SoC Projekt MitM Attacks Hardwareaufbau (intern)

Allgemeines:

Die Datenbearbeitungspipeline besteht aus 4 groben Blöcken:
1) Daten vom RX_PHY einlesen (Raphael)
2) Daten verzögern zum Vergleichen und Verteilen (Martin)
3) AHBL Busmaster und 8kB AHBL SRAM (singleport) (Ingo)
4) Daten zum TX_PHY senden (Frederick)

Zur Erkennung von Paketende und Anfang wird das Data_Valid Signal des PHY durch die gesamte Hardwarepipeline als FRAME Signal durchgeleitet. Damit ein Paketende erkannt werden kann müssen FRAME Signal erzeugende Komponenten noch ein Nullbyte in den zugewiesenen Speicher schreiben mit dem FRAME Signal auf low.

Over oder Underruns der ASYNC FIFOs auf PHY Seite setzen das FRAME Signal auf low und sorgen für ein defektes Paket welches von externer Ethernethardware gedroppt wird.

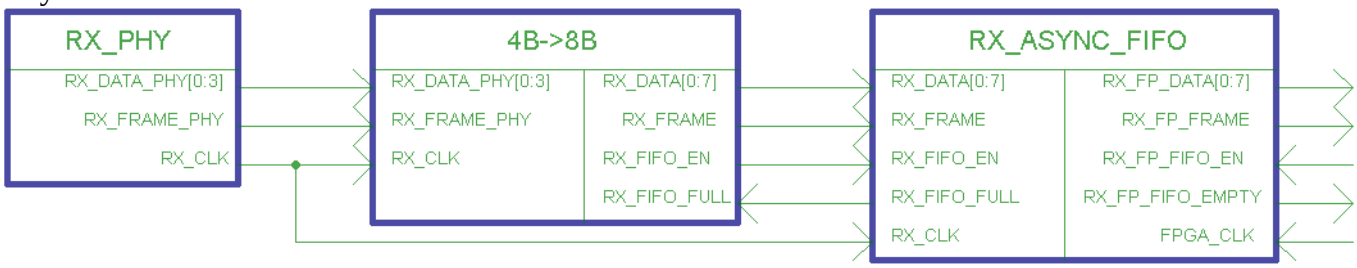
Pakete werden vollständig durch die Pipelines übertragen, also mit Präambel und CRC Checksumme. Die Checksumme wird nur bei veränderten Paketen neu berechnet.
Packete mit VLAN Tags werden nicht berücksichtigt (sorgen aber für nettes Fehlverhalten).

Resetsignale sind nicht beschrieben, es versteht sich von selbst, dass Reset benötigt werden.
Auf die Polarität im Projekt achten!

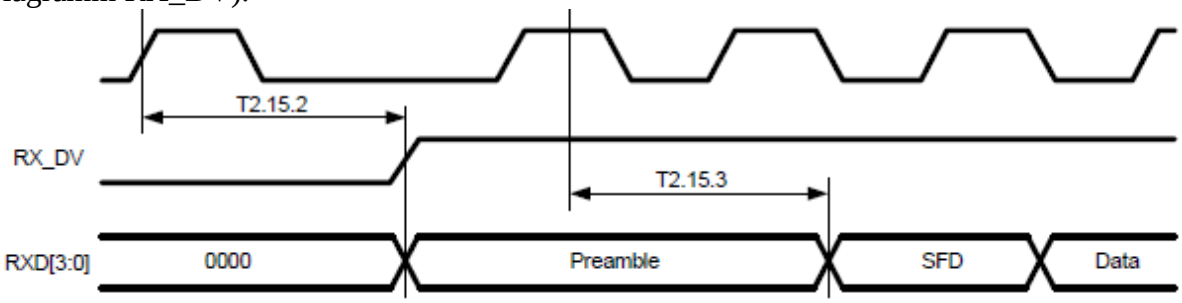
Die angegebenen Signalnamen sind verbindend um später in Simulationen alles wiederzufinden.

Block 1: Daten vom RX_PHY einlesen

Dieser Block unterteilt sich in den RX_PHY selbst, einem 4 Bit zu 8 Bit Konverter und einer Asynchronen FIFO zu 9 Bit Datenbreite.



