| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [PatternProcessor/include](http://docs.google.com/index.html) - PatternProcessor.h (source / [functions](http://docs.google.com/PatternProcessor.h.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 867 | 1056 | 82.1 % | | Date: | 2012-03-23 |  | Functions: | 12 | 12 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 166 | 334 | 49.7 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : #ifndef NFT\_PATTERNPROCESSOR\_H  2 : : #define NFT\_PATTERNPROCESSOR\_H  3 : :   4 : : #include <boost/function.hpp>  5 : :   6 : : #include "DPL.h"  7 : :   8 : : #include "ParsedPattern.h"  9 : : #include "Position.h"  10 : : #include "PatternEdgeVectors.h"  11 : : #include "DivisionParams.h"  12 : : #include "ThreadID.h"  13 : : #include "RetryFunc.h"  14 : :   15 : : #include "PatternExploder.h"  16 : : #include "SimplePattern.h"  17 : : #include "ShotShiftCalculator.h"  18 : : #include "Logger.h"  19 : : #include "ErrorHandler.h"  20 : :   21 : : #include "print.h"  22 : : #include "SDDB.h"  23 : : namespace nft { class PatternExploder; }   24 : : namespace nft { class SimplePattern; }   25 : :   26 : : namespace nft {  27 : :   28 : : typedef boost::function<void(const SimplePattern&, const PatternPosition &, uint32\_t, ThreadID, uint32\_t)>  29 : : TShotDistributor;  30 : :   31 : : class PatternProcessor : public NftNewDeleteBase {  32 : : public:  33 : : //Sets the various parameters used for pattern division  34 : : inline void  35 : : setDivisionParams(DivisionParams \*divParam);  36 : :   37 : : inline void  38 : : setMode(PatternDivisionMode inMode);  39 : :   40 : : inline void  41 : : setEdgeVector(PatternEdgeVectors inEdgeVector);  42 : :   43 : : inline void  44 : : setThreadID( ThreadID inThreadID );  45 : :   46 : : inline void registerDistributor(const TShotDistributor &func);  47 : :   48 : : inline void adjustPatternOrigin(ParsedPattern& inPat) const;  49 : :   50 : : //Converts an input pattern to Shot format.  51 : : inline void  52 : : process(const ParsedPattern & inputPattern, uint32\_t inSFLNo) throw(DPL::Error \*);  53 : : //Writes the shot to all sub-fields  54 : : //void  55 : : //writePattern(const SimplePattern & wShotPattern);  56 : :   57 : : ////Writes the shot to all sub-fields after shifting and copying  58 : : //void  59 : : //duplicateAndWritePattern(const SimplePattern & dwShotPattern);  60 : :   61 : : inline PatternProcessor ();  62 : : inline ~PatternProcessor ();  63 : :   64 : : private:  65 : : inline void slicePattern(const ParsedPattern& pattern, uint32\_t inSFLNo);  66 : :   67 : : inline void slicePatternFixedDivisionMode(const ParsedPattern& pattern,  68 : : uint32\_t inSFLNo);  69 : :   70 : : inline void slicePatternSlopeDivisionMode(const ParsedPattern& pattern,  71 : : uint32\_t inSFLNo);   72 : :   73 : : //void adjustPatternOrigin(ParsedPattern& pat) const  74 : : //;  75 : :   76 : : inline void sliceHorizontalNonLNPlgPattern (const ParsedPattern& pat,   77 : : const DIV\_TH& sliceParam,  78 : : const DIV\_TH& divideParam,  79 : : uint32\_t inSFLNo);  80 : :   81 : : inline void sliceHorizontalLNPlgPattern (const ParsedPattern& pat,  82 : : const DIV\_TH& sliceParam,  83 : : const DIV\_TH& divideParam,  84 : : uint32\_t inSFLNo);  85 : :   86 : : inline void prepareNextHorizontalSlice (ParsedPattern& patternSlice,   87 : : uint32\_t divisionLength);  88 : :   89 : : inline uint32\_t getLxForFirstNonLNPlgHorizontalSlice  90 : : (const ParsedPattern& pat,  91 : : uint32\_t divisionLength);  92 : :   93 : : inline uint32\_t getLxForNextNonLNPlgHorizontalSlice  94 : : (const ParsedPattern& pat,  95 : : uint32\_t prevLy,  96 : : uint32\_t thisLy);  97 : :   98 : : inline void divideHorizontalPatternSlice (const ParsedPattern& pat,   99 : : const DIV\_TH& divisionParam,  100 : : uint32\_t inSFLNo);  101 : :   102 : : inline void createHorizontalRectangle  103 : : (ParsedPattