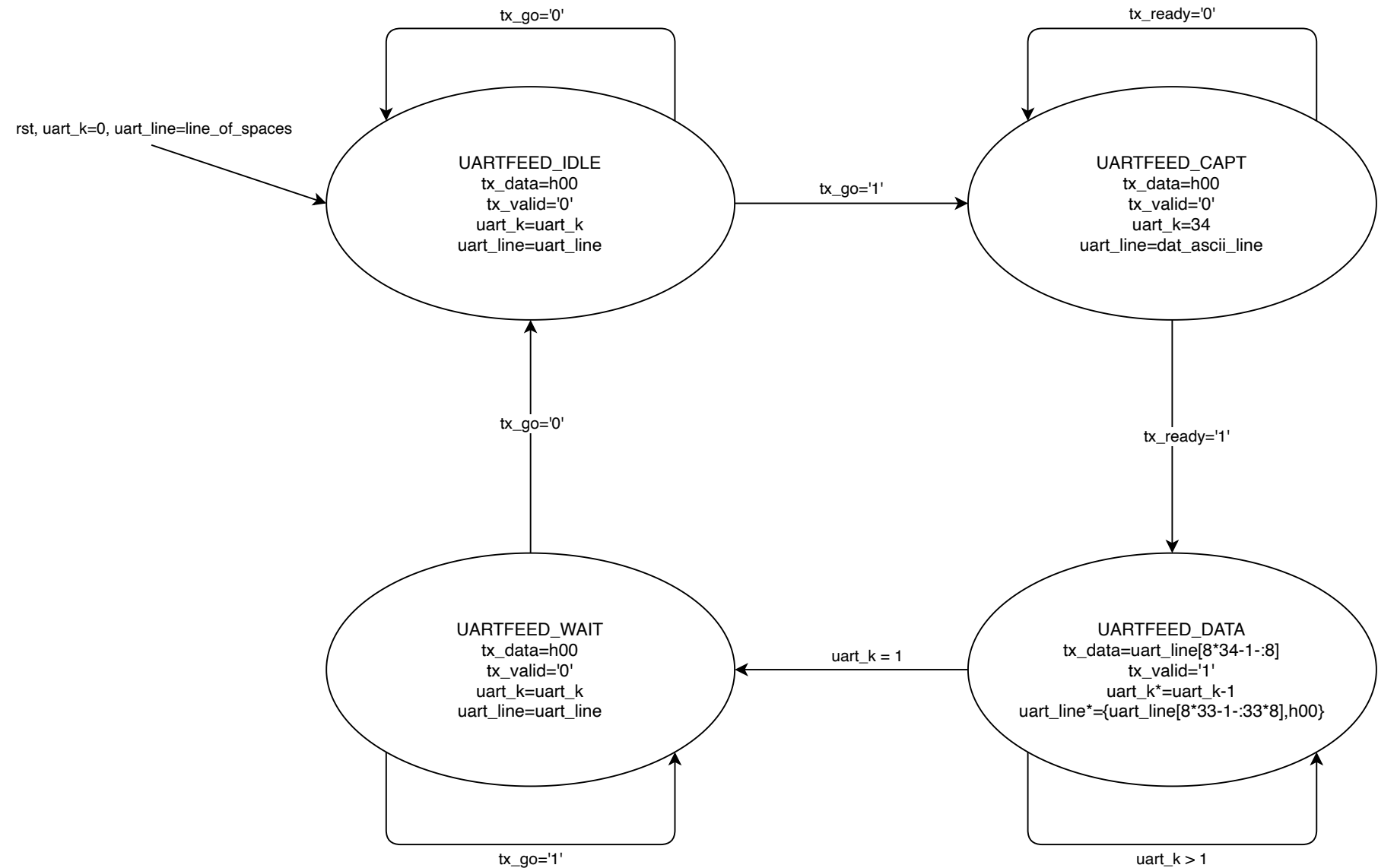




LCD Text Feed FSM for driving the commands of the custom Pmod CLS driver.
 Timer Strategy #1 - timer resets on state change and runs from zero to `i_subsecond` .

Correction from original diagram:
 - details of the FSM outputs and timer "i"
 - transitions were rewritten to hold commands according to `command_ready` handshake instead of a fixed number of clock cycles during a state
 - the updated FSM is a more proper FSM diagram design
 - there is no longer a double delay between the end of Line 2 and the start of Clear.

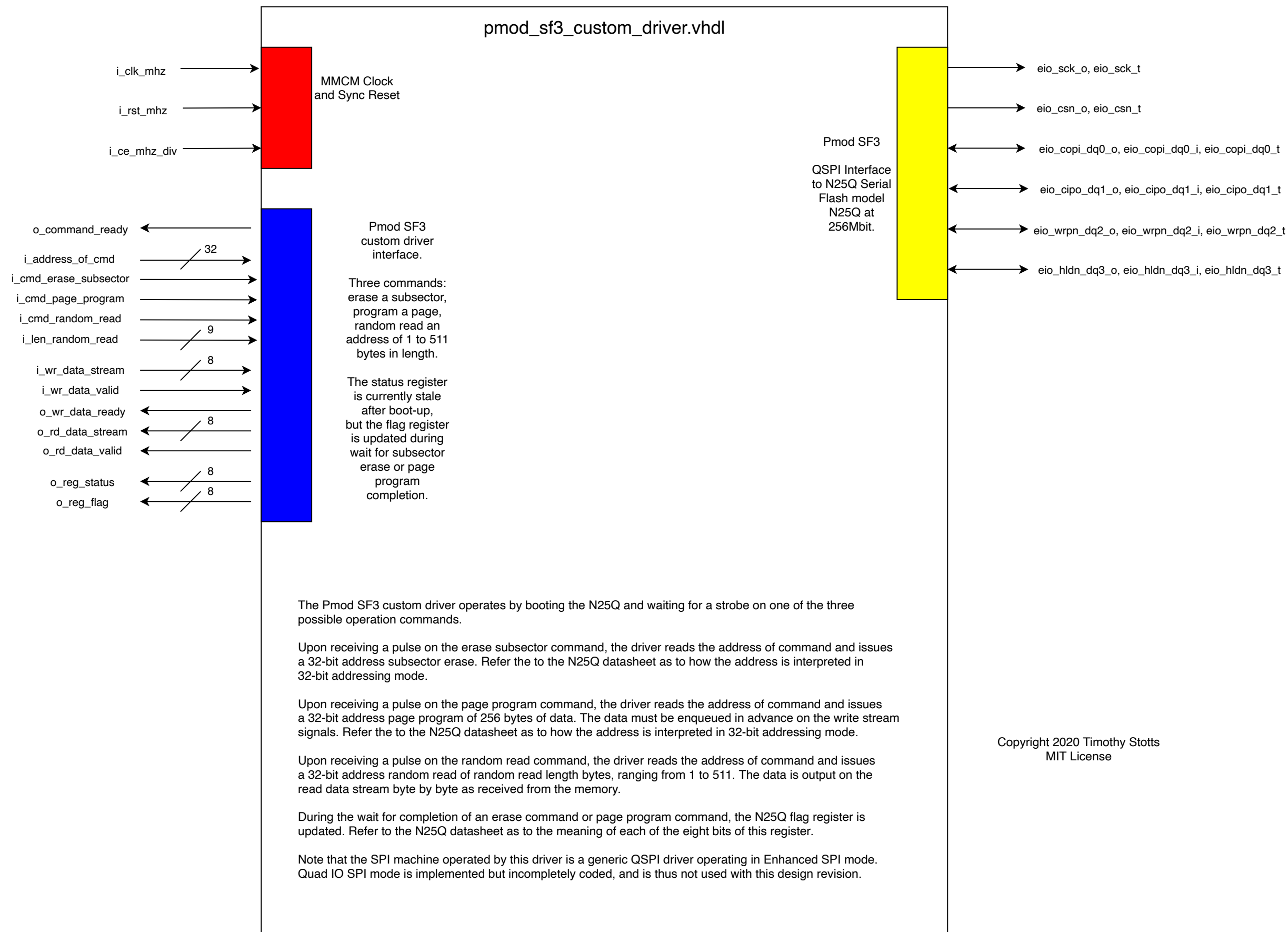
Copyright 2020 Timothy Stotts
 MIT License

