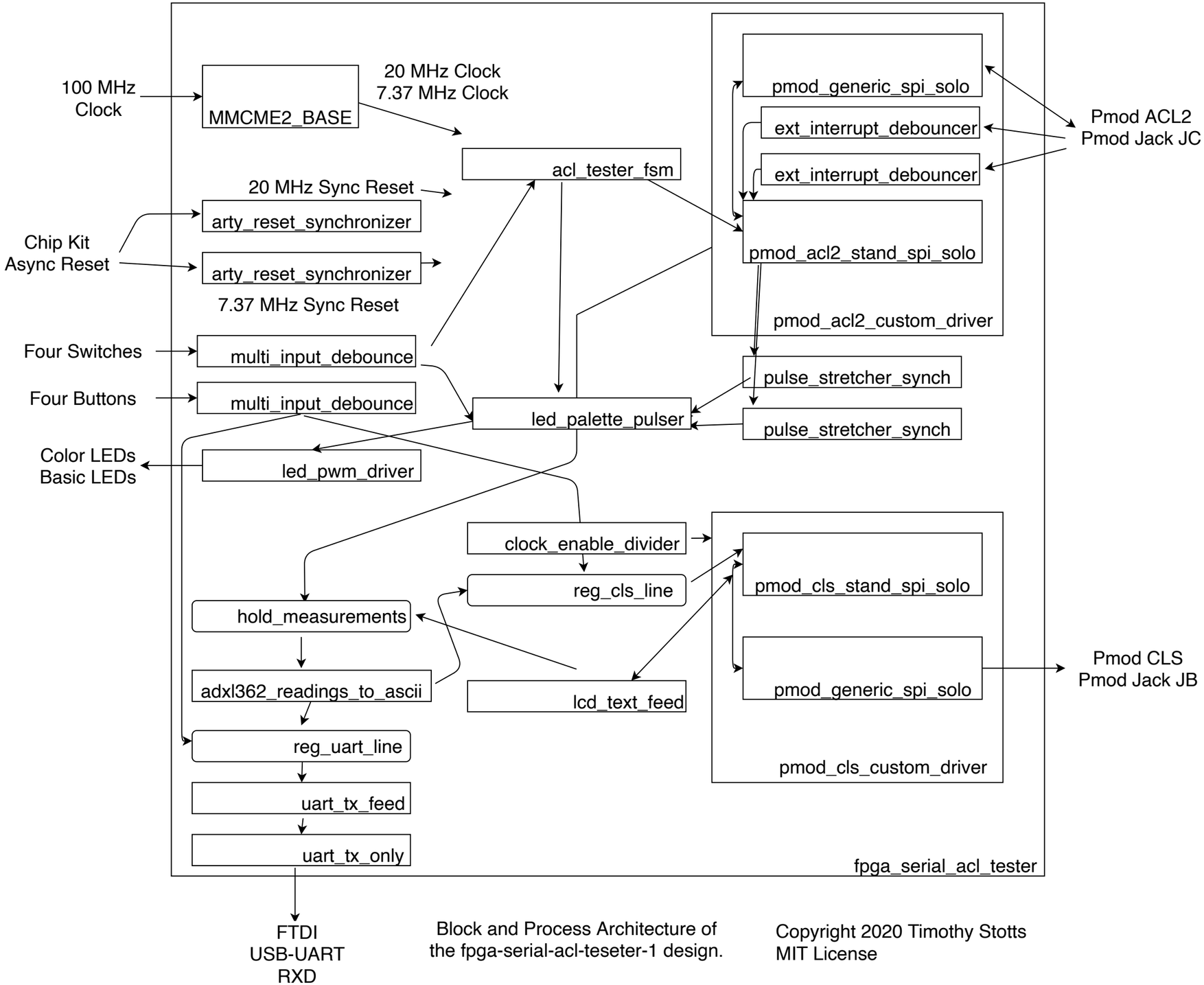
FPGA Serial Accelerometer Tester, Version 1

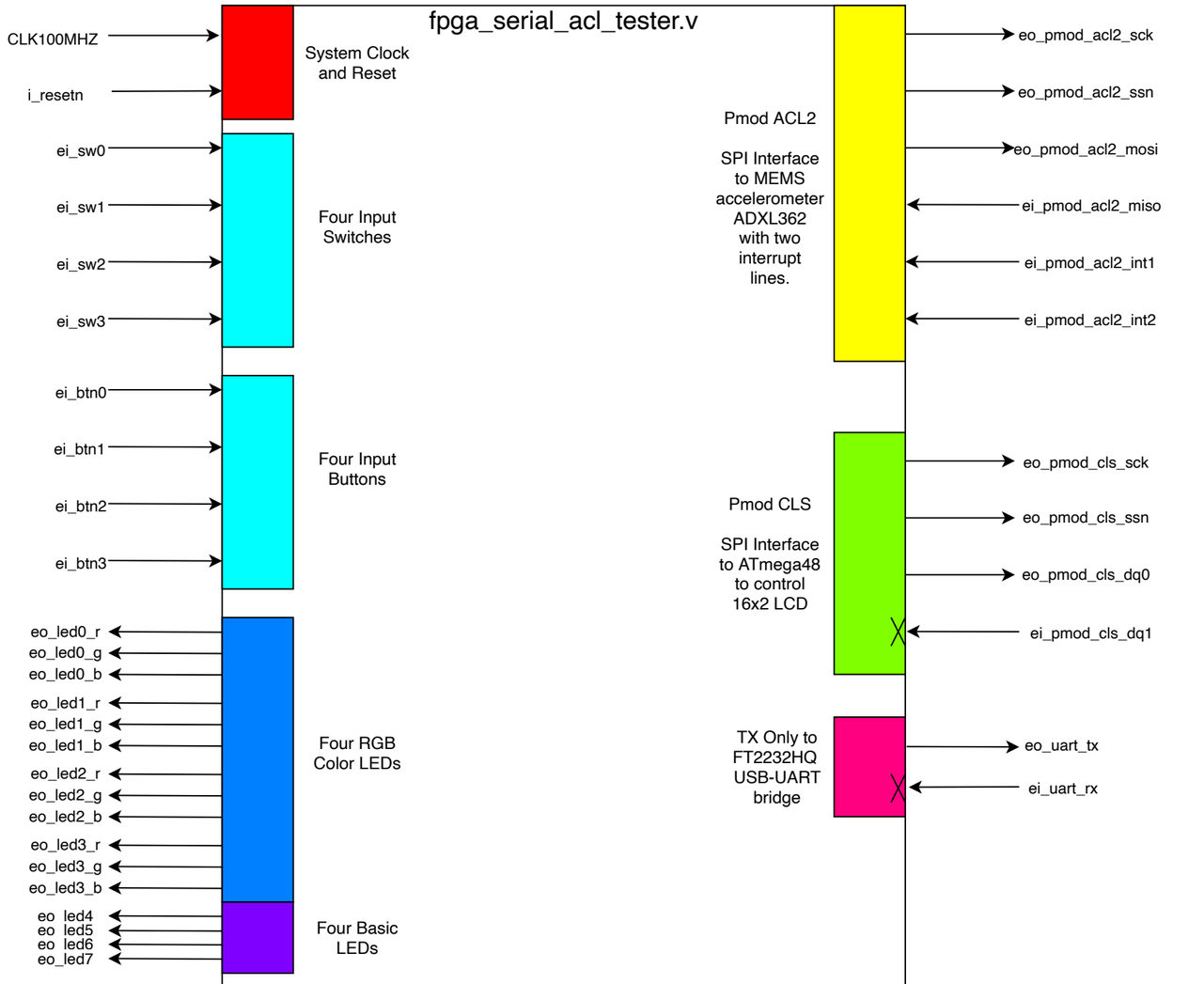
by Timothy Stotts

Hosted at:
https://github.com/timothystotts/fpga-serial-acl-tester-1

ACL-Tester-Design-Diagrams document revision 14A

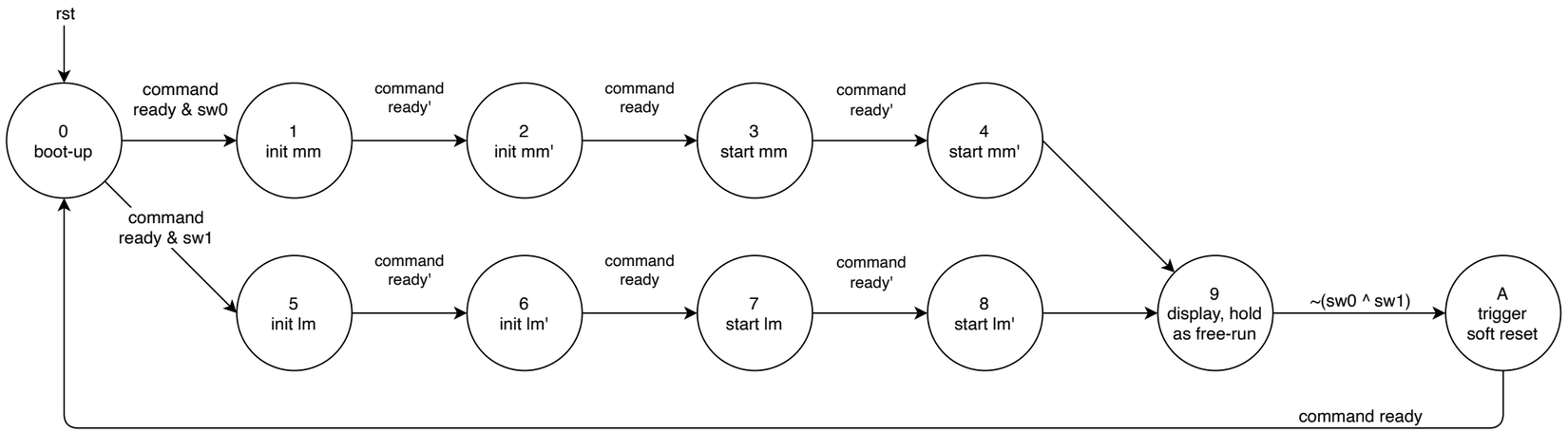


Block and Process Architecture of the fpga-serial-acl-teseter-1 design.

