4/22

- Explicitly connecting all wires being sniffed from ps, smart connect, and sniffer stops freezing the board, and pm commands work again
- A single pm to target address only moves sniffer to IDLE state, additional pms don't leave idle state
    - Assumption because counters are still 0 and state is 1
    - pms *do* get through to BRAM so transaction process is occuring
    - IDLE to RESET works by turning off timer_enable

4/23

- Last may stay high even after transfer is finished, but one of the 3 signals will be off
    - Does last stay on whole time or is transfer too fast?
    - So far flag is saying off to last state

4/26

- Usual burst size 2, burst type 1
- Usual burst length = 0x1F
    - Can be less if transferring less than a "page"
    - But transferring 0x_00, like for example transferring 0xa00 bytes -> last transaction address is 0x980
        - Implies 128 bytes in transaction for 0x1F burst length
            - burst size ⇒ $2^2 = 4$ ⇒ 4 bytes per transfer
            - If burst length ⇒ 32 ⇒ 32 transfers per transaction
            - 128 bytes = $4 * 32$
                - Currently in vivado CDMA data width is 32 bits and burst size is 32, seems to check out
    - [0xb0028000, 0xb0028300) seem like a single transaction though
        - Any x100 range from 0xb0028300 and up shows up as expected
    - Example 0x320 BTT from 0xb0028000 to 0xb0028300
        - Last transaction address is 0xb0028300, but burst length is 0x7
            - burst size ⇒ $2^2 = 4$
            - If burst length ⇒ 8
            - Total bytes in last transaction⇒ $4 * 8 = 32 = 0x20$
- on_count is never set ⇒ off state never occurs
    - since we're testing w/o any interference ⇒ transfers happen directly after each other?

4/27

- https://www.intel.com/content/www/us/en/docs/programmable/683130/22-2/axi-interface-timing-diagram.html ???
- Transaction count ok
    - 2 transactions for 0xb0028400 to 0xb0028500
    - 4 transactions for 0xb0028400 to 0xb0028600
- Transfer count not
    - 0x2f for 0xb0028400 to 0xb0028500, expected 0x3f or 0x40
    - 0x5f for 0xb0028400 to 0xb0028600, expected 0x7f or 0x80
    - 0x17 for 0xb0028400 to 0xb0028420, expected 0x7 or 0x8
    - ⇒ Counting every valid&ready signal like the TA's over counts within a transaction, but something happens between transactions that causes undercount
- Uptime values -- since on_stamp is always 0, next bit file will increment uptime while in on state instead of doing difference when leaving on state
    - For 0xb0028400 to 0xb0028500
        - Transaction 1 ends at 0x17
        - Transaction 2 ends at 0x45
    - For 0xb0028400 to 0xb0028600
        - Transaction 1 ends at 0x17
        - Transaction 2 ends at 0x45
        - Transaction 3 ends at 0x8a
        - Transaction 4 ends at 0xe6
    - For 0xb0028400 to 0xb0028420
        - Ends at 0x17
- Uptime values after changing uptime logic - 0x17 per transaction, i.e. 0x17, 0x2e, 0x45, 0x5c, …
- Putting a condition to jump directly to COUNT_ON instead of WAIT if valid&wready increases transfer count
    - ⇒ the intel timing diagram may be what's actually happening
    - Additionally if cdma is starting at any other address, the same thing occurs where [0xb0028_00, 0xb0028(_+3)00) shows up as a single transaction
        - ⇒ state machine may be built on false assumption, next bit file will check transaction and transfer count outside of state machine and see
- Transaction count outside state machine catches the [0xb0028_00, 0xb0028(_+3)00) cases, transfer count higher than expected
    - Transfer count catching other transfers, seen from it increasing after dming the sniffer again
- Since transfer count can be calculated user side from burst info register, removing transfer count for now
    - No reliable way to get transfer timings without this, but maybe can still do stuff with transactions
    - Changing on_stamp and off_stamp to transaction_start and transaction_end?

- Throwing out the state machine and just checking signals gets the right numbers of transactions and transfers
    - To accommodate, new types of data recorded
        - Transaction start - when an address in range is valid and ready
        - Transaction end - when last signal is up (and ∃ unfinished transaction)
        - Transfer time stamp - take time stamp when wvalid and wready during transaction

4/28

- Signals/data options for write diagram
    - Clk ✅
    - Awaddr ✅
    - Burst ✅
        - Awlen
        - Awsize
        - Awburst
    - Awvalid ✅
    - Awready ✅
    - Wdata ✅ supporting only 128 bits data width for now
    - Wstrb ✅ if 128 data width max, 16 bit max
    - Wlast ✅
    - Wvalid ✅
    - Wready ✅
    - Bresp ✅
    - Bvalid ✅
    - Bready ✅
- Current slv_reg reads/input

Reg0	0xa0070000	Enter Start Address
Reg1	0xa0070004	Enter End address
Reg2	0xa0070008	Timer enable bit [0]
Reg3	0xa007000c	Total Transaction count
Reg4	0xa0070010	Bresp count
Reg5	0xa0070014	Total Transfer count
Reg6	0xa0070018	Enter transaction # for data
Reg7	0xa007001c	Enter transfer # for data
Reg8	0xa0070020	Enter clk # for (signal) data
Reg9	0xa0070024	Count/total clk cycles

Reg10	0xa0070028	awaddress[R6]
Reg11	0xa007002c	Burst info[R6]
Reg12	0xa0070030	Bresp[R6]
Reg13	0xa0070034	chunk_wdata[R7]
Reg14	0xa0070038	wstrb[R7]
Reg15	0xa007003c	{24'hFACADE, all_bready[R8], all_bvalid[R8], all_wready[R8], all_wvalid[R8], all_wlast[R8], all_awready[R8], all_awvalid[R8]};
Reg16	0xa0070040	Enter wdata_chunk

4/30

- Cases that can break the grapher
    - Sniffer will not record address that are out of range
        - I.E. if we're looking for 0xa000 to 0xb000, if the order is 0xa000, 0xc000, 0xb000, then 0xc000 isn't recorded
            - But the aw signals for 0xc000 may still be recorded
        - The sniffer currently does not do place holders for this scenario, so the grapher will grab 0xb000 since it is the next transaction address
            - Thus the grapher rn can only check transactionNum < totalTransaction
            - This most likely also affects transfers
        - Possible solution to this problem is to have the sniffer record time stamps of **all** valid transactions start, end, and transfers, which would require a lot of additional hardware resources
        - An alternative solution is to throw out range functionality and observe all traffic after a given start address, or even just after enabling, and just keep recording until full
    - Another problem for graphing is trying to sniff in between, because catching tail end of preceding transaction
        - Can be mitigated by same suggestions above
    - Something weird happens when trying a range that cuts off in the middle of a transaction