| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [Output/src](http://docs.google.com/index.html) - OutputGenerator.cpp (source / [functions](http://docs.google.com/OutputGenerator.cpp.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 502 | 596 | 84.2 % | | Date: | 2012-03-23 |  | Functions: | 35 | 35 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 135 | 268 | 50.4 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #include <algorithm>  2 : : #include <iostream>  3 : : #include <boost/bind.hpp>  4 : : #include <boost/mem\_fn.hpp>  5 : : #include <boost/ref.hpp>  6 : :   7 : : #include "AreaAccumulator.h"  8 : : #include "ArbitraryModeCellContainer.h"  9 : : #include "ConverterParams.h"  10 : : #include "FlagField.h"  11 : : #include "OutputGenerator.h"  12 : : #include "SFIterator.h"  13 : : #include "Sdf12iTypes.h"  14 : : #include "SimplePattern.h"  15 : : #include "SFOrder.h"  16 : : #include "ThreadManager.h"  17 : : #include "UtilTypes.h"  18 : : #include "SDDB.h"  19 : : #include "TimeData.h"  20 : : #include "print.h"  21 : : #include "HealthCheckCounter.h"  22 : : #include "Logger.h"  23 : : #include "MainSettlingTimeCalculator.h"  24 : : #include "SubSettlingTimeCalculator.h"  25 : : #include "TDSizerSettlingTimeCalculator.h"  26 : :   27 : : namespace {  28 : : const uint64\_t cMaxShotCountPerTF = 1<<28;  29 : : }  30 : :   31 : : namespace nft {  32 : :   33 : : void  34 : 1048 : OutputGenerator::setFixedModeSFTable( SFTable::Pointer inSFTable ) {  35 : 1048 : mFixedModeSFTable = inSFTable;  36 : : }  37 : :   38 : : void  39 : 1316 : OutputGenerator::addSFGToOutputQueue(nft::SFGroup::Pointer inSFG) {  40 : 1316 : boost::mutex::scoped\_lock l(mMutex);  41 : 1316 : e9k\_debug("Received SFG at %p for output\n", inSFG);  42 : 1316 : mCompletedSFGroups.push\_back( inSFG );  43 : 1316 : mInputAvailable.notify\_all();  44 : : }  45 : :   46 : :   47 : : void  48 : : OutputGenerator::addCellForOutput(  49 : 63 : ArbitraryModeCellContainer::Pointer inCell) {  50 : 63 : boost::mutex::scoped\_lock l(mMutex);  51 : 63 : mCells.push\_back( inCell );  52 : : }  53 : :   54 : : void  55 : 1094 : OutputGenerator::stop() {  56 : 1094 : boost::mutex::scoped\_lock l(mMutex);  57 : 1094 : mStop = true;  58 : 1094 : mInputAvailable.notify\_all();  59 : : }  60 : :   61 : : void  62 : 1044 : OutputGenerator::setIsComplete(bool value) {  63 : 1044 : boost::mutex::scoped\_lock l(mMutex);  64 : 1044 : mIsComplete = value;  65 : 1044 : mInputAvailable.notify\_all();  66 : : }  67 : :   68 : : void  69 : 1094 : OutputGenerator::operator()() throw() {  70 : 1094 : ConverterParams \*pParams = ConverterParams::instance();  71 : 1094 : ArbitraryModeCellContainer \*pTopCell = 0;  72 : 2434 : while(true)  73 : : {  74 : : {  75 : 2434 : boost::mutex::scoped\_lock l(mMutex);  76 : : // while( !mStop  77 : : // && !mIsComplete  78 : : // && ( (mFixedMode && mCompletedSFGroups.empty())))  79 : : //TODO::OPEN || (!mFixedMode && mCells.empty() )))  80 [ + - ][ + + ]: 2434 : if( !mStop  [ + + ]  81 : : && !mIsComplete  82 : : && ( (mFixedMode && mCompletedSFGroups.empty())  83 : : || (!mFixedMode  84 : : && ( mCells.empty()   85 : : || (pTopCell  86 : : && !pTopCell->getIsComplete())))))  87 : : {  88 : 2419 : mInputAvailable.wait(l);  89 : : }  90 [ + + ]: 2419 : if( mStop )  91 : : {  92 : 50 : return;  93 : : }  94 [ + + ]: 2384 : if( mIsComplete )  95 : : {  96 [ + + ]: 1053 : if( (mFixedMode && mCompletedSFGroups.empty() )  97 : : || (!mFixedMode && mCells.empty() ))  98 : : {  99 : : SDF12iCheckSumData \*pChecksum  100 : : = reinterpret\_cast<SDF12iCheckSumData \*>(  101 : : mOutputBuffer.getWritableMemory(  102 : 1044 : sizeof( SDF12iCheckSumData ) ));  103 : 1044 : pChecksum->mFGid = eFlagCheckSum;  104 : 1044 : pChecksum->mChkSumS0  105 : : = mChecksumCalculator.getSum0();  106 : 1044 : pChecksum->mChkSumS1  107 : : = mChecksumCalculator.getSum1();  108 : 1044 : pChecksum->mChkSumS2  109 : : = mChecksumCalculator.getSum2();  110 : : SDF12iDpbHeader \*pDPB  111 : : = reinterpret\_cast<SDF12iDpbHeader \*>(  112 : 1044 : mOutputBuffer.getBuffer() + mDPBHeader);  113 [ + - ][ + + ]: 1044 : if(pDPB && !mIsNULLDPB ){  114 : : size\_t dataLength  115 : 1015 : = mOutputBuffer.getOutputData().getMemSize();  116 : 1015 : dataLength -=   117 : : ( getFlagMask( FLAG\_DUMP\_OUTPUT )  118 : : & mStartMsg->usFlag )  119 : : ? sizeof(SDF12iDumpHeader) : 0;  120 : 1015 : pDPB->mDataLength = dataLength - sizeof(SDF12iDpbHeader);  121 : 1015 : e9k\_debug("Generated data size %u\n",  122 : : pDPB->mDataLength );  123 : : }  124 [ # # ]: 0 : else if( mIsNULLDPB )  125 : : {  126 : 29 : e9k\_debug("NULL DPB .. No data generated");  127 : 29 : Logger::instance()->log(LogMessageTable::eNULL\_DPB);  128 : 29 : mOutputBuffer.reset();  129 : : }  130 : :   131 : 1044 : getThreadManager()->stop(); //Stop everything  132 : 1044 : e9k\_debug("Requesting everyone to stop\n");  133 : 1044 : return;  134 : : }  135 : : }  136 [ + + ]: 1331 : if( mFixedMode )  137 : : {  138 : : //Enqueue completed SFGS to be written  139 : 1296 : for(std::list<nft::SFGroup::Pointer>::iterator sfg  140 : 5224 : = mCompletedSFGroups.begin();  141 : : sfg != mCompletedSFGroups.end(); sfg++ )  142 : : {  143 : 1316 : mLocalSFGs.push( \*sfg );  144 : : }  145 : 1296 : mCompletedSFGroups.clear();  146 : : }  147 : : else  148 : : {  149 : 107 : while( !mCells.empty() )  150 : : {  151 : : std::list<ArbitraryModeCellContainer \*>::iterator cell  152 : 63 : = mCells.begin();  153 : 63 : pTopCell = \*cell;  154 : 63 : if( pTopCell->getIsComplete() )  155 : : {  156 : 63 : mCells.erase( cell );  157 : 63 : mLocalCells.push( pTopCell );  158 : : }  159 : : else  160 : : {  161 : 63 : break;  162 : : }  163 : : }  164 : : }  165 : : }  166 : : //Now write the SFGs  167 : 2656 : while( !mLocalSFGs.empty() )  168 : : {  169 : 1316 : SFGroup::Pointer sfg = mLocalSFGs.front();  170 : 1316 : mLocalSFGs.pop();  171 : 1316 : try  172 : : {  173 : 1316 : writeSFGRepeatedly( sfg );  174 : : }  175 [ # # ]: 0 : catch( DPL::Error \*pChild )  176 : : {  177 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  178 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  179 : : ErrorHandler::eErrFunction,  180 : : pErr->getErrorMsg(  181 : : ErrorHandler::eErrFunction,  182 : : "OutputGenerator::writeSFGRepeatedly"),  183 : : getThreadManager());  184 : 0 : return;  185 : : }  