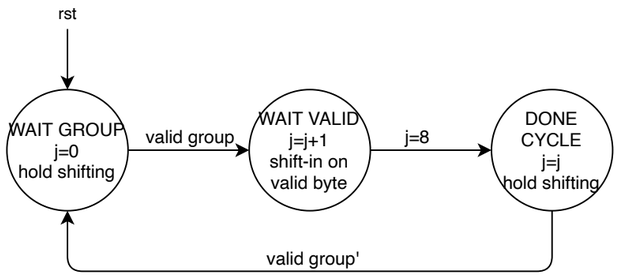


This top-level FPGA module operates the Pmod ACL2 to obtain measurement readings, converts these readings to ASCII data and text, and then outputs the ASCII data on the UART TX and the ASCII text on the Pmod CLS LCD. The switches select one of two possible operational modes: Stream Measurements to Display, or Detect Activity/Inactivity events and display event-trigger measurements. The event triggering is coded into the Pmod ACL2 driver with constants that define operation and thresholds. The color LEDs LD0 and LD1 display information about internal FSM states to indicate mode of operation and if the top-level FSMs are operating data reception from the Pmod ACL2. The color LEDs LD2 ad LD3 indicate Activity Event and Inactivity Event respectively. The basic LED LD4 indicates Awake status of the Pmod ACL2; and basic LD6 and LD7 indicate selection of SW0 and SW1 respectively.

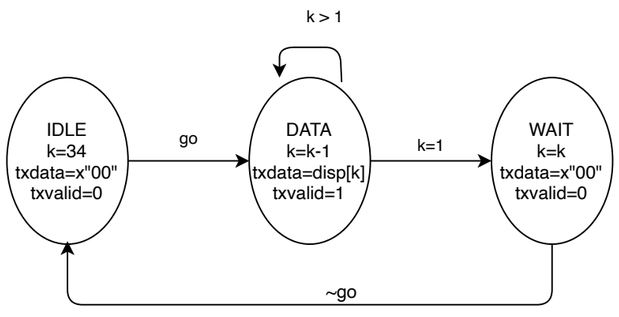
The Pmod CLS is selected to operate with SPI output and no SPI input, even though the Pmod CLS additionally supports UART and I2C modes.



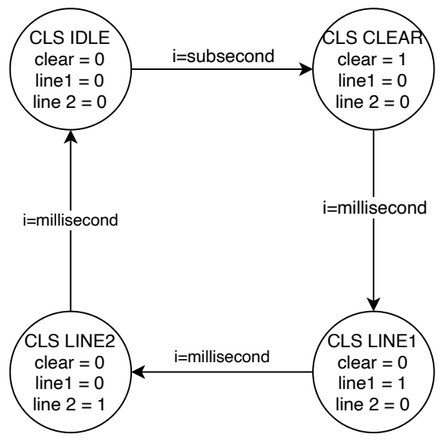
Tester FSM for operating the PMOD ACL2 driver commands.



Tester FSM to receive the streamed measurements and shift them into a bit vector.

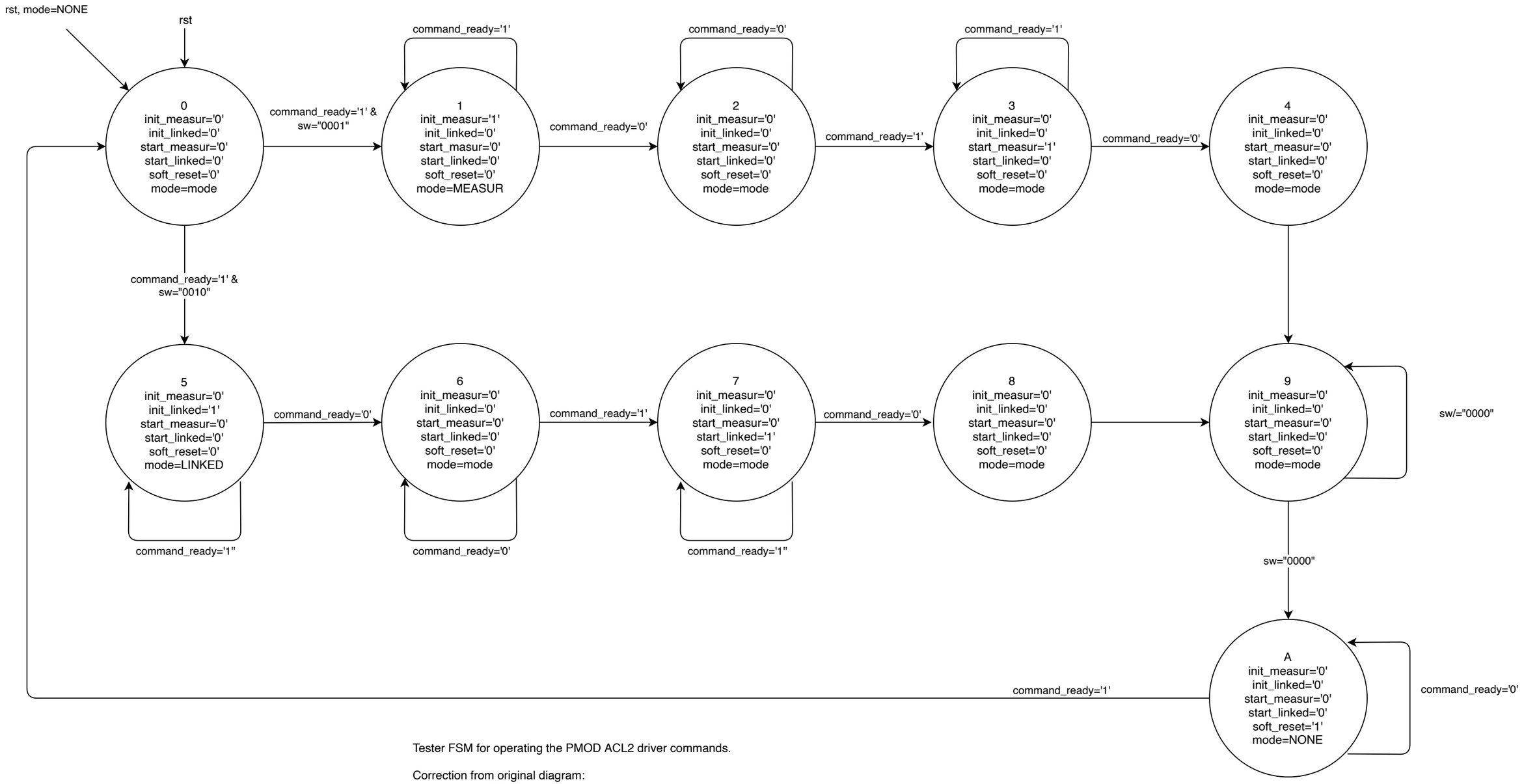


Tester FSM to load the TX ONLY UART with a 32 character text line, plus carriage return and new line.



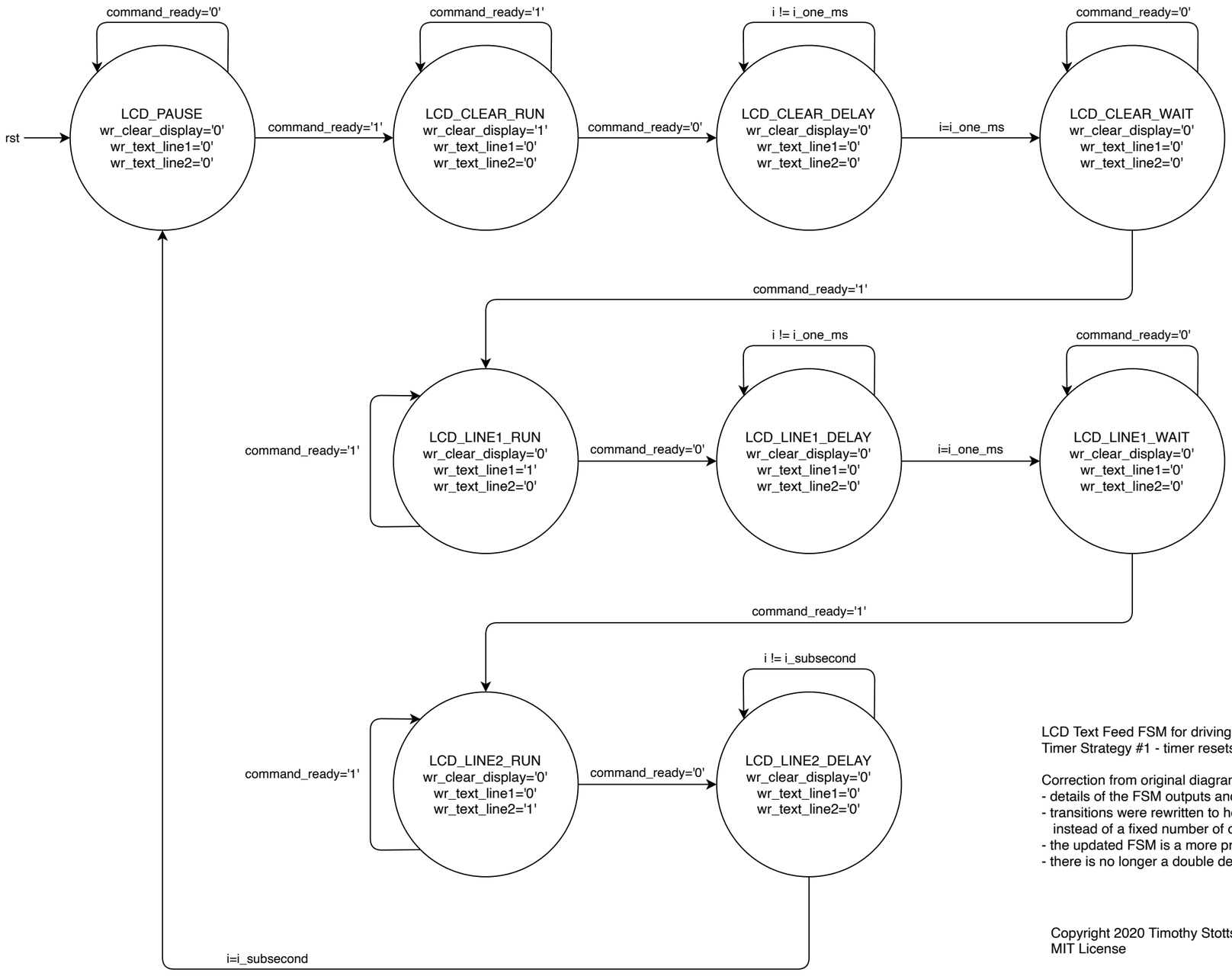
Tester FSM for updating the PMOD CLS display.