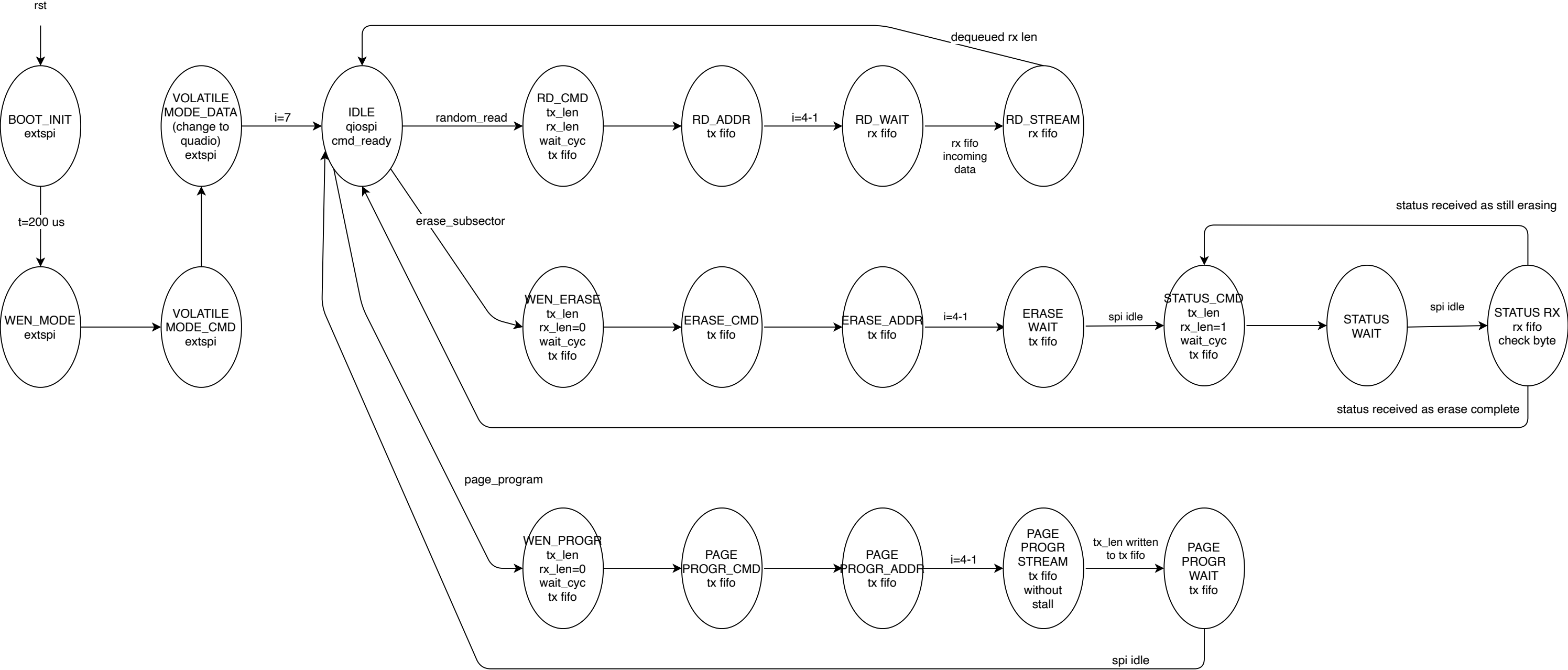


UART TX Feed FSM

This FSM feeds the TX FIFO of the `uart_tx_only` module. The data to feed to the TX FIFO is always a 34 8-bit character line of ASCII text. The `tx_go` input is triggered by the corresponding `wr_clear_display` pulse on the Pmod CLS custom driver, such that the UART TX Feed occurs when the LCD is starting to update on the FSM cycle of that driver.

Copyright 2020 Timothy Stotts
MIT License





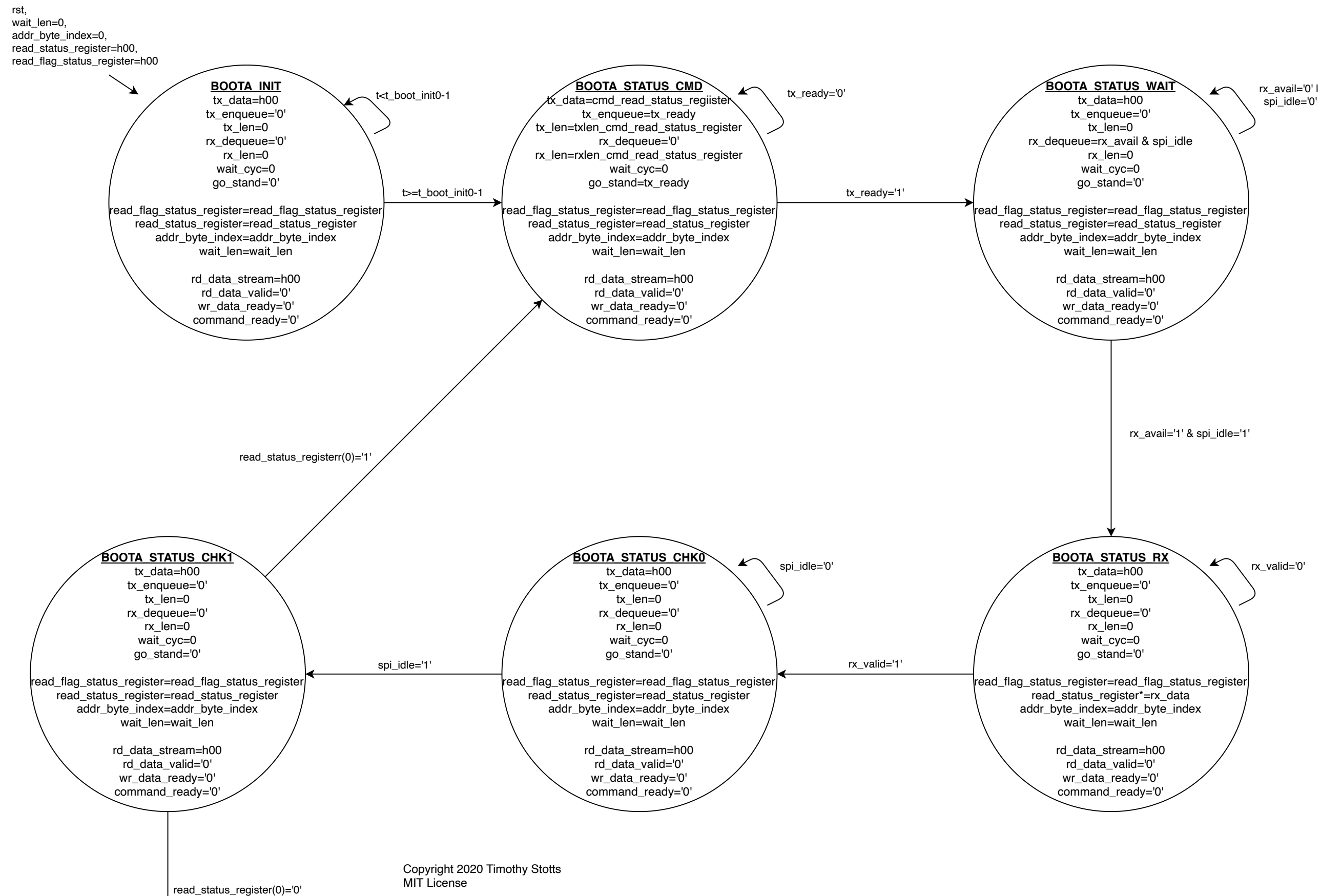
In each transition, tx_len and rx_len are to be multiplied by 8 from the FSM input signals, as it only makes sense to input into the FSM a byte count, while the FSM requires transitioning based upon a bit count.

N25Q chip FSM for communicating with the SPI Machine, with the N25Q as only SPI slave on the bus.

Refer to project source code for more details of the FSM.
This drawing is a simplified conceptual representation of a more complex machine.