Der RX_PHY Block stellt nur die Signale RX_DATA_PHY[0:3], RX_FRAME_PHY und RX_CLK bereit.
Dabei liegt die RX_CLK dauerhaft an und beträgt 25MHz.
Insofern die PHY Daten empfängt geht das RX_FRAME_PHY Signal auf high (Im Diagramm RX_DV).



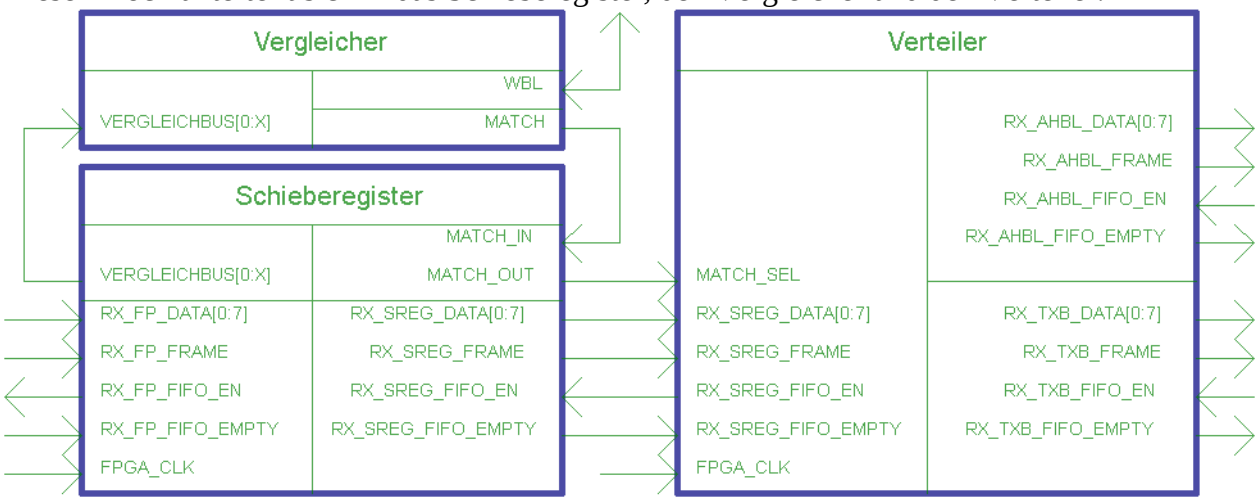
Da die Daten als 4 Bit Nibble vorliegen und intern als 8 Bit verarbeitet werden, müssen diese vom **4B/8B Konverter** aneinander gereiht werden.
Dieser Block besitzt als Inputs RX_CLK, RX_FRAME_PHY, RX_FIFO_FULL und RX_DATA_PHY[0:3].
Als Outputs besitzt dieser RX_FRAME, RX_FIFO_EN und RX_DATA[0:7].

Insofern das RX_FRAME_PHY Signal high ist setzt dieser Block immer 2 Nibbles zu einem Byte zusammen und schreibt dieses in die RX_ASYNC_FIFO, dabei wird das RX_FRAME Signal als Flag mit in die RX_ASYNC_FIFO geschrieben. Wenn RX_FRAME_PHY von high auf low wechselt muss noch ein Nullbyte in die RX_ASYNC_FIFO geschrieben werden. Denn das RX_FRAME Signal wird als Flag in der RX_ASYNC_FIFO mitgespeichert und so wird sichergestellt, dass der auslesende aus der RX_ASYNC_FIFO auch mitbekommt, dass ein Paket zu Ende ist. Sollte während des Nullbyte schreiben die RX_ASYNC_FIFO voll werden/sein so muss gewartet werden.
Sollte es vorkommen, dass während eines RX_FRAME_PHY high die RX_ASYNC_FIFO voll ist so wird so vorgegangen als währe das Paket zu Ende und zusätzlich werden alle Nibbles die vom RX_PHY kommen gedroppt bis zum nächsten RX_FRAME_PHY high.

Die RX_ASYNC_FIFO nimmt die Daten vom Konverter entgegen zum Speichern und Übertragen über die Clockdomäne.
In der RX_CLK Domäne besitzt die RX_ASYNC_FIFO die Eingänge RX_CLK, RX_FRAME, RX_FIFO_EN und RX_DATA[0:7]. Sowie den Ausgang RX_FIFO_FULL.
In der FPGA_CLK Domäne besitzt die FIFO die Eingänge RX_FP_FIFO_EN und FPGA_CLK. Sowie die Ausgänge RX_FP_FIFO_EMPTY, RX_FP_FRAME und RX_FP_DATA[0:7].