These diagrams are
incomplete. Some
state-bypass preventions
and iterations may not be shown.
These are the original diagram
draft prior to implementation.



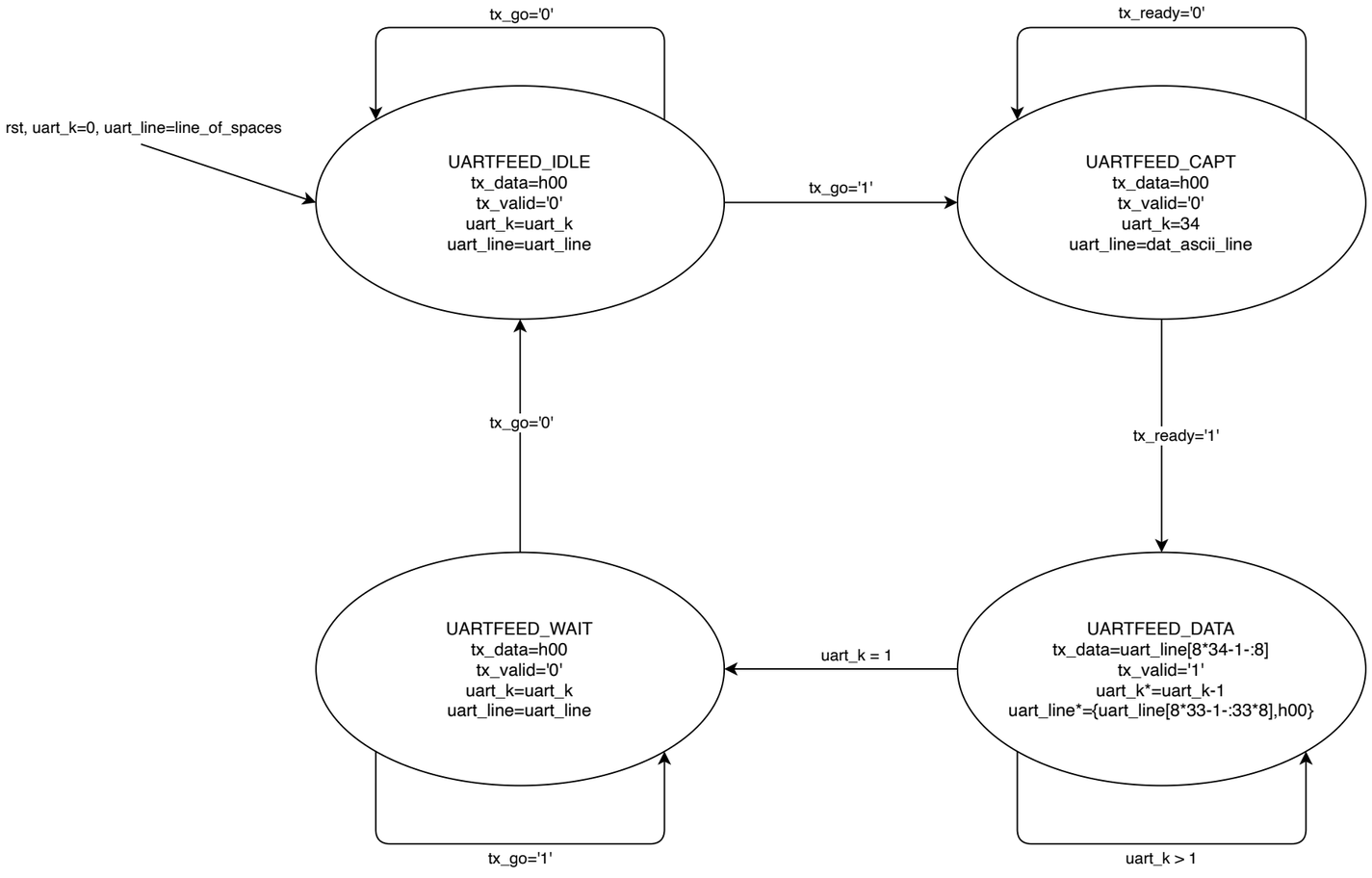
Tester FSM for operating the PMOD ACL2 driver commands.

Correction from original diagram:
- details of the FSM outputs and recursive output
- transitions 9-A and A-0 are a unique sensitivity to input command_ready in order to pulse the output soft_reset for width of as many clock cycles as required, instead of only one clock cycle. The custom ACL2 driver inputs the soft_reset signal as a level interrupt to transition to SOFTRESET_CMD after completing the current SPI operations. The command_ready remains zero before and after assertion of the soft_reset signal, and changes to one when the soft reset is complete. Preventing state bypass for these two transitions is different than the other transitions of this FSM.



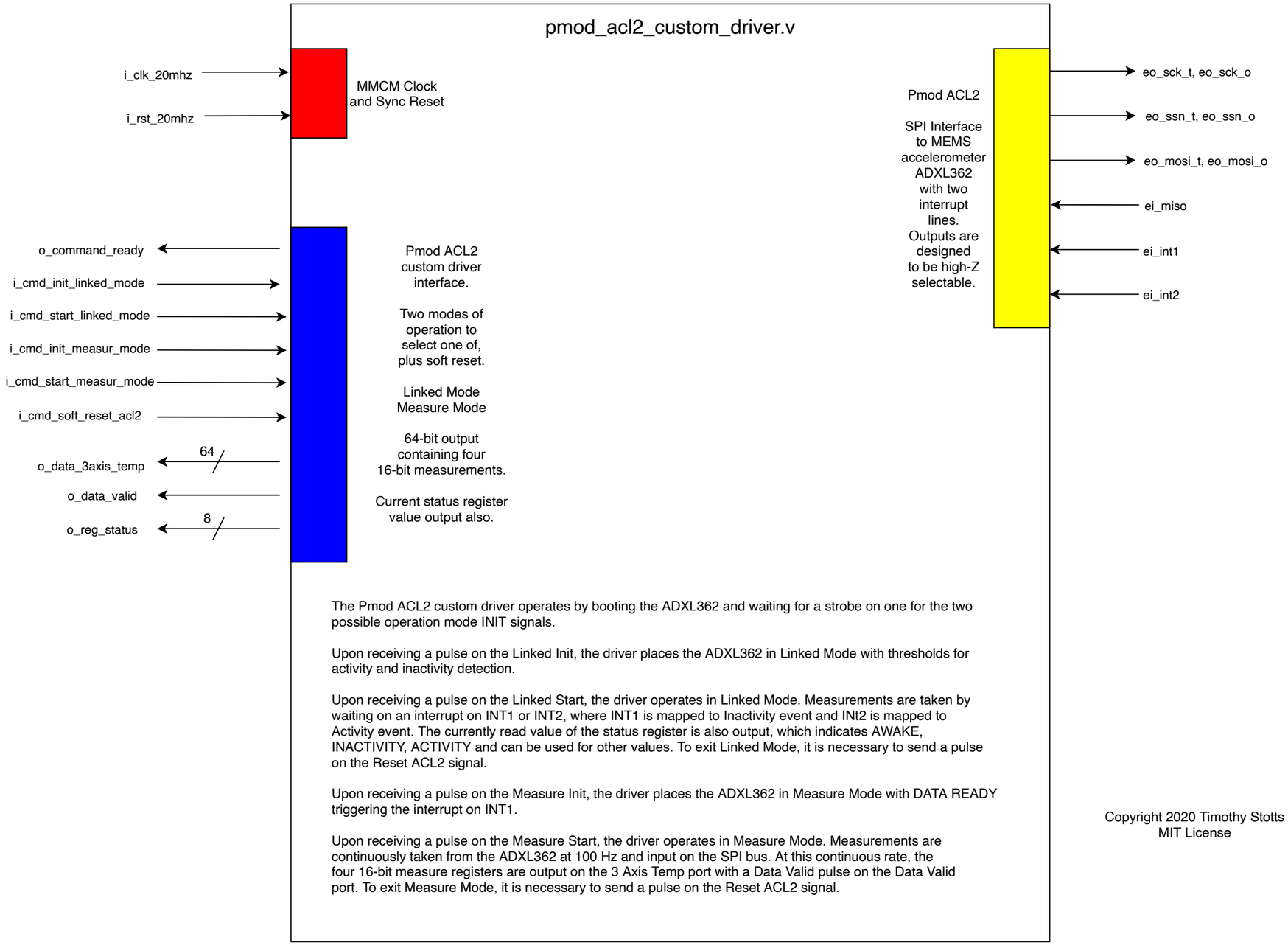
LCD Text Feed FSM for driving the commands of the custom Pmod CLS driver.
Timer Strategy #1 - timer resets on state change and runs from zero to i_subsecond .

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.



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.