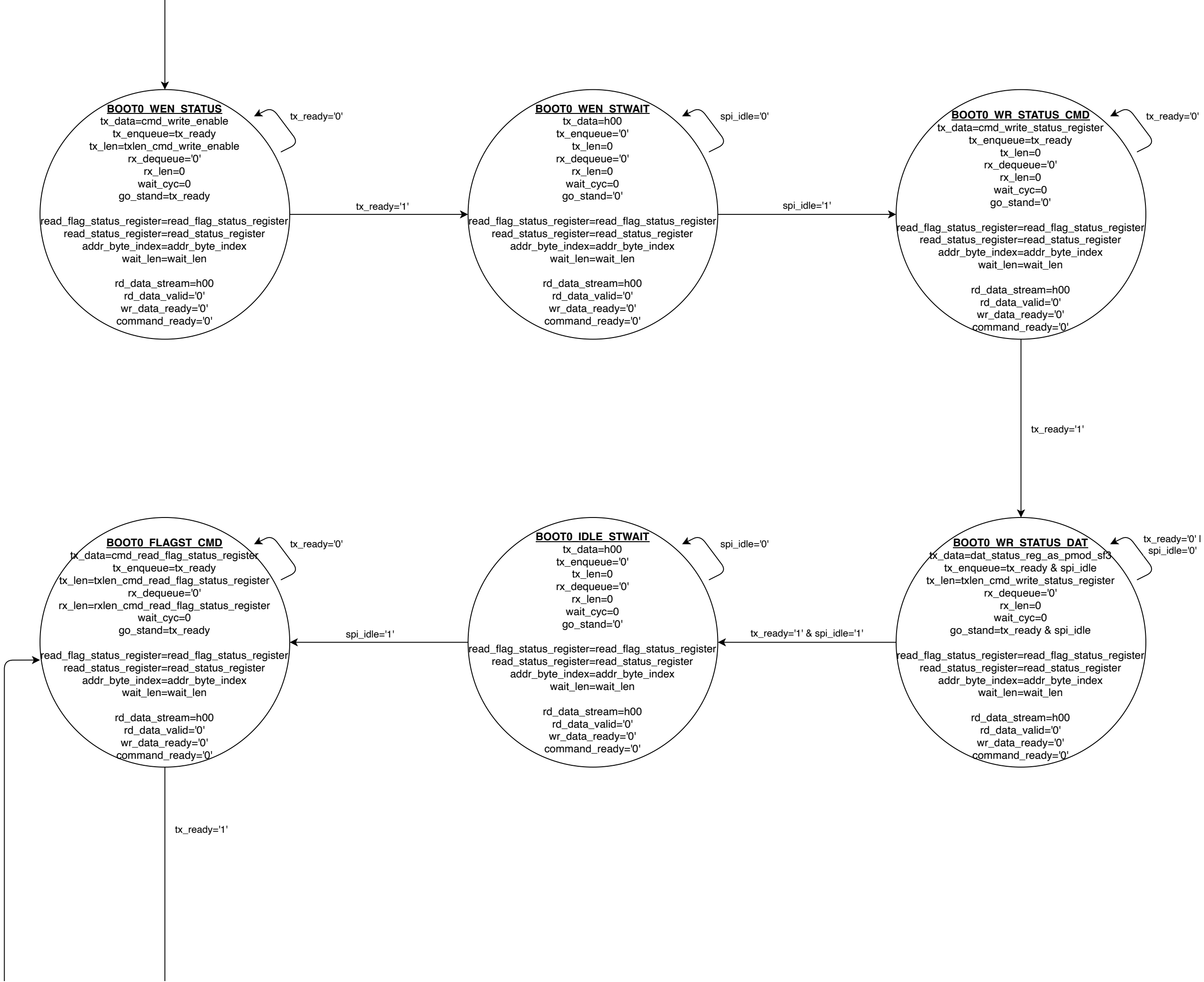
Copyright 2019 Timothy Stotts

MIT License. Refer to LICENSE
file included with this software.

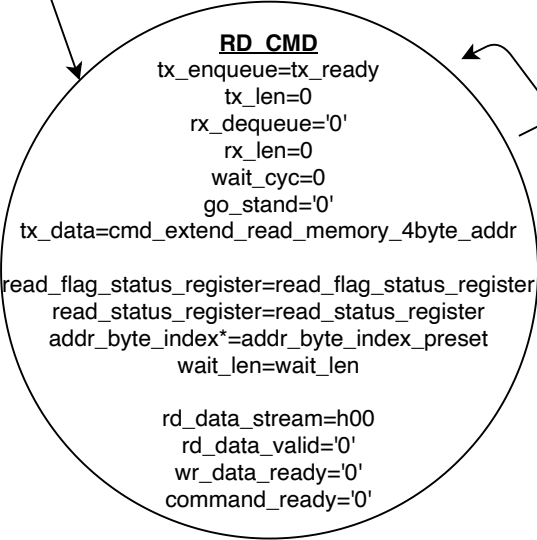
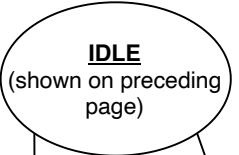


SF3 Extended SPI Driver of the SF3 Custom Driver. The FSM works by waiting for a command to (a) random read an address, (b) erase an addressed subsector, or (c) program an addressed page.

The first group of Moore machine outputs in each state bubble are FSM outputs, some of them with combinatorial output that is normally only seen in a Mealy machine. (This FSM could be said to be a hybrid Moore and Mealy machine.) The second group of outputs are recursive auxiliary outputs that retain state from one state to the next when assigned to itself. The third group of outputs are FSM outputs in same fashion as the first group.

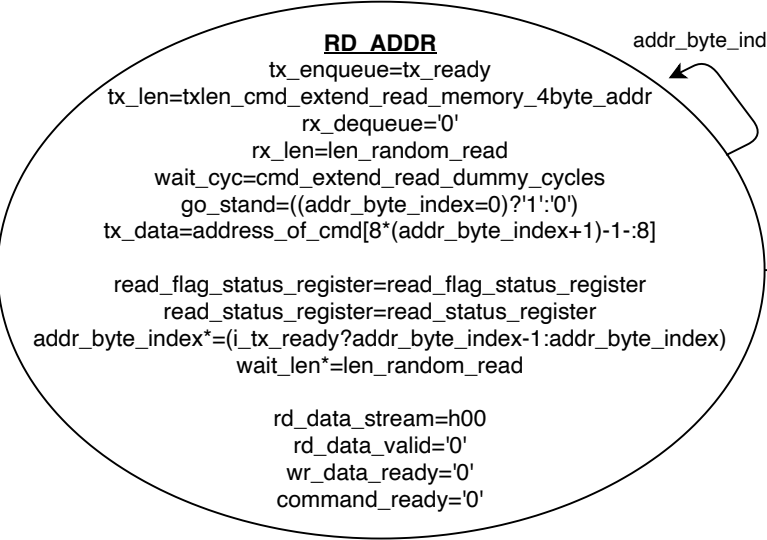






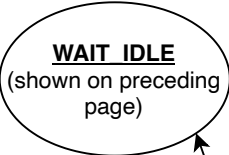
tx_ready='0'

tx_ready='1'

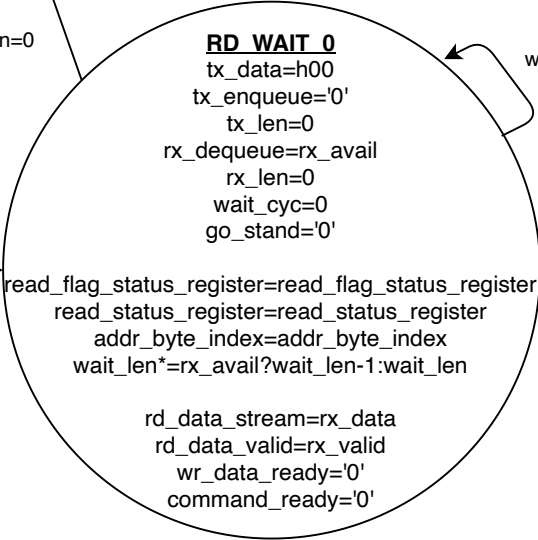


addr_byte_index/=0

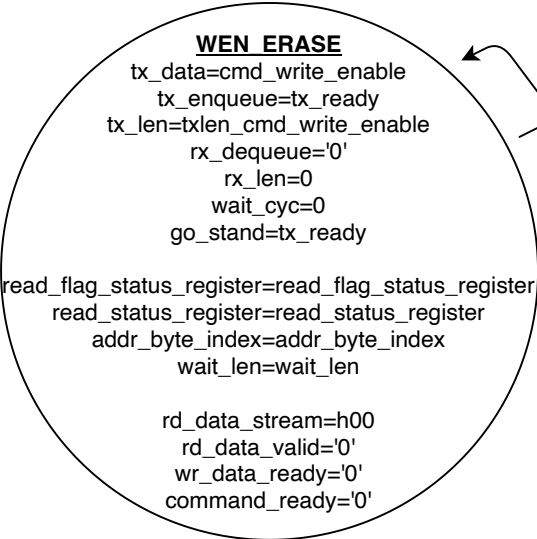
addr_byte_index=0



wait_len=0

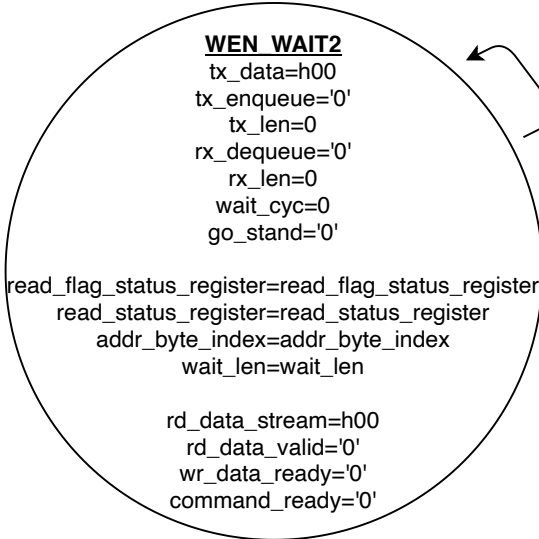


wait_len/=0



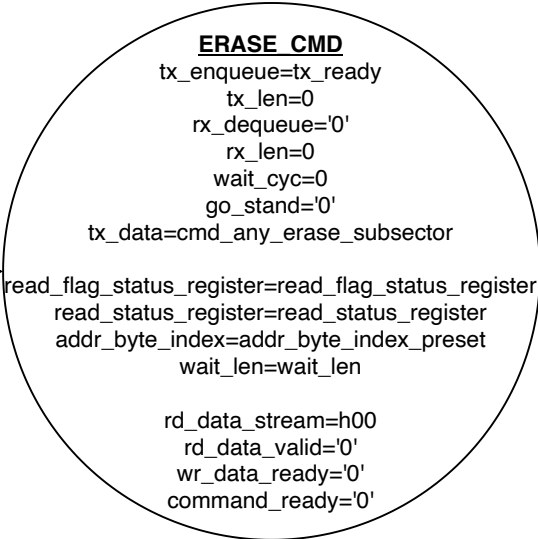
tx_ready='0'

