#!/tools/sw/perl/bin/perl -w

# Author Lutz Filor

# Phone 408 807 6915

#

# Purpose Analysis AHB-tracker tracefile

#

# Revision VERSION DATE Description

# ================================================================================================

# History 1.01.01 09/10/2018 Parsing trace files

# 1.01.02 09/14/2018 First report file, build data structure first check in

# Min, Max, Average

# 1.01.03 - 09/21/2018 Refactor parsing, analysis, reporting second check in

# split -> REGEX, Header parsing, Parsing.log

# 1.01.04 - 09/24/2018 Add and fix payload analysis, timebase, beat unaligned

# Next ADDR transfer report before concurrent DATA transfer

# 1.01.05 - 09/25/2018 Add transfer analysis, transfer, beat, WAIT states, IDLE,

# and concealed ADDR transfers

# 1.01.06 09/26/2018 Refactor latency analysis

# Refactoring latency\_analysis()

# Accounting 1st DATA, Last Data latency

# 3 1.01.07 09/27/2018 Update analysis report

# Debugging latency analysis()

# Refactoring bandwidth analysis()

# 4 1.01.08 09/28/2018 Document transfer pointer list/data structure

# Refactor build command/data object()

# Create elaborate\_tracefile()

# 5 1.02.01 10/01/2018 Add Command line interface

# Revise Analysis Report

# 6 1.02.02 10/01/2018 Refactor transaction\_analysis(),

# refactor report\_transaction()

# 7 1.02.03 10/02/2018

my $VERSION = "1.02.03";

use strict;

use warnings;

use feature 'state'; # Static local variables

use Readonly;

use Getopt::Long;

#use Padwalker; # Not available/installed

#use Package::Stash; # available

#use Pod::Usage; # available

#use DateTime; # Not used

sub usage;

sub report;

sub parse\_header;

sub calculate\_payload;

sub elaborate\_tracefile;

Readonly my $DIGITS => qr { \d+ (?: [.] \d\*)? | [.] \d+ }xms;

Readonly my $SIGN => qr { [+-] }xms;

Readonly my $EXPO => qr { [eE] $SIGN? \d+ }xms;

Readonly my $FLOAT => qr { ($SIGN?) ($DIGITS) ($EXPO?) }xms;

#

# Convert global variable to local state variable

#

my $ptr\_fst\_cmd = undef; # pointer to first command object/transaction

my $ptr\_ant\_cmd = undef; # pointer to antepenultimate command object/transaction

my $ptr\_pen\_cmd = undef; # pointer to penultimate command object/transaction <= SINGLE Transfer

my $ptr\_prv\_cmd = undef; # pointer to previous command object/transaction

my $ptr\_cur\_cmd = undef; # pointer to current command object/transaction

my $ptr\_nxt\_cmd = undef; # pointer to next command object/transaction

my $ptr\_lst\_cmd = undef; # pointer to last command object/transaction

my $win\_str\_ptr = undef; # pointer to start analysis window

my $win\_stp\_ptr = undef; # pointer to stop analysis window

my $ptr\_fst\_dta = undef; # pointer to first data obj

my $ptr\_prv\_dta = undef;

my $ptr\_cur\_dta = undef; # pointer to current data obj/transaction

my $ptr\_nxt\_dta = undef;

my $ptr\_lst\_dta = undef; # pointer to last data obj

my %defines; # Command line defines

my %pointer = ( first => \$ptr\_fst\_cmd, # Head of transaction list

previous => \$ptr\_prv\_cmd, # predecessor transaction

current => \$ptr\_cur\_cmd, # pointer actual transaction

next => \$ptr\_nxt\_cmd, # successor transaction

antepenultimate => \$ptr\_ant\_cmd, # third last

penultimate => \$ptr\_pen\_cmd, # second last

ultimate => \$ptr\_lst\_cmd, # last

last => \$ptr\_lst\_cmd, # Tail of transaction list

start => \$win\_str\_ptr, # Begin of analysis window

stop => \$win\_stp\_ptr, # End of analysis window

);

my %opts = ( #from => 0, # Command line options, default values

#until => 100,

window => 'window.log', # Default logname for determine the trace window within the trace file

);

my %Trace; # Trace file header information

my %AHB\_hash; # Trace file transfer information access AHB command transfers via time stamp

my %ahb\_transfer; # AHB transfer analysis new

my %ahb\_latency\_new; # AHB latency analysis new/refactored

my %ahb\_bandwidth\_new; # AHB bandwidth analysis new/refactored

my %ahb\_transaction\_new; # AHB transaction analysis new/refactored

my %ahb\_anal; # AHB transaction analysis

my %ahb\_band; # AHB bandwidth analysis

my %ahb\_latency; # AHB latency analysis

#my $filename = $ARGV[0]; # deprecated

#my $key\_value = $ARGV[1]; #

#my $access\_data = $ARGV[2];

my $parser\_log = 'parser.log'; # log parsing input file parsing

my $build\_log = 'build.log'; # log building of transaction tree, hash

my $progress\_log = 'progress.log'; # log progress phases of program

my $payload\_log = 'payload.log'; # log payload calculation as beat are error prone

my $transfer\_log = 'transfer.log'; # log transfers, type and duration

my $transaction\_log = 'transaction.log'; # log transaction, by direction, type and size

my $latency\_log = 'latency.log'; # log analysing transaction latency

my $bandwidth\_log = 'bandwidth.log'; # log analysing transaction bandwidth

my $pointer\_log = 'pointer.log'; # test logfile for point structure

my $ahb\_testlog = 'ahb\_test.log'; # access data via ahb\_hash{keys}

my $analysis = 'analysis.rpt'; # access data via pointer; report file

#################################################################################################################################

my $error\_log = 'error.log'; # log trace file reporting errors

#| 6127040 ps| 6132066 ps|RD-DATA | 84D5214C | 3/ 4 | -- | 00000000 | OKAY | ---- | ------- | - | ---------- |

#| 6132066 ps| 6137066 ps|WR-NONSEQ | 84D52332 | ----- | 16 | -------- | ---- |INCR16| 1 | 8 | 0|

#| 6132066 ps| 6137066 ps|RD-DATA | 84D52150 | 1/ 4 | -- | 00000000 | OKAY | ---- | ------- | - | ---------- |

#| 6137066 ps| 6142066 ps|WR-DATA | 84D52332 | 2/16 | -- | ..A3.... | OKAY | ---- | ------- | - | ---------- |

# ....

#| 6206870 ps| 6211870 ps|WR-DATA | 84D52340 | 16/16 | -- | ......EE | OKAY | ---- | ------- | - | ---------- |

#| 6211870 ps| 6216946 ps|WR-DATA | 84D52341 | 17/16 | -- | ....48.. | OKAY | ---- | ------- | - | ---------- |

#| 6216946 ps| 6221846 ps|RD-NONSEQ | 84D52248 | ----- | 16 | -------- | ---- |INCR16| 4 | 4 | 0|

#| 6221846 ps| 6226746 ps|RD-DATA | 84D52248 | 1/16 | -- | 00000000 | OKAY | ---- | ------- | - | ---------- |

#################################################################################################################################

# 'transfer.log # tbd handle for cvs file handle transfer log

my $trans\_log = 'transaction.log'; # tbd handle not used transaction log/command log

my $data\_tracer = 'ahb\_data.log'; # tbd handle for csv file handle transfer log

my $owner = 'Lutz Filor';

my ($s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst) = localtime;

open (my $parh, ">$parser\_log") || die " Can not create log $parser\_log";

open (my $buih, ">$build\_log") || die " Can not create log $build\_log";

open (my $proh, ">$progress\_log") || die " Can not create log $progress\_log";

open (my $payh, ">$payload\_log") || die " Can not create log $payload\_log";

open (my $trah, ">$transfer\_log") || die " Can not create log $transfer\_log"; # Are the bus transfers accounted for ? Mendatory

open (my $cmdh, ">$transaction\_log")|| die " Can not create log $transaction\_log"; # Split & sort transactions/commands by direction, type & size

open (my $lath, ">$latency\_log") || die " Can not create log $latency\_log";

open (my $banh, ">$bandwidth\_log") || die " Can not create log $bandwidth\_log";

open (my $errh, ">$error\_log") || die " Can not create log $error\_log"; # Report errors trace file formats

#################################################################################################################################

open (my $dath, ">$data\_tracer") || die " Can not create log $data\_tracer"; # If transported data becomes relevant,

open (my $tsth, ">$ahb\_testlog") || die " Can not create log $ahb\_testlog"; # Testing AHB command hash data structure

open (my $ptrh, ">$pointer\_log") || die " Can not create log $pointer\_log"; # To be deprecated

open (my $anah, ">$analysis") || die " Can not create log $analysis"; # To be deprecated

#

# main entry

#

#=================================================================================================================================

GetOptions ( 'help|h|?' => \&help,

'info' => \$opts{info},

'debug' => \$opts{debug},

'trace|t=s' => \$opts{trace},

'from=i' => \$opts{from},

'until=i' => \$opts{until},

); # Command Line Processor

if ( !defined $opts{trace} ) {

printf " ... command line option --trace=filename not found\n";

exit;

} else {

printf " ... tracefile :: %s\n", $opts{trace};

#open ( FILE , "<$filename") || die " Can not open input $filename";

open ( FILE , "<$opts{trace}") || die " Can not open input $opts{trace}";

#exit;

printf {$proh} " Author : %s\n", $owner;

printf {$proh} "\n";

printf {$proh} " ... opening %s\n", $opts{trace}; # $filename;

printf {$proh} " creating BuildLogFile %s\n", $build\_log;

printf {$proh} " creating ProgressFile %s\n", $progress\_log;

printf {$proh} " creating AnalysisReport %s\n", $analysis;

}

elaborate\_tracefile ( \%pointer ); # Parsing the tracefile and

# building 2D directed graph of ahb transfers

if ( defined $opts{info} ) {

report\_info ( \%Trace # Trace information from file Header

, \%pointer # Pointer information

, \%opts ); # Command line

exit;

}#

initialize\_transaction\_analysis ( \%ahb\_anal ); # Initialize reporting structure

initialize\_transaction\_analysis ( \%ahb\_transaction\_new );

initialize\_payload\_analysis ( $ptr\_fst\_cmd ); # Calculating payload, burst length

initialize\_transfer\_analysis ( $ptr\_fst\_cmd

, $ptr\_lst\_cmd

, $transfer\_log # logfilename

, \%Trace # Trace Header information

, \%ahb\_transfer );

initialize\_latency\_analysis ( \%ahb\_latency ); # To be deprecated

initialize\_latency\_analysis ( \%ahb\_latency\_new );

### finding\_trace\_window ( $ptr\_fst\_cmd # Pointer to trace data structure

### , $from # Staring point in time

### , $until ); # End point in time

latency\_analysis ( $ptr\_fst\_cmd

, \%Trace # Trace Header information

, \%ahb\_transfer

, \%ahb\_latency\_new );

bandwidth\_analysis ( $ptr\_fst\_cmd # Needs payload

# $stop\_ptr # ??? future

, \%Trace # Trade Header information

, \%ahb\_transfer

# \%payload # ??? future

, \%ahb\_bandwidth\_new

, \%ahb\_transaction\_new );

test\_AHB\_hash ( ); # Use of AHB-Transaction-HASH

transaction\_analysis ( \%Trace # Trace Header information

, \%AHB\_hash # Reference to global variable

, $transaction\_log # logfile name

, \%ahb\_transaction\_new ); # Transaction reporting, replacing %ahb\_anal

protocol\_trace\_analysis ( $ptr\_fst\_cmd

, \%ahb\_band

, \%ahb\_latency ); # To be deprecated

analysis\_report ( \%Trace # Trade Header information

, \%ahb\_band

, \%ahb\_latency

, \%ahb\_latency\_new

, \%ahb\_transfer

, \%ahb\_bandwidth\_new

# ahb\_anal # is a remaining global variable - to be deprecated !!

, \%ahb\_transaction\_new);

info\_report ( \%Trace

, \%pointer # all pointer

, \%opts );

#=================================================================================================================================

# End of main()

# Testing specified data structures

#test\_AHB\_hash(); # Testing data structure

#report(); # This report is for debugging purposes

#

# Subroutine implementation

#

#=================================================================================================================================

sub usage{

printf "\n";

#printf " Usage : <command> <tracker\_trace.file> <start\_time> <trace>\n";

#printf " parse\_ahb\_tracker.pl log\_file\_name key\_value access\_data [0..12] range\n";

printf " Usage : <command> <tracker\_trace.file> \n";

printf " parse\_ahb\_tracker.pl log\_file\_name \n";

printf "\n";

printf " Trace : Tracker-field selected by column# or Name <string> \n";

printf " ------------------------------------------------------------------------\n";

printf " Nonseq/Seq assert Time : 0 || nonseq\_assert\_time\n";

printf " HREADY assert Time : 1 || hready\_assert\_time\n";

printf " DIR PHASE : 2 || dir\_phase\n";

printf " ADDRESS : 3 || address\n";

printf " BEAT NUM : 4 || beat\_num\n";

printf " LEN : 5 || burst\_len\n";

printf " DATA : 6 || data\n";

printf " RESPONSE : 7 || response\n";

printf " BURST Type : 8 || burst\_type\n";

printf " BURST Size : 9 || burst\_size\n";

printf " HPORT : 10 || hport\n";

printf " Next command : 11 || next\_command\n";

printf " Previous command : 12 || prev\_command\n";

printf "\n";

exit 0;