186 : : }  187 : 1403 : while( !mLocalCells.empty() )  188 : : {  189 : 63 : ArbitraryModeCellContainer \*pTopCell = mLocalCells.front();  190 : 63 : mLocalCells.pop();  191 : 63 : try  192 : : {  193 : 63 : distributeCellForWriting( pTopCell );  194 : : }  195 [ # # ]: 0 : catch( DPL::Error \*pChild )  196 : : {  197 [ # # ]: 0 : delete pTopCell;  198 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  199 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  200 : : ErrorHandler::eErrFunction,  201 : : pErr->getErrorMsg(  202 : : ErrorHandler::eErrFunction,  203 : : "OutputGenerator::distributeCellForWriting"),  204 : : getThreadManager());  205 : 0 : return;  206 : : }  207 [ + - ]: 63 : delete pTopCell;  208 : : }  209 : : #if 0  210 : : {  211 : : boost::mutex::scoped\_lock l(mMutex);  212 : : if( mIsComplete )  213 : : {  214 : : if( mFixedMode && mCompletedSFGroups.empty() )  215 : : {  216 : : getThreadManager()->stop(); //Stop everything  217 : : return;  218 : : }  219 : : }  220 : : }  221 : : #endif  222 : : }  223 : : }  224 : :   225 : : void  226 : : OutputGenerator::notifyCellProcessingCompletion(  227 : 63 : ArbitraryModeCellContainer::Pointer inCell) {  228 : 63 : boost::mutex::scoped\_lock l(mMutex);  229 : 63 : inCell->setIsComplete(true);  230 : 63 : mInputAvailable.notify\_all();  231 : : }  232 : :   233 : : nft::MemInfo  234 : 1030 : OutputGenerator::getOutputMemory() const {  235 [ + + ]: 1030 : return (mIsNULLDPB)?MemInfo():mOutputBuffer.getOutputData();  236 : : }  237 : :   238 : : const DPMANAGER\_MESSAGE\_SHOT\_TIMEDATA \*  239 : 1030 : OutputGenerator::getTimeData() const {  240 [ + + ]: 1030 : return (mTimeData)?mTimeData->getTimeData() : 0;  241 : : }  242 : :   243 : : //true if fixed mode  244 : :   245 : : OutputGenerator::OutputGenerator(  246 : : ThreadID inThreadID, bool inMode, size\_t inProjectedSize,  247 : : double inBlkUnit, uint16\_t inShotTimeStep,  248 : : uint16\_t inShotTimeThr, ShotTimeCalculator inShotTimeCalc)  249 : 1094 : :Thread(),  250 : : mCompletedSFGroups(),  251 : : mMutex(),  252 : : mInputAvailable(),  253 : : mOutputBuffer( inProjectedSize ),  254 : : mChecksumCalculator(),  255 : : mStop( false ),  256 : : mFixedMode( inMode ),  257 : : mIsComplete( false ),  258 : : mSFOrderYNext( eUpwardYOrder ),  259 : : mFlipNextYDirection( false ),  260 : : mTFOrderer( ConverterParams::instance() ),  261 : : mFixedModeSFTable(),  262 : : mCurrentSF(),  263 : : mNumTFinCurrSF(),  264 : : mDPBHeader(),  265 : : mLastWrittenSF(),  266 : : mLastWrittenTF(),  267 : : mLastWrittenShot(),  268 : : mCurrentRepetiton(),  269 : : mIsHeaderWritten(),  270 : : mIsTFWritten(),  271 : : mIsShotWritten(),  272 : : mStartMsg( ConverterParams::instance()->getStartupMessage() ),  273 : : mColumnParams( ConverterParams::instance()->getColumnParams() ),  274 : : mSubSetlCalc(),  275 : : mTDSizerSetlCalc(),  276 : : mShotCount(),  277 : : mBlkUnit( inBlkUnit ),  278 : : mShotTimeStep( inShotTimeStep ),  279 : : mShotTimeThr( inShotTimeThr ),  280 : : mShotTimeCalc( inShotTimeCalc),  281 : : mIsFirstShot(),  282 : : mTimeData(),  283 : : mIsFirstNULLShotInSF(true),  284 : : mLastSFShotTCcode(),  285 : : mIsNULLDPB( true ),  286 : : mDPBBoundingBox()  287 : 8752 : {  288 : 1094 : setThreadID( inThreadID );  289 : :   290 : 1094 : ConverterParams \*mParams = ConverterParams::instance();  291 : 1094 : const SETL\_TBL \*pSetlTbl = &(mParams->getColumnParams()->def.setl\_tbl);  292 : 1094 : uint16\_t flag = mStartMsg->usFlag;  293 : :   294 : 1094 : if (flag & getFlagMask( FLAG\_SETL\_ENABLE\_MAIN )) {  295 : 1093 : Logger::instance()->log(LogMessageTable::eMAIN\_ENABLE,  296 : : "Reading from main setting enable");  297 : : }  298 : 1094 : if (flag & getFlagMask( FLAG\_SETL\_ENABLE\_SUB )) {  299 : 1093 : Logger::instance()->log(LogMessageTable::eSUB\_ENABLE,  300 : : "Reading from sub setting enable");  301 : : }  302 : 1094 : if (flag & getFlagMask( FLAG\_SETL\_ENABLE\_TD )) {  303 : 1093 : Logger::instance()->log(LogMessageTable::eTD\_ENABLE,  304 : : "Reading from td setting enable");  305 : : }  306 : 1094 : if (flag & getFlagMask( FLAG\_SETL\_ENABLE\_SIZER )) {  307 : 1093 : Logger::instance()->log(LogMessageTable::eSIZER\_ENABLE,  308 : : "Reading from sizer setting enable");  309 : : }  310 : :   311 : 1094 : mMainSetlCalc = boost::shared\_ptr<MainSettlingTimeCalculator>(  312 : : new MainSettlingTimeCalculator(  313 : : &(pSetlTbl->main),  314 : : flag & getFlagMask( FLAG\_SETL\_ENABLE\_MAIN ),  315 : : mParams->getMainSetlTbl(),  316 : : mStartMsg->dSettlingTimeResolution ));  317 : 1094 : mSubSetlCalc = boost::shared\_ptr<SubSettlingTimeCalculator>(  318 : : new SubSettlingTimeCalculator(  319 : : &(pSetlTbl->sub),  320 : : flag & getFlagMask( FLAG\_SETL\_ENABLE\_SUB ),  321 : : mParams->getSubSetlTbl(),  322 : : pSetlTbl->selector,  323 : : mStartMsg->dSettlingTimeResolution ));  324 : 1094 : mTDSizerSetlCalc = boost::shared\_ptr<TDSizerSettlingTimeCalculator>(  325 : : new TDSizerSettlingTimeCalculator(  326 : : &(pSetlTbl->td),  327 : : flag & getFlagMask( FLAG\_SETL\_ENABLE\_TD ),  328 : : mParams->getTDSetlTbl(),  329 : : &(pSetlTbl->sizer),  330 : : flag & getFlagMask( FLAG\_SETL\_ENABLE\_SIZER ),  331 : : mParams->getSizerSetlTbl(),  332 : : mStartMsg->dSettlingTimeResolution ));  333 : :   334 : 1094 : YDirection yDir = eUpwardYOrder;  335 : :   336 [ + - ]: 1094 : switch( mStartMsg->usSFOrder )  [ + + # # ]  [ # # ]  337 : : {  338 : : case eSFOrderDwdDwdFwd:  339 : : case eSFOrderDwdUwdFwd:  340 : : case eSFOrderDwdUwdBwd:  341 : : case eSFOrderDwdDwdBwd:  342 : : {  343 : 15 : mSFOrderYNext = eDownwardYOrder;  344 : : }  345 : : }  346 : 1094 : mFlipNextYDirection = 0 == (mStartMsg->usSFOrder% 2 );  347 : 1094 : mTFOrderer.constructComparators();  348 [ + + # # ]: 1094 : if( mStartMsg->uiDrawTimeCalcMode )  349 : : {  350 : 1092 : mTimeData = new TimeData(  351 : : mStartMsg->uiNumRowDrawTimeCalc,  352 : : mStartMsg->llDrawTimeCalcOriginX,  353 : : mStartMsg->llDrawTimeCalcPitch);  354 : : }  355 : 1094 : mDPBBoundingBox.setLowerLeftCorner(  356 : : LongPosition( mStartMsg->llDpbX, mStartMsg->llDpbY ) );  357 : 1094 : mDPBBoundingBox.setSize(  358 : : LongPosition( mStartMsg->ullDpbWidth, mStartMsg->ullDpbHeight) );  359 : : }  360 : :   361 : :   362 : 2188 : OutputGenerator::~OutputGenerator() {  363 : : //Do not free output memory here .. this will