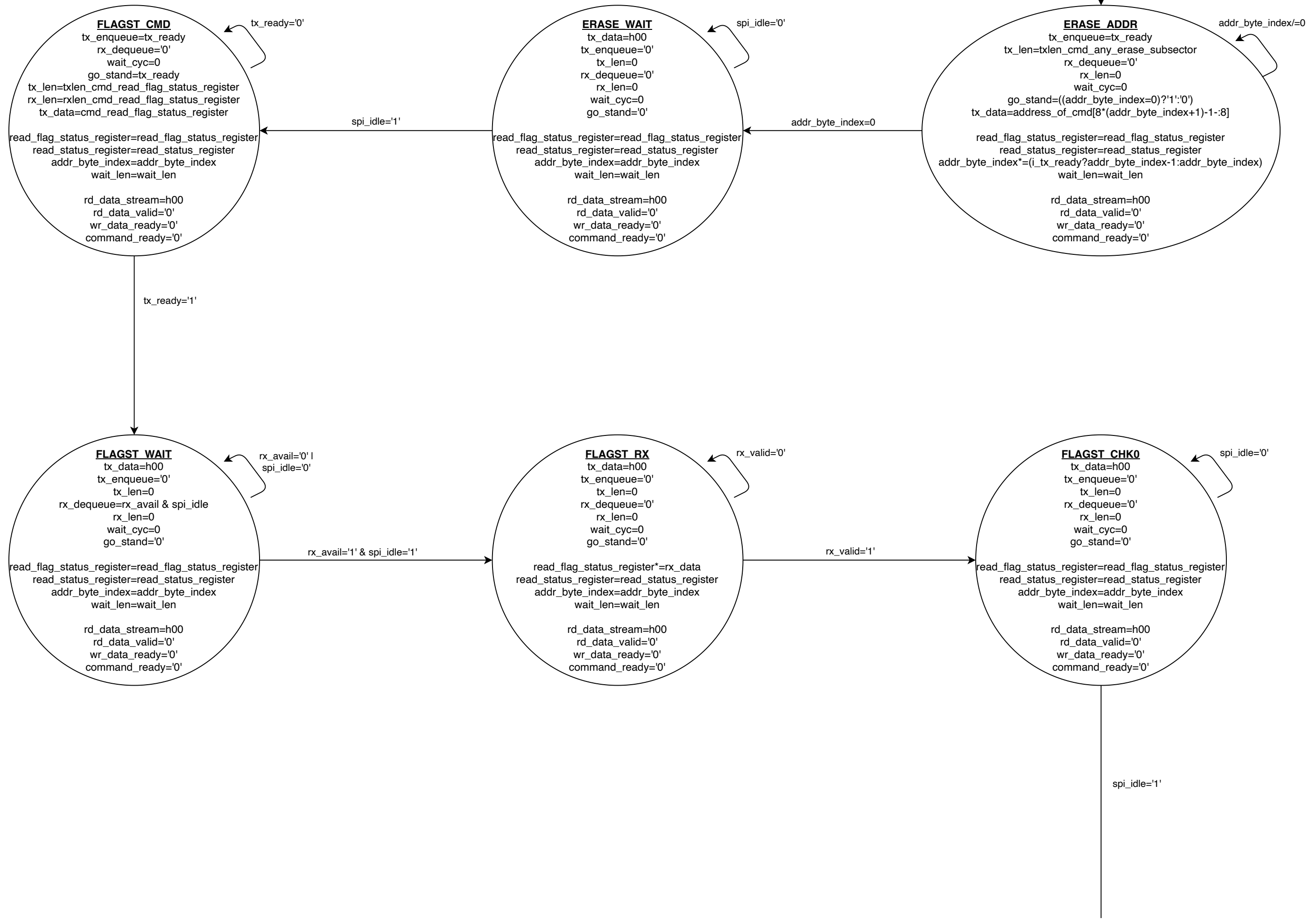
tx_ready='1'