}#sub usage

sub elaborate\_tracefile {

printf $proh " ... reading tracefile\n";

printf $proh " creating ParserLogFile %s\n", $parser\_log;

my @LINES = <FILE>;

for (my $i = 0; $i < $.; $i++)

{

my @COL;

my $row = $LINES[$i];

chomp $row;

my @line = split (/\|/, $LINES[$i]); # deprecated, maintained for backward verification

if ( $i <= 9) {

printf $buih " ... HEADER decoding\n" if ($i == 0);

printf $proh " ... HEADER decoding\n" if ($i == 0);

parse\_header( $row # input trace file row/line

, $i # row/line index/count

, \%Trace ); # output extracted trace file header information

printf $buih " line :: %5s :: %s\n", $i, $row;

printf $buih " ... HEADER decoding done\n" if ($i == 9);

printf $proh " ... HEADER decoding done\n" if ($i == 9);

}

if ( $i >= 10) {

printf $proh " ... TABLE decoding ... \n" if ($i == 10);

printf $buih " ... TABLE decoding ... \n" if ($i == 10);

printf $buih " line :: %5s :: %s\n", $i, $row;

parse\_body ( $row, \@COL ); # decode column data of transfer table rows

if ( $COL[4] =~ /NONSEQ/ ) { # PHASE Address

$AHB\_hash{$COL[0]} # hash of AHB command, w/ command ASSERT time as access keys

= build\_command\_obj( \@COL ); # addr transfers/AHB command objects, in double ptr list V.01.01.08

}# NONSEQ aka address phase or command

if ( $COL[4] =~ /DATA|BUSY/ ) { # PHASE Data

build\_data\_obj ( \@COL ); # data transfers --> via AHB address/cmd transfers

}# Data phase or Data

}# process trace table $i >= 10

}# for AHB trace tracker

close( FILE );

printf "\n";

#printf " ... done reading AHB trace tracker\n";

printf "\n";

printf $buih " ... TABLE decoding done\n";

printf $proh " ... TABLE decoding done\n";

}#sub

sub parse\_header{

my ( $row # input row Tables have rows and columns

, $lc # row/line count

, $trace\_header\_ref ) = @\_; # output extracted trace file header info

#

# Instance: uvm\_test\_top.top\_env\_i.ahbfab\_env\_i.mvc\_ahb\_mst\_5\_env\_i.logger\_handle\_ahb

#

if ($lc == 0) {

if ( $row =~ /^Instance:\s+([a-z0-9\_.]+)/x ) {

printf $parh " Picked : %s :: Header\n", $row;

printf $parh " Instance: %s\n", $1;

${$trace\_header\_ref}{tracefile} = $1; # extract trace filename

} else {

printf $parh " Skipped : %s\n", $row;

}

}# parse first line

#

# AHB Clk Cycle = 5 ns; AHB Clk Frequency = 200.00 MHz; Data bus width = 32 bits

#

elsif ($lc == 1) {

if ( $row =~ m{ ^(\w+)[ ]Clk[ ]Cycle [ ]=[ ]($FLOAT)[ ] (ns|ps) # <== Add Time base

;[ ](\w+)[ ]Clk[ ]Frequency[ ]=[ ]($FLOAT)[ ](MHz|GHz) # <== Add Frequency base

;[ ] Data[ ]bus[ ]width [ ]=[ ] (\d+) [ ] bits # increase readability w/ x-modifier

}x )

{

printf $parh " Picked : %s\n", $row;

printf $parh "%\*s%s Clk Cycle = %s %s; %s Clk Frequency = %3.2f %s; Data bus width = %s bits\n"

,13,'',$1, $2, $6, $7, $8, $12, $13;

${$trace\_header\_ref}{protocol} = $1; # (AHB)

${$trace\_header\_ref}{period} = $2; # (float)

${$trace\_header\_ref}{timebase} = $6; # (ns|ps)

${$trace\_header\_ref}{redundant} = $7; # (AHB)

${$trace\_header\_ref}{frequency} = $8; # (float)

${$trace\_header\_ref}{freqbase} = $12; # (MHz|GHz)

${$trace\_header\_ref}{databus} = $13; # bit width

}# regex

else {

printf $parh " Skipped : %s\n", $row;

$row =~ m{ ^(\w+)[ ]Clk[ ]Cycle [ ]=[ ]($FLOAT)[ ] (ns|ps)

;[ ](\w+)[ ]Clk[ ]Frequency[ ]=[ ]($FLOAT)[ ](MHz|GHz)

;[ ] Data[ ]bus[ ]width [ ]=[ ] (\d+) [ ] bits

}x ;

printf $parh "%\*s%s Clk Cycle = %s %s; %s Clk Frequency = %3.2f %s; Data bus width = %s bits\n"

,13,'',$1, $2, $6, $7, $8, $12, $13;

printf $parh " \$1 : %s\n", $1; # Protocol Period (AHB)

printf $parh " \$2 : %s\n", $2; # FLOAT

printf $parh " \$3 : %s\n", $3; # Sign

printf $parh " \$4 : %s\n", $4; # Digits

printf $parh " \$5 : %s\n", $5; # Exponent

printf $parh " \$6 : %s\n", $6; # Timebase

printf $parh " \$7 : %s\n", $7; # Protocol Frequency (AHB)

printf $parh " \$8 : %s\n", $8; # FLOAT

printf $parh " \$9 : %s\n", $9; # SIGN

printf $parh " \$10: %s\n", $10;# DIGITS

printf $parh " \$11: %s\n", $11;# Exponent

printf $parh " \$12: %s\n", $12;# Exponent

printf $parh " \$13: %s\n", $13;# Databus Width (bit)

}

}# parse second line

else {

printf $parh " Skipped : %s\n", $row;

}

}#sub parse\_header

sub parse\_body{

my ( $row # input row/line from trace file

, $a\_ref ) = @\_; # output extracted trace information

# The commented code documents a failing regex and how to improve the parsing code, control and extract

#if ( $LINES[$i] =~ m{\A # From start of line beginning with <|>

#if ( $row =~ m{#\A # From start of line beginning with <|>

#

# |\s+(\d+) ns # Column 2: Sample time HREADY Assert Time

# |(\w+)\s+ # Column 3: PHASE

# |\s([A-F0-9]{8})\s+ # Column 4: Address 32bit exact 8 Hex

# |\s([0-9\/\- ]+) # Column 5: Beat number, extract string w/ subset char

# |\s([0-9-]+) # Column 6: LEN/Length, extract string w/ char subset

# |\s([A-F0-9\-\.]) # Column 7: DATA, text string !! Not data

# |\s([A-F0-9\-)\s+ # Column 8: RESP/Response

# |(\w+) # Column 9: Burst Type SINGLE, WRAP4, INCR8 .. WRAP16

# |\s([0-9\-])\s # Column 10: Burst Size, text string !! Not data

# |\s([A-F0-9]{1}) # Column 11: HPROT, Hex number

# |\s+([0-9\-]+)\s\*| # Column 12: HSEL, Slave index text string !! Not data

# \z # Until end of line ending EOS Shall match \A usage

#COL 1 2 3 4 5 6 7 8 9 10 11 12

#| 620521 ns| 620531 ns|WR-NONSEQ | 3FF115E7 | ----- | 8 | -------- | ---- |WRAP8 | 1 | 3 | 0|

#| 625571 ns| 625581 ns|WR-NONSEQ | 3FF114F0 | ----- | ? | -------- | ---- |INCR | 2 | 8 | 0|

#| 6026708 ps| 6031684 ps|WR-NONSEQ | 84D522B8 | ----- | 16 | -------- | ---- |INCR16| 4 | B | 0|

if ( $row =~ m{^[|]\s+ (\d+) [ ] (ns|ps) # Column 1: Assert time SEQ Assert Time <== Parsing Extention timebase

[|]\s+ (\d+) [ ] (ns|ps) # Column 2: Sample time HREADY Assert Time <== Parsing Extention timebase

[|] ([A-Z-]+) [ ]+ # Column 3: PHASE

[|]\s ([A-F0-9]{8}) [ ]+ # Column 4: Address 32bit exact 8 Hex literals

[|]\s ([0-9/ ?-]{5}) [ ] # Column 5: Beat number, extract string w/ char subset <== Parsing Error [0-9\/\- ] class subset

[|]\s+ ([0-9-?]+) [ ] # Column 6: LEN/Length , extract string w/ char subset <== Parsing Error, \s+, missing ?

[|]\s ([A-F0-9\-\.]{8})[ ] # Column 7: DATA, text string !! Not data

[|]\s+ ([\-A-Z]+) [ ]+ # Column 8: RESP/Response

[|]\s\* ([-A-Z0-9]+) [ ]\* # Column 9: Burst Type SINGLE, WRAP4, INCR8 .. WRAP16, ' ---- '

[|]\s+ ([0-9\-]+) [ ]+ # Column 10: Burst Size, text string !! Not data

[|]\s ([-A-F0-9]{1}) [ ] # Column 11: HPROT, Hex number

[|]\s+ ([0-9\-]+)\s\* [|] # Column 12: HSEL, Slave index text string !! Not data

}xmsi # Allow comment, multi line (^,$), New line (.), case insensitive

) { # Parenterize correctly the matching condition

printf $parh " Picked : %s\n", $row;

push $a\_ref, ( $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14 ); # Index no longer match the columns

printf $parh "%\*s" , 13, ''; # Pre timebase tracking

printf $parh "|%20s %s" , $a\_ref->[ 0],$a\_ref->[ 1]; #$1 Assert time; $a\_ref->[ 0]

printf $parh "|%20s %s" , $a\_ref->[ 2],$a\_ref->[ 3]; # Complettime $a\_ref->[ 1]

printf $parh "|%-10s" , $a\_ref->[ 4]; #Transfer $a\_ref->[ 2]

printf $parh "| %-8s " , $a\_ref->[ 5]; #ADDR $a\_ref->[ 3]

printf $parh "| %5s " , $a\_ref->[ 6]; #BEAT $a\_ref->[ 4]

printf $parh "| %2s " , $a\_ref->[ 7]; #LEN $a\_ref->[ 5]

printf $parh "| %8s " , $a\_ref->[ 8]; #DATA $a\_ref->[ 6]

printf $parh "|%6s " , $a\_ref->[ 9]; #RESP $a\_ref->[ 7]

if ( $a\_ref->[10] =~ m/[-]/){ printf $parh "| %s " , $a\_ref->[10]; #BURST TYPE $a\_ref->[ 8]

} else { printf $parh "|%-6s" , $a\_ref->[10]; } #BURST TYPE $a\_ref->[ 8]

if ( $a\_ref->[11] =~ m/[-]/){ printf $parh "| %7s " , $a\_ref->[11]; #BURST SIZE $a\_ref->[ 9]

} else { printf $parh "| %4s ", $a\_ref->[11]; } #BURST SIZE $a\_ref->[ 9]

printf $parh "|%2s " , $a\_ref->[12]; #HPROT $a\_ref->[10]

if ( $a\_ref->[13] =~ m/[-]/){ printf $parh "| %s |\n" , $a\_ref->[13]; #HSEL $a\_ref->[11]

} else { printf $parh "|%12s|\n" , $a\_ref->[13]; } #HSEL $a\_ref->[11]

} else {

#

# Debug code, catch the failure and parse again

#

printf $parh " Skipped : %s\n", $row;

$row =~ m{^[|]\s+(\d+)[ ](ns|ps)

[|]\s+(\d+)[ ](ns|ps)

[|]([A-Z-]+) [ ]+

[|]\s([A-F0-9]{8}) [ ]+

[|]\s([0-9/ ?-]{5}) [ ] # Column 5: Beat# 5 characters [? /0-9]

[|]\s+([0-9-?]+) [ ]

[|]\s([A-F0-9\-\.]{8})[ ]

[|]\s+([\-A-Z]+) [ ]+

[|]\s\*([-A-Z0-9]+) [ ]\* # Column 9: Burst Type SINGLE, WRAP4, INCR8 .. WRAP16, ' ---- '

[|]\s+([0-9\-]+) [ ]+ # Column 10: Burst Size, text string !! Not data

[|]\s([-A-F0-9]{1}) [ ] # Column 11: HPROT, Hex number

[|]\s+([0-9\-]+)\s\* [|]

}ixms;

printf $parh " \$1 : %s\n", $1;

printf $parh " \$2 : %s\n", $2;

printf $parh " \$3 : %s\n", $3;

printf $parh " \$4 : %s\n", $4;

printf $parh " \$5 : %s\n", $5;

printf $parh " \$6 : %s\n", $6;

printf $parh " \$7 : %s\n", $7;

printf $parh " \$8 : %s\n", $8;

printf $parh " \$9 : %s\n", $9;

printf $parh " \$10: %s\n", $10;

printf $parh " \$11: %s\n", $11;

printf $parh " \$12: %s\n", $12;

printf $parh " \$13: %s\n", $13;

printf $parh " \$14: %s\n", $14;

}# tracefile parser

}#sub parse\_body

sub build\_command\_obj {

my ( $COL ) = @\_; # input column array with extracted values from tabel row

state $first\_cmd = 1; # Exception handling for first command transaction

my %command; # Create new cmd transfer obj:186

my $ptr\_cur\_cmd = \%command; # Pointer to command object/transaction

#=============================================================================================================

#

# ----> ptr\_fst\_cmd %AHB\_hash

# ----> FIRST

#

# ----------|

# | |

# | V ptr\_lst\_data

# prev ^ | | ptr\_fst\_data | ptr\_pre\_data

# | | | |

# | | V V

# ------------------------- |

# ----> ptr\_pen\_cmd | previous cmd object | |

# ----> ANTEPENULTIMATE |-----------------------| |

# | | -------------------- -------------------- -------------------- --------------------

# | | | prev | <---- | prev | <---- | prev | <---- | prev |

# | | |------------------| |------------------| |------------------| |------------------|

# | | | | | | | | | |

# |-----------------------| | | | | | | | |

# | data | ----> | | | | | | | |

# |-----------------------| | next | ----> | next | ----> | next | ----> | next | ----> ptr\_fst\_data

# | | |------------------| |------------------| |------------------| |------------------| !ptr\_lst\_data

# | |

# |-----------------------|

# | next cmd object |

# -------------------------

# |

# |

# V next

# prev ^

# |

# |

# -------------------------

# ----> ptr\_pre\_cmd | previous cmd object |

# ----> PREVIOUS |-----------------------|

# ----> PENULTIMATE | |

# | |

# | |

# | |

# | |

# |-----------------------|

# | data | ----> undefined

# |-----------------------|

# | |

# | |

# |-----------------------|

# | next cmd object |

# -------------------------

# |

# |

# V next

# prev ^

# |

# |

# -------------------------

# ----> ptr\_cur\_cmd | previous cmd object |

# ----> CURRENT |-----------------------|

# ----> LAST/ULTIMATE | |

# | |

# | |

# | |

# | |

# |-----------------------|

# | data | ----> undefined

# |-----------------------|

# | |

# | |

# |-----------------------|

# | next cmd object |

# ----> ptr\_lst\_cmd -------------------------

# |

# |

# V next

# ptr\_fst\_cmd

#

#=============================================================================================================

$ptr\_lst\_cmd = $ptr\_cur\_cmd; # No look ahead, current ptr is last ptr

$command{assert} = $COL->[ 0]; #$line[ 0]; # assertion time / addr\_assertion

$command{complete} = $COL->[ 2]; #$line[ 1]; # completion time / addr\_sampling

$command{direction} = $COL->[ 4]; #$line[ 2]; # direction READ/WRITE BUSY/DATA

$command{addr} = $COL->[ 5]; #$line[ 3]; # ADDRESS aligned, unaligned

#$command{beat\_num} NA 6 # BEAT

$command{burst\_len} = $COL->[ 7]; #$line[ 5]; # 1, 4, 8, 16, any >1

#$command{data} NA 8 # DATA strobed

#$command{response} NA 9 # Slave response

$command{burst\_type} = $COL->[10]; #$line[ 8]; # SINGLE,INCR,INCR4,WRAP4,INCR8,WRAP8,INCR16,WRAP16

$command{burst\_size} = $COL->[11]; #$line[ 9]; # 1, 2 or 4 Byte

$command{hport} = $COL->[12]; #$line[10]; # HPROT protection CACHE,BUFFER,PRIVILEGE,OPCODE/DATA

#$command{slave} = $COL->[13]; #$line[11]; # Slave index

#

#$command{data} = undef; # A AHB command has no DATA transfers associated yet !!

printf $buih "%\*s%s :: first command\n", 22, '', $first\_cmd;

if ( $first\_cmd ) {

$ptr\_fst\_cmd = $ptr\_cur\_cmd; # Set entry point first Command

$ptr\_pen\_cmd = $ptr\_cur\_cmd; # Set point penultimate Command

$ptr\_prv\_cmd = $ptr\_cur\_cmd; # Set point previous Command

$command{prev} = $ptr\_fst\_cmd; # points to itself prev == curr

$command{next} = $ptr\_fst\_cmd; #

$first\_cmd = 0; # Update SM parse AHB tracker

} else { # first command

${$ptr\_prv\_cmd}{next} = $ptr\_lst\_cmd; # Update previous cmd obj next reference

#$command{next} = $ptr\_lst\_cmd; # Referencing self not prefered Option

$command{next} = $ptr\_fst\_cmd; # Referencing circular to first pointer

$command{prev} = $ptr\_prv\_cmd; # Pointing back

$ptr\_pen\_cmd = $ptr\_prv\_cmd; # Update point penultimate Command

$ptr\_prv\_cmd = $ptr\_cur\_cmd; # Update point previous Command

}

return $ptr\_cur\_cmd; # Return pointer to current object

}#sub build\_command\_obj

sub build\_data\_obj{

my ( $COL ) = @\_; # Input column array Pointer to arrays w/ parsed trace data

state $last\_data = 0; # After a last data, Need to create first data !!

my %data\_obj; # Create new data transfer obj

my $ptr\_cur\_dta = \%data\_obj; # Pointer to data transfer obj

#

# $ptr\_prv\_cmd points to the LAST command/addr transfer, as it is the previous & current & last command !!

#

$data\_obj{assert} = $COL->[ 0]; #$line[0]; # assertion time / data transfer assertion

$data\_obj{complete} = $COL->[ 2]; #$line[1]; # completion time / data transfer completion

$data\_obj{phase} = $COL->[ 4]; #$line[2]; # direction READ/WRITE phase DATA/BUSY

$data\_obj{address} = $COL->[ 5]; #$line[3]; # ADDRESS updated, select SLAVE, data strobe, warp

$data\_obj{beat\_num} = $COL->[ 6]; #$line[4]; # BEAT

#$data\_obj{burst\_len} NA 7 # 1, 4, 8, 16, any >1

$data\_obj{data} = $COL->[ 8]; #$line[6]; # DATA ( 1, 2, 4) Byte strobed via address

$data\_obj{response} = $COL->[ 9]; #$line[7]; # Slave response OKAY/ERROR

#$data\_obj{burst\_type} NA 10 # SINGLE,INCR,INCR4,WRAP4,INCR8,WRAP8,INCR16,WRAP16

#$data\_obj{burst\_size} NA 11 # 1, 2 or 4 Byte

#$data\_obj{protection} NA 12 # HPROT protection CACHE,BUFFER,PRIVILEGE,OPCODE/DATA

#$data\_obj{slave\_index} NA 13 # Slave index

#

# The problem of concurrent DATA PHASE w/ ADDR PHASE of next command, creates TWO instead of ONE OT transactions

# this leads to an incomplete occupied state machine, to identify the data phase as either LAST or FIRST transfer

#

#if ( ${$ptr\_prv\_cmd}{data} eq undef ) {

if ( !defined ${$ptr\_prv\_cmd}{data} ) {

printf $buih "%\*s:: AHB address transfer phase detected\n", 22, '';

if ( $data\_obj{assert} == ${$ptr\_prv\_cmd}{assert} ) {

#if ( !defined ${${$ptr\_prv\_cmd}{prev}}{data} ) {

if ( !defined ${$ptr\_pen\_cmd}{data} ) {

printf $buih "%\*s:: penultimate AHB address transfer has no FIRST data phase detected\n", 22, '';

printf $buih "%\*s:: sequential AHB data transfer phase detected FIRST\n", 22, '';

printf $buih " %\*sDATA transfer assertion time : %8s ns\n", 24, '',$data\_obj{assert};

printf $buih " %\*sADDR transfer assertion time : %8s ns\n", 24, '',${$ptr\_pen\_cmd}{assert};

${$ptr\_pen\_cmd}{data} # update penultimate & previous last command pointer to

= $ptr\_cur\_dta; # first/current data transaction

$ptr\_fst\_dta # Store first data pointer to

= $ptr\_cur\_dta; # current/first data transaction

$data\_obj{prev} # update previous data object pointer

= $ptr\_fst\_dta; # points to itself, first data object/transaction

# = undef; # undefined NO previous data obj

# = $ptr\_lst\_cmd; # points to the previous command

$data\_obj{next} # Initial pointer to next data object

= $ptr\_fst\_dta; # points circular to first data object

# to be updated w/ next data object

$data\_obj{last} # Initial pointer to last data object/transfer

= $ptr\_cur\_dta; # with current data object

$ptr\_prv\_dta # Store pointer to previous data object

= $ptr\_fst\_dta; # points to current data object

} else {

printf $buih "%\*s:: penultimate AHB address transfer had FIRST data phase detected\n", 22, '';

#

# $ptr\_prv\_dta, $ptr\_lst\_dta, $ptr\_fst\_dta are not updated - thus behave correctly sequential

# still pointing to the sequential DATA transfers of the second last/antepenultimate AHB CMD

#

${$ptr\_prv\_dta}{next} # Update/overwrite previous data object next\_pointer

= $ptr\_cur\_dta; # with pointer to current data object/transaction

$data\_obj{next} # Initial pointer to next data object next\_pointer

= $ptr\_fst\_dta; # points circular to first data object/transaction

$data\_obj{prev} # Store pointer to previous data object/transaction

= $ptr\_prv\_dta; # points to previous data object/transaction

$ptr\_prv\_dta # Update pointer to previous data object/transaction

= $ptr\_cur\_dta; # with current data object/transaction

$ptr\_lst\_dta # Update pointer to last data object/transaction

= $ptr\_cur\_dta; # with current data object/transaction

${$ptr\_fst\_dta}{last} # Update first data object last\_(transaction)\_pointer

= $ptr\_lst\_dta; # with pointer to last data object/transaction

}

} else {

printf $buih "%\*s:: sequential AHB data transfer phase detected FIRST\n", 22, '';

printf $buih " %\*sDATA transfer assertion time : %8s ns\n", 24, '',$data\_obj{assert};

printf $buih " %\*sADDR transfer assertion time : %8s ns\n", 24, '',${$ptr\_prv\_cmd}{assert};

${$ptr\_prv\_cmd}{data} # update previous&last command pointer to

= $ptr\_cur\_dta; # first/current data transaction

$ptr\_fst\_dta # Store first data pointer to

= $ptr\_cur\_dta; # current/first data transaction

$data\_obj{prev} # update previous data object pointer

= $ptr\_fst\_dta; # points to itself, first data object/transaction

# = undef; # undefined NO previous data obj

# = $ptr\_lst\_cmd; # points to the previous command

$data\_obj{next} # Initial pointer to next data object

= $ptr\_fst\_dta; # points circular to first data object

# to be updated w/ next data object

$data\_obj{last} # Initial pointer to last data object/transfer

= $ptr\_cur\_dta; # with current data object

$ptr\_prv\_dta # Store pointer to previous data object

= $ptr\_fst\_dta; # points to current data object

}# Insert FIRST

} else {

printf $buih "%\*s:: sequential AHB data transfer phase detected\n", 22, '';

${$ptr\_prv\_dta}{next} # Update/overwrite previous data object next\_pointer

= $ptr\_cur\_dta; # with pointer to current data object/transaction

$data\_obj{next} # Initial pointer to next data object next\_pointer

= $ptr\_fst\_dta; # points circular to first data object/transaction

$data\_obj{prev} # Store pointer to previous data object/transaction

= $ptr\_prv\_dta; # points to previous data object/transaction

$ptr\_prv\_dta # Update pointer to previous data object/transaction

= $ptr\_cur\_dta; # with current data object/transaction

$ptr\_lst\_dta # Update pointer to last data object/transaction

= $ptr\_cur\_dta; # with current data object/transaction

${$ptr\_fst\_dta}{last} # Update first data object last\_(transaction)\_pointer

= $ptr\_lst\_dta; # with pointer to last data object/transaction

}

}#sub build\_data\_obj

sub initialize\_latency\_analysis {

my ($a\_ref) = @\_; # Reference to hash

#printf " ... initialzing latency analysis\n";

printf $proh " ... initialzing latency analysis\n";

my $ridiculous = 1000000;

#

# This set was used for calculation including ADDR phase

#

${$a\_ref}{ALL}{minimum} = $ridiculous;

${$a\_ref}{ALL}{average} = 0;

${$a\_ref}{ALL}{maximum} = 0;

${$a\_ref}{ALL}{number} = 0;

${$a\_ref}{READ}{minimum} = $ridiculous;

${$a\_ref}{READ}{average} = 0;

${$a\_ref}{READ}{maximum} = 0;

${$a\_ref}{READ}{number} = 0;

${$a\_ref}{WRITE}{minimum} = $ridiculous;

${$a\_ref}{WRITE}{average} = 0;

${$a\_ref}{WRITE}{maximum} = 0;

${$a\_ref}{WRITE}{number} = 0;

#

# First data transfer

#

${$a\_ref}{1}{ALL}{minimum} = $ridiculous;

${$a\_ref}{1}{ALL}{average} = 0;

${$a\_ref}{1}{ALL}{maximum} = 0;

${$a\_ref}{1}{ALL}{number} = 0;

${$a\_ref}{1}{ALL}{total} = 0;

${$a\_ref}{1}{READ}{minimum} = $ridiculous;

${$a\_ref}{1}{READ}{average} = 0;

${$a\_ref}{1}{READ}{maximum} = 0;

${$a\_ref}{1}{READ}{number} = 0;

${$a\_ref}{1}{READ}{total} = 0;

${$a\_ref}{1}{WRITE}{minimum}= $ridiculous;

${$a\_ref}{1}{WRITE}{average}= 0;