Block 2: Daten verzögern zum Vergleichen und Verteilen

Dieser Block unterteilt sich in das Schieberegister, den Vergleicher und den Verteiler.



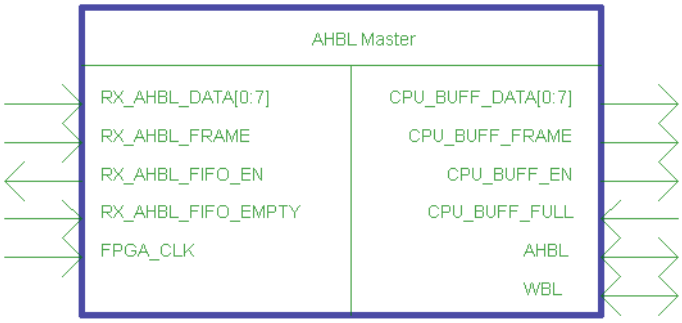
Das Schieberegister parallelisiert die Daten damit diese verglichen werden können.
Dieses Schieberegister hat die Länge von Präambel, SFD, Ethernet II Header und dem IP Header. Gleichzeitig werden die Daten auch verzögert mit der Verteiler die Daten erst weiterleitet wenn entschieden ist wer diese bekommt.
Dabei verhält sich das Schieberegister wie eine Synchrone FIFO um für den Rest der Datenpipeline durchsichtig zu sein. Also am Ausgang stehen dieselben Signale zur Verfügung wie bei der FPGA_CLK Seite der RX_ASYNC_FIFO. Diese tragen nur nicht mehr den Identifier _FP_ sondern _SREG_.
Zusätzlich dazu noch die Signale MATCH_IN, MATCH_OUT und VERGLEICHBUS[0:X].
Dabei müssen folgende Fälle berücksichtigt werden:
1) Es kommt ein neues Packet welches in das Schieberegister eingetaktet wird und noch nicht zum Ausgang vorgedrungen ist.
2) Der Anfang des Paketes aus 1) liegt nun vollständig im Schieberegister und MATCH_IN wird gespeichert und mit MATCH_OUT dem Verteiler mitgeteilt.
3) Das Paket wird durch das Schieberegister geschoben.
4) Die ASYNC_FIFO am Eingang meldet das Ende des Paketes und ein Teil ist noch im Schieberegister -> FIFO nicht weiter auslesen, aber weiter schieben.
5) Die FIFO enthält wieder ein neues Paket aber das Ende des alten Paketes liegt noch im Schieberegister.

Der Vergleicher bekommt vom Schieberegister den Ethernet II Header und den IP Header.
Als Eingänge besitzt dieser FPGA_CLK, VERGLEICHBUS[0:X] und WBL_I.
Als Ausgänge besitzt dieser MATCH und WBL_O.
Der Vergleicher besitzt zwei X Bit lange Bitmasken unterteilt in 32 Byte breite Register.
Eine Bitmaske enthält die Bits die gleich sein Müssen mit den Headern und eine Bitmaske enthält die Bits die beschreiben ob das Bit aus dem Header mit der Bitmaske verglichen oder ignoriert wird. Ist ein Vergleich erfolgreich so wird das MATCH Signal auf high gesetzt, sonst low. Es wird angestrebt, dass ein Vergleicher 2 oder 4 Registerbänke für die 2 Bitmasken hat. Weiterhin ist es möglich mehrere Vergleicher anzuschließen.

Der Verteiler schaltet den Ausgang je nach Wert von MATCH_OUT auf den AHBL Master oder auf den SYNC_TX_BUFF.
Bei MATCH_OUT high wird das Signal auf den AHBL Master durchgereicht sonst auf den SYNC_TX_BUFF. Die nicht ausgewählte Baugruppe bekommt eine leere FIFO vorgegaukelt.
Das MATCH_OUT Signal muss für eine ganze Paketlänge stabil bleiben.

Block 3: AHBL Busmaster und 8kB AHBL SRAM

Dieser Block unterteilt sich in den AHBL Master und den AHBL Slave SRAM mit 8kB.