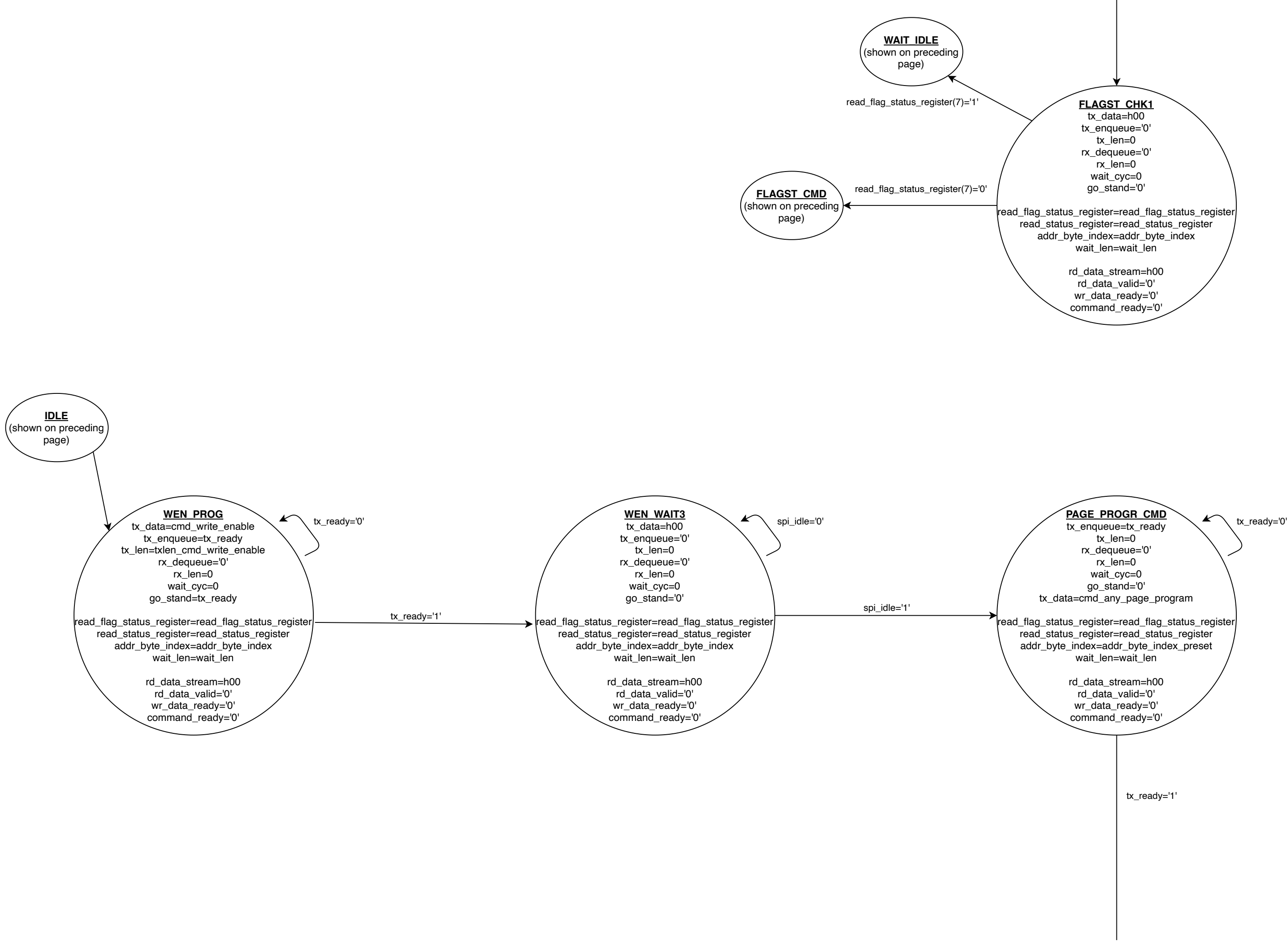


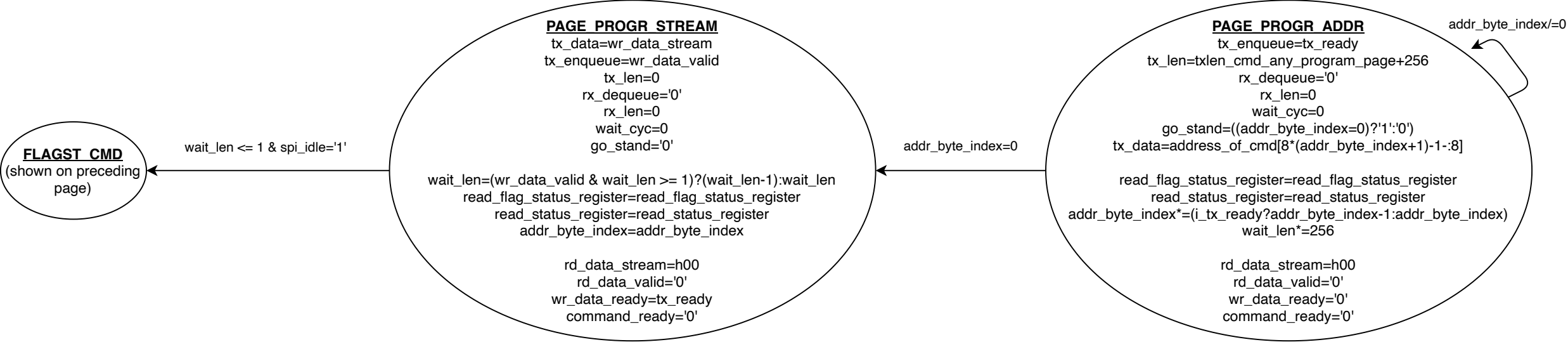
spi_idle='0'

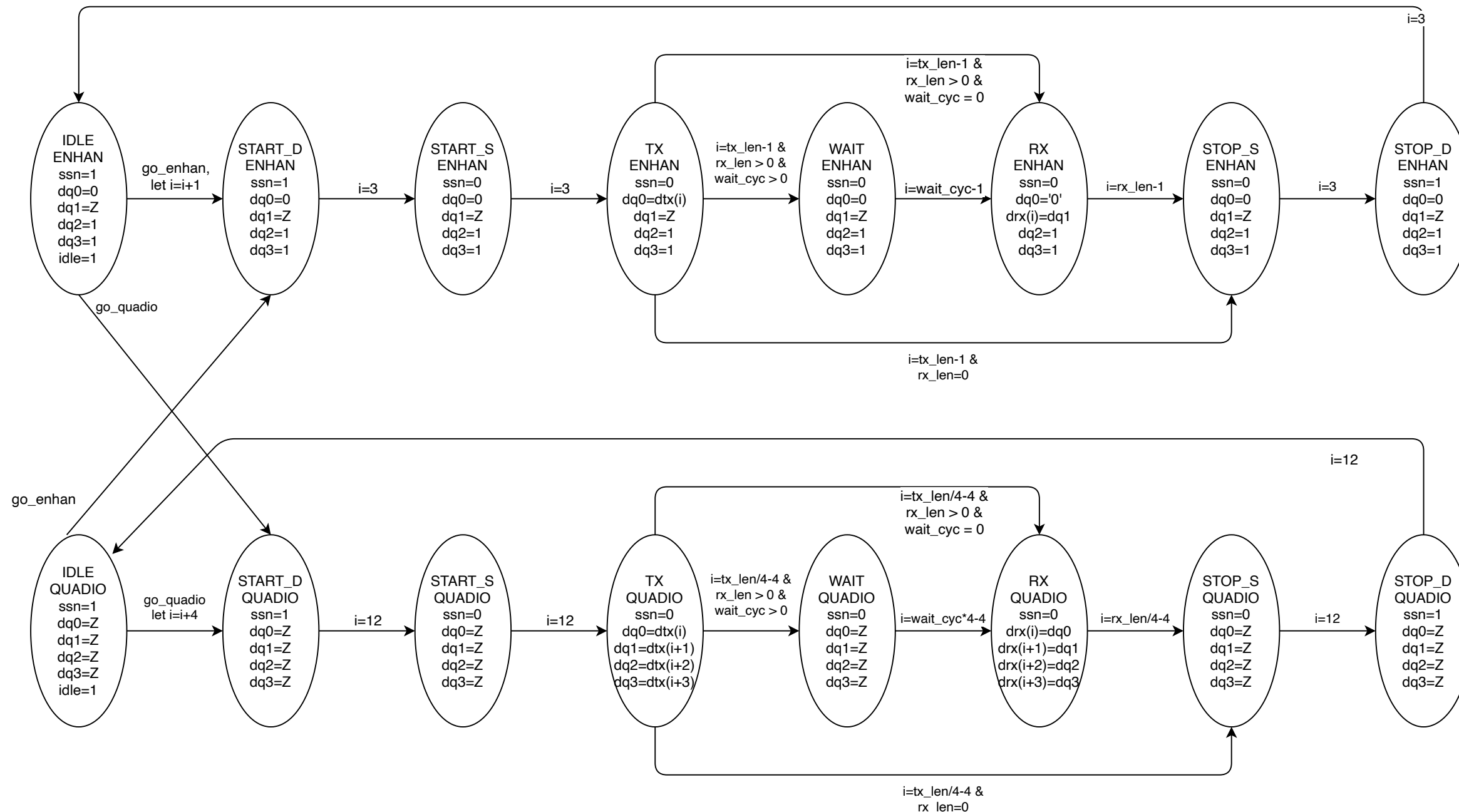
spi_idle='1'











In each transition, tx_len and rx_len are to be multiplied by 8 from the FSM input signals, as it only makes sense to input into the FSM a byte count, while the FSM requires transitioning based upon a bit count.

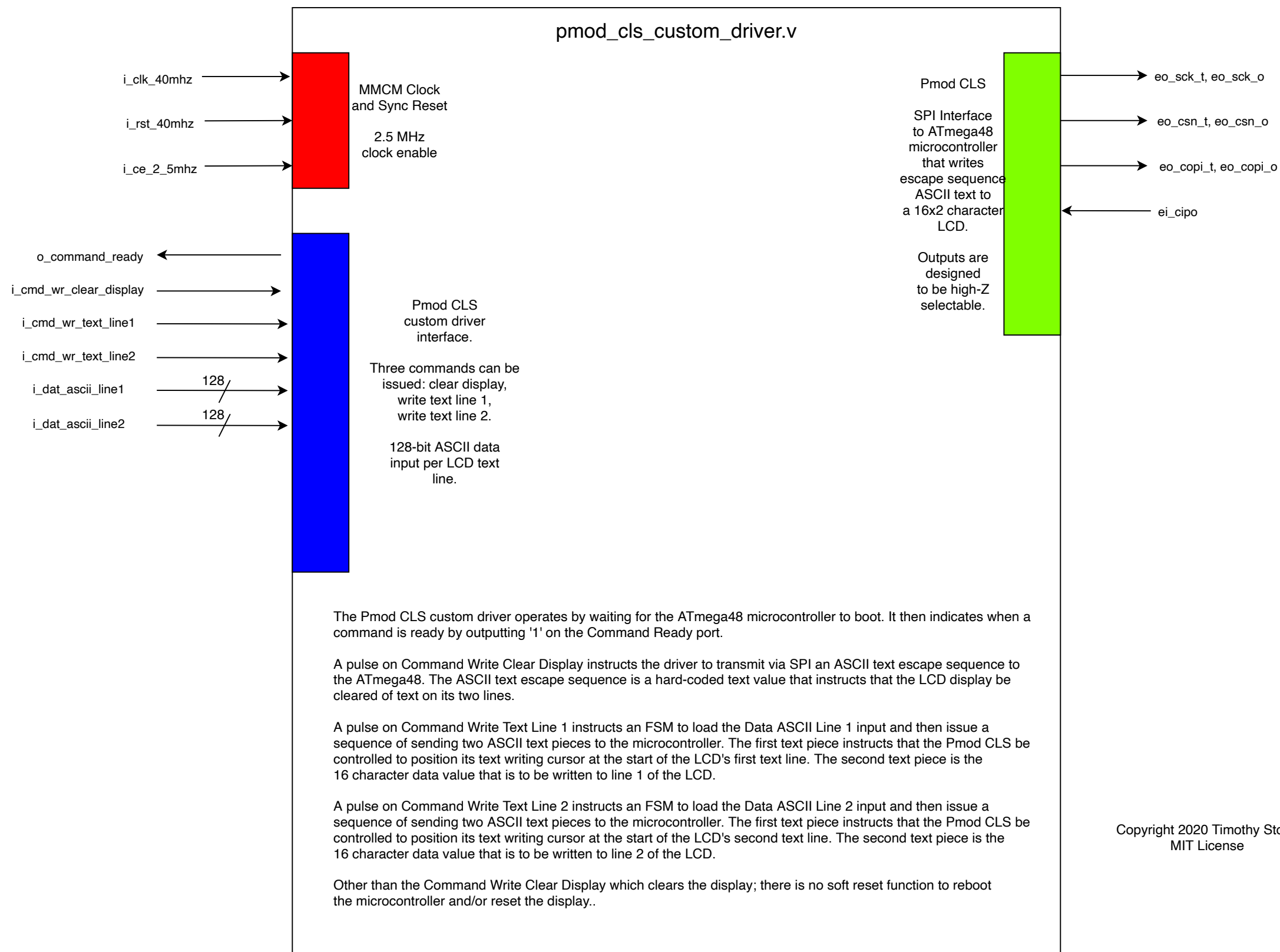
Generic SPI FSM, with the only one SPI slave on the bus.