be done by application  364 [ + + ][ # # ]: 1094 : delete mTimeData;  [ # # ]  365 : 1094 : }  366 : :   367 : : void  368 : 1399 : OutputGenerator::writeSFGRepeatedly(SFGroup::Pointer inSFG ) throw(DPL::Error\*) {  369 : 1399 : HealthCheckCounter::instance()->incrCounter(  370 : : HealthCheckCounter::eUpdateFrequent |  371 : : HealthCheckCounter::eUpdateInfrequent |  372 : : HealthCheckCounter::eUpdateSeldom );  373 : 1399 : SDDB \*pSDDB = SDDB::instance();  374 : 1399 : uint32\_t repCount = mStartMsg->uiRepCount;  375 [ + + ]: 1399 : if( !mIsHeaderWritten)  376 : : {  377 : 1015 : mIsHeaderWritten = true;  378 : : DPMANAGER\_MESSAGE\_HEADER \*pHeader  379 : 1015 : = ConverterParams::instance()->getHeader();  380 : 1015 : if( getFlagMask(FLAG\_DUMP\_OUTPUT) & mStartMsg->usFlag )  381 : : {  382 : : SDF12iDumpHeader \*pDumpHeader  383 : : = reinterpret\_cast<SDF12iDumpHeader \*>(  384 : : mOutputBuffer.getWritableMemory(  385 : 1014 : sizeof( SDF12iDumpHeader) ) );  386 : 1014 : pDumpHeader->mFGid = eFlagDumpHeader;  387 : 1014 : pDumpHeader->mShotUnit = mStartMsg->uiAU;  388 : 1014 : pDumpHeader->mDPBSizeX = mStartMsg->ullDpbWidth;  389 : 1014 : pDumpHeader->mDPBSizeY = mStartMsg->ullDpbHeight;  390 : 1014 : pDumpHeader->mDPBLocationX = mStartMsg->llDpbX;  391 : 1015 : pDumpHeader->mDPBLocationY = mStartMsg->llDpbY;  392 : : }  393 : : SDF12iDpbHeader \*pDPB  394 : : = reinterpret\_cast<SDF12iDpbHeader \*>(  395 : : mOutputBuffer.getWritableMemory(  396 : 1015 : sizeof( SDF12iDpbHeader ) ) );  397 : 1015 : pDPB->mFG1 = eFlagDPBHeader;  398 : :   399 : 1015 : pDPB->mStripeNo = pHeader->uiStripeNo;  400 : 1015 : pDPB->mDPBProcNo = pHeader->uiDpbNo; //FIX::E9K-27  401 : 1015 : pDPB->mFG2 = 0; //TODO?  402 : 1015 : pDPB->mDataLength = 0;  403 : 1015 : mDPBHeader = reinterpret\_cast<char \*>(pDPB)  404 : : - mOutputBuffer.getBuffer();  405 : : //Offset from memory start  406 : : }  407 : 1399 : try  408 : : {  409 : 1399 : pSDDB->SfgToBeWritten(inSFG);  410 : : }  411 [ # # ]: 0 : catch(DPL::Error \*pChild)  412 : : {  413 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  414 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  415 : : ErrorHandler::eErrFunction,  416 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  417 : : "SDDB::SfgToBeWritten"),  418 : : getThreadManager());  419 : 0 : return;  420 : : }  421 : :   422 [ + - ][ + + ]: 2791 : for( uint32\_t currRep = 0; currRep < repCount; currRep++ )  423 : : {  424 : 1411 : mCurrentRepetiton = currRep;  425 : 1411 : try  426 : : {  427 : 1411 : writeSFG( inSFG );  428 : : }  429 [ + - ]: 19 : catch( DPL::Error \*pChild )  430 : : {  431 : 19 : ErrorHandler \*pErr = ErrorHandler::instance();  432 : 19 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  433 : : ErrorHandler::eErrFunction,  434 : : pErr->getErrorMsg(  435 : : ErrorHandler::eErrFunction,  436 : : "OutputGenerator::writeSFG"),  437 : : getThreadManager());  438 : 19 : return;  439 : : }  440 : 1392 : try  441 : : {  442 : 1392 : pSDDB->setSfgRepeat(true);  443 : : }  444 [ # # ]: 0 : catch(DPL::Error \*pChild)  445 : : {  446 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  447 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  448 : : ErrorHandler::eErrFunction,  449 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  450 : : "SDDB::setSfgRepeat"),  451 : : getThreadManager());  452 : 0 : return;  453 : : }  454 : : }  455 : 1380 : try  456 : : {  457 : 1380 : pSDDB->setColumnSwitch();  458 : : }  459 [ # # ]: 0 : catch(DPL::Error \*pChild)  460 : : {  461 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  462 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  463 : : ErrorHandler::eErrFunction,  464 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  465 : : "SDDB::setColumnSwitch()"),  466 : : getThreadManager());  467 [ # # ]: 1399 : return;  468 : : }  469 : : }  470 : :   471 : : void  472 : : OutputGenerator::writeSFG( SFGroup::Pointer inoutSFG ) throw(DPL::Error \*)  473 : 1411 : {  474 : 1411 : HealthCheckCounter::instance()->incrCounter(  475 : : HealthCheckCounter::eUpdateFrequent);  476 : 1411 : mTFOrderer.registerSegmentWriter(  477 : 1411 : boost::bind( &OutputGenerator::writeSegment, boost::ref(\*this),\_1 ));  478 : : //The following conditions should not occur  479 [ + - ]: 1411 : if( NULL == inoutSFG)  480 : : {  481 : 1411 : return;  482 : : }  483 : 1411 : SFIterator sf(inoutSFG, mSFOrderYNext);  484 : : const sgs\_column\_t \*pColumnParams  485 : 1411 : = ConverterParams::instance()->getColumnParams();  486 [ + + ]: 29279 : for(SubField::Pointer subField = sf.begin();  487 : : subField != sf.end(); sf++, subField = sf )  488 : : {  489 : 27887 : if( subField->getIsEmpty() )  490 : : {  491 : 2199 : continue;  492 : : }  493 : 2199 : int32\_t sfXpos = subField->getXwrtVChip();  494 : 2199 : int32\_t sfYpos = subField->getYwrtVChip();  495 : :   496 : 2199 : if( mStartMsg->usFlag & getFlagMask( FLAG\_MISC\_KEEP\_OUTSIDE\_SF ) )  497 : : {  498 : : //Proceed normally  499 : : }  500 : : else  501 : : {  502 : : //Remove SF if outside DPB  503 : 0 : LongPosition sfAbsOrigPos( sfXpos, sfYpos );  504 [ # # ]: 0 : if( !mDPBBoundingBox.checkIsPointIncluded( sfAbsOrigPos ) )  505 : : {  506 : 0 : Logger::instance()->log(  507 : : LogMessageTable::eSF\_OUSIDE\_DPB, sfXpos, sfYpos );  508 : 0 : e9k\_debug("Sub Field falls outside DPB boundary.");  509 : 0 : continue;  510 : : }  511 : : }  512 : : double conversionFactor  513 : : = static\_cast<double>(mColumnParams->unit.laser\_count)  514 [ + - ]: 2199 : / mColumnParams->unit.laser\_wavelength;  515 : : //LU Conversion  516 : 2199 : sfXpos = static\_cast<int32\_t>((sfXpos  517 : : + static\_cast<int32\_t>(pColumnParams->mask.mask\_center\_x))  518 : : \* conversionFactor) ;  519 : 2199 : sfYpos = static\_cast<int32\_t>((sfYpos  520 : : + static\_cast<int32\_t>(pColumnParams->mask.mask\_center\_y))  521 : : \* conversionFactor) ;  522 : :   523 : : SDF12iSFHeader \*pSF  524 : : = reinterpret\_cast<SDF12iSFHeader \*>(  525 : : mOutputBuffer.getWritableMemory(  526 : 2199 : sizeof( SDF12iSFHeader ) ) );  527 : 2199 : pSF->mFG1 = eFlagSF;  528 : 2199 : pSF->mSFLocationX = sfXpos;  529 : 2199 : pSF->mSFLocationY = sfYpos;  530 : 2199 : AreaAccumulator::instance()->incrementSFCount();  531 : 2199 : mCurrentSF = subField;  532 : 2199 : mLastWrittenSF = reinterpret\_cast<char \*>(pSF)  533 : : - mOutputBuffer.getBuffer();  534 : 2199 : mNumTFinCurrSF = 0;  535 [ + + ]: 2199 : if( mFixedMode )  536 : : {  537 : 2116 : try  538 : : {  539 : 2116 : mTFOrderer.performOrdering( subField );  540 : : }  541 [ + - ]: 18 : catch(DPL::Error \*pChild)  542 : : {  543 : 18 : ErrorHandler \*pErr = ErrorHandler::instance();  544 : 18 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  545 : : ErrorHandler::eErrFunction,  546 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  547 : : "TFOrderEnforcer::performOrdering"));  548 : : }  549 : : }  550 : : else  551 : : {  552 : 83 : try  553 : : {  554 : 83 : writeSegment( subField->getSegmentsForLocationPriority() );  555 : : }  556 [ # # ]: 0 : catch(DPL::Error \*pChild)  557 : : {  558 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  559 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  560 : : ErrorHandler::eErrFunction,  561 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  562 : : "OutputGenerator::writeSegment"));  563 : : }  564 : : }  565 : 2181 : finalizeSF();   566 [ + - ]: 2180 : if( mIsTFWritten )  567 : : {  568 : : SDF12iTFHeader \*pTF = reinterpret\_cast<SDF12iTFHeader \*>(  569 : 2180 : mOutputBuffer.getBuffer() + mLastWrittenTF);  570 : 2180 : mChecksumCalculator.removeChecksums( pTF );  571 : 2180 : pTF->mCcode = pTF->mCcode | 0x01;  572 : 2180 : mChecksumCalculator.updateChecksums( pTF );  573 : : }  574 : :   575 : 2180 : e9k\_debug("Wrote a SF at %d %d\n", sfXpos, sfYpos);  576 : : }  577 [ + + ]: 1411 : mSFOrderYNext = (mFlipNextYDirection)  578 : : ?static\_cast<YDirection>( ( mSFOrderYNext + 1 )% 2)  579 : : :mSFOrderYNext; //Flip  580 [ + - ]: 1411 : e9k\_debug("Wrote a SFG at %p\n", inoutSFG );  581 : : }  582 : :   583 : : void  584 : : OutputGenerator::finalizeSF()  585 : 2181 : {  586 : 2181 : uint64\_t minSFTime = mStartMsg->llMinSFTime;  587 : 2181 : SDDB \*pSDDB = SDDB::instance();  588 : 2181 : double nullShotTime = mStartMsg->dSFEndNullShotTime;  589 : 2181 : double nullSettlingTime = mStartMsg->dSFEndNullSettlingTime;  590 : 2181 : uint64\_t nullShotCount = mStartMsg->uiSFEndNullShotNum;  591 : 2181 : bool isTFLessMode = mStartMsg->usProcessingMode & 0x02;  592 [ + + ]: 2181 : if ( nullShotCount > 0 )  593 : : {  594 : 28 : SDF12iTFHeader \*pTF = 0;  595 : 28 : SDF12iShot \*lastShot = NULL;  596 : 28 : AreaAccumulator \*pAreaAcc = AreaAccumulator::instance();  597 : : SDF12iTFHeader \*pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  598 : 28 : mOutputBuffer.getBuffer() + mLastWrittenTF);  599 : 28 : uint64\_t writtenShotCount = pLastTF->mNShot;   600 [ + + ]: 28 : if( !isTFLessMode )  601 : : {  602 : 14 : lastShot = reinterpret\_cast<SDF12iShot \*>(  603 : : mOutputBuffer.getBuffer() + mLastWrittenShot);  604 : 14 : mChecksumCalculator.removeChecksums( lastShot );  605 : 14 : lastShot->mTCcode = mLastSFShotTCcode;  606 : 14 : mChecksumCalculator.updateChecksums( lastShot );  607 : : }  608 [ + - + + ]: 108 : for( uint64\_t index = 0; index < nullShotCount; index++)  609 : : {  610 [ - + ]: 80 : if( writtenShotCount == cMaxShotCountPerTF )  611 : : {  612 : 0 : try  613 : : {  614 [ # # ][ - + ]: 40 : pTF = reinterpret\_cast<SDF12iTFHeader \*>(  [ # # ][ # # ]  [ - + ]  615 : : mOutputBuffer.getWritableMemory(  616 : : sizeof(SDF12iTFHeader)) );  617 : : }  618 [ # # ]: 0 : catch(DPL::Error \*pChild)  619 : : {  620 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  621 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  622 : : ErrorHandler::eErrFunction,  623 : : pErr->getErrorMsg(  624 : : ErrorHandler::eErrFunction,  625 : : "OutputBuffer::getWritableMemory"));  626 : : }  627 : : SDF12iTFHeader \*pLastTF  628 : : = reinterpret\_cast<SDF12iTFHeader \*>(  629 : 0 : mOutputBuffer.getBuffer() + mLastWrittenTF);  630 : 0 : \*pTF = \*pLastTF;  631 : 0 : pLastTF->mNShot = cMaxShotCountPerTF - 1;  632 : 0 : mChecksumCalculator.updateChecksums( pLastTF );  633 : 0 : pAreaAcc->incrementTFCount();  634 : 0 : mLastWrittenTF = reinterpret\_cast<char \*>(pTF)  635 : : - mOutputBuffer.getBuffer();  636 : 0 : writtenShotCount = 0;  637 : : }  638 [ + + ]: 80 : if(isTFLessMode)  639 : : {  640 : 40 : mNumTFinCurrSF++;  641 : 40 : SDF12iTFHeader \*pTF = 0;  642 : 40 : try  643 : : {  644 : 40 : pTF = reinterpret\_cast<SDF12iTFHeader \*>(  645 : : mOutputBuffer.getWritableMemory(  646 : : sizeof(SDF12iTFHeader)) );  647 : : }  648 [ # # ]: 0 : catch(DPL::Error \*pChild)  649 : : {  650 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  651 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  652 : : ErrorHandler::eErrFunction,  653 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  654 : : "OutputBuffer::getWritableMemory"));  655 : : }  656 : : SDF12iTFHeader \*pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  657 : 40 : mOutputBuffer.getBuffer() + mLastWrittenTF);  658 : 40 : pTF->mFG1 = eFlagTF;  659 : 40 : pTF->mKcode = pLastTF->mKcode;  660 : 40 : pTF->mCcode = 0;  661 : 40 : pTF->mTFLocationX = 0;  662 : 40 : pTF->mTFLocationY = 0;  663 : 40 : pTF->mSFLNo = pLastTF->mSFLNo;  664 : 40 : pTF->mNShot = 1;  665 : 40 : pAreaAcc->incrementTFCount();  666 : 40 : mLastWrittenTF = reinterpret\_cast<char \*>(pTF)  667 : : - mOutputBuffer.getBuffer();  668 : : SDF12iTFHeader \*pLastTF1 = reinterpret\_cast<SDF12iTFHeader \*>(  669 : 40 : mOutputBuffer.getBuffer() + mLastWrittenTF);  670 : 80 : mChecksumCalculator.updateChecksums( pLastTF1 );  671 : :   672 : : }  673 : : SDF12iShot \*pNullShot  674 : : = reinterpret\_cast<SDF12iShot\*>(  675 : : mOutputBuffer.getWritableMemory(  676 : 80 : sizeof( SDF12iShot ) ) );  677 : 80 : pNullShot->mFG1 = eFlagShot;  678 : 80 : pNullShot->mXshot = 0;  679 : 80 : pNullShot->mYshot = 0;  680 : 80 : pNullShot->mLX = 0;  681 : 80 : pNullShot->mLY = 0;  682 : 80 : uint32\_t shotTime = (uint32\_t)nullShotTime;  683 : 80 : pNullShot->mTshot = shotTime;  684 : 80 : pAreaAcc->addShotTime( shotTime );  685 : 80 : mSumTShotTime += (shotTime - mShotTimeThr)  686 : : / mShotTimeStep;  687 : :   688 : 80 : pNullShot->mTsettling = (uint32\_t)nullSettlingTime;  689 : 80 : mSumTSettlingTime += pNullShot->mTsettling;  690 [ + + ]: 80 : if(isTFLessMode)  691 : : {  692 : 40 : pNullShot->mTCcode |= 0x0C;  693 : : }  694 : : else  695 : : {  696 : 40 : pNullShot->mTCcode = 0;  697 : : }  698 : :   699 : 80 : mLastWrittenShot = reinterpret\_cast<char \*>(pNullShot)  700 : : - mOutputBuffer.getBuffer();  701 : 80 : pAreaAcc->incrementShotCount();  702 : 80 : pAreaAcc->addSettlingTime( pNullShot->mTsettling );  703 : 80 : mChecksumCalculator.updateChecksums( pNullShot );  704 : 80 : writtenShotCount++;  705 : : }  706 : 28 : pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  707 : : mOutputBuffer.getBuffer() + mLastWrittenTF);  708 [ + + ]: 28 : if (isTFLessMode == true)  709 : : {  710 : 14 : pLastTF->mNShot = 1;  711 : : }  712 : : else  713 : : {  714 : 14 : pLastTF->mNShot = writtenShotCount;  715 : : }  716 : 28 : mChecksumCalculator.updateChecksums( pLastTF );  717 : 28 : mShotCount += writtenShotCount;  718 : :   719 : 28 : lastShot = reinterpret\_cast<SDF12iShot \*>(  720 : : mOutputBuffer.getBuffer() + mLastWrittenShot);  721 : 28 : mChecksumCalculator.removeChecksums( lastShot );  722 : 28 : lastShot->mTCcode = lastShot->mTCcode | 0x08;  723 : 28 : mChecksumCalculator.updateChecksums( lastShot );  724 [ + - ]: 28 : mLastWrittenShot = ( lastShot )  725 : : ? (reinterpret\_cast<char\*>(lastShot)  726 : : - mOutputBuffer.getBuffer())  727 : : : mLastWrittenShot;  728 : : }  729 : 2181 : mIsFirstNULLShotInSF = true;   730 : :   731 : : //In the foll .. pSF can't be used as it may have been  732 : : //moved due to realloc  733 : : SDF12iSFHeader \*oldSF = reinterpret\_cast<SDF12iSFHeader \*>(  734 : 2181 : mOutputBuffer.getBuffer() + mLastWrittenSF);  735 [ + + ]: 2181 : if (isTFLessMode)   736 : : {  737 : 35 : oldSF->mNtf = 0;   738 : : }   739 : : else   740 : : {  741 : 2146 : oldSF->mNtf = mNumTFinCurrSF;  742 : : }  743 [ + - ][ + - ]: 2181 : mSumTSettlingTime = static\_cast<uint64\_t>(  [ + - ]  744 : : mSumTSettlingTime \* mBlkUnit  745 : : + mShotTimeStep \* mShotCount );  746 : : uint64\_t sct = static\_cast<uint64\_t>(  747 : : mSumTShotTime \* mShotTimeStep  748 : 2181 : + mSumTSettlingTime );  749 : 2181 : sct = (sct \* 125) / 10000; //BLK UNIT to NS  750 [ + + ]: 2181 : sct = ( sct < minSFTime ) ? minSFTime : sct;  751 : :   752 [ + - ]: 2181 : oldSF->mTtsc = static\_cast<uint32\_t>( sct  753 : : \* mStartMsg->dShotCycleTimeScaleFactor);  754 : 2181 : mChecksumCalculator.updateChecksums( oldSF );  755 : :   756 [ + + ]: 2181 : if( mStartMsg->uiDrawTimeCalcMode )  757 : : {  758 : : LongPosition absSFPosition( mCurrentSF->getXwrtVChip(),  759 : 2180 : mCurrentSF->getYwrtVChip() );  760 : :   761 : : double mdSettlingTime  762 : 2180 : = mMainSetlCalc->calculate( absSFPosition );  763 : 2179 : e9k\_debug("Main deflector settling time for SF at (%lld,%lld) is %lf\n",  764 : : mCurrentSF->getXwrtVChip(),  765 : : mCurrentSF->getYwrtVChip(), mdSettlingTime);  766 : 2179 : Logger::instance()->log(LogMessageTable::eSHOT\_MAIN\_DEFLECTOR\_ST,  767 : : absSFPosition.getX(), absSFPosition.getY(),  768 : : mdSettlingTime );  769 : : uint64\_t drawingTime  770 [ + - ]: 2179 : = static\_cast<uint64\_t>(mdSettlingTime) + sct;  771 : 2179 : e9k\_debug("Drawing time for SF at (%lld,%lld) is %llu\n",  772 : : mCurrentSF->getXwrtVChip(),  773 : : mCurrentSF->getYwrtVChip(), drawingTime );  774 : 2179 : try  775 : : {  776 : 2179 : mTimeData->updateDrawingTimes(  777 : : mCurrentSF->getXwrtVChip()  778 : : - ( mStartMsg->iSFSizeX >> 1 ),  779 : : drawingTime );  780 : : }  781 [ # # ]: 0 : catch(DPL::Error \*pChild)  782 : : {  783 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  784 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  785 : : ErrorHandler::eErrFunction,  786 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  787 : : "TimeData::updateDrawingTimes"));  788 : : }  789 : : }  790 : :   791 [ + - ]: 2180 : if( mIsShotWritten )  792 : : {  793 : : SDF12iShot \*pShot = reinterpret\_cast<SDF12iShot \*>(  794 : 2180 : mOutputBuffer.getBuffer() + mLastWrittenShot);  795 : 2180 : mChecksumCalculator.removeChecksums( pShot );  796 : 2180 : pShot->mTCcode = pShot->mTCcode | 0x01;  797 : 2180 : mChecksumCalculator.updateChecksums( pShot );  798 : : }  799 : 2180 : try  800 : : {  801 : 2180 : pSDDB->SFWrittenIntoColumn(\*mCurrentSF);  802 : : }  803 [ # # ]: 0 : catch(DPL::Error \*pChild)  804 : : {  805 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  806 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  807 : : ErrorHandler::eErrFunction,  808 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  809 : : "SDDB::SFWrittenIntoColumn"));  810 : : }  811 : 2180 : mSubSetlCalc->reset();  812 : : }  813 : :   814 : :   815 : : void  816 : 4879 : OutputGenerator::writeSegment( Segment::Pointer inSegment ) throw(DPL::Error \*) {  817 : 4879 : TFBuffer::Pointer bufPtr = inSegment->getTFs();  818 [ + - ]: 4879 : if(!bufPtr)  819 : : {  820 : 4879 : return;  821 : : }  822 : 4879 : TF::Pointer \*TFs = bufPtr->getTFs();  823 : 4879 : uint32\_t numTFs = bufPtr->getSize();  824 : 4879 : if(0 == bufPtr->getSize())  825 : : {  826 : 4879 : return;  827 : : }  828 : : const DPMANAGER\_MESSAGE\_SFL\_INFO \*pLayers = ConverterParams::instance()  829 : 4879 : ->getLayerParams();  830 : 4879 : bool isTFLessMode = mStartMsg->usProcessingMode & 0x02;  831 : 4879 : uint32\_t repCount = mStartMsg->uiRepCount;  832 : 4879 : mSumTShotTime = mSumTSettlingTime = mShotCount = 0;  833 : 4879 : SDDB \*pSDDB = SDDB::instance();  834 : 4879 : try  835 : : {  836 : 4879 : pSDDB->WriteNewSegment();  837 : : }  838 [ # # ]: 0 : catch(DPL::Error \*pChild)  839 : : {  840 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  841 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  842 : : ErrorHandler::eErrFunction,  843 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  844 : : "SDDB::WriteNewSegment"));  845 : : }  846 : 4879 : AreaAccumulator \*pAreaAcc = AreaAccumulator::instance();  847 : :   848 [ + - + + ]: 15361 : for( TF::Pointer \*tfIt = TFs; tfIt != TFs + numTFs; tfIt++ )  849 : : {  850 : 10500 : TF::Pointer tf = \*tfIt;  851 : 10500 : try  852 : : {  853 : 10500 : pSDDB->TFWrittenIntoSEgment(\*tf);  854 : : }  855 [ # # ]: 0 : catch(DPL::Error \*pChild)  856 : : {  857 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  858 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  859 : : ErrorHandler::eErrFunction,  860 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  861 : : "SDDB::TFWrittenIntoSEgment"));  862 : : }  863 : :   864 : 10500 : SimplePatternBuffer &shotList = tf->getShotList();  865 : 10500 : uint64\_t nullShotCount = 0;  866 : 10500 : double nullShotTime = mStartMsg->dSFStartNullShotTime;  867 : 10500 : double nullSettlingTime = mStartMsg->dSFStartNullSettlingTime;  868 [ + + ]: 10500 : if ( mIsFirstNULLShotInSF == true )  869 : : {  870 : 2195 : nullShotCount = mStartMsg->uiSFStartNullShotNum;  871 : :   872 : : }  873 : : const DPMANAGER\_MESSAGE\_SFL\_INFO \*pLayers = ConverterParams::instance()  874 : 10500 : ->getLayerParams();  875 [ + + ]: 10500 : if(nullShotCount > 0) {  876 [ + + ]: 28 : if(isTFLessMode) {  877 [ + - ][ + + ]: 54 : for( uint64\_t index = 0; index < nullShotCount; index++) {  878 : 40 : mNumTFinCurrSF++;  879 : 40 : SDF12iTFHeader \*pTF = 0;  880 : 40 : try  881 : : {  882 : 40 : pTF = reinterpret\_cast<SDF12iTFHeader \*>(  883 : : mOutputBuffer.getWritableMemory(  884 : : sizeof(SDF12iTFHeader)) );  885 : : }  886 [ # # ]: 0 : catch(DPL::Error \*pChild)  887 : : {  888 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  889 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  890 : : ErrorHandler::eErrFunction,  891 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  892 : : "OutputBuffer::getWritableMemory"));  893 : : }  894 : 40 : pTF->mFG1 = eFlagTF;  895 : 40 : pTF->mKcode = tf->getPatCode();  896 : 40 : pTF->mCcode = 0;  897 : 40 : pTF->mTFLocationX = 0;  898 : 40 : pTF->mTFLocationY = 0;  899 : 40 : pTF->mNShot = 1;  900 : 40 : uint32\_t SFLNo = tf->getParentSF()->getSFLNo();  901 : 40 : pTF->mSFLNo = (pLayers + SFLNo)->uiSFLNo;  902 : 40 : pAreaAcc->incrementTFCount();  903 : 40 : try  904 : : {  905 : 40 : mSubSetlCalc->setNullShotTFLessModeTF(  906 : : tf->getPatCode());  907 : : }  908 [ # # ]: 0 : catch(DPL::Error \*pChild)  909 : : {  910 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  911 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  912 : : ErrorHandler::eErrFunction,  913 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  914 : : "SubSettlingTimeCalculator::setNullShotTFLessModeTF"));  915 : : }  916 : 40 : mLastWrittenTF = reinterpret\_cast<char \*>(pTF)  917 : : - mOutputBuffer.getBuffer();  918 : : SDF12iTFHeader \*pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  919 : 40 : mOutputBuffer.getBuffer() + mLastWrittenTF);  920 : 40 : mChecksumCalculator.updateChecksums( pLastTF );  921 : 40 : uint64\_t writtenShotCount = 0;  922 : 40 : writeShot( shotList, writtenShotCount, SFLNo );  923 : : }  924 : 14 : mIsFirstNULLShotInSF = false;  925 : : }  926 : : }  927 : 10500 : mNumTFinCurrSF++;  928 [ - + ]: 10500 : if( mNumTFinCurrSF > 1 << 28 ) //SF Replication  929 : : {  930 : 0 : finalizeSF();  931 : 0 : try  932 : : {  933 : : SDF12iSFHeader \*pSF = reinterpret\_cast<SDF12iSFHeader \*>(  934 : : mOutputBuffer.getWritableMemory(  935 : 0 : sizeof(SDF12iSFHeader)) );  936 : : SDF12iSFHeader \*pLastSF = reinterpret\_cast<SDF12iSFHeader \*>(  937 : 0 : mOutputBuffer.getBuffer() + mLastWrittenSF);  938 : 0 : \*pSF = \*pLastSF;  939 : 0 : pAreaAcc->incrementSFCount();  940 : 0 : mLastWrittenSF = reinterpret\_cast<char \*>(pSF)  941 : : - mOutputBuffer.getBuffer();  942 : 0 : mNumTFinCurrSF = 0;  943 : : }  944 [ # # ]: 0 : catch(DPL::Error \*pChild)  945 : : {  946 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  947 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  948 : : ErrorHandler::eErrFunction,  949 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  950 : : "OutputBuffer::getWritableMemory"));  951 : : }  952 : : }  953 : 10500 : SDF12iTFHeader \*pTF = 0;  954 : 10500 : try  955 : : {  956 : 10500 : pTF = reinterpret\_cast<SDF12iTFHeader \*>(  957 : : mOutputBuffer.getWritableMemory(  958 : : sizeof(SDF12iTFHeader)) );  959 : : }  960 [ # # ]: 0 : catch(DPL::Error \*pChild)  961 : : {  962 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  963 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  964 : : ErrorHandler::eErrFunction,  965 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  966 : : "OutputBuffer::getWritableMemory"));  967 : : }  968 : 10500 : mIsNULLDPB = false;  969 : 10500 : mIsFirstShot = true;  970 : : int32\_t tfXWrtSF = tf->getLocX()  971 : 10500 : - (mStartMsg->iSFSizeX >> 1);  972 : : int32\_t tfYWrtSF = tf->getLocY()  973 : 10500 : - (mStartMsg->iSFSizeY >> 1);  974 : 10500 : pTF->mFG1 = eFlagTF;  975 : 10500 : pTF->mKcode = tf->getPatCode();  976 : 10500 : pTF->mCcode = 0;  977 : 10500 : pTF->mTFLocationX = tfXWrtSF;  978 : 10500 : pTF->mTFLocationY = tfYWrtSF;  979 : 10500 : pTF->mSFLNo = (pLayers + tf->getParentSF()->getSFLNo())->uiSFLNo;  980 : 10500 : mCurrentSubSettlingTime = mSubSetlCalc->calculate( tf );  981 : 10498 : uint32\_t SFLNo = tf->getParentSF()->getSFLNo();  982 : 10498 : pAreaAcc->incrementTFCount();  983 : 10498 : uint32\_t currKcode = pTF->mKcode;  984 : 10498 : mLastWrittenTF = reinterpret\_cast<char \*>(pTF)  985 : : - mOutputBuffer.getBuffer();  986 : : //Store offset to avoid problems due to reallocation  987 : 10498 : mIsTFWritten = true;  988 : 10498 : uint64\_t writtenShotCount = 0;  989 : 10498 : writeShot( shotList, writtenShotCount, SFLNo );  990 : : SDF12iTFHeader \*pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  991 : 10482 : mOutputBuffer.getBuffer() + mLastWrittenTF);  992 : 10482 : pLastTF->mNShot = writtenShotCount;  993 : 10482 : mChecksumCalculator.updateChecksums( pLastTF );  994 : 10482 : mShotCount += writtenShotCount;  995 : :   996 : : SDF12iShot \*lastShot = reinterpret\_cast<SDF12iShot \*>(  997 : 10482 : mOutputBuffer.getBuffer() + mLastWrittenShot);  998 : : //mLastSFShotTCcode value will be assigned to lastShot->mTCcode in finalizeSF()  999 : : //if null shot is added to last TF of SF  1000 : 10482 : mLastSFShotTCcode = lastShot->mTCcode;  1001 : :   1002 : 10482 : mChecksumCalculator.removeChecksums( lastShot );  1003 : 10482 : lastShot->mTCcode = lastShot->mTCcode | 0x08;  1004 : 10482 : mChecksumCalculator.updateChecksums( lastShot );  1005 [ + + ]: 10482 : if( mCurrentRepetiton == repCount - 1 )  1006 : : {  1007 : 10446 : tf->releaseShotMem();  1008 [ + + ][ + + ]: 10446 : if( mFixedMode && !isTFLessMode )  1009 : : {  1010 : : SFLayer::Pointer sfl  1011 : : = mFixedModeSFTable->getSFLayers()  1012 : 10151 : + tf->getParentSF()->getSFLNo();  1013 : 10151 : TFLayers::Pointer tfl = sfl->getTFL();  1014 : 10151 : tfl->releaseTF( tf );  1015 : 10151 : \*tfIt = NULL;  1016 : : }  1017 : : }  1018 : 10482 : mTDSizerSetlCalc->reset();  1019 : : }  1020 : : // mLastWrittenShot = ( lastShot )  1021 : : // ? (reinterpret\_cast<char\*>(lastShot)  1022 : : // - mOutputBuffer.getBuffer())  1023 : : // : mLastWrittenShot;  1024 : 4861 : mIsShotWritten = true;  1025 [ + + ]: 4861 : if( mCurrentRepetiton == repCount - 1 )  1026 : : {  1027 [ # # ]: 4861 : inSegment->releaseBuffer();  1028 : : }  1029 : : }  1030 : :   1031 : : void  1032 : : OutputGenerator::writeShot( SimplePatternBuffer &shotList,  1033 : 10538 : uint64\_t &writtenShotCount, uint32\_t SFLNo) {  1034 : 10538 : AreaAccumulator \*pAreaAcc = AreaAccumulator::instance();  1035 : 10538 : uint16\_t shotTime = 0;  1036 : 10538 : double tddcCoeff = mStartMsg->dTimeDependDoseCorrection;  1037 : 10538 : const sgs\_doc\_param\_t &docParam = mColumnParams->doc;  1038 : 10538 : bool isTFLessMode = mStartMsg->usProcessingMode & 0x02;  1039 : 10538 : uint32\_t repCount = mStartMsg->uiRepCount;  1040 : 10538 : uint16\_t numThreads = mStartMsg->usConcurrency;  1041 : 10538 : SDF12iTFHeader \*pTF = 0;  1042 : 10538 : uint32\_t currKcode = 0;  1043 : 10538 : SDF12iShot \*lastShot = NULL;  1044 : 10538 : bool isFirstShotForTCcode = true;  1045 [ + - ][ + + ]: 21068 : for( uint16\_t i = 0; i < numThreads; i++ )  1046 : : {  1047 : 10586 : const SimplePattern \*patterns = shotList.getPatterns(i);  1048 : 10586 : uint64\_t shotCount = shotList.getShotCount(i);  1049 : 10586 : uint64\_t nullShotCount = 0;  1050 : 10586 : double nullShotTime = mStartMsg->dSFStartNullShotTime;  1051 : 10586 : double nullSettlingTime = mStartMsg->dSFStartNullSettlingTime;  1052 [ + + ]: 10586 : if ( mIsFirstNULLShotInSF == true )  1053 : : {  1054 : 2269 : nullShotCount = mStartMsg->uiSFStartNullShotNum;  1055 : :   1056 : : }  1057 [ + + ][ + + ]: 26588 : for( uint64\_t index = 0; index < shotCount + nullShotCount; index++)  1058 : : {  1059 [ - + ]: 16058 : if( writtenShotCount == cMaxShotCountPerTF )  1060 : : {  1061 : 0 : try  1062 : : {  1063 : 0 : pTF = reinterpret\_cast<SDF12iTFHeader \*>(  1064 : : mOutputBuffer.getWritableMemory(  1065 : : sizeof(SDF12iTFHeader)) );  1066 : : }  1067 [ # # ]: 0 : catch(DPL::Error \*pChild)  1068 : : {  1069 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1070 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  1071 : : ErrorHandler::eErrFunction,  1072 : : pErr->getErrorMsg(  1073 : : ErrorHandler::eErrFunction,  1074 : : "OutputBuffer::getWritableMemory"));  1075 : : }  1076 : : SDF12iTFHeader \*pLastTF  1077 : : = reinterpret\_cast<SDF12iTFHeader \*>(  1078 : 0 : mOutputBuffer.getBuffer() + mLastWrittenTF);  1079 : 0 : \*pTF = \*pLastTF;  1080 : 0 : pLastTF->mNShot = cMaxShotCountPerTF - 1;  1081 : 0 : mChecksumCalculator.updateChecksums( pLastTF );  1082 : 0 : pAreaAcc->incrementTFCount();  1083 : 0 : mLastWrittenTF = reinterpret\_cast<char \*>(pTF)  1084 : : - mOutputBuffer.getBuffer();  1085 : 0 : writtenShotCount = 0;  1086 : : }  1087 [ + + ]: 16058 : if ( index < nullShotCount )  1088 : : {  1089 : : SDF12iShot \*pNullShot  1090 : : = reinterpret\_cast<SDF12iShot\*>(  1091 : : mOutputBuffer.getWritableMemory(  1092 : 80 : sizeof( SDF12iShot ) ) );  1093 : 80 : pNullShot->mFG1 = eFlagShot;  1094 : 80 : pNullShot->mXshot = 0;  1095 : 80 : pNullShot->mYshot = 0;  1096 : 80 : pNullShot->mLX = 0;  1097 : 80 : pNullShot->mLY = 0;  1098 : 80 : uint32\_t shotTime = (uint32\_t)nullShotTime;  1099 : 80 : pNullShot->mTshot = shotTime;  1100 : 80 : pAreaAcc->addShotTime( shotTime );  1101 : 80 : mSumTShotTime += (shotTime - mShotTimeThr)  1102 : : / mShotTimeStep;  1103 : :   1104 : 80 : pNullShot->mTsettling = (uint32\_t)nullSettlingTime;  1105 : 80 : mSumTSettlingTime += pNullShot->mTsettling;  1106 [ + + ]: 80 : pNullShot->mTCcode = (isFirstShotForTCcode)?0x04:0x00;  1107 [ + + ]: 80 : pNullShot->mTCcode |= (isTFLessMode)?0x08:0x00;  1108 : 80 : isFirstShotForTCcode = false;  1109 : 80 : mLastWrittenShot = reinterpret\_cast<char \*>(pNullShot)  1110 : : - mOutputBuffer.getBuffer();  1111 : 80 : pAreaAcc->incrementShotCount();  1112 : 80 : pAreaAcc->addSettlingTime( pNullShot->mTsettling );  1113 : : //The following is just to let the calc know that the fisrt  1114 : : //shot is a null shot  1115 : 80 : mTDSizerSetlCalc->calculate( LongPosition(), LongPosition());  1116 : 80 : mChecksumCalculator.updateChecksums( pNullShot );  1117 : :   1118 : :   1119 : 80 : writtenShotCount++;  1120 [ + + ]: 80 : if(isTFLessMode) {  1121 : 40 : return;  1122 : : }  1123 : : }  1124 : : else   1125 : : {   1126 : 15978 : const SimplePattern \*pattern = patterns + index - nullShotCount;  1127 : 15978 : pAreaAcc->accumulateArea( \*pattern );  1128 : : SDF12iShot \*pShot  1129 : : = reinterpret\_cast<SDF12iShot\*>(  1130 : : mOutputBuffer.getWritableMemory(  1131 : 15978 : sizeof( SDF12iShot ) ) );  1132 : 15978 : pShot->mFG1 = eFlagShot;  1133 : 15978 : pShot->mXshot = pattern->getPositionX();  1134 : 15978 : pShot->mYshot = pattern->getPositionY();  1135 : 15978 : pShot->mLX = pattern->getSizeX();  1136 : 15978 : pShot->mLY = pattern->getSizeY();  1137 : 15978 : uint16\_t shotTime = 0;  1138 : 15978 : try  1139 : : {  1140 : 15978 : shotTime  1141 : : = mShotTimeCalc.calculateShotTimeRepetition(  1142 : : pattern, mCurrentRepetiton, SFLNo);  1143 : : }  1144 [ - + ]: 6 : catch(DPL::Error \*pChild)  1145 : : {  1146 : 6 : ErrorHandler \*pErr = ErrorHandler::instance();  1147 : 6 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  1148 : : ErrorHandler::eErrFunction,  1149 : : pErr->getErrorMsg(  1150 : : ErrorHandler::eErrFunction,  1151 : : "ShotTimeCalculator::calculateShotTimeRepetition"));  1152 : : }  1153 : :   1154 : 15972 : pShot->mTshot = shotTime;  1155 : 15972 : pAreaAcc->addShotTime( shotTime );  1156 : 15972 : shotTime = static\_cast<uint16\_t>( shotTime \* tddcCoeff  1157 : : + docParam.shot\_offset );  1158 : 15972 : mSumTShotTime += (shotTime - mShotTimeThr)  1159 : : / mShotTimeStep;  1160 : : double shotSettlTime = mTDSizerSetlCalc->calculate(  1161 : : LongPosition( pShot->mXshot, pShot->mYshot ),  1162 : 15972 : LongPosition( pShot->mLX, pShot->mLY ) );  1163 : : double finalShotSettlingTime = (mIsFirstShot  1164 : : &&shotSettlTime < mCurrentSubSettlingTime )  1165 [ + + ][ + + ]: 15970 : ? mCurrentSubSettlingTime : shotSettlTime ;  1166 : : //#ifdef ERROR\_SEED //<<Test\_Id:OutputGenerator\_finalShotSettlingTime\_Set\_To\_True>>  1167 : : // finalShotSettlingTime = true;  1168 : : // uint32\_t fsst = static\_cast<uint16\_t>(finalShotSettlingTime);  1169 : : // fsst = 65536;  1170 : : //#endif  1171 [ + - ]: 15970 : if( finalShotSettlingTime )  1172 : : {  1173 [ + + ]: 15970 : if( (1<<10) < static\_cast<uint16\_t>(finalShotSettlingTime) )  1174 : : {  1175 : 8 : Logger::instance()->log(  1176 : : LogMessageTable::eSHOT\_RANGE\_OVERFLOW,  1177 : : (int64\_t)finalShotSettlingTime,  1178 : : 0, 1<<10);  1179 : 8 : ErrorHandler \*pErr = ErrorHandler::instance();  1180 : 8 : pErr->errorAdd(0, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  1181 : : ErrorHandler::eErrRangeOverflow,  1182 : : pErr->getErrorMsg(  1183 : : ErrorHandler::eErrRangeOverflow,  1184 : : (int64\_t)finalShotSettlingTime, 0, 1<<10));  1185 : : }  1186 : : }  1187 : 15962 : pShot->mTsettling = static\_cast<uint16\_t>(  1188 : : finalShotSettlingTime);  1189 : : #ifdef OLD\_SETTL\_CALC  1190 : : if( mIsFirstShot ){  1191 : : pShot->mTsettling = mShotSettlingTime.calculate(  1192 : : pShot, pShot, currKcode, currKcode);   1193 : : }else{  1194 : :   1195 : : SDF12iTFHeader \*pLastTF  1196 : : = reinterpret\_cast<SDF12iTFHeader \*>(  1197 : : mOutputBuffer.getBuffer() + mLastWrittenTF);  1198 : : lastShot = reinterpret\_cast<SDF12iShot \*>(  1199 : : mOutputBuffer.getBuffer()  1200 : : + mLastWrittenShot);  1201 : : pShot->mTsettling = mShotSettlingTime.calculate(  1202 : : pShot, lastShot, currKcode, pLastTF->mKcode);   1203 : : }  1204 : : #endif  1205 : 15962 : mSumTSettlingTime += pShot->mTsettling;  1206 [ + + ]: 15962 : pShot->mTCcode = (isFirstShotForTCcode)?0x04:0x00;  1207 : 15962 : isFirstShotForTCcode = false;  1208 : 15962 : mLastWrittenShot = reinterpret\_cast<char \*>(pShot)  1209 : : - mOutputBuffer.getBuffer();  1210 : 15962 : mIsFirstShot = false;  1211 : 15962 : pAreaAcc->incrementShotCount();  1212 : 15962 : pAreaAcc->addSettlingTime( pShot->mTsettling );  1213 [ + + ]: 15962 : if( mCurrentRepetiton == repCount - 1 )  1214 : : {  1215 : 15926 : pattern->~SimplePattern();  1216 : : }  1217 [ + + ]: 15962 : if( isTFLessMode )  1218 : : {  1219 : 99 : pShot->mXshot = 0;  1220 : 99 : pShot->mYshot = 0;  1221 : : }  1222 : 15962 : mChecksumCalculator.updateChecksums( pShot );  1223 : 15962 : writtenShotCount++;  1224 : : }  1225 : : }  1226 : : }  1227 : 10522 : mIsFirstNULLShotInSF = false;  1228 : : }  1229 : :   1230 : : namespace {  1231 : : struct SFGData  1232 : : {  1233 : : int64\_t mPosition;  1234 : : SFGroup::Pointer mSFG;  1235 : : bool  1236 [ # # ][ # # ]: 0 : operator < ( const SFGData &inRhs ) const {  [ # # ][ # # ]  [ # # ][ # # ]  [ # # ]  1237 : : const DPMANAGER\_MESSAGE\_CONVERTER \* startMsg  1238 [ # # ][ # # ]: 40 : = ConverterParams::instance()->getStartupMessage();  [ - + ][ # # ]  [ # # ][ # # ]  [ # # ][ # # ]  [ # # ][ # # ]  [ # # ][ - + ]  1239 : : return startMsg->usSFOrder > 4  1240 : : ? mPosition > inRhs.mPosition  1241 : : : mPosition < inRhs.mPosition;  1242 : : }  1243 : : };  1244 : : }  1245 : :   1246 : : void  1247 : : OutputGenerator::distributeCellForWriting(  1248 : 63 : ArbitraryModeCellContainer \*inCell ) throw(DPL::Error \*) {  1249 : 63 : std::vector<SFGData> sfgsToBeWritten;  1250 : 63 : SFTable::Pointer sfTable = inCell->getSFT();  1251 : 63 : SFLayer::Pointer sfls = sfTable->getSFLayers();  1252 : 63 : uint32\_t numSFL = mStartMsg->uiSFLCountInDpb;  1253 : 63 : int32\_t sfWidth = mStartMsg->iSFSizeX;  1254 [ + - ][ + + ]: 146 : for( SFLayer::Pointer layer = sfls; layer != sfls + numSFL; layer++ )  1255 : : {  1256 : 83 : SFGroup::Pointer sfgs = layer->getSFGroups();  1257 : 83 : uint32\_t numCols = layer->getNoOfCols();  1258 [ + - ][ - + ]: 166 : for( uint32\_t col = 0; col < numCols; col++ )  1259 : : {  1260 : 83 : SFGData data;  1261 : 83 : data.mPosition = layer->getLLx() + col \* sfWidth;  1262 : 83 : data.mSFG = sfgs + col;  1263 : 83 : sfgsToBeWritten.push\_back( data );  1264 : : }  1265 : : }  1266 : 63 : sort( sfgsToBeWritten.begin(), sfgsToBeWritten.end() );  1267 : 63 : for(std::vector<SFGData>::iterator it = sfgsToBeWritten.begin();  1268 : : it != sfgsToBeWritten.end(); ++it )  1269 : : {  1270 : 83 : try  1271 : : {  1272 : 83 : writeSFGRepeatedly( (\*it).mSFG );  1273 : : }  1274 [ # # ]: 0 : catch( DPL::Error \*pChild )  1275 : : {  1276 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1277 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  1278 : : ErrorHandler::eErrFunction,  1279 : : pErr->getErrorMsg(  1280 : : ErrorHandler::eErrFunction,  1281 : : "OutputGenerator::writeSFGRepeatedly"));  1282 : : }  1283 : : }  1284 : 63 : try  1285 : : {  1286 : 63 : SDDB::instance()->cellWritten();  1287 : : }  1288 [ # # ]: 0 : catch(DPL::Error \*pChild)  1289 : : {  1290 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1291 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_, \_\_LINE\_\_,  1292 : : ErrorHandler::eErrFunction,  1293 : : pErr->getErrorMsg( ErrorHandler::eErrFunction,  1294 : : "SDDB::cellWritten"));  1295 : : }  1296 : : }  1297 : :   1298 : : bool  1299 : 7 : OutputGenerator::setLastStripeFlag() {  1300 : 7 : if( mOutputBuffer.getOutputData().getMemSize() == 0 )  1301 : : {  1302 : 1 : return false;  1303 : : }  1304 : : SDF12iShot \*lastShot = reinterpret\_cast<SDF12iShot \*>(  1305 : 6 : mOutputBuffer.getBuffer() + mLastWrittenShot);  1306 : 6 : mChecksumCalculator.removeChecksums( lastShot );  1307 : 6 : lastShot->mTCcode = lastShot->mTCcode | 0x02;  1308 : 6 : mChecksumCalculator.updateChecksums( lastShot );  1309 : : SDF12iTFHeader \*pLastTF = reinterpret\_cast<SDF12iTFHeader \*>(  1310 : 6 : mOutputBuffer.getBuffer() + mLastWrittenTF);  1311 : 6 : mChecksumCalculator.removeChecksums( pLastTF );  1312 : 6 : pLastTF->mCcode = pLastTF->mCcode | 0x02;  1313 : 6 : mChecksumCalculator.updateChecksums( lastShot );  1314 : : SDF12iCheckSumData \*pChecksum = reinterpret\_cast<SDF12iCheckSumData \*>(  1315 : : mOutputBuffer.getBuffer()  1316 : : + mOutputBuffer.getOutputData().getMemSize()  1317 : 6 : - sizeof( SDF12iCheckSumData ));  1318 : 6 : pChecksum->mChkSumS0 = mChecksumCalculator.getSum0();  1319 : 6 : pChecksum->mChkSumS1 = mChecksumCalculator.getSum1();  1320 : 6 : pChecksum->mChkSumS2 = mChecksumCalculator.getSum2();  1321 : 7 : return true;  1322 : : }  1323 : :   1324 : 3282 : } //namespace nft |

|  |
| --- |
| Generated by: [LCOV version 1.9](http://ltp.sourceforge.net/coverage/lcov.php) |