Der SRAM wird am Student AHBL Slave Port angeschlossen und stellt 8kB SRAM bereit welcher aus BlockRAM des FPGA besteht. Dieser SRAM muss unbedingt 8/16/32 Bit Zugriffe unterstützen.

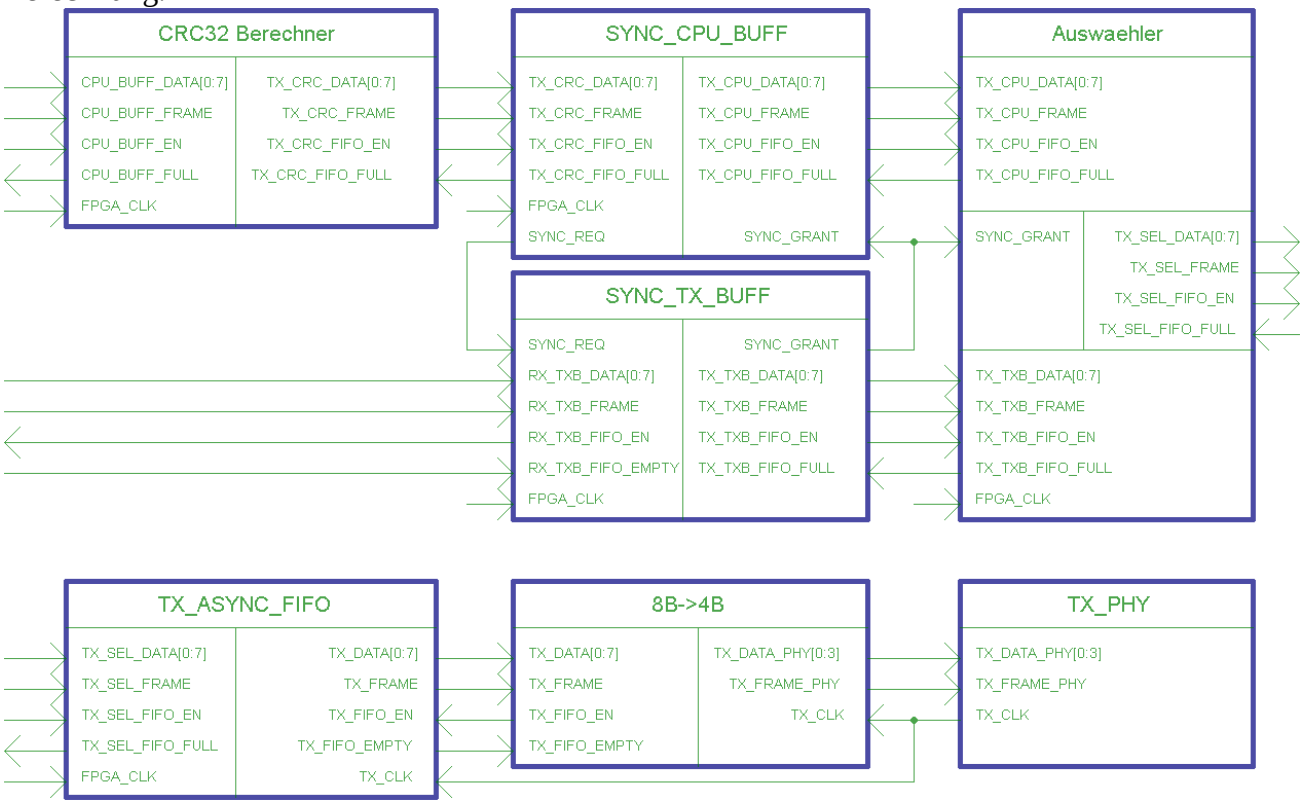
Der AHBL Master schreibt Packete in den SRAM oder ließt diese aus.
Als Eingänge besitzt dieser die Ausgänge der ASYNC_FIFO welche durch Block 2 durchgereicht sind und den Identifier _AHBL_ haben. Weiterhin noch den AHBL Master Anschluss und einen WBL Slave Anschluss.
Als Ausgang besitzt der AHBL Master einen Port um die SYNC_CPU_BUFF zu beschreiben bestehend aus CPU_BUFF_EN, CPU_BUFF_FRAME, CPU_BUFF_DATA[0:7] und CPU_BUFF_FULL.
Sowie den Interrupts für neues Packet im Buffer (IRQ_NEW_PACKET), Paket senden fertig (IRQ_PACKET_SEND) und memmove fertig (IRQ_MEMMOVE_DONE).