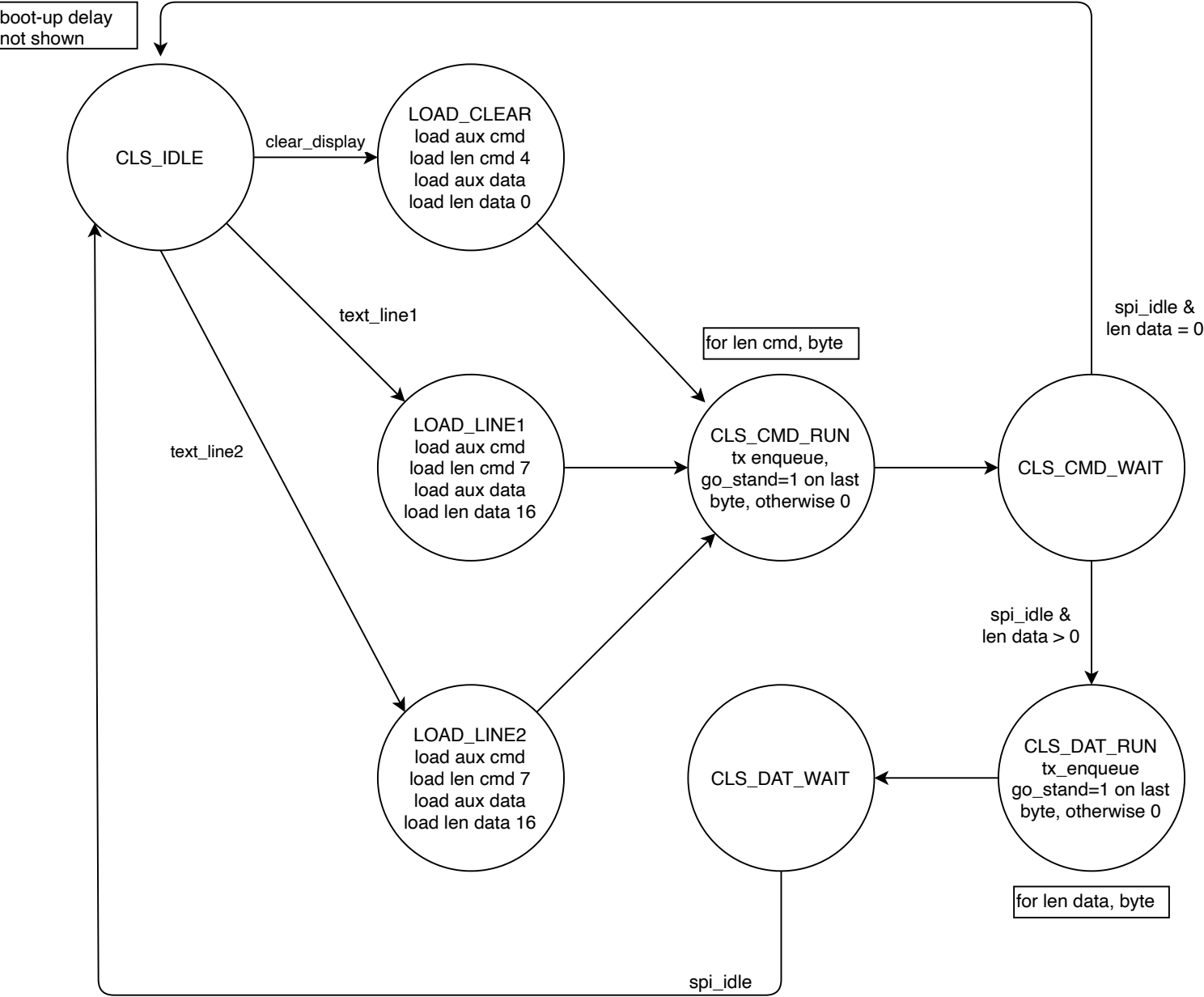
Refer to project source code for more details of the FSM.
This drawing is a simplified conceptual representation of a more complex machine.
Note that this drawing indicates operation of either Extended SPI or Quad IO SPI.
The source code only implemented Extended SPI due to the Quad IO for N25Q requiring some transmission parts to only output data on MOSI and other parts to be output using all four bits. This diagram shows the original design intent, but only Extended SPI was implemented. In other words, this diagram is incorrect.

Copyright 2019 Timothy Stotts

MIT License. Refer to LICENSE file included with this software.

Quad SPI FSM for Enhanced SPI operation of 4-data-line SPI Flash, diagram pending.



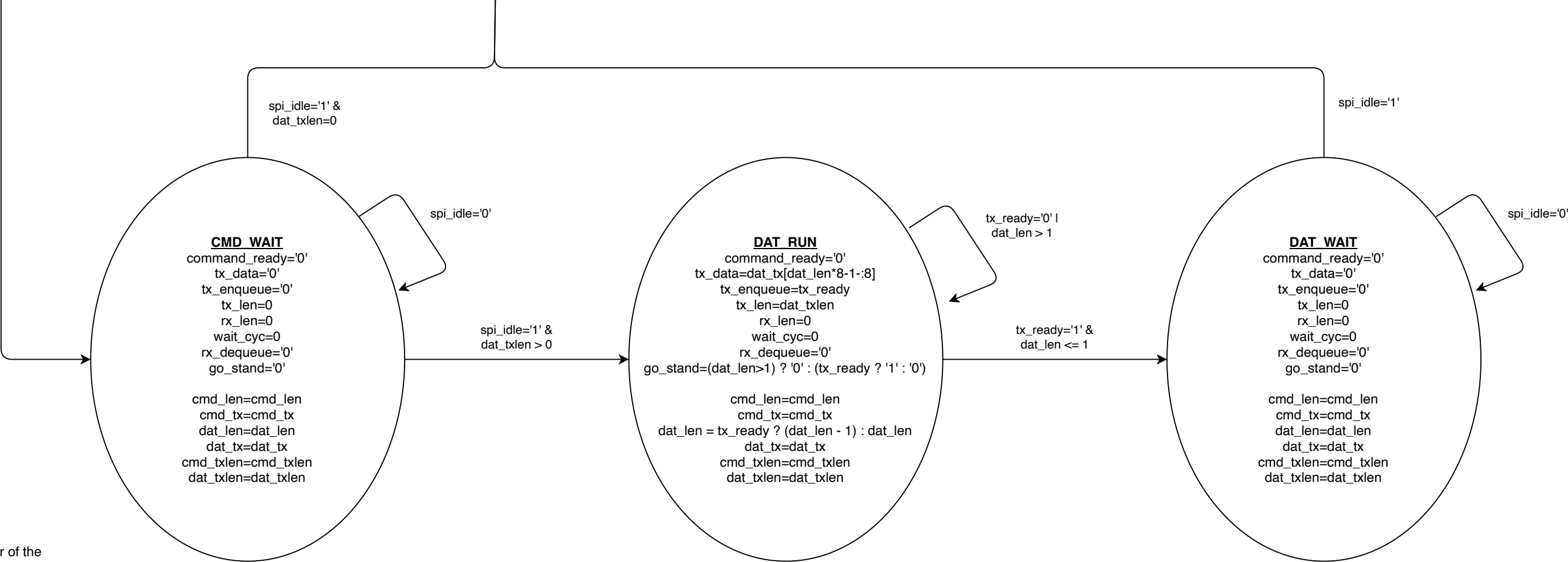


A FSM to operate the Digilent Inc. PMOD CLS LCD display communication via the single slave SPI-machine FSM of this document.

This diagram is incomplete and does not show boot-time delay. Also, some state-bypass preventions and iterations may not be shown. This is the first design draft, and the complete FSM diagram is shown on another page.

Copyright 2020 Timothy Stotts
MIT License

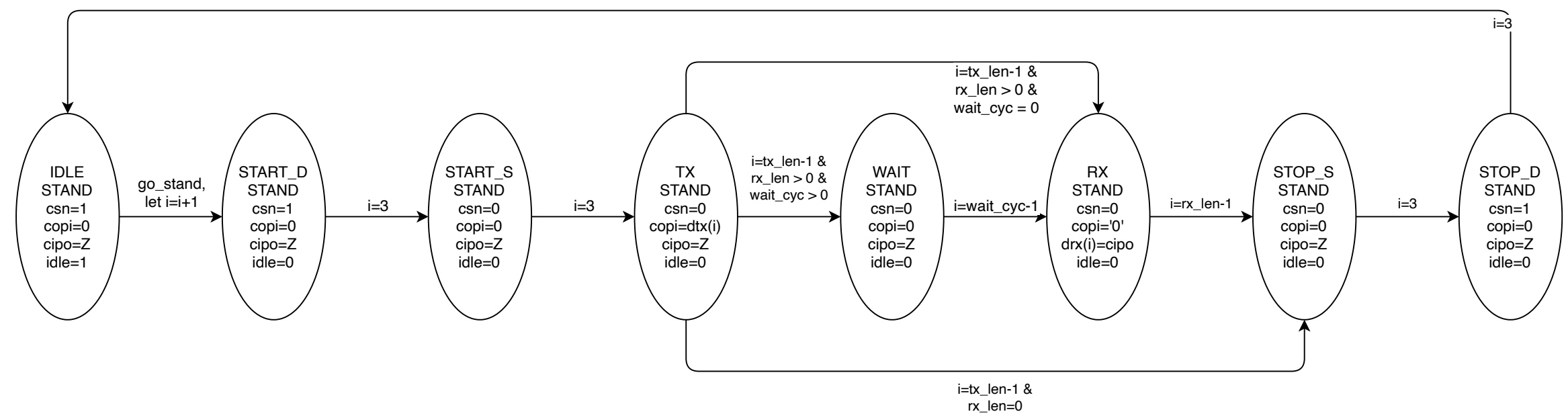




CLS Stand SPI Driver of the CLS Custom Driver. The FSM works by waiting for a command to (a) write a clear display command to the CLS, (b) write a 16-character line to the first line of the CLS, or (c) write a 16-character line to the second line of the CLS. The clear display command only writes an ANSI escape sequence with no textual data after it. The write line 1 and write line 2 commands write an ANSI escape sequence to position the cursor at the beginning of one of the two lines, and then 16 characters of text. The CLS microcontroller processes each command and line data. The clear display clears the 16x2 LCD; and the write line writes new text to the specified line of the 16x2 LCD.

Copyright 2020 Timothy Stotts
MIT License

The first group of Moore machine outputs in each state bubble are FSM outputs, some of them with combinatorial output that is normally only seen in a Mealy machine. (This FSM could be said to be a hybrid Moore and Mealy machine.) The second group of outputs are recursive auxiliary outputs that retain state from one state to the next when assigned to itself.



In each transition, tx_len and rx_len are to be multiplied by 8 from the FSM input signals, as it only makes sense to input into the FSM a byte count, while the FSM requires transitioning based upon a bit count.

Copyright 2020 Timothy Stotts
MIT License

Generic SPI FSM, with only one SPI slave on the bus.

This diagram is incomplete and is shown as the first draft of designing the Standard SPI Single Slave Device driver.

Generic SPI FSM, with only one SPI
slave on the bus.

In each transition, tx_len and rx_len are
to be multiplied by 8 from the FSM input
signals, as it only makes sense to input
into the FSM a byte count, while the
FSM requires transitioning
based upon a bit count.

The main FSM combinatorial operates
all states, but processes data on in the
TX state when TX data is written out the
copi port.

A side sequential process processes
data only on the RX state when RX data
is read from the cipo input port.

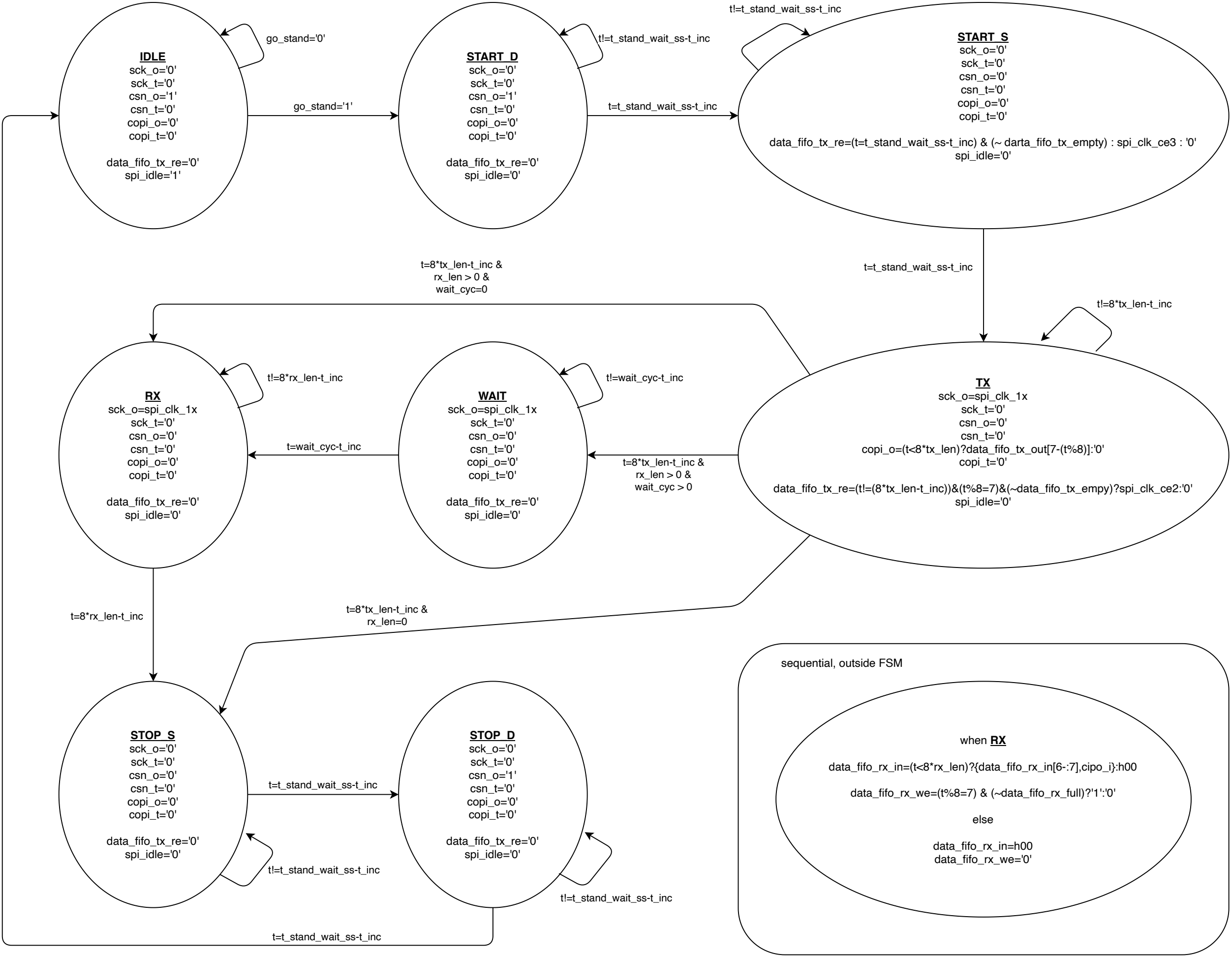
A general timer is used to time the SPI
clock cycles (and thus bits). The timer
is reset to zero at the transition from
a state to a different state.

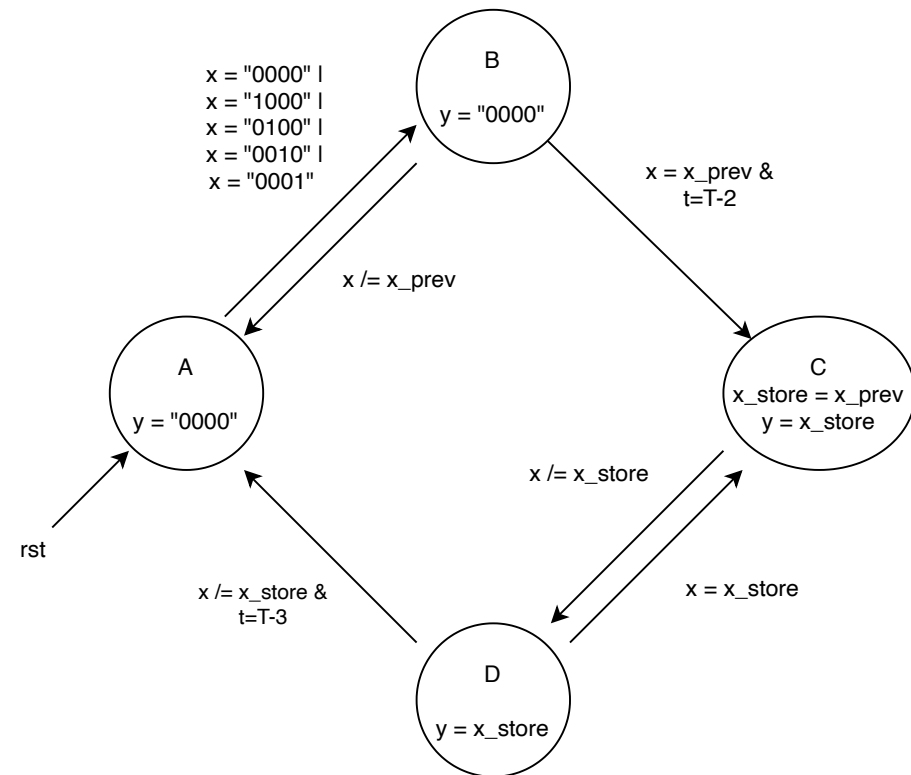
This machine could be considered a
hybrid Moore and Mealy machine.
The outputs during a state can
change based upon timer and FIFO
control inputs; but the diagram is
drawn as a Moore machine.