${$a\_ref}{1}{WRITE}{maximum}= 0;

${$a\_ref}{1}{WRITE}{number} = 0;

${$a\_ref}{1}{WRITE}{total} = 0;

# Last data transfer, complete burst

${$a\_ref}{L}{ALL}{minimum} = $ridiculous;

${$a\_ref}{L}{ALL}{average} = 0;

${$a\_ref}{L}{ALL}{maximum} = 0;

${$a\_ref}{L}{ALL}{number} = 0;

${$a\_ref}{L}{ALL}{total} = 0;

${$a\_ref}{L}{READ}{minimum} = $ridiculous;

${$a\_ref}{L}{READ}{average} = 0;

${$a\_ref}{L}{READ}{maximum} = 0;

${$a\_ref}{L}{READ}{number} = 0;

${$a\_ref}{L}{READ}{total} = 0;

${$a\_ref}{L}{WRITE}{minimum} = $ridiculous;

${$a\_ref}{L}{WRITE}{average} = 0;

${$a\_ref}{L}{WRITE}{maximum} = 0;

${$a\_ref}{L}{WRITE}{number} = 0;

${$a\_ref}{L}{WRITE}{total} = 0;

#printf " ... initialzing latency analysis done !!\n"; # Silent progress report

#printf " ... initialzing transaction analysis\n";

printf "\n";

printf $proh " ... initialzing latency analysis done !!\n";

printf $proh " ... initialzing transaction analysis\n";

}#sub initialize\_latency\_analysis

sub initialize\_transaction\_analysis {

my ($a\_ref) = @\_; # Reference to hash --> $ahb\_anal ysis

my @\_list = ( "SINGLE", "INCR", "INCR4", "INCR8", "INCR16", "WRAP4", "WRAP8", "WRAP16", "total" );

my @\_dirs = ( "READ", "WRITE" );

my @\_size = ( 1, 2, 4 ); # byte

foreach my $size (@\_size) {

foreach my $dir (@\_dirs) {

foreach my $trans (@\_list) {

${$a\_ref}{transaction}{$trans} = 0;

${$a\_ref}{transaction}{$dir}{$trans} = 0;

${$a\_ref}{transaction}{$size}{$dir}{$trans} = 0;

}# trans

}# dirs

}# size

#printf " ... initialzing transaction analysis done !!\n";

printf $proh " ... initialzing transaction analysis done !!\n";

}#sub initialize\_transaction\_analysis

sub initialize\_payload\_analysis {

my ($ptr\_fst\_cmd) = @\_; # Starting pointer

my $transaction = 0;

my $data\_beat = 0;

my $busy\_beat = 0;

my $totalbeat = 0;

my $ptr\_cur\_cmd = $ptr\_fst\_cmd;

my $total\_payload = 0;

my $write\_payload = 0;

my $read\_payload = 0;

printf $proh " ... initialize payload analysis\n";

printf $errh " %s\n", '='x80;

printf $errh " Tracefile : %s\n", $Trace{tracefile};

printf $errh " Script : %s %s\n", $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf $errh " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $errh " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $errh " %s\n", '='x80;

do {

$transaction++;

$ptr\_cur\_dta = ${$ptr\_cur\_cmd}{data};

$ptr\_fst\_dta = ${$ptr\_cur\_dta}{prev};

$ptr\_lst\_dta = ${$ptr\_cur\_dta}{last};

$data\_beat = 0;

$busy\_beat = 0;

$totalbeat = 0;

printf $payh "%\*s %s %6s\n" , 8, '', 'Transaction :', $transaction;

printf $payh "%\*s %s %6s\n" , 8, '', 'Direction :', ${$ptr\_cur\_cmd}{direction};

printf $payh "%\*s %s %6s\n" , 8, '', 'Burst Type :', ${$ptr\_cur\_cmd}{burst\_type};

printf $errh "%\*s %s %6s\n" , 8, '', 'Transaction :', $transaction;

printf $errh "%\*s %s %6s\n" , 8, '', 'Burst Type :', ${$ptr\_cur\_cmd}{burst\_type};

do {

#my ($dir, $dtype) = split /-/ , ${$ptr\_cur\_dta}{phase}; # DATA, BUSY phase

if ( ${$ptr\_cur\_dta}{phase} =~ m/DATA/ ) {

$totalbeat++; # ONLY data transfer count

$data\_beat++;

printf $payh "%\*s %s %2s of %2s :: %s" , 29, '', 'transfer '

,$data\_beat,$totalbeat,${$ptr\_cur\_dta}{phase};

}

if ( ${$ptr\_cur\_dta}{phase} =~ m/BUSY/ ) {

$busy\_beat++; # A wait state is no BEAT, it is an extention

printf $payh "%\*s %s %2s of %2s :: %s" , 29, '', 'wait state'

,$busy\_beat,$totalbeat,${$ptr\_cur\_dta}{phase};

}

printf $payh " :: %s %s\n" ,'beat ',${$ptr\_cur\_dta}{beat\_num};

my ($beat,$length) = split /\//, ${$ptr\_cur\_dta}{beat\_num};

if ( $beat != $data\_beat ) {

printf $errh "%\*s %s %2s vs %2s :: %s\n" , 29, '', 'Error Beat'

,$data\_beat,${$ptr\_cur\_dta}{beat\_num}, 'reported';

}

# printf $payh "%\*s %s %s\n" , 29, '', 'assert ' , ${$ptr\_cur\_dta}{assert}; # Timestemp for back annotation into trace file

# printf $payh "%\*s %s %s\n" , 29, '', 'complete' , ${$ptr\_cur\_dta}{complete};

$ptr\_cur\_dta = ${$ptr\_cur\_dta}{next}; # circular list points to the beginning

}until ($ptr\_cur\_dta == $ptr\_fst\_dta); # If first is the last&only pointer, then the next is the first as well

${$ptr\_cur\_cmd}{burstlength} = $totalbeat; # Last beat/data transfer/burst length

${$ptr\_cur\_cmd}{payload} = $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size}; # data transfered in burst

$total\_payload += $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size};

$write\_payload += $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size} if ( ${$ptr\_cur\_cmd}{direction} =~ m/WR/ );

$read\_payload += $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size} if ( ${$ptr\_cur\_cmd}{direction} =~ m/RD/ );

printf $payh "%\*s %s %6s byte/beat\n", 8, '', 'Burst Size :', ${$ptr\_cur\_cmd}{burst\_size};

printf $payh "%\*s %s %6s beat\n" , 8, '', 'Burst Length :', ${$ptr\_cur\_cmd}{burstlength};

printf $payh "%\*s %s %6s byte\n" , 8, '', 'Burst Payload:', ${$ptr\_cur\_cmd}{payload };

printf $payh "\n";

printf $errh "\n";

$ptr\_cur\_cmd = ${$ptr\_cur\_cmd}{next}; # Iterate to the next transaction/transfer AHB command

} until ($ptr\_cur\_cmd == $ptr\_fst\_cmd ); #

printf $payh "\n";

printf $payh "%\*s %s %6s byte\n" , 8, '', 'Payload Total:', $total\_payload;

printf $payh "%\*s %s %6s byte\n" , 8, '', 'Payload Write:', $write\_payload;

printf $payh "%\*s %s %6s byte\n" , 8, '', 'Payload Read :', $read\_payload;

printf $proh " ... initialize payload done\n";

}#sub initialize\_payload\_analysis

sub initialize\_transfer\_analysis {

my ( $fst\_ptr # Start pointer

, $lst\_ptr # End pointer

, $logfile # future log filename

, $trace\_header\_ref # Trace file information

, $transfer\_ref) = @\_; # Transfer analysis

my $ptr\_cur\_cmd = $fst\_ptr;

my $cycle = ${$trace\_header\_ref}{period}; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

my $frequency = ${$trace\_header\_ref}{frequency}; # (float)

my $fb = ${$trace\_header\_ref}{freqbase}; # (MHz|GHz)

my $transaction = 0;

my $beat = 0;

my $previous\_transfer\_stop;

printf $proh " ... initialize transfer analysis\n";

printf $trah " %s\n", '='x80;

printf $trah " Tracefile : %s\n" , $Trace{tracefile};

printf $trah " Logfile : %s\n" , $logfile;

printf $trah " Script : %s %s\n" , $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf $trah " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $trah " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $trah " Cycle : %10s %s\n" , $cycle, $tb;

printf $trah " Frequency : %10s %s\n", , $frequency, $fb;

printf $trah " %s\n", '='x80;

$previous\_transfer\_stop = ${$ptr\_cur\_cmd}{assert};

do {

$transaction++;

$beat = 0;

#

# IDLE transfer is implicite in trace file and zero in back to back transactions

#

#

#

# previous\_transfer\_stop < ${$ptr\_cur\_cmd}{assert} ==> IDLE not ZERO

# ADDR not ZERO, Not consealed

#

# previous\_transfer\_stop == ${$ptr\_cur\_cmd}{assert} ==> IDLE == ZER0

# ADDR not ZERO, Not consealed

# previous\_transfer\_stop == ${$ptr\_cur\_cmd}{complete}

# previous\_transfer\_stop > ${$ptr\_cur\_cmd}{assert} ==> IDLE == ZERO

# ADDR == ZERO, consealed behind DATA

# ADDR is consealed/hidden behind previous DATA

#

if ( ${$ptr\_cur\_cmd}{assert} > $previous\_transfer\_stop ) {

${$transfer\_ref}{$transaction}{ADDR}{concealed} = 'NOT';

${$transfer\_ref}{$transaction}{IDLE}{transferstart} = $previous\_transfer\_stop;

${$transfer\_ref}{$transaction}{IDLE}{transferstop} = ${$ptr\_cur\_cmd}{assert};

}elsif( ${$ptr\_cur\_cmd}{assert} == $previous\_transfer\_stop ) {

${$transfer\_ref}{$transaction}{ADDR}{concealed} = 'NOT';

${$transfer\_ref}{$transaction}{IDLE}{transferstart} = $previous\_transfer\_stop;

${$transfer\_ref}{$transaction}{IDLE}{transferstop} = $previous\_transfer\_stop;

}elsif( ${$ptr\_cur\_cmd}{assert} < $previous\_transfer\_stop ) {

${$transfer\_ref}{$transaction}{ADDR}{concealed} = 'YES';

${$transfer\_ref}{$transaction}{IDLE}{transferstart} = ${$ptr\_cur\_cmd}{assert};

${$transfer\_ref}{$transaction}{IDLE}{transferstop} = ${$ptr\_cur\_cmd}{assert};

}

${$transfer\_ref}{$transaction}{ADDR}{transferstart} = ${$ptr\_cur\_cmd}{assert};

${$transfer\_ref}{$transaction}{ADDR}{transferstop } = ${$ptr\_cur\_cmd}{complete};

${$transfer\_ref}{$transaction}{IDLE}{duration} = ${$transfer\_ref}{$transaction}{IDLE}{transferstop}

- ${$transfer\_ref}{$transaction}{IDLE}{transferstart};

${$transfer\_ref}{$transaction}{ADDR}{duration} = ${$transfer\_ref}{$transaction}{ADDR}{transferstop}

- ${$transfer\_ref}{$transaction}{ADDR}{transferstart};

$previous\_transfer\_stop = ${$ptr\_cur\_cmd}{complete};

$ptr\_cur\_dta = ${$ptr\_cur\_cmd}{data}; # Set pointer to data transfer also the first

$ptr\_fst\_dta = ${$ptr\_cur\_dta}{prev}; # Initialize pointer to first data transfer

$ptr\_lst\_dta = ${$ptr\_cur\_dta}{last}; # Initialize pointer to last data transfer

${$ptr\_cur\_cmd}{AddrComp} = ${$ptr\_cur\_cmd}{complete} - ${$ptr\_cur\_cmd}{assert};

#

# Transfer analysis - Each AHB transaction can has WAIT states, aka BUSY transfers

#

printf $trah "%\*s %s %10s\n" , 8, '', 'Transaction:', $transaction;

printf $trah "%\*s %s %10s %s %s\n", 8, '', 'IdleTransfer', ${$transfer\_ref}{$transaction}{IDLE}{duration}, $tb

,(${$transfer\_ref}{$transaction}{IDLE}{duration} == 0)

? 'Back2Back'

: 'IDLE';

printf $trah "%\*s %s %10s %s %s\n", 8, '', 'AddrTransfer', ${$transfer\_ref}{$transaction}{ADDR}{duration}, $tb

,(${$transfer\_ref}{$transaction}{ADDR}{concealed} =~ m/NOT/)

? 'Transfer visible'

: 'Transfer consealed';

${$transfer\_ref}{$transaction}{BUSY}{duration} = 0; # Implicite WAIT states, accounted explicite, initialized to ZERO

${$transfer\_ref}{$transaction}{BUSY}{total}{duration} = 0; # Initialize WAIT states as ZERO

do {

#

# The wait state or BUSY transfer comes before the DATA transfer complete

#

if ( ${$ptr\_cur\_dta}{phase} =~ m/BUSY/ ) {

my $wait\_state = ${$ptr\_cur\_dta}{complete} - ${$ptr\_cur\_dta}{assert}; # The trace protocol allows 1 or more WAIT state

# being inserted, trace reports transfers not clock cycle

${$transfer\_ref}{$transaction}{BUSY}{$beat+1}{duration} = $wait\_state; # The wait state comes before the DATA transfer,