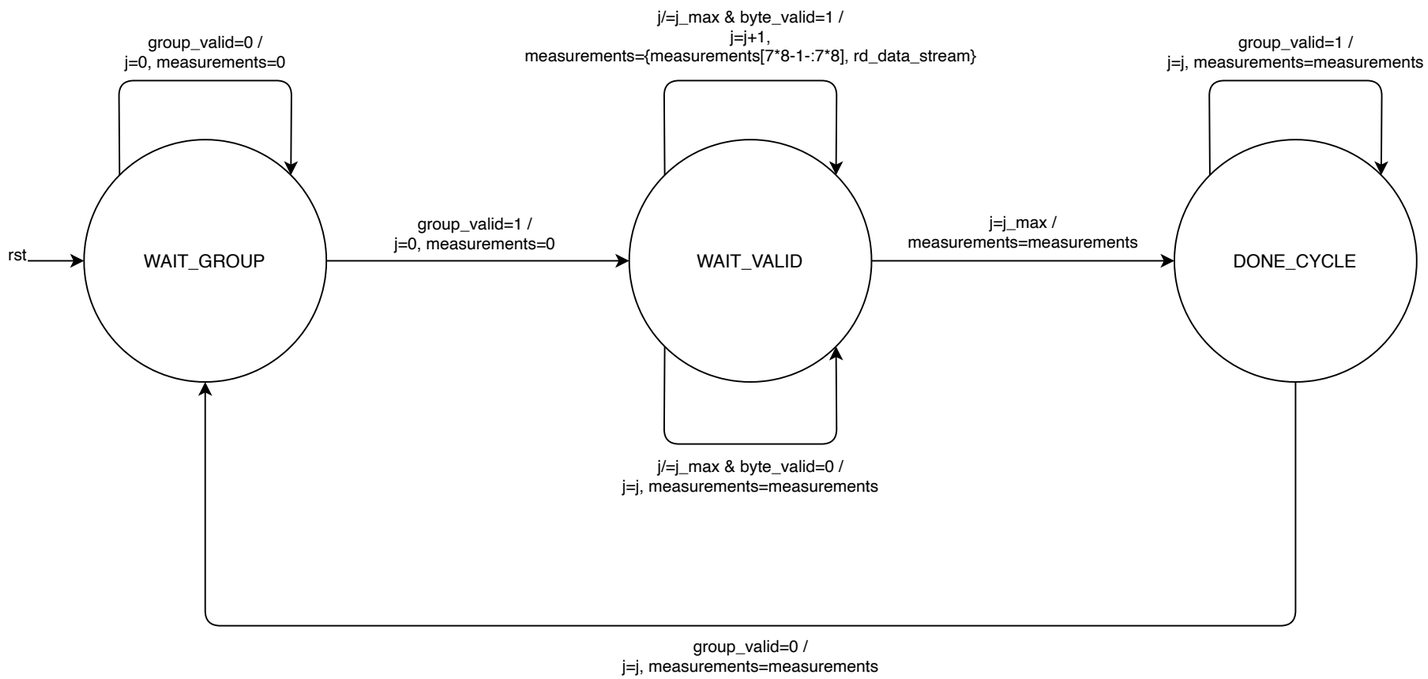
The Pmod ACL2 custom driver operates by booting the ADXL362 and waiting for a strobe on one for the two possible operation mode INIT signals.

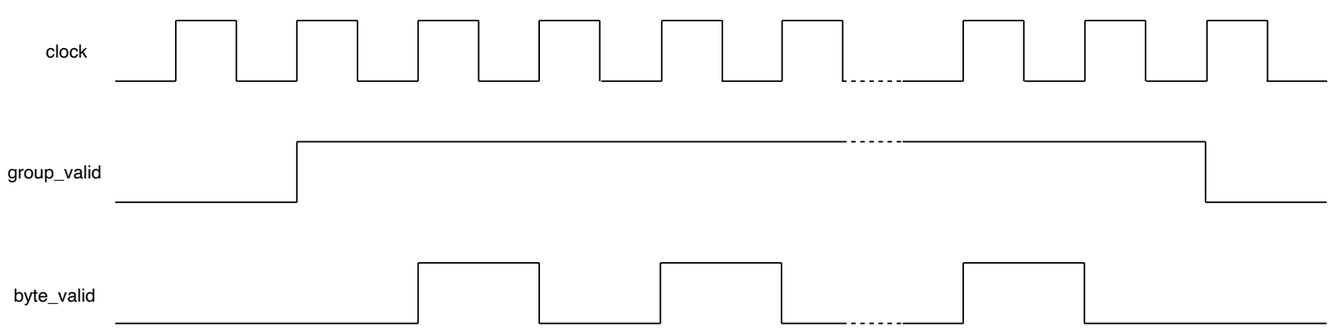
Upon receiving a pulse on the Linked Init, the driver places the ADXL362 in Linked Mode with thresholds for activity and inactivity detection.

Upon receiving a pulse on the Linked Start, the driver operates in Linked Mode. Measurements are taken by waiting on an interrupt on INT1 or INT2, where INT1 is mapped to Inactivity event and INt2 is mapped to Activity event. The currently read value of the status register is also output, which indicates AWAKE, INACTIVITY, ACTIVITY and can be used for other values. To exit Linked Mode, it is necessary to send a pulse on the Reset ACL2 signal.

Upon receiving a pulse on the Measure Init, the driver places the ADXL362 in Measure Mode with DATA READY triggering the interrupt on INT1.

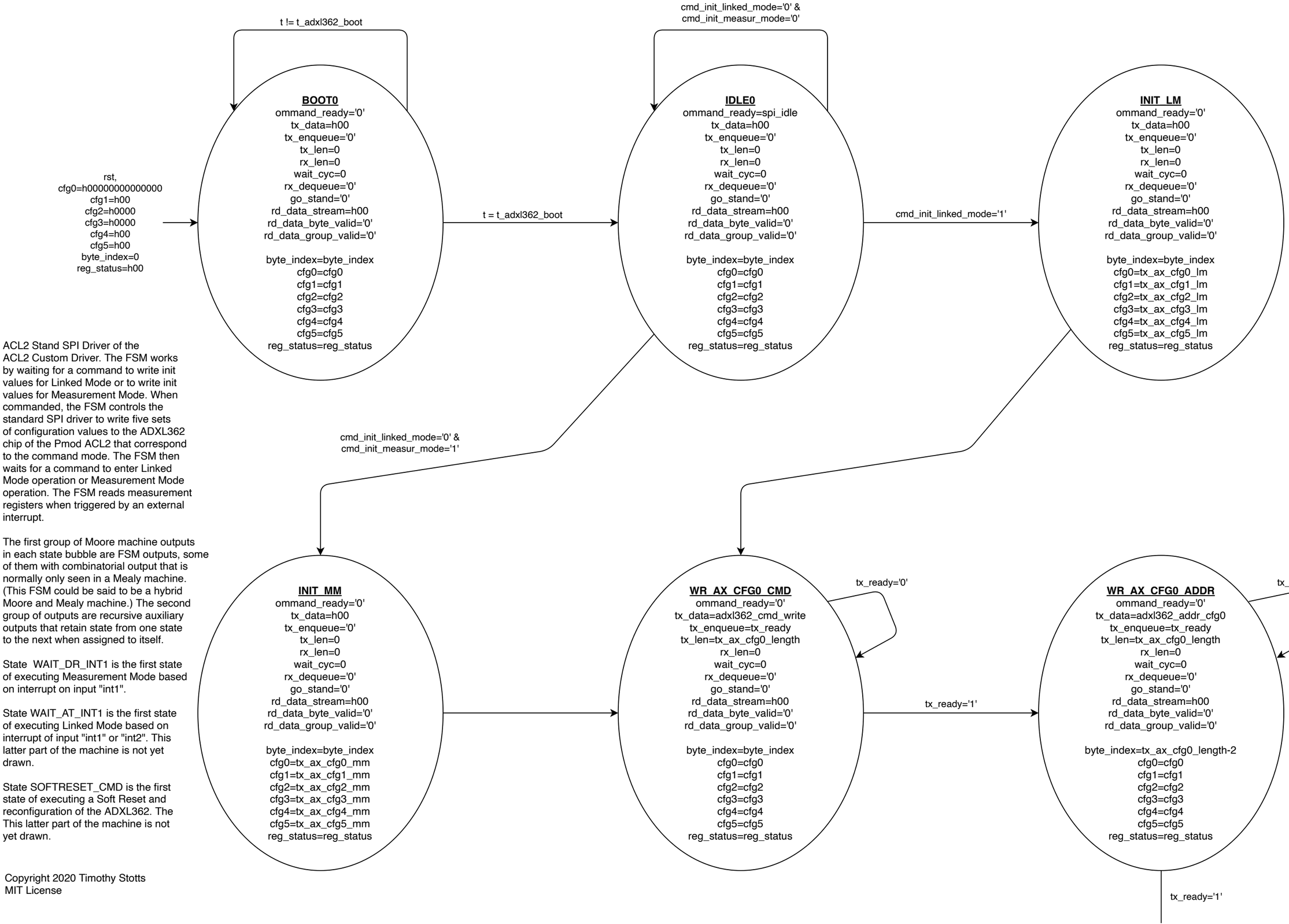
Upon receiving a pulse on the Measure Start, the driver operates in Measure Mode. Measurements are continuously taken from the ADXL362 at 100 Hz and input on the SPI bus. At this continuous rate, the four 16-bit measure registers are output on the 3 Axis Temp port with a Data Valid pulse on the Data Valid port. To exit Measure Mode, it is necessary to send a pulse on the Reset ACL2 signal.





Stream FSM for capturing RX FIFO output from the ACL2 Custom Driver.
A group_valid input signal is held high to indicate a set of measurement values.
A byte_valid input signal is pulsed for each new byte to shift into the measurement register.

Operational driver for the Digilent PMOD ACL2, that drives the Standard SPI FSM. This diagram is incomplete and does not show Soft Reset operation, or boot-time delay. Also, not all state-bypass preventions and not all iterations are show. This is the first design draft, and the complete FSM diagram is shown on another page.