The TX state controls reading byte
by byte from a TX fifo and write its
bits to COPI until the timer has
reached 8*tx_len cycles.

The RX standalone sequential
controls reading a byte bit by bit
from CIPO and then writing the
full byte to a RX fifo when the
full byte has been received.
The main FSM still controls when
the FSM is in the **RX** state or has
transitioned to **STOP S** state.

Note that clock enables such as
spi_clk_ce2 are omitted from most
of the diagram, but are still required
for proper function of the design to
write data on the falling edge of the
SPI clock output and read data on
the rising edge of the SPI clock
output. Refer to the source code.





Full 4-button combined debouncer.

x is defined as a four-bit value.

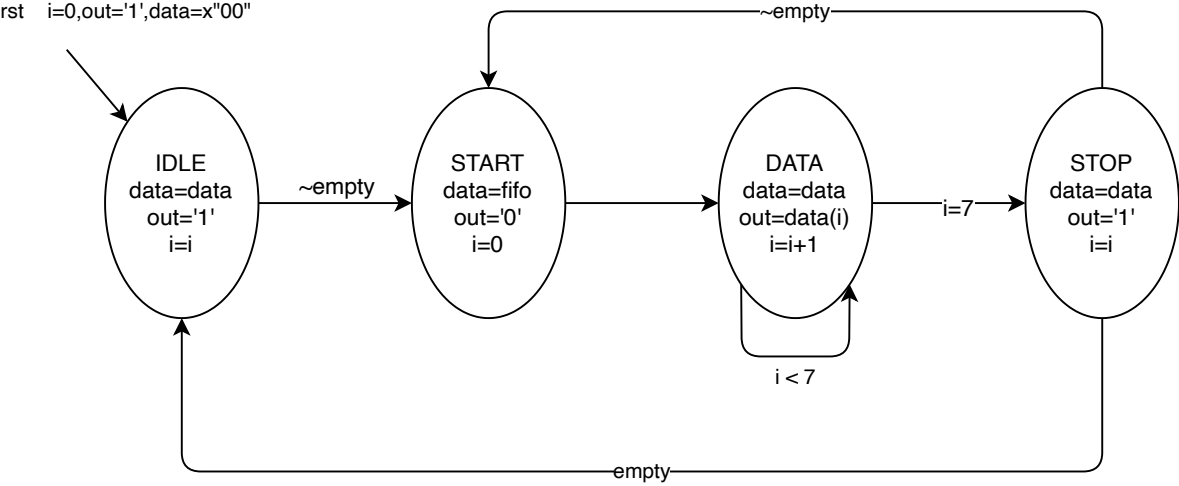
x_prev is defined as a four-bit value that holds the previous clock cycle value of x.

x_store is defined as a four-bit value that holds the value of x and updates the debouncer FSM entered state C during the transition BC..

The registers x_prev and x_store could be combined into one register, with its capture of X being a clock-enable during transitions and states of a more complex diagram.

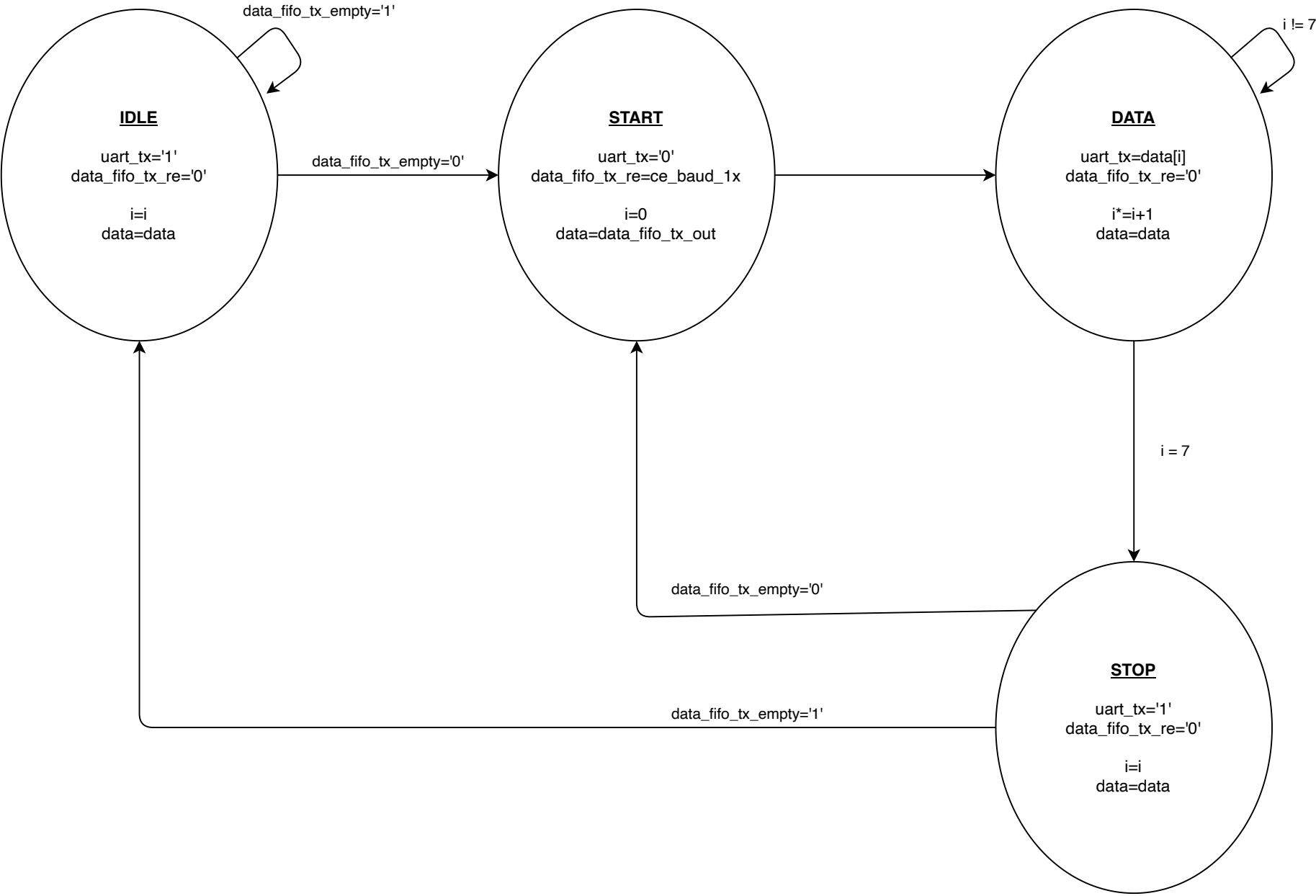
Copyright 2020 Timothy Stotts

MIT License. Refer to LICENSE
file included with this software.



A TX ONLY UART output to UART chip from the FPGA, with the FSM executing at BAUD rate as its clock enable.

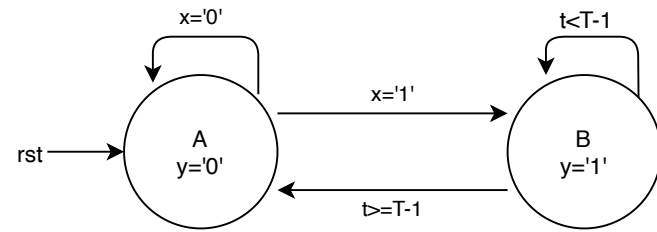
This is the first design draft,
and the complete FSM diagram
is shown on another page.



A TX ONLY UART output to UART chip from the FPGA, with the FSM executing at BAUD rate as its clock enable.

The FSM dequeues the UART TX FIFO when there is at least one more byte in the FIFO. No extra STOP/IDLE bits are generated except when the UART TX FIFO reaches empty.

The TX ONLY UART FSM generates a single START bit, 8 bits of DATA, a single STOP bit, and no parity bits.



Moore FSM for a synchronous pulse stretcher of signal X that lasts for a duration less than T, with Y lasting exactly T cycles.

Textbook Figure 8.28. quoted from:

Finite State Machines in Hardware: Theory and Design (with VHDL and SystemVerilog)
by Volnei A. Pedroni,
reprinted courtesy of The MIT Press