Dabei muss das CPU_BUFF_FRAME Signal vom AHBL Master beim Senden neu erzeugt werden da dieses beim schreiben des Paketes im SRAM verloren geht. Da das Paket sowieso von der CPU verändert wird. Weiterhin muss beim senden darauf geachtet werden, dass nach dem FRAME auf low geschalten wurde noch ein Nullbyte in die FIFO geschrieben wird mit FRAME auf low damit die nachfolgende Hardware das Paketende erkennt.
Der AHBL Master sollte optional ein memmove unterstützen, falls die CPU im Paket Daten verschieben muss.
Insofern vorhanden speichert der AHBL Master ein Paket welches er aus Block 2 ausließt im SRAM und wirft einen IRQ wenn der Vorgang beendet ist. Ein Paket fängt an wenn RX_FP_FRAME auf high wechselt und ein Paket ist zu Ende wenn RX_FP_FRAME auf low wechselt. Während des Kopiervorgangs kann es passieren, dass die FIFO empty anzeigt, aber ein Paketende wird erst von RX_FP_FRAME low signalisiert.
Ein im SRAM gespeichertes Paket wird angeführt von einem Struct aus Pointern welches auf die wichtigen Bereiche des Paketes zeigen (Packetlänge in Bytes, MAC Adressen, Packetype, Datenbereicharray[1500] sowie ein Pointer für pinkmagic).
Die Struct Basisadresse muss wenn der IRQ geworfen wurde per WBL auslesbar sein.

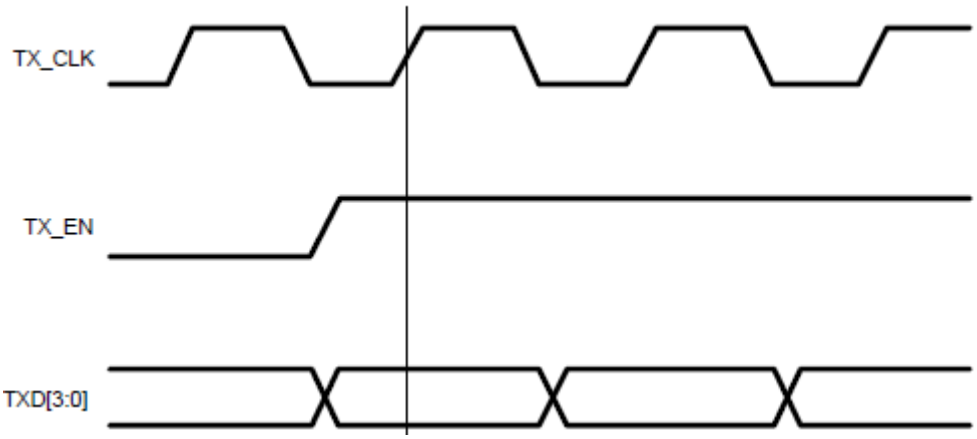
Per WBL ist im AHBL Busmaster einstellbar auf welchen 4 Adressen er Packetroundrobin machen soll (2k aligned). Zudem ist per WBL mitzuteilen von welchem der Slots ein Packet versendet werden soll (struct Basisadresse), ein schreiben der Basisadresse löst sofort das Senden des Pakets aus, die Länge wird von der CPU in das struct geschrieben.

Block 4: Daten zum PHY senden

Dieser Block besteht aus dem TX_PHY, einem 8Bit zu 4Bit Konverter, einer TX_ASYNC_FIFO, einem SYNC_TX_BUFF und dem SYNC_CPU_BUFF mit CRC32 Berechnung.



Der TX_PHY Block stellt nur die Signale TX_DATA_PHY[0:3], TX_FRAME_PHY und TX_CLK bereit. Dabei liegt die TX_CLK dauerhaft an und beträgt 25MHz.
Insofern die PHY Daten senden soll muss das TX_FRAME_PHY Signal auf high gehen (im Diagramm TX_EN).



Der 8Bit zu 4Bit Konverter zerlegt die Bytes aus der TX_ASYNC_FIFO wieder in 2 Nibbles.