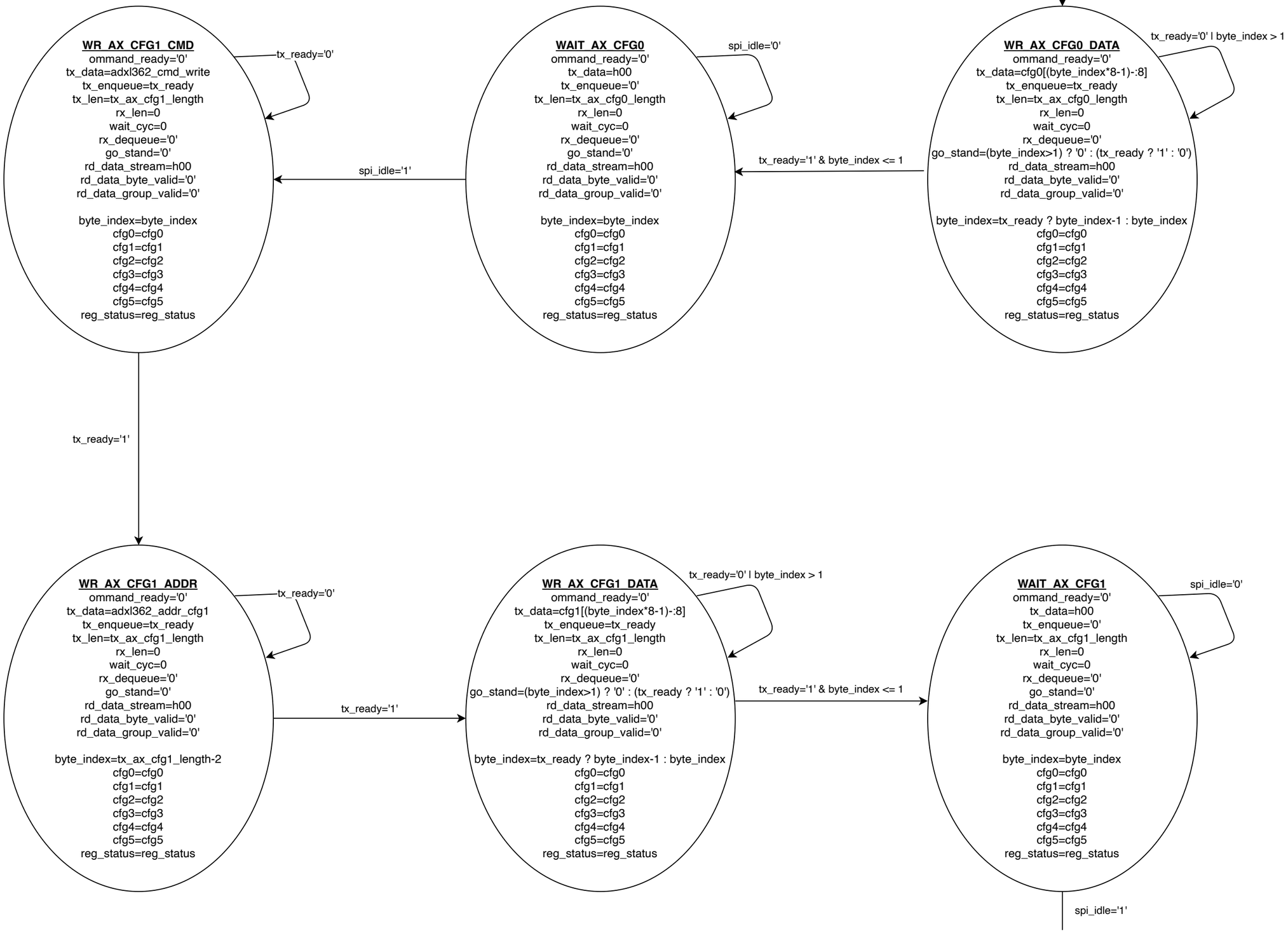
ACL2 Stand SPI Driver of the ACL2 Custom Driver. The FSM works by waiting for a command to write init values for Linked Mode or to write init values for Measurement Mode. When commanded, the FSM controls the standard SPI driver to write five sets of configuration values to the ADXL362 chip of the Pmod ACL2 that correspond to the command mode. The FSM then waits for a command to enter Linked Mode operation or Measurement Mode operation. The FSM reads measurement registers when triggered by an external interrupt.

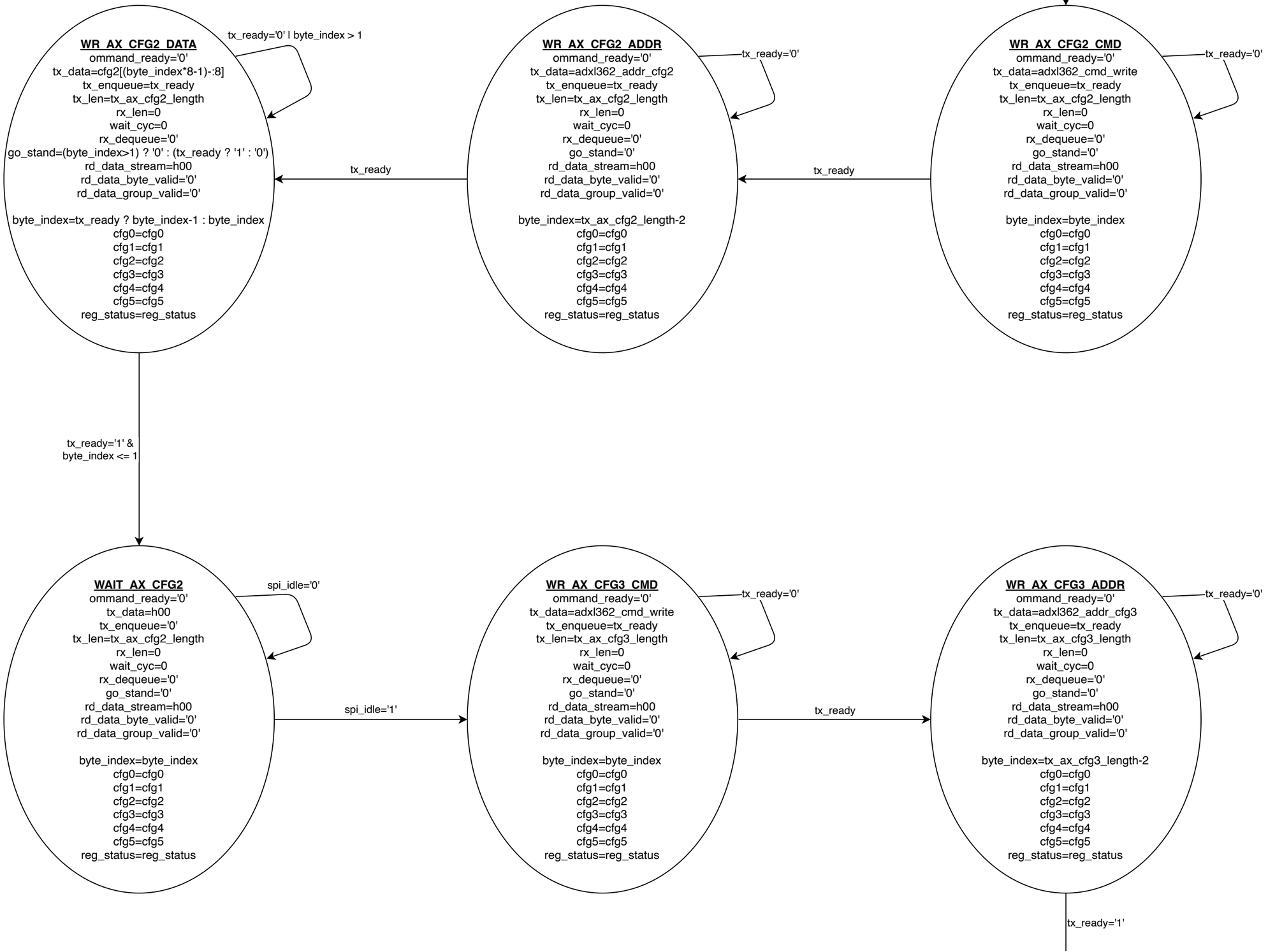
The first group of Moore machine outputs in each state bubble are FSM outputs, some of them with combinatorial output that is normally only seen in a Mealy machine. (This FSM could be said to be a hybrid Moore and Mealy machine.) The second group of outputs are recursive auxiliary outputs that retain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

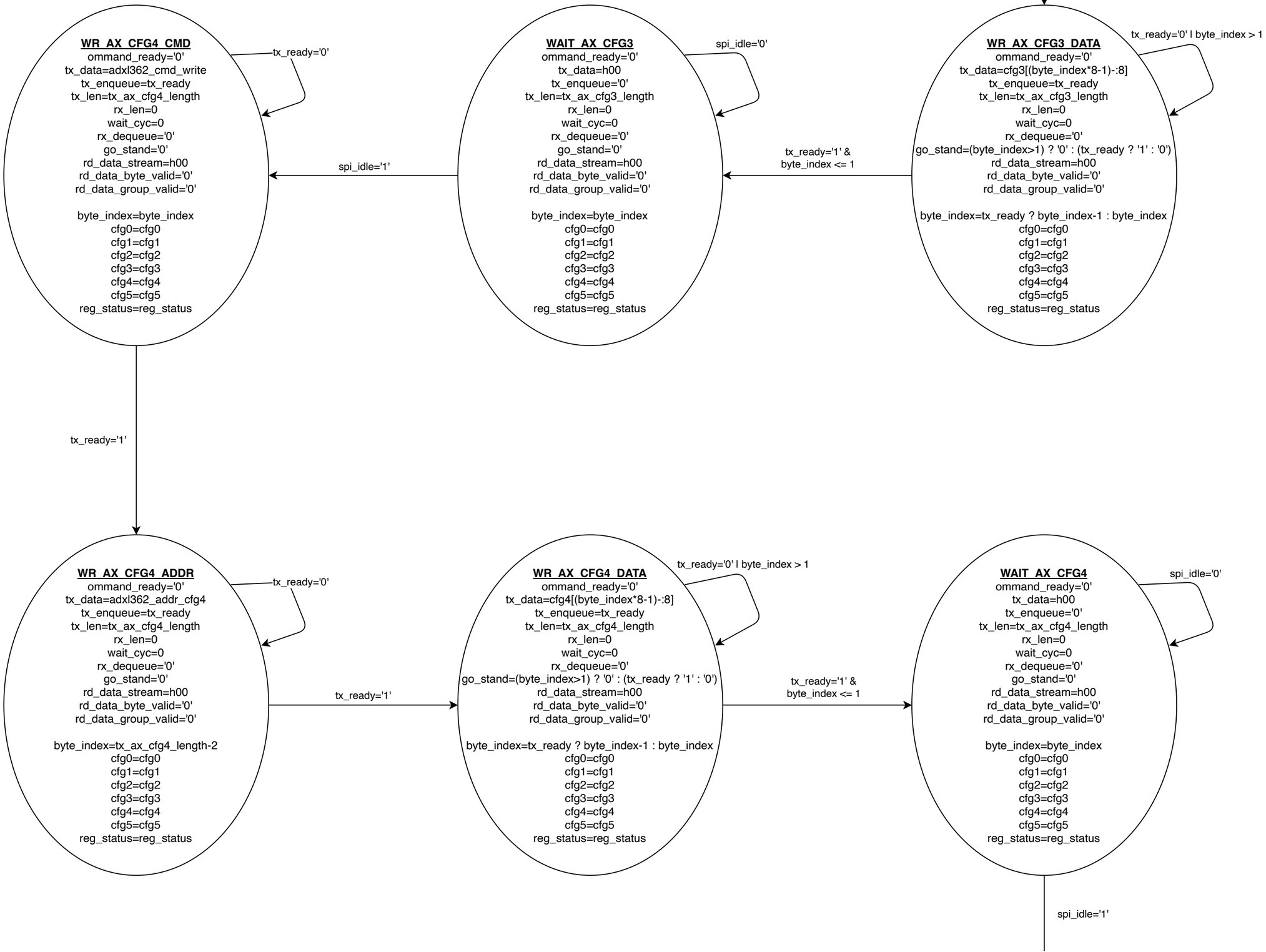
State WAIT_DR_INT1 is the first state of executing Measurement Mode based on interrupt on input "int1".