${$transfer\_ref}{$transaction}{BUSY}{total}{duration} += $wait\_state; # thus the wait state is associate w/ next beat

}# BUSY beats, accounting for one or more WAIT states

#

# Only DATA transfers are beats, Trace protocol inserts/accounts expletice for WAIT states as BUSY cycles

#

if ( ${$ptr\_cur\_dta}{phase} =~ m/DATA/ ) {

$beat++; # Update data transfer beat count

#

# The first data transfer starts with the end of the ADDR transfer,

# the first beat could be a BUSY transfer

#

if ( $previous\_transfer\_stop < ${$ptr\_cur\_dta}{assert} ) { # There must have been a WAIT state

${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstart} = $previous\_transfer\_stop; # Trace protocol counts BUSY cycle, blurs beat count

} else {

${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstart} = ${$ptr\_cur\_dta}{assert}; # extract transfer start

}# determin DATA transfer start with WAIT states

${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstop} = ${$ptr\_cur\_dta}{complete}; # extract transfer stop

${$transfer\_ref}{$transaction}{DATA}{$beat}{duration} = ${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstop }

- ${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstart};

$previous\_transfer\_stop = ${$ptr\_cur\_dta}{complete}; # Remember transaction completion

}# DATA beats # Last transfer must be DATA transfer

printf $trah "%\*s %s %s :: %s" , 28, '', 'beat ' , $beat, ${$ptr\_cur\_dta}{beat\_num};

printf $trah "%19s %s\n" , ${$ptr\_cur\_dta}{complete} - ${$ptr\_cur\_dta}{assert}, $tb;

printf $trah "%\*s %s %s %s\n" , 50, '', 'complete' , ${$ptr\_cur\_dta}{complete}, $tb;

printf $trah "%\*s %s %s %s\n" , 50, '', 'assert ' , ${$ptr\_cur\_dta}{assert} , $tb;

$ptr\_cur\_dta = ${$ptr\_cur\_dta}{next}; # circular list points to the beginning

}until ($ptr\_cur\_dta == $ptr\_fst\_dta); # If first is the last&only pointer,

# then the next is the first as well

${$transfer\_ref}{$transaction}{ADDR}{length} = $beat; # Record the beat count of transaction, burst lenght

${$transfer\_ref}{$transaction}{DATA}{total}{duration} = ${$transfer\_ref}{$transaction}{DATA}{$beat}{transferstop }

- ${$transfer\_ref}{$transaction}{ADDR}{transferstop };

printf $trah "%\*s %s %10s %s\n", 8, '', 'BusyTransfer', ${$transfer\_ref}{$transaction}{BUSY}{duration}, $tb;

printf $trah "\n";

$ptr\_cur\_cmd = ${$ptr\_cur\_cmd}{next}; # Iterate to the next transaction/transfer AHB command

} until ($ptr\_cur\_cmd == $ptr\_fst\_cmd );#

printf $trah "\n";

#printf $trah "%\*s %s %10s\n" , 8, '', 'Total IDLE :', ${$b\_ref}{IDLE}{total};

printf $proh " ... initialize transfer analysis done\n";

}#sub initialize\_transfer\_analysis

sub latency\_analysis{

my ( $start\_ptr # interval start analysis

#,$stop\_ptr # future interval stop analysis

, $trace\_header\_ref # Trace file information

, $transfer\_ref # %ahb\_transfer analysis

, $latency\_ref) = @\_; # %ahb\_latency\_new

my $cycle = ${$trace\_header\_ref}{period}; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

my $frequency = ${$trace\_header\_ref}{frequency}; # (float)

my $fb = ${$trace\_header\_ref}{freqbase}; # (MHz|GHz)

my $ptr\_cur\_cmd = $start\_ptr;

my $transaction = 0;

my $duration; # duration of each DATA transfer

my $l\_duration; # duration of all DATA transfers combined/total burst

my $execution\_p = 0;

printf $proh " ... latency analysis new\n";

printf $lath " %s\n", '='x80;

printf $lath " Tracefile : %s\n" , $Trace{tracefile};

printf $lath " Script : %s %s\n" , $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf $lath " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $lath " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $lath " Cycle : %10s %s\n" , $cycle, $tb;

printf $lath " Frequency : %10s %s\n", , $frequency, $fb;

printf $lath " %s\n", '='x80;

do {

$transaction++;

#

# ${$transfer\_ref}{$transaction}{IDLE}

# ${$transfer\_ref}{$transaction}{IDLE}{duration}

# ${$transfer\_ref}{$transaction}{ADDR}{concealed} YES/NOT

# ${$transfer\_ref}{$transaction}{ADDR}{duration}

# ${$transfer\_ref}{$transaction}{ADDR}{length}

# ${$transfer\_ref}{$transaction}{ADDR}{lenght} = $beat; # Record the beat count of transaction, burst lenght

# ${$transfer\_ref}{$transaction}{DATA}{1}{duration}

#

my ($direction, $ctype) = split /-/ , ${$ptr\_cur\_cmd}{direction};

my $phase = ($ctype eq 'NONSEQ') ? 'ADDR' : 'IDLE'; # NONSEQ, IDLE, SEQ/DATA, BUSY Recoding ADDR

my $cmd = ($direction eq 'RD') ? 'READ' : 'WRITE'; # Encode READ vs RD; WRITE vs WR Recoding READ / WRITE

my $L = ${$transfer\_ref}{$transaction}{ADDR}{length}; # LAST beat

my $concealed = ${$transfer\_ref}{$transaction}{ADDR}{concealed};

printf $lath "%\*s %s %10s\n" ,12, '', 'Transaction:', $transaction;

printf $lath "%\*s %s %10s\n" ,12, '', 'Direction :', $cmd;

printf $lath "%\*s %s %10s beat\n" ,12, '', 'BurstLength:', $L;

printf $lath "%\*s %s %10s\n" ,12, '', 'Addr Phase :', ($concealed =~ m/NOT/ )

? 'visible'

: 'concealed';

#

# Warning the latency considers only DATA transfers including WAIT states or BUSY cycles

# No ADDR phase is considered in this analysis

#

$duration = ${$transfer\_ref}{$transaction}{IDLE}{duration};

${$latency\_ref}{IDLE}{total} += $duration;

${$latency\_ref}{IDLE}{number} += 1 if ($duration > 0);

printf $lath "%\*s %s %10s %s\n" ,12, '', 'ADDRtransfr:',${$transfer\_ref}{$transaction}{ADDR}{duration} , $tb;

# First DATA transfer latency

$duration = ${$transfer\_ref}{$transaction}{DATA}{1}{duration};

printf $lath "%\*s %s %10s %s\n" ,12, '', 'FirstData :', $duration, $tb;

# Last DATA transfer latency

my $l\_duration = ${$transfer\_ref}{$transaction}{DATA}{$L}{transferstop}

- ${$transfer\_ref}{$transaction}{ADDR}{transferstop};

printf $lath "%\*s %s %10s %s\n" ,12, '', 'AdrPhaseCmp:', ${$transfer\_ref}{$transaction}{ADDR}{transferstart} , $tb;

printf $lath "%\*s %s %10s %s\n" ,12, '', 'LastDataCmp:', ${$transfer\_ref}{$transaction}{DATA}{$L}{transferstop}, $tb;

printf $lath "%\*s %s %10s %s\n" ,12, '', 'DtaPhaseCmp:', $l\_duration, $tb;

#

# First data transfer

#

${$latency\_ref}{1}{ALL}{total} += $duration; # ALL READ & WRITE

${$latency\_ref}{1}{ALL}{number} += 1; # ALL transaction

${$latency\_ref}{1}{ALL}{minimum} = $duration if (${$latency\_ref}{1}{ALL}{minimum} >= $duration);

${$latency\_ref}{1}{ALL}{maximum} = $duration if (${$latency\_ref}{1}{ALL}{maximum} <= $duration);

${$latency\_ref}{1}{$cmd}{total} += $duration;

${$latency\_ref}{1}{$cmd}{number} += 1; # ALL READ, ALL WRITE

${$latency\_ref}{1}{$cmd}{minimum} = $duration if (${$latency\_ref}{1}{$cmd}{minimum} >= $duration);

${$latency\_ref}{1}{$cmd}{maximum} = $duration if (${$latency\_ref}{1}{$cmd}{maximum} <= $duration);

#

# Last data transfer

#

${$latency\_ref}{L}{ALL}{total} += $l\_duration;

${$latency\_ref}{L}{ALL}{number} += 1;

${$latency\_ref}{L}{ALL}{minimum} = $l\_duration if (${$latency\_ref}{L}{ALL}{minimum} >= $l\_duration);

${$latency\_ref}{L}{ALL}{maximum} = $l\_duration if (${$latency\_ref}{L}{ALL}{maximum} <= $l\_duration);

${$latency\_ref}{L}{$cmd}{total} += $l\_duration;

${$latency\_ref}{L}{$cmd}{number} += 1;

${$latency\_ref}{L}{$cmd}{minimum} = $l\_duration if (${$latency\_ref}{L}{$cmd}{minimum} >= $l\_duration);

${$latency\_ref}{L}{$cmd}{maximum} = $l\_duration if (${$latency\_ref}{L}{$cmd}{maximum} <= $l\_duration);

printf $lath "\n"; # Spacer

$ptr\_cur\_cmd = ${$ptr\_cur\_cmd}{next};

} until ($ptr\_cur\_cmd == $ptr\_fst\_cmd );

printf $lath "%\*s %s %10s\n" ,12, '', 'ALL 1st Number:', ${$latency\_ref}{1}{ALL }{number};

printf $lath "%\*s %s %10s\n" ,12, '', 'READ 1st Number:', ${$latency\_ref}{1}{READ }{number};

printf $lath "%\*s %s %10s\n" ,12, '', 'WRITE 1st Number:', ${$latency\_ref}{1}{WRITE}{number};

printf $lath "%\*s %s %10s\n" ,12, '', 'READ 1st Total :', ${$latency\_ref}{1}{READ }{total };

printf $lath "%\*s %s %10s\n" ,12, '', 'WRITE 1st Total :', ${$latency\_ref}{1}{WRITE}{total };

printf $lath "%\*s %s %10s\n" ,12, '', 'ALL Lst Number:', ${$latency\_ref}{L}{ALL }{number};

printf $lath "%\*s %s %10s\n" ,12, '', 'READ Lst Number:', ${$latency\_ref}{L}{READ }{number};

printf $lath "%\*s %s %10s\n" ,12, '', 'WRITE Lst Number:', ${$latency\_ref}{L}{WRITE}{number};

${$latency\_ref}{1}{ALL }{average} = ${$latency\_ref}{1}{ALL }{total} / ${$latency\_ref}{1}{ALL }{number};

${$latency\_ref}{1}{READ }{average} = ${$latency\_ref}{1}{READ }{total} / ${$latency\_ref}{1}{READ }{number};

${$latency\_ref}{1}{WRITE}{average} = ${$latency\_ref}{1}{WRITE}{total} / ${$latency\_ref}{1}{WRITE}{number};

${$latency\_ref}{L}{ALL }{average} = ${$latency\_ref}{L}{ALL }{total} / ${$latency\_ref}{L}{ALL }{number};

${$latency\_ref}{L}{READ }{average} = ${$latency\_ref}{L}{READ }{total} / ${$latency\_ref}{L}{READ }{number};

${$latency\_ref}{L}{WRITE}{average} = ${$latency\_ref}{L}{WRITE}{total} / ${$latency\_ref}{L}{WRITE}{number};

#printf $lath "%\*s %s %10s %s\n" ,12, '', 'AdrPhaseCmp:', $tb;

printf $proh " ... latency analysis new done\n";

}#sub latency\_analysis

sub bandwidth\_analysis {

my ( $start\_ptr # interval start analysis

#,$stop\_ptr # ??? future interval stop analysis

, $trace\_header\_ref # Trace file information

, $transfer\_ref # input %ahb\_transfer analysis

# $payload\_ref # ???

, $bandwidth\_ref # output %ahb\_bandwidth\_new

, $transaction\_ref) = @\_; # %ahb\_transaction\_new

my $ptr\_cur\_cmd = $start\_ptr; # Initialize starting point in transaction list

my $cycle = ${$trace\_header\_ref}{period}; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

my $frequency = ${$trace\_header\_ref}{frequency}; # (float)

my $fb = ${$trace\_header\_ref}{freqbase}; # (MHz|GHz)

my $transaction = 0;

my $beat = 0;

printf $proh " ... bandwidth analysis new\n";

printf $banh " %s\n", '='x80;

printf $banh " Tracefile : %s\n" , $Trace{tracefile};

printf $banh " Script : %s %s\n" , $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf $banh " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $banh " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $banh " Cycle : %10s %s\n" , $cycle, $tb;

printf $banh " Frequency : %10s %s\n", , $frequency, $fb;

printf $banh " %s\n", '='x80;

my $bytes\_transfered = 0;

my $bytes\_read = 0;

my $bytes\_write = 0;

my $total\_time = 0;

my $total\_idle = 0;

my $total\_addr = 0;

my $total\_data = 0;

my $total\_busy = 0;

# $write\_payload += $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size} if ( ${$ptr\_cur\_cmd}{direction} =~ m/WR/ );

# $read\_payload += $totalbeat \* ${$ptr\_cur\_cmd}{burst\_size} if ( ${$ptr\_cur\_cmd}{direction} =~ m/RD/ );

do {

$transaction++;

#

# ${$transfer\_ref}{$transaction}{IDLE}

# ${$transfer\_ref}{$transaction}{IDLE}{duration}

# ${$transfer\_ref}{$transaction}{ADDR}{concealed} YES/NOT

# ${$transfer\_ref}{$transaction}{ADDR}{duration}

# ${$transfer\_ref}{$transaction}{ADDR}{length}

# ${$transfer\_ref}{$transaction}{ADDR}{lenght} = $beat; # Record the beat count of transaction, burst lenght

# ${$transfer\_ref}{$transaction}{DATA}{1}{duration}

#

my ($direction, $ctype) = split /-/ , ${$ptr\_cur\_cmd}{direction};

my $payload = ${$ptr\_cur\_cmd}{payload};

my $phase = ($ctype eq 'NONSEQ') ? 'ADDR' : 'IDLE'; # NONSEQ, IDLE, SEQ/DATA, BUSY Recoding ADDR

my $cmd = ($direction eq 'RD') ? 'READ' : 'WRITE'; # Encode READ vs RD; WRITE vs WR Recoding READ / WRITE

my $idle = ${$transfer\_ref}{$transaction}{IDLE}{duration}; # IDLE phase

my $addr = ${$transfer\_ref}{$transaction}{ADDR}{duration}; # ADDR phase

my $data = ${$transfer\_ref}{$transaction}{DATA}{total}{duration}; # DATA phase

my $busy = ${$transfer\_ref}{$transaction}{BUSY}{total}{duration}; # WAIT states

my $L = ${$transfer\_ref}{$transaction}{ADDR}{length}; # LAST beat Burst length

my $concealed = ${$transfer\_ref}{$transaction}{ADDR}{concealed}; # Visible/concealed ADDR phase

printf $banh "%\*s %s %10s\n" ,12, '', 'Transaction:', $transaction;

printf $banh "%\*s %s %10s\n" ,12, '', 'Direction :', $cmd;

printf $banh "%\*s %s %10s%s\n",12, '', 'BurstLength:', $L,' beat';

printf $banh "%\*s %s %10s\n" ,12, '', 'Addr Phase :', ($concealed =~ m/NOT/ )

? 'visible'

: 'concealed';

printf $banh "%\*s %s %10s\n" ,12, '', 'IDLE Phase :', $idle if ( $idle > 0 ); # NOT Back2Back commands

printf $banh "%\*s %s %10s\n" ,12, '', 'ADDR Phase :', $addr if ($concealed =~ m/NOT/ ); # No interleaving commands

printf $banh "%\*s %s %10s\n" ,12, '', 'DATA Phase :', $data if ( $data > 0 ); # Should be always true

printf $banh "%\*s %s %10s\n" ,12, '', 'BUSY Phase :', $busy if ( $busy > 0 ); # In case of WAIT states

printf $banh "\n"; # Spacer

$ptr\_cur\_cmd = ${$ptr\_cur\_cmd}{next};

$total\_time += $idle + $data;

$total\_time += $addr if ($concealed =~ m/NOT/ );

$total\_idle += $idle;

$total\_addr += $addr if ($concealed =~ m/NOT/ );

$total\_data += $data;

$total\_busy += $busy;

${$bandwidth\_ref}{$cmd}{addr} += $addr if ($concealed =~ m/NOT/ );

${$bandwidth\_ref}{$cmd}{data} += $data;

${$bandwidth\_ref}{$cmd}{busy} += $busy;

${$bandwidth\_ref}{$cmd}{time} += $addr if ($concealed =~ m/NOT/ );

${$bandwidth\_ref}{$cmd}{time} += $data;

} until ($ptr\_cur\_cmd == $ptr\_fst\_cmd );

${$bandwidth\_ref}{ALL}{totaltime} = $total\_time;

${$bandwidth\_ref}{ALL}{totalidle} = $total\_idle;

${$bandwidth\_ref}{ALL}{totaladdr} = $total\_addr;

${$bandwidth\_ref}{ALL}{totaldata} = $total\_data;

${$bandwidth\_ref}{ALL}{totalbusy} = $total\_busy;

printf $banh "\n\n";

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Time :',$total\_time;

printf $banh "\n";

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Idle :',$total\_idle;

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Addr :',$total\_addr;

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Data :',$total\_data;

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Busy :',$total\_busy;

printf $banh "\n";

printf $banh "%\*s %s %10s\n" ,12, '', 'Total Idle :',${$bandwidth\_ref}{ALL}{totalidle};

printf $banh "%\*s %s %10s\n" ,12, '', 'READ Time :',${$bandwidth\_ref}{READ}{time};

printf $banh "%\*s %s %10s\n" ,12, '', 'READ Busy :',${$bandwidth\_ref}{READ}{busy};

printf $banh "%\*s %s %10s\n" ,12, '', 'WRITE Time :',${$bandwidth\_ref}{WRITE}{time};

printf $banh "%\*s %s %10s\n" ,12, '', 'WRITE Busy :',${$bandwidth\_ref}{WRITE}{busy};

printf $proh " ... bandwidth analysis new done\n";

}#sub bandwidth\_analysis

sub protocol\_trace\_analysis {

my ( $start\_ptr

, $b\_ref # $b\_ref --> %ahb\_band

, $l\_ref ) = @\_; # $l\_ref --> %ahb\_latency

my $completion; # Sample time of transaction completion

my $transaction = 0; # Ordinal Number of transaction

my $cmd; # READ or WRITE command

my $type; # Transfertype NONSEQ

my $phase;

my $transfer; # Transfertype IDLE, BUSY, ADDR=NONSEQ, DATA=SEQ

my $beat; # Cordial number of data transfer 1, 2, 3 ...

my $beats; # Total number of data transfers 4, 8 or else

my $payload; # transported byte

my $execution\_p; # Command completion time

#printf " ... initialize bandwidth analysis\n";

printf $proh " ... bandwidth analysis\n";

printf $proh " ... latency analysis\n";

#$ptr\_cur\_cmd = $ptr\_fst\_cmd;

$ptr\_cur\_cmd = $start\_ptr;

printf $ptrh " %s -----> %s\n\n", 'ptr\_fst\_cmd', $ptr\_fst\_cmd;

printf $ptrh " %s -----> %s\n\n", 'ptr\_fst\_cmd', $start\_ptr;

$completion = ${$ptr\_cur\_cmd}{assert}; # Start with non IDLE transfer

#while ( $ptr\_cur\_cmd != $ptr\_lst\_cmd ) {

#}# iterate over all commands

do {

if ( ${$ptr\_cur\_cmd}{assert} > $completion) {

${$b\_ref}{$transaction}{IDLE}{transferstart} = $completion;

${$b\_ref}{$transaction}{IDLE}{transferstop} = ${$ptr\_cur\_cmd}{assert};

${$b\_ref}{$transaction}{IDLE}{duration} = ${$ptr\_cur\_cmd}{assert} - $completion;

${$l\_ref}{IDLE}{total} += ${$b\_ref}{$transaction}{IDLE}{duration};

}# Account IDLE transfers, not explicite in AHB trace file

$transaction++;

printf $ptrh "%\*s %s %s\n" , 8, '', 'previous ', ${$ptr\_cur\_cmd}{prev};

printf $ptrh "%\*s %s %s\n" , 8, '', 'current ', $ptr\_cur\_cmd;

printf $ptrh "%\*s %s %s\n" , 8, '', 'Transaction', $transaction;

printf $ptrh "%\*s %s %s\n" , 8, '', 'transfer ', ${$ptr\_cur\_cmd}{direction};

my ($direction, $ctype) = split /-/ , ${$ptr\_cur\_cmd}{direction};

printf $ptrh "%\*s %s %s\n" , 8, '', 'bursttype ', ${$ptr\_cur\_cmd}{burst\_type};

printf $ptrh "%\*s %s %s byte\n" , 8, '', 'burstsize ', ${$ptr\_cur\_cmd}{burst\_size};

printf $ptrh "%\*s %s ---------------> %s\n", 8, '', 'data ', ${$ptr\_cur\_cmd}{data};

$ptr\_cur\_dta = ${$ptr\_cur\_cmd}{data};

$ptr\_fst\_dta = ${$ptr\_cur\_dta}{prev};

$ptr\_lst\_dta = ${$ptr\_cur\_dta}{last};

printf $ptrh "%\*s %s %s\n\n" , 29, '', 'last ', ${$ptr\_cur\_dta}{last};

$beat = 1;

do {

# Scope of $type, inside/outside the loop

my ($dir, $dtype) = split /-/ , ${$ptr\_cur\_dta}{phase}; # DATA, BUSY

printf $ptrh "%\*s %s %s\n" , 29, '', 'previous' , ${$ptr\_cur\_dta}{prev};

printf $ptrh "%\*s %s %s\n" , 29, '', 'current ' , $ptr\_cur\_dta;

printf $ptrh "%\*s %s %s\n" , 29, '', 'assert ' , ${$ptr\_cur\_dta}{assert};

printf $ptrh "%\*s %s %s\n" , 29, '', 'complete' , ${$ptr\_cur\_dta}{complete};

printf $ptrh "%\*s %s %s\n" , 29, '', 'beat ' , ${$ptr\_cur\_dta}{beat\_num};

printf $ptrh "%\*s %s %3s ns\n" , 29, '', 'timing ' , ${$ptr\_cur\_dta}{complete} - ${$ptr\_cur\_dta}{assert};

printf $ptrh "%\*s %s %s %s\n" , 29, '', 'valid ' , ${$ptr\_cur\_dta}{phase}, $dtype; # Transfer pending

printf $ptrh "%\*s %s %s\n" , 29, '', 'next ' , ${$ptr\_cur\_dta}{next};

printf $ptrh "\n";

${$b\_ref}{$transaction}{$dtype}{$beat}{description} = ${$ptr\_cur\_dta}{beat\_num};

${$b\_ref}{$transaction}{$dtype}{$beat}{duration} = ${$ptr\_cur\_dta}{complete} - ${$ptr\_cur\_dta}{assert};

${$b\_ref}{$transaction}{$dtype}{$beat}{transferstart} = ${$ptr\_cur\_dta}{assert};

${$b\_ref}{$transaction}{$dtype}{$beat}{transferstop} = ${$ptr\_cur\_dta}{complete};

$transfer = ($dtype eq 'DATA') ? 'DATA'

: ($dtype eq 'BUSY') ? 'BUSY' : 'ERRR';

$completion = ${$ptr\_cur\_dta}{complete};

$beats = ${$ptr\_cur\_dta}{beat\_num};

$ptr\_cur\_dta = ${$ptr\_cur\_dta}{next}; # circular list points to the beginning

$beat++;

#$ptr\_cur\_dta = ${$ptr\_cur\_cmd}{next} if ($ptr\_cur\_dta == $ptr\_lst\_dta

} until ($ptr\_cur\_dta == $ptr\_fst\_dta); # If the first is the last&only pointer, then the next is the first as well

#} until ($ptr\_cur\_dta == $ptr\_lst\_dta); ## Coming out one transaction short

#} until ($ptr\_cur\_dta == ${$ptr\_cur\_dta}{next}); #

#

$phase = ($ctype eq 'NONSEQ') ? 'ADDR' : 'IDLE'; # NONSEQ, IDLE, SEQ/DATA, BUSY

$cmd = ($direction eq 'RD') ? 'READ' : 'WRITE'; # Encode READ vs RD; WRITE vs WR

$execution\_p = $completion - ${$ptr\_cur\_cmd}{assert}; # period / elapse time of command/transaction

my ($burstlength,$length) = split /\//, $beats ; #

#$payload = ${$ptr\_cur\_cmd}{burst\_size} \* $burstlength; # payload [byte]

$payload = ${$ptr\_cur\_cmd}{payload}; # payload [byte]

printf $ptrh "%\*s %s %\*s ns\n" , 8, '', 'Addr Phase ', 8, ${$ptr\_cur\_cmd}{complete} - ${$ptr\_cur\_cmd}{assert};

printf $ptrh "%\*s %s %\*s ns\n" , 8, '', 'CmdFinish ', 8, $completion;

printf $ptrh "%\*s %s %\*s ns\n" , 8, '', 'CmdStart ', 8, ${$ptr\_cur\_cmd}{assert};

printf $ptrh "%\*s %s %\*s ns\n" , 8, '', 'Completion ', 8, $completion - ${$ptr\_cur\_cmd}{assert};

printf $ptrh "%\*s %s %\*s beat\n", 8, '', 'Transfers ', 8, $beats;

printf $ptrh "%\*s %s %\*s byte\n", 8, '', 'Burstsize ', 8, ${$ptr\_cur\_cmd}{burst\_size};

printf $ptrh "%\*s %s %\*s beat\n", 8, '', 'Burstlength', 8, $burstlength;

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Direction ', 8, $direction;

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Transftype ', 8, $ctype;

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Command ', 8, $cmd;

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Average ', 8, ${$l\_ref}{ALL}{average};

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Minimum ', 8, ${$l\_ref}{ALL}{minimum};

printf $ptrh "%\*s %s %\*s \n" , 8, '', 'Maximum ', 8, ${$l\_ref}{ALL}{maximum};

printf $ptrh "%\*s %s %s\n\n" , 8, '', 'next ' , ${$ptr\_cur\_cmd}{next};

${$b\_ref}{$transaction}{number} = $transaction; # key == value, ordinal number of transaction

${$b\_ref}{$transaction}{payload} = $payload; # in units of [byte] transfered READ or WRITE

${$b\_ref}{$transaction}{$phase}{direction} = $cmd; # direction refers to READ or WRITE

${$b\_ref}{$transaction}{$phase}{bursttype} = ${$ptr\_cur\_cmd}{burst\_type}; # SINGLE, INCR4, WRAP4, ... INCR

${$b\_ref}{$transaction}{$phase}{burstsize} = ${$ptr\_cur\_cmd}{burst\_size}; # 1, 2 or 4 byte width/size [byte/beat]

${$b\_ref}{$transaction}{$phase}{burstlength} = ${$ptr\_cur\_cmd}{beats}; # burst length 1, 4, 8, 16, 2, 3, 4, .. 1024 [beat] or transfers

${$b\_ref}{$transaction}{$phase}{transferstart} = ${$ptr\_cur\_cmd}{assert}; # point in time begin/assertion of new command [ns]

${$b\_ref}{$transaction}{$phase}{transferstop} = ${$ptr\_cur\_cmd}{complete}; # point in time finish/completion of address phase [ns]

${$b\_ref}{$transaction}{$phase}{transactionstop}= $completion; # point in time finish/completion of transaction/command [ns]

${$b\_ref}{$transaction}{$phase}{duration} = $execution\_p; # period to finish/completion of transaction/command [ns]

${$l\_ref}{ALL}{payload} += $payload;

${$l\_ref}{ALL}{total} += $execution\_p; # execution time

${$l\_ref}{ALL}{number} += 1; # transaction

${$l\_ref}{ALL}{minimum} = $execution\_p if (${$l\_ref}{ALL}{minimum} >= $execution\_p);

${$l\_ref}{ALL}{maximum} = $execution\_p if (${$l\_ref}{ALL}{maximum} <= $execution\_p);

${$l\_ref}{$cmd}{payload} += $payload;

${$l\_ref}{$cmd}{total} += $execution\_p; # execution time

${$l\_ref}{$cmd}{number} += 1; # transaction

${$l\_ref}{$cmd}{minimum} = $execution\_p if (${$l\_ref}{$cmd}{minimum} >= $execution\_p);

${$l\_ref}{$cmd}{maximum} = $execution\_p if (${$l\_ref}{$cmd}{maximum} <= $execution\_p);

$ptr\_cur\_cmd = ${$ptr\_cur\_cmd}{next};

} until ($ptr\_cur\_cmd == $ptr\_fst\_cmd );

printf $ptrh " %s -----> %s\n", 'ptr\_lst\_cmd', $ptr\_lst\_cmd;

close $ptrh;

printf $proh " ... bandwidth analysis done\n";

printf $proh " ... latency analysis done\n";

}#sub protocol\_trace\_analysis

sub report{

my @\_list = ( "SINGLE", "INCR", "INCR4", "INCR8", "INCR16", "WRAP4", "WRAP8", "WRAP16", "total" );

printf "%s\n", '='x80;

printf " TOTAL transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$type};

}

printf "%s\n", '-'x80;

printf " WRITE transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{WRITE}{$type};

}

printf "%s\n", '-'x80;

printf " READ transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{READ}{$type};

}

printf "%s\n", '='x80;

printf " WRITE transactions :: (by size, type)\n";

foreach my $size ( ( 1, 2, 4 ) ) {

printf " %s byte wide burst\n", $size;

foreach my $type ( @\_list ) {

printf " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$size}{WRITE}{$type};

}# by type

printf "%s\n", '-'x80;

}#by size

printf "%8s : %4s transactions\n", "Total WRITE", $ahb\_anal{transaction}{WRITE}{total};

printf "%s\n", '='x80;

printf " READ transactions :: (by size, type)\n";

foreach my $size ( ( 1, 2, 4 ) ){

printf " %s byte wide burst\n", $size;

foreach my $type ( @\_list ) {

printf " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$size}{READ}{$type};

}# by type

printf "%s\n", '-'x80;

}# by size

printf "%8s : %4s transactions\n", "Total READ", $ahb\_anal{transaction}{READ}{total};

printf "%s\n", '='x80;

}#sub report

sub report\_info {

my ( $trace\_header\_ref # Trace information

, $ptr\_ref # transaction trace pointer

, $opts\_ref ) = @\_; # command line arguments

my $cycle = ${$trace\_header\_ref}{period}; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

my $frequency = ${$trace\_header\_ref}{frequency}; # (float)

my $fb = ${$trace\_header\_ref}{freqbase}; # (MHz|GHz)

my $logfile = ${$opts\_ref}{window};

my $transaction = 0;

printf " ... initialize transfer analysis\n";

printf " %s\n", '='x80;

printf " Tracefile : %s\n" , $Trace{tracefile};

printf " Logfile : %s\n" , $logfile;

printf " Script : %s %s\n" , $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf " Cycle : %10s %s\n" , $cycle, $tb;

printf " Frequency : %10s %s\n", , $frequency, $fb;

printf " %s\n", '='x80;

${$opts\_ref}{from} = ${${${$ptr\_ref}{first}}}{assert} if ( !defined ${$opts\_ref}{from} );

${$opts\_ref}{until} = ${${${$ptr\_ref}{last}}}{assert} if ( !defined ${$opts\_ref}{until} );

printf " TraceStart: %10s %s\n" , ${${${$ptr\_ref}{first}}}{assert}, $tb;

printf " TraceEnd : %10s %s\n" , ${${${$ptr\_ref}{last }}}{assert}, $tb;

printf " Starting : %10s %s\n" , ${$opts\_ref}{from} , $tb;

printf " Ending : %10s %s\n" , ${$opts\_ref}{until} , $tb;

open (my $winh, ">$logfile") || die " Can not create log $logfile";

if ( ${$opts\_ref}{from} < ${${${$ptr\_ref}{first}}}{assert} ) {

printf " Starting time point out of bound to the lower side of trace window\n";

}

if ( ${$opts\_ref}{from} > ${${${$ptr\_ref}{last }}}{assert} ) {

printf " Starting time point out of bound to the upper side of trace window\n";

}

if ( ${$opts\_ref}{until} < ${${${$ptr\_ref}{first}}}{assert} ) {

printf " Ending time point out of bound to the lower side of trace window\n";

}

if ( ${$opts\_ref}{until} > ${${${$ptr\_ref}{last }}}{assert} ) {

printf " Ending time point out of bound to the upper side of trace window\n";

}

#if ( ${$opts\_ref}{until} <= ${$opts\_ref}{from} ) {

if ( ${$opts\_ref}{until} < ${$opts\_ref}{from} ) { # The start and end time would select just one Transfer

printf " Ending time is earlier than the Starting time :: Contradiction !!\n";

}

my $stop\_ptr = ${${$ptr\_ref}{first}}; # Set buffer stop pointer

my $search\_ptr = ${${$ptr\_ref}{first}}; # Set search pointer to the FIRST pointer of Transfer List

my $next\_ptr = ${${${$ptr\_ref}{first}}}{next}; # Set lookahead pointer to the NEXT pointer of Transfer List

do {

#

# 1) Transfer assertion is earlier as the START window

#

# 2) Transfer assertion is identical with the START window

#

# 3) Transfer assertion is past the START window

#

$transaction++;

printf $winh "%\*s %s %15s\n" , 8,'', 'Transaction ', $transaction;

printf $winh "%\*s %s %15s\n" , 8,'', 'TransferTime', ${$search\_ptr}{assert};

printf $winh "%\*s %s %15s" , 8,'', 'Pointer ', $search\_ptr;

if ( ${$search\_ptr}{assert} == ${$opts\_ref}{from} # Set START pointer as the BEGIN pointer in Transfer List

# specifically if at FIRST and ULTIMARE transfer

||(${$search\_ptr}{assert} < ${$opts\_ref}{from} && # if between any two transfers

${$next\_ptr }{assert} > ${$opts\_ref}{from} ) # specifically if between ULTIMATE and PENULTIMATE transfer

) {

${${$ptr\_ref}{start}} = $search\_ptr;

printf $winh "\t%s %15s", 'STARTWINDOW ', ${${$ptr\_ref}{start}};

}

if ( ${$search\_ptr}{assert} == ${$opts\_ref}{until} # Set END pointer as the ENDING pointer in Transfer List

# specifically if at ULTIMATE or at any other transfer

||(${$search\_ptr}{assert} < ${$opts\_ref}{until} && # if between any two transfers

${$next\_ptr }{assert} > ${$opts\_ref}{until} ) # specifically if between ULTIMATE and PENULTIMATE transfer

) {

${${$ptr\_ref}{stop}} = $search\_ptr;

printf $winh "\t%s %15s", 'STOP WINDOW ', ${${$ptr\_ref}{stop}};

}

printf $winh "\n";

$search\_ptr = ${$search\_ptr}{next};

$next\_ptr = ${$search\_ptr}{next}; #

} until ($search\_ptr == $stop\_ptr );

printf "START pointer %s\n", ${${$ptr\_ref}{start}};

printf "STOP pointer %s\n", ${${$ptr\_ref}{stop}};

}#sub report\_info

#my %pointer = ( first => \$ptr\_fst\_cmd, # Head of transaction list

# previous => \$ptr\_prv\_cmd, # predessor transaction

# current => \$ptr\_cur\_cmd, # pointer actual transaction

# next => \$ptr\_nxt\_cmd, # successor transaction

# antepenultimate => \$ptr\_ant\_cmd, # third last

# penultimate => \$ptr\_pen\_cmd, # second last

# ultimate => \$ptr\_lst\_cmd, # last

# last => \$ptr\_lst\_cmd, # Tail of transaction list

# start => \$win\_str\_ptr, # Begin of analysis window

# stop => \$win\_end\_ptr, # End of analysis window

sub analysis\_report{

my ( $trace\_header\_ref # Trace information

, $ref # %ahb\_bandwidth

, $latr # %ahb\_latency

, $latency\_ref # %ahb\_latency\_new

, $transfer\_ref # %ahb\_transfer

, $bandwidth\_ref # %ahb\_bandwidth\_new

, $transaction\_ref ) = @\_; # %ahb\_transaction\_new

#foreach my $k ( keys %{$latr} ) {

# printf $anah " <%s>\n" , $k;

# foreach my $k1 ( keys %{${$latr}{$k}} ) {

# printf $anah " <%10s> : %s\n" , $k1, ${${$latr}{$k}}{$k1};

# }

#}

#my $tr = ${$trace\_header\_ref}{tracefile} = $1; # trace filename

#my $prot = ${$trace\_header\_ref}{protocol} = $1; # (AHB)

#my ${$trace\_header\_ref}{period} = $2; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

#my ${$trace\_header\_ref}{redundant} = $7; # (AHB)

#my ${$trace\_header\_ref}{frequency} = $8; # (float)

#my ${$trace\_header\_ref}{freqbase} = $12; # (MHz|GHz)

#my ${$trace\_header\_ref}{databus} = $13; # bit width

my $scale = ($tb eq 'ms')? 1000 # ms

: ($tb eq 'us')? 1000 \* 1000 # us

: ($tb eq 'ns')? 1000 \* 1000 \* 1000 # ns

: ($tb eq 'ps')? 1000 \* 1000 \* 1000 \* 1000 # ps

: 1000 \* 1000 \* 1000 \* 1000 \* 1000; # fs

my $giga = 1000 \* 1000 \* 1000;

my $length\_of\_line = 91;

printf $proh " ... printing analysis report \n";

printf $anah "Trace file : %s\n", ${$trace\_header\_ref}{tracefile} ;

printf $anah "Protocol : %s\n", ${$trace\_header\_ref}{protocol};

printf $anah "Data bus width : %s bit\n", ${$trace\_header\_ref}{databus};

printf $anah "Convention : SI : giga = 1 000 000 000 = 10^9\n"; # IEC 2^10 gibibit JEDEC

printf $anah " mega = 1 000 000 = 10^6\n";

printf $anah " kilo = 1 000 = 10^3\n";

printf $anah "\n\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "Last Data Latency ( Latency = Last Data HREADY Assert time - Address HREADY Assert time )\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' total transaction number', ${$latency\_ref}{L}{ALL }{number};

printf $anah " %s : %13.3f %s\n" , 'minimum transaction completion latency', ${$latency\_ref}{L}{ALL }{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average transaction completion latency', ${$latency\_ref}{L}{ALL }{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum transaction completion latency', ${$latency\_ref}{L}{ALL }{maximum}, $tb;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' READ transaction number', ${$latency\_ref}{L}{READ }{number};

printf $anah " %s : %13.3f %s\n" , 'minimum READ completion latency', ${$latency\_ref}{L}{READ }{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average READ completion latency', ${$latency\_ref}{L}{READ }{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum READ completion latency', ${$latency\_ref}{L}{READ }{maximum}, $tb;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' WRITE transaction number', ${$latency\_ref}{L}{WRITE}{number};

printf $anah " %s : %13.3f %s\n" , 'minimum WRITE completion latency', ${$latency\_ref}{L}{WRITE}{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average WRITE completion latency', ${$latency\_ref}{L}{WRITE}{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum WRITE completion latency', ${$latency\_ref}{L}{WRITE}{maximum}, $tb;

printf $anah "\n";

printf $anah "\n\n";

#printf $anah "