Dieser Block besitzt als Inputs TX_CLK, TX_FRAME, TX_FIFO_EMPTY und TX_DATA[0:7].
Als Outputs besitzt dieser TX_FRAME_PHY und TX_DATA_PHY[0:4].

Sollte es beim senden zu einem TX_ASYNC_FIFO underrun kommen (TX_FIFO_EMPTY auf low) so wird der Sendevorgang abgebrochen und alle weiteren Daten aus der TX_ASYNC_FIFO werden gedroppt bis zum nächsten Paket. Also TX_FRAME_PHY auf low schalten und solange TX_FRAME auf high ist alle Daten aus der TX_ASYNC_FIFO verwerfen.

Die TX_ASYNC_FIFO nimmt die Daten vom SYNC_TX_BUFF oder vom SYNC_CPU_BUFF entgegen zum Speichern und Übertragen über die Clockdomäne.

In der TX_CLK Domäne besitzt die TX_ASYNC_FIFO die Eingänge TX_CLK, und TX_FIFO_EN. Sowie die Ausgänge TX_FIFO_EMPTY, TX_DATA[0:7] und TX_FRAME.
In der FPGA_CLK Domäne besitzt die FIFO die Eingänge TX_FP_FIFO_EN, TX_FP_FRAME, FPGA_CLK und TX_FP_DATA[0:7]. Sowie den Ausgang TX_FP_FIFO_FULL.

Hierbei handelt es sich um eine normale Asynchrone FIFO, es muss also nicht mehr dazu gesagt werden.

Der Auswähler entscheidet mithilfe des SYNC_GRANT Signals welcher Buffer in die TX_ASYNC_FIFO schreiben kann. Ist SYNC_GRANT high so schreibt der SYNC_CPU_BUFF in die TX_ASYNC_FIFO, ansonsten der SYNC_TX_BUFF. Dem nicht ausgewählten Block wird eine volle FIFO vorgegaukelt.

Der SYNC_TX_BUFF ließt die FIFO aus Block 2 aus und leitet diese an die TX_ASYNC_FIFO weiter.

Am Eingangsport besitzt der Block die Signale <siehe Blockschaltbild>
Am Ausgangsport besitzt der Block die Signale <siehe Blockschaltbild>

Weiterhin hat der Block das Ausgangssignal SYNC_GRANT und das Eingangssignal SYNC_REQ.
Ist das Signal SYNC_REQ auf high so darf das Paket noch zu Ende gesendet werden und danach wird SYNC_GRANT auf high gesetzt. Ist SYNC_REQ auf low so ist auch SYNC_REQ auf low.
Ist das TX_FP_FIFO_FULL high so wird gewartet.

Der SYNC_CPU_BUFF bekommt vom AHBL Master seine Daten und leitet diese an die TX_ASYNC_FIFO weiter.

Am Eingangsport besitzt der Block die Signale <siehe Blockschaltbild>
Am Ausgangsport besitzt der Block die Signale <siehe Blockschaltbild>

Weiterhin hat der Block das Eingangssignal SYNC_GRANT und das Ausgangssignal SYNC_REQ. Besitzt der SYNC_CPU_BUFF genug Daten zum Senden so wird SYNC_REQ auf high gesetzt und gewartet bis SYNC_GRANT auf high ist, dann darf gesendet werden.
Ist das TX_FP_FIFO_FULL high so wird gewartet.

Die CRC32 Berechnung liegt nur im Pfad vom AHBL Master zum SYNC_CPU_BUFF und berechnet in Echtzeit die CRC32 Prüfsumme zum Ethernet II Frame.

Am Eingangsport besitzt der Block die Signale <siehe Blockschaltbild>

Am Ausgangsport besitzt der Block die Signale <siehe Blockschaltbild>

Wechselt das Signal CPU_BUFF_FRAME auf high so startet die CRC32 Berechnung. Diese Berechnung wird für jede FPGA_CLK durchgeführt bei der CPU_BUFF_EN high ist (also ein neues Byte anliegt). Wechselt CPU_BUFF_FRAME schließlich auf low so muss das Ausgangssignal CPU_BUFF_FRAME_OUT auf high gehalten werden und die 4 Prüfsummenbytes werden auf den Bus gelegt.