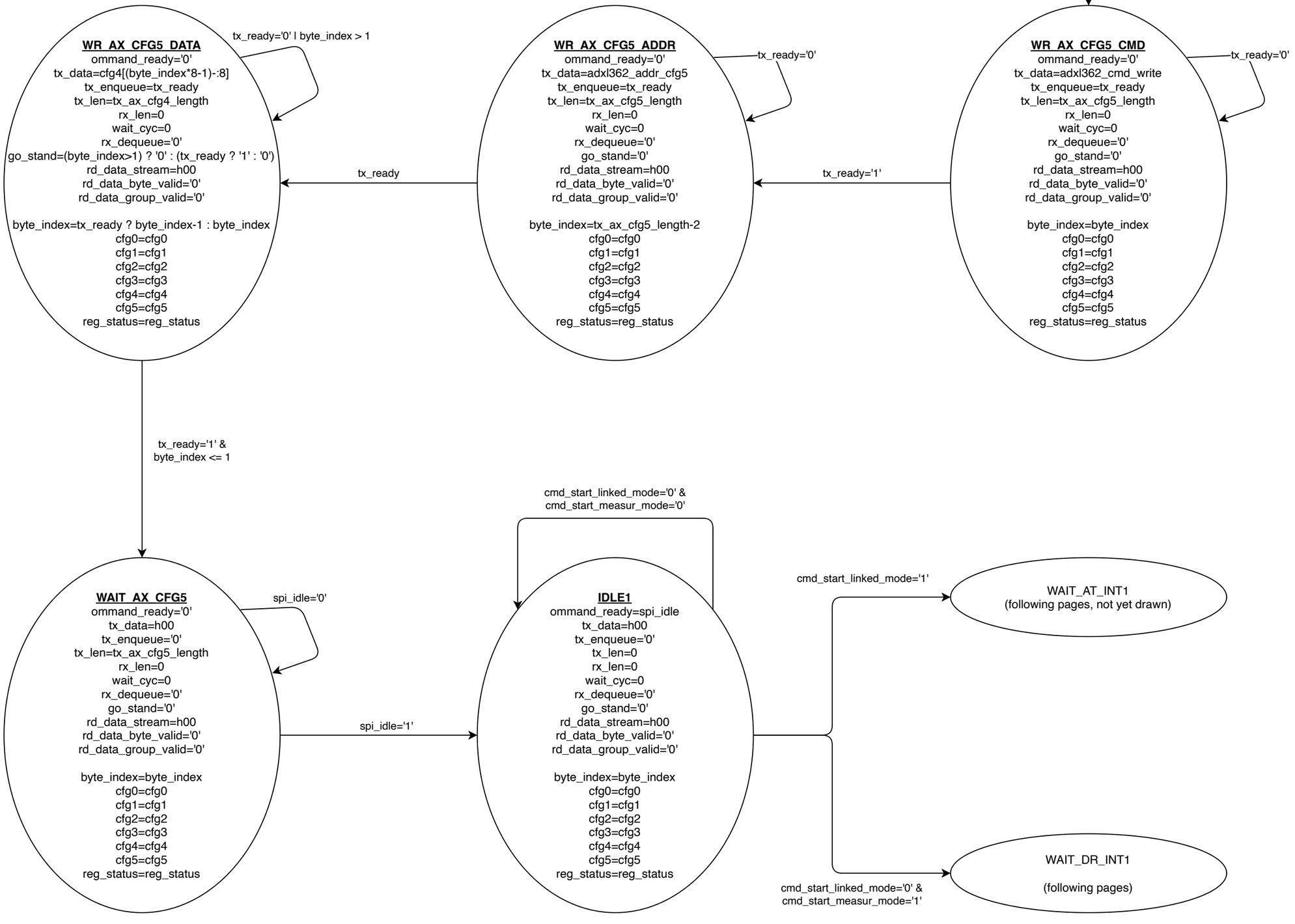
State WAIT_AT_INT1 is the first state of executing Linked Mode based on interrupt of input "int1" or "int2". This latter part of the machine is not yet drawn.

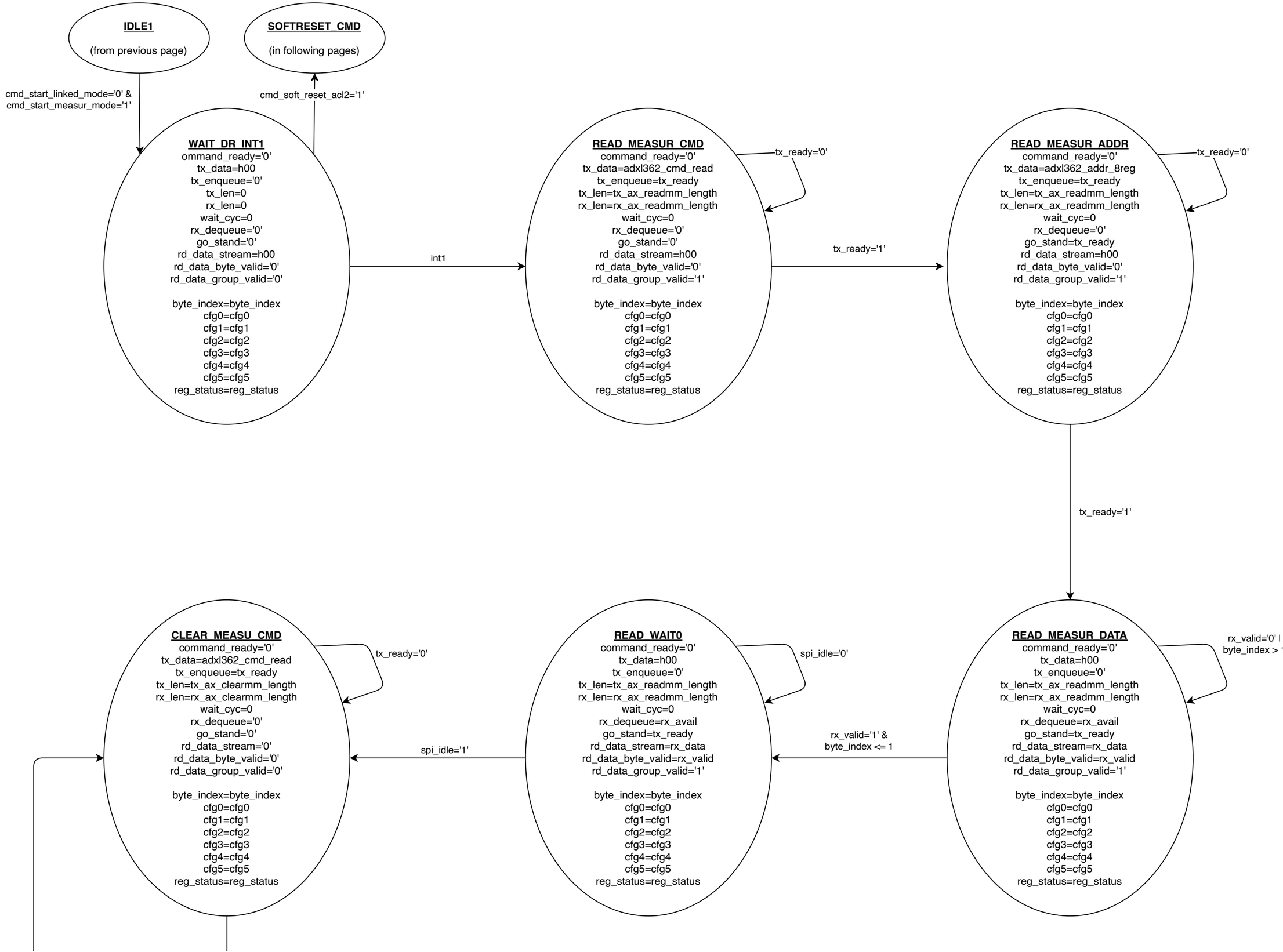
State SOFTRESET_CMD is the first state of executing a Soft Reset and reconfiguration of the ADXL362. The This latter part of the machine is not yet drawn.