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "First Data Latency ( Latency = First Data HREADY Assert time - Address HREADY Assert time )\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' total transaction number', ${$latency\_ref}{1}{ALL }{number};

printf $anah " %s : %13.3f %s\n" , 'minimum transaction completion latency', ${$latency\_ref}{1}{ALL }{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average transaction completion latency', ${$latency\_ref}{1}{ALL }{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum transaction completion latency', ${$latency\_ref}{1}{ALL }{maximum}, $tb;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' READ transaction number', ${$latency\_ref}{1}{READ }{number};

printf $anah " %s : %13.3f %s\n" , 'minimum READ completion latency', ${$latency\_ref}{1}{READ }{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average READ completion latency', ${$latency\_ref}{1}{READ }{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum READ completion latency', ${$latency\_ref}{1}{READ }{maximum}, $tb;

printf $anah "\n";

printf $anah " %s : %9s\n" , ' WRITE transaction number', ${$latency\_ref}{1}{WRITE}{number};

printf $anah " %s : %13.3f %s\n" , 'minimum WRITE completion latency', ${$latency\_ref}{1}{WRITE}{minimum}, $tb;

printf $anah " %s : %13.3f %s\n" , 'average WRITE completion latency', ${$latency\_ref}{1}{WRITE}{average}, $tb;

printf $anah " %s : %13.3f %s\n" , 'maximum WRITE completion latency', ${$latency\_ref}{1}{WRITE}{maximum}, $tb;

printf $anah "\n";

printf $anah "\n\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah " Bandwidth Analysis\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "\n";

printf $anah " %s : %9s %\*s\n" , 'Total transfered amount of data ',${$latr}{ALL}{payload} , 8,'byte';

printf $anah " %s : %9s %\*s\n" , 'Total READ amount of data ',${$latr}{READ}{payload} , 8,'byte';

printf $anah " %s : %9s %\*s\n" , 'Total WRITE amount of data ',${$latr}{WRITE}{payload} , 8,'byte';

printf $anah "\n\n";

printf $anah " %s : %13.3f %s\n" , 'Total traced period ',${$bandwidth\_ref}{ALL}{totaltime} , $tb;

printf $anah " %s : %13.3f %s\n" , 'Total READ time ',${$bandwidth\_ref}{READ}{time} , $tb;

printf $anah " %s : %13.3f %s\n" , 'Total WRITE time ',${$bandwidth\_ref}{WRITE}{time} , $tb;

printf $anah " %s : %13.3f %s\n" , 'Total IDLE time ',${$bandwidth\_ref}{ALL}{totalidle} , $tb;

printf $anah "\n\n";

printf $anah " %s : %13.3f byte/s", 'Total Bandwith consumed ',${$latr}{ALL}{payload} / ${$bandwidth\_ref}{ALL}{totaltime} \* $scale;

printf $anah " %10.3f Gbit/s\n" ,${$latr}{ALL}{payload} / ${$bandwidth\_ref}{ALL}{totaltime} \* $scale / $giga \* 8;

printf $anah " %s : %13.3f byte/s", 'READ Bandwith consumed ',${$latr}{READ}{payload} / ${$bandwidth\_ref}{READ}{time} \* $scale;

printf $anah " %10.3f Gbit/s\n" ,${$latr}{READ}{payload} / ${$bandwidth\_ref}{READ}{time} \* $scale / $giga \* 8;

printf $anah " %s : %13.3f byte/s", 'WRITE Bandwith consumed ',${$latr}{WRITE}{payload}/ ${$bandwidth\_ref}{WRITE}{time} \* $scale;

printf $anah " %10.3f Gbit/s\n" ,${$latr}{WRITE}{payload}/ ${$bandwidth\_ref}{WRITE}{time} \* $scale / $giga \* 8;

printf $anah "\n\n";

report\_transactions ( $anah # filehandle

, $transaction\_ref # %ahb\_transaction

, 91 ); # length of a report line

#======================================================================================================================================

printf $anah "\n\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah " Latency per transaction\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "\n\n";

printf $anah "Transaction\t\tTransaction\t\tSample time\t\tSample time\t\t Latency\t\t Latency\n";

printf $anah "\t number\t\t assertion\t\t first\_data\t\t last\_data\t\t first\_data\t\t last\_data\n"; # Version 1.01.05 header

printf $anah "%s\n", '='x$length\_of\_line;

# Version 1.01.06

foreach my $transaction ( sort { $a <=> $b } keys %{$transfer\_ref } ) { # Sort numerically

my $L = ${$transfer\_ref}{$transaction}{ADDR}{length}; # Last DATA beat, burst length

printf $anah "#%10s" , $transaction; # First column Transaction Number

printf $anah "\t%12s %s", ${$transfer\_ref}{$transaction}{ADDR}{transferstop} , $tb; # Second column ADDR transfer done

printf $anah "\t%12s %s", ${$transfer\_ref}{$transaction}{DATA}{ 1}{transferstop}, $tb; # Third column 1st DATA transfer done

printf $anah "\t%12s %s", ${$transfer\_ref}{$transaction}{DATA}{$L}{transferstop}, $tb; # Fourth column Last DATA transfer done

printf $anah "\t%12s %s", ${$transfer\_ref}{$transaction}{DATA}{ 1}{duration} , $tb; # Fifth column 1st DATA transfer period

printf $anah "\t%12s %s", ${$transfer\_ref}{$transaction}{DATA}{total}{duration} , $tb; # Sixth column total DATA transfer period

printf $anah "\n";

}#foreach transaction/cmd NOT transfer

printf $anah "%s\n", '='x$length\_of\_line;

printf $anah "End of Analysis Report\n";

printf $anah "%s\n", '='x$length\_of\_line;

printf " ... %s trace analysis done\n",$Trace{protocol} ;

}#sub analysis\_report

sub test\_AHB\_hash{

print {$proh} "\n ... testing \%AHB\_hash structure\n";

print {$tsth} " ... testing \%AHB\_hash structure\n";

my $cnt = 0;

#foreach my $timestamp (sort { $a <=> $b } keys %AHB\_hash) { # The order is chronologic

foreach my $timestamp (sort keys %AHB\_hash) { # The order is alphabetic

$cnt++;

$ahb\_anal{transaction}{total} = $cnt;

my $burst\_type = ${$AHB\_hash{$timestamp}}{burst\_type};

my $burst\_size = ${$AHB\_hash{$timestamp}}{burst\_size};

$ahb\_anal{transaction}{$burst\_type} += 1;

my $addr\_delay = ${$AHB\_hash{$timestamp}}{complete} - ${$AHB\_hash{$timestamp}}{assert};

my $direction = ${$AHB\_hash{$timestamp}}{direction};

if ($direction =~ m{WR-NONSEQ} ){

$ahb\_anal{transaction}{WRITE}{$burst\_type} += 1;

$ahb\_anal{transaction}{WRITE}{total} += 1;

$ahb\_anal{transaction}{$burst\_size}{WRITE}{$burst\_type} += 1;

$ahb\_anal{transaction}{$burst\_size}{WRITE}{total} += 1;

}

if ($direction =~ m{RD-NONSEQ} ){

$ahb\_anal{transaction}{READ}{$burst\_type} += 1;

$ahb\_anal{transaction}{READ}{total} += 1;

$ahb\_anal{transaction}{$burst\_size}{READ}{$burst\_type} += 1;

$ahb\_anal{transaction}{$burst\_size}{READ}{total} += 1;

}

printf

$tsth " %s ns | %s | %4s | %8s | %s byte | %3s ns\n"

, $timestamp, ref $AHB\_hash{$timestamp}, $cnt, ${$AHB\_hash{$timestamp}}{burst\_type}, $burst\_size, $addr\_delay;

# foreach my $id ( sort keys %{$AHB\_hash{$timestamp}} ) {

# printf "%\*s%\*s | %s\n", 10,'',-16, $id, ${$AHB\_hash{$timestamp}}{$id};

# }

}# Readout

}#sub test\_AHB\_hash()

sub transaction\_analysis {

my ( $trace\_header\_ref # Trace information

, $trace\_ref # Reference to global %AHB\_hash, holding all ADDR transfers/ AHB transactions

# no DATA transfers are included in %AHB\_hash

, $logfile # logfile name

, $transaction\_ref ) = @\_; # %ahb\_transaction\_new

my $cycle = ${$trace\_header\_ref}{period}; # (float)

my $tb = ${$trace\_header\_ref}{timebase}; # = $6; # (ns|ps)

my $frequency = ${$trace\_header\_ref}{frequency}; # (float)

my $fb = ${$trace\_header\_ref}{freqbase}; # (MHz|GHz)

printf $proh " ... initialize transfer analysis\n";

printf $cmdh " %s\n", '='x80;

printf $cmdh " Tracefile : %s\n" , $Trace{tracefile};

printf $cmdh " Logfile : %s\n" , $logfile;

printf $cmdh " Script : %s %s\n" , $0, $VERSION; # 0 1, 2, 3, 4, 5, 6, 7, 8

printf $cmdh " Date : %4s-%02s-%02s\n", 1900+$year,$mon,$cday; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $cmdh " Time : %4s:%02s:%02s\n", $h, $m,$s; # $s,$m,$h,$cday,$mon,$year,$wday,$yday,$isdst

printf $cmdh " Cycle : %10s %s\n" , $cycle, $tb;

printf $cmdh " Frequency : %10s %s\n", , $frequency, $fb;

printf $cmdh " %s\n", '='x80;

foreach my $timestamp (sort { $a <=> $b } keys %{$trace\_ref}) { # The order is chronologic

my $direction = ${${$trace\_ref}{$timestamp}}{direction};

my $burst\_type = ${${$trace\_ref}{$timestamp}}{burst\_type};

my $burst\_size = ${${$trace\_ref}{$timestamp}}{burst\_size};

${$transaction\_ref}{transaction}{total} += 1; # accounting starts with counting

${$transaction\_ref}{transaction}{$burst\_type} += 1;

if ($direction =~ m{WR-NONSEQ} ){

${$transaction\_ref}{transaction}{WRITE}{total} += 1;

${$transaction\_ref}{transaction}{WRITE}{$burst\_type} += 1;

${$transaction\_ref}{transaction}{$burst\_size}{WRITE}{total} += 1;

${$transaction\_ref}{transaction}{$burst\_size}{WRITE}{$burst\_type} += 1;

}

if ($direction =~ m{RD-NONSEQ} ){

${$transaction\_ref}{transaction}{READ}{total} += 1;

${$transaction\_ref}{transaction}{READ}{$burst\_type} += 1;

${$transaction\_ref}{transaction}{$burst\_size}{READ}{total} += 1;

${$transaction\_ref}{transaction}{$burst\_size}{READ}{$burst\_type} += 1;

}

printf $cmdh "%6s # %12s\n", ${$transaction\_ref}{transaction}{total}, $timestamp;

}#foreach

report\_transactions ( $cmdh # filehandle

, $transaction\_ref # %ahb\_transaction

, 91 ); # length of a report line

close $cmdh;

}#sub transaction\_analysis

sub report\_transactions {

my ( $fileh

, $transaction\_ref

, $length\_of\_line

) = @\_;

printf $fileh "%s\n", '='x$length\_of\_line;

printf $fileh " Transaction Overview\n";

printf $fileh "%s\n", '='x$length\_of\_line;

printf $fileh "\n";

my @\_list = ( "SINGLE", "INCR", "INCR4", "INCR8", "INCR16", "WRAP4", "WRAP8", "WRAP16", "total" );

printf $fileh " TOTAL transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf $fileh " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$type};

}

printf $fileh "%s\n", '-'x$length\_of\_line;

printf $fileh " WRITE transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf $fileh " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{WRITE}{$type};

}

printf $fileh "%s\n", '-'x$length\_of\_line;

printf $fileh " READ transactions :: (by type)\n";

foreach my $type ( @\_list ){

printf $fileh " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{READ}{$type};

}

printf $fileh "\n";

printf $fileh "%s\n", '='x$length\_of\_line;

printf $fileh " WRITE transactions :: (by size, type)\n";

foreach my $size ( ( 1, 2, 4 ) ){ # byte

printf $fileh " %s byte wide burst\n", $size;

foreach my $type ( @\_list ) {

printf $fileh " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$size}{WRITE}{$type};

}# by type

printf $fileh "%s\n", '-'x$length\_of\_line;

}#by size

printf $fileh "%8s : %4s transactions\n", "Total WRITE", $ahb\_anal{transaction}{WRITE}{total};

printf $fileh "\n";

printf $fileh "%s\n", '='x$length\_of\_line;

printf $fileh " READ transactions :: (by size, type)\n";

foreach my $size ( ( 1, 2, 4 ) ){ # byte

printf $fileh " %s byte wide burst\n", $size;

foreach my $type ( @\_list ) {

printf $fileh " %8s : %4s transactions\n", $type, $ahb\_anal{transaction}{$size}{READ}{$type};

}# by type

printf $fileh "%s\n", '-'x$length\_of\_line;

}# by size

printf $fileh "%8s : %4s transactions\n", "Total READ", $ahb\_anal{transaction}{READ}{total};

printf $fileh "%s\n", '='x$length\_of\_line;

}#sub report\_transactions

sub list\_hash {

my ($href) = @\_;

foreach my $entry ( keys %{$href} ){

printf "\n", ;

}#foreach

}#sub list\_hash