WR_AX_CFG1_CMD
ommand_ready='0'
tx_data=adxl362_cmd_write
tx_enqueue=tx_ready
tx_len=tx_ax_cfg1_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
spi_idle='1'
WAIT_AX_CFG0
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg0_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
tx_ready='1' & byte_index <= 1
WR_AX_CFG0_DATA
ommand_ready='0'
tx_data=cfg0[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg0_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready='1'
WR_AX_CFG1_ADDR
ommand_ready='0'
tx_data=adxl362_addr_cfg1
tx_enqueue=tx_ready
tx_len=tx_ax_cfg1_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ax_cfg1_length-2
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
WR_AX_CFG1_DATA
ommand_ready='0'
tx_data=cfg1[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg1_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready='1' & byte_index <= 1
WAIT_AX_CFG1
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg1_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'

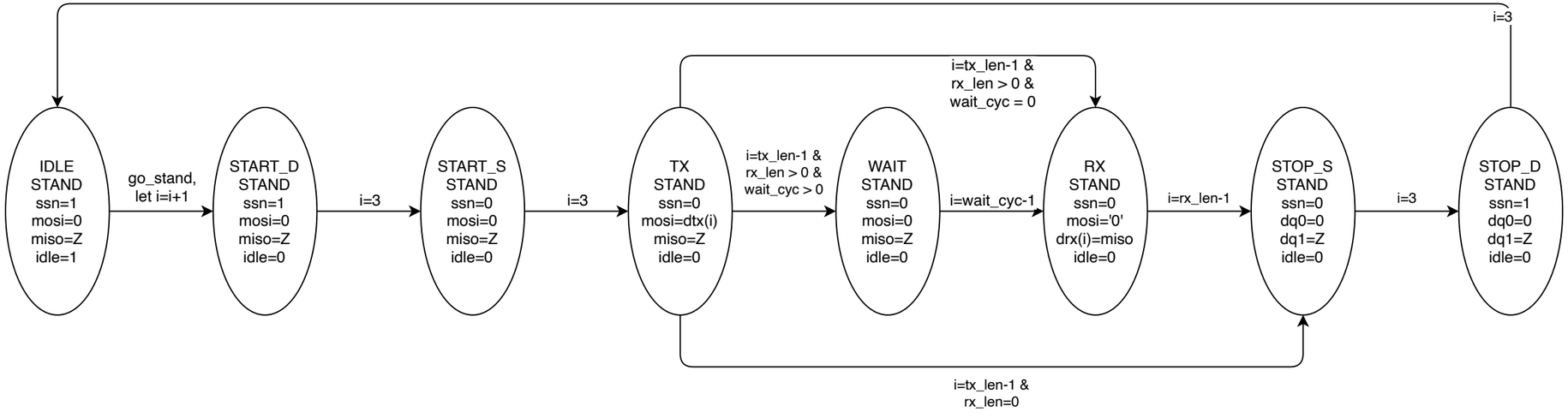

WR_AX_CFG2_DATA
ommand_ready='0'
tx_data=cfg2[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg2_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready
WR_AX_CFG2_ADDR
ommand_ready='0'
tx_data=adxl362_addr_cfg2
tx_enqueue=tx_ready
tx_len=tx_ax_cfg2_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ax_cfg2_length-2
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready
WR_AX_CFG2_CMD
ommand_ready='0'
tx_data=adxl362_cmd_write
tx_enqueue=tx_ready
tx_len=tx_ax_cfg2_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1' &
byte_index <= 1
WAIT_AX_CFG2
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg2_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'
WR_AX_CFG3_CMD
ommand_ready='0'
tx_data=adxl362_cmd_write
tx_enqueue=tx_ready
tx_len=tx_ax_cfg3_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready
WR_AX_CFG3_ADDR
ommand_ready='0'
tx_data=adxl362_addr_cfg3
tx_enqueue=tx_ready
tx_len=tx_ax_cfg3_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ax_cfg3_length-2
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'


WR_AX_CFG4_CMD
ommand_ready='0'
tx_data=adxl362_cmd_write
tx_enqueue=tx_ready
tx_len=tx_ax_cfg4_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
spi_idle='1'
WAIT_AX_CFG3
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg3_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
tx_ready='1' &
byte_index <= 1
WR_AX_CFG3_DATA
ommand_ready='0'
tx_data=cfg3[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg3_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready='1'
WR_AX_CFG4_ADDR
ommand_ready='0'
tx_data=adxl362_addr_cfg4
tx_enqueue=tx_ready
tx_len=tx_ax_cfg4_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ax_cfg4_length-2
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
WR_AX_CFG4_DATA
ommand_ready='0'
tx_data=cfg4[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg4_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready='1' &
byte_index <= 1
WAIT_AX_CFG4
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg4_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'


WR_AX_CFG5_DATA
ommand_ready='0'
tx_data=cfg4[(byte_index*8-1)-:8]
tx_enqueue=tx_ready
tx_len=tx_ax_cfg4_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand=(byte_index>1) ? '0' : (tx_ready ? '1' : '0')
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ready ? byte_index-1 : byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0' | byte_index > 1
tx_ready
WR_AX_CFG5_ADDR
ommand_ready='0'
tx_data=adxl362_addr_cfg5
tx_enqueue=tx_ready
tx_len=tx_ax_cfg5_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=tx_ax_cfg5_length-2
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
WR_AX_CFG5_CMD
ommand_ready='0'
tx_data=adxl362_cmd_write
tx_enqueue=tx_ready
tx_len=tx_ax_cfg5_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1' &
byte_index <= 1
WAIT_AX_CFG5
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_cfg5_length
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'
IDLE1
ommand_ready=spi_idle
tx_data=h00
tx_enqueue='0'
tx_len=0
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
cmd_start_linked_mode='0' &
cmd_start_measur_mode='0'
cmd_start_linked_mode='1'
WAIT_AT_INT1
(following pages, not yet drawn)
cmd_start_linked_mode='0' &
cmd_start_measur_mode='1'
WAIT_DR_INT1
(following pages)

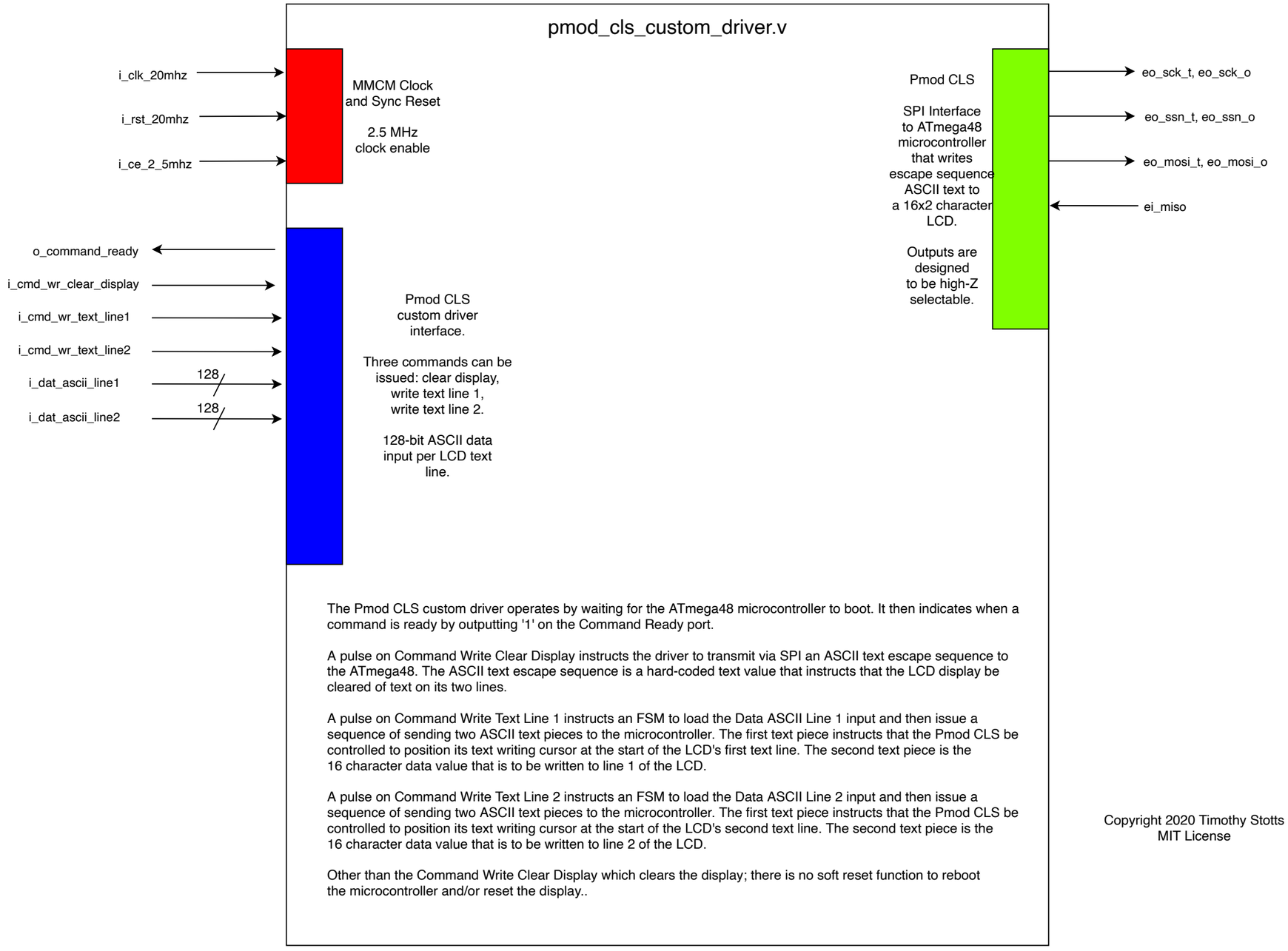

IDLE1
(from previous page)
SOFTRESET_CMD
(in following pages)
cmd_start_linked_mode='0' &
cmd_start_measur_mode='1'
cmd_soft_reset_acl2='1'
WAIT_DR_INT1
ommand_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=0
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
int1
READ_MEASUR_CMD
command_ready='0'
tx_data=adxl362_cmd_read
tx_enqueue=tx_ready
tx_len=tx_ax_readmm_length
rx_len=rx_ax_readmm_length
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='1'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
READ_MEASUR_ADDR
command_ready='0'
tx_data=adxl362_addr_8reg
tx_enqueue=tx_ready
tx_len=tx_ax_readmm_length
rx_len=rx_ax_readmm_length
wait_cyc=0
rx_dequeue='0'
go_stand=tx_ready
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='1'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
READ_MEASUR_DATA
command_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_readmm_length
rx_len=rx_ax_readmm_length
wait_cyc=0
rx_dequeue=rx_avail
go_stand=tx_ready
rd_data_stream=rx_data
rd_data_byte_valid=rx_valid
rd_data_group_valid='1'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
rx_valid='0' |
byte_index > 1
rx_valid='1' &
byte_index <= 1
READ_WAIT0
command_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_readmm_length
rx_len=rx_ax_readmm_length
wait_cyc=0
rx_dequeue=rx_avail
go_stand=tx_ready
rd_data_stream=rx_data
rd_data_byte_valid=rx_valid
rd_data_group_valid='1'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'
CLEAR_MEASU_CMD
command_ready='0'
tx_data=adxl362_cmd_read
tx_enqueue=tx_ready
tx_len=tx_ax_clearmm_length
rx_len=rx_ax_clearmm_length
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream='0'
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'


tx_ready='1'
CLEAR_MEASU_ADDR
command_ready='0'
tx_data=adxl362_addr_stat
tx_enqueue=tx_ready
tx_len=tx_ax_clearmm_length
rx_len=rx_ax_clearmm_length
wait_cyc=0
rx_dequeue='0'
go_stand=tx_ready
rd_data_stream='0'
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
tx_ready='0'
tx_ready='1'
CLEAR_MEASU_DATA
command_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=tx_ax_clearmm_length
rx_len=rx_ax_clearmm_length
wait_cyc=0
rx_dequeue=rx_avail
go_stand='0'
rd_data_stream='0'
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=rx_valid ? byte_index-1 : byte_index
reg_status=rx_valid ? rx_data : reg_status
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
rx_valid='0'
rx_valid='1'
READ_WAIT1
command_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=0
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
spi_idle='0'
spi_idle='1'
READ_WAIT2
command_ready='0'
tx_data=h00
tx_enqueue='0'
tx_len=0
rx_len=0
wait_cyc=0
rx_dequeue='0'
go_stand='0'
rd_data_stream=h00
rd_data_byte_valid='0'
rd_data_group_valid='0'
byte_index=byte_index
cfg0=cfg0
cfg1=cfg1
cfg2=cfg2
cfg3=cfg3
cfg4=cfg4
cfg5=cfg5
reg_status=reg_status
int1='1'
int1='0'
WAIT_DR_INT1
(from previous page)



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 only one SPI slave on the bus.



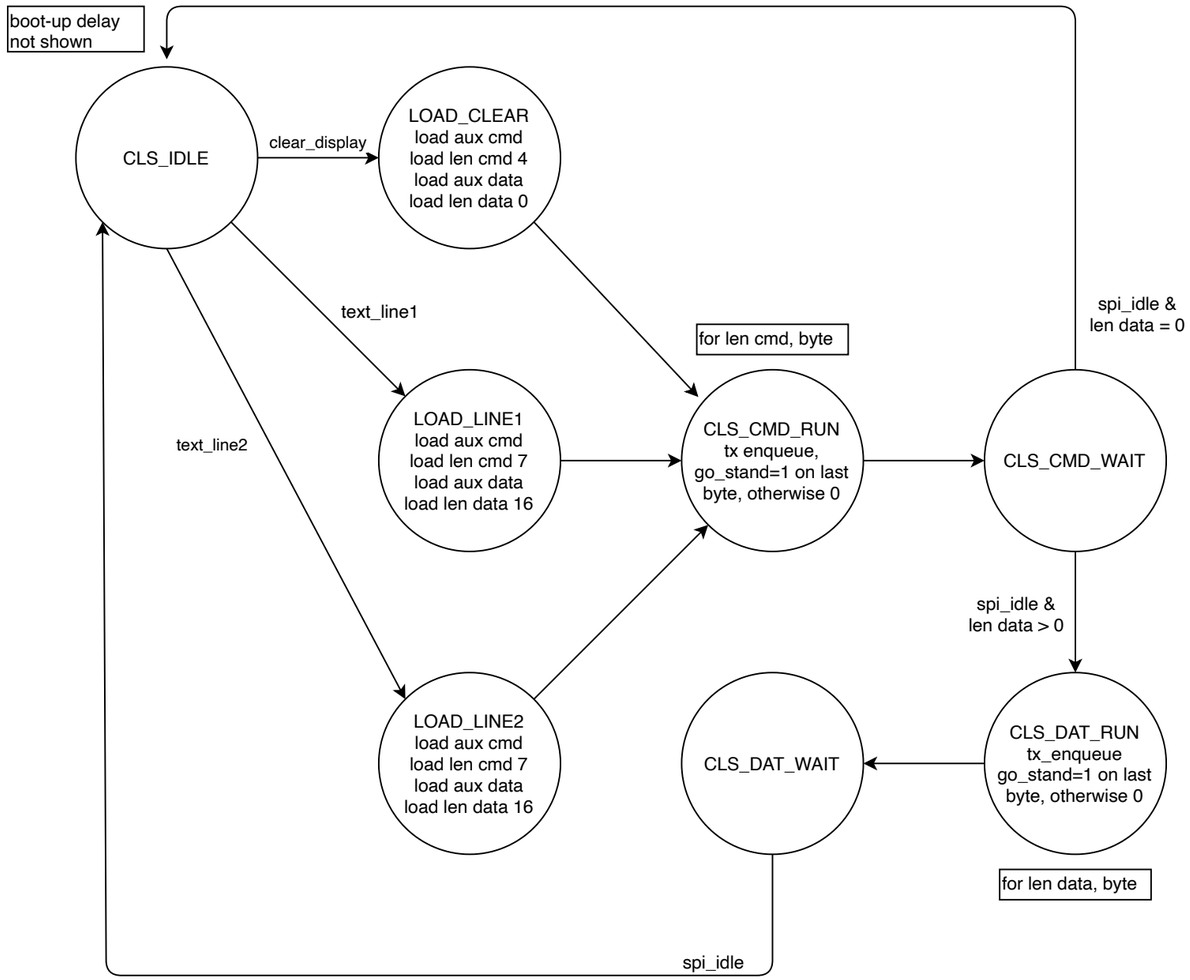
The Pmod CLS custom driver operates by waiting for the ATmega48 microcontroller to boot. It then indicates when a command is ready by outputting '1' on the Command Ready port.

A pulse on Command Write Clear Display instructs the driver to transmit via SPI an ASCII text escape sequence to the ATmega48. The ASCII text escape sequence is a hard-coded text value that instructs that the LCD display be cleared of text on its two lines.

A pulse on Command Write Text Line 1 instructs an FSM to load the Data ASCII Line 1 input and then issue a sequence of sending two ASCII text pieces to the microcontroller. The first text piece instructs that the Pmod CLS be controlled to position its text writing cursor at the start of the LCD's first text line. The second text piece is the 16 character data value that is to be written to line 1 of the LCD.

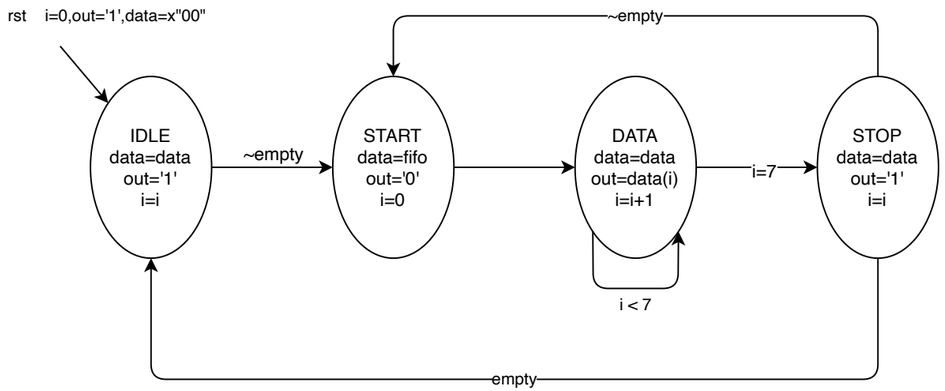
A pulse on Command Write Text Line 2 instructs an FSM to load the Data ASCII Line 2 input and then issue a sequence of sending two ASCII text pieces to the microcontroller. The first text piece instructs that the Pmod CLS be controlled to position its text writing cursor at the start of the LCD's second text line. The second text piece is the 16 character data value that is to be written to line 2 of the LCD.

Other than the Command Write Clear Display which clears the display; there is no soft reset function to reboot the microcontroller and/or reset the display..

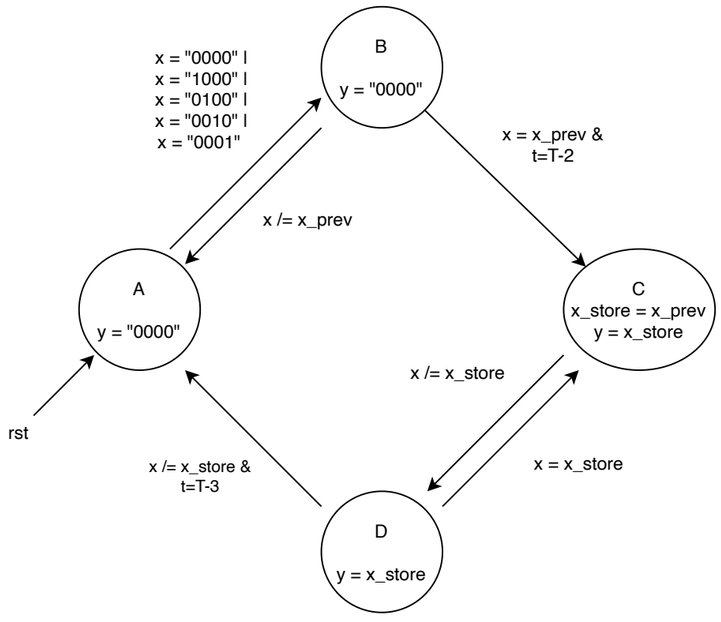


A FSM to operate the Digilent Inc. PMOD CLS LCD display communication via the single slave SPI-machine FSM of this document.

This diagram is
incomplete and does not
show boot-time delay. Also, some
state-bypass preventions
and iterations may not be shown.



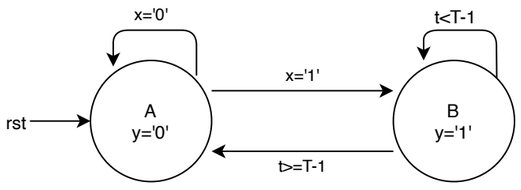
A TX ONLY UART output to UART chip from the FPGA, with the FSM executing at BAUD rate as its clock enable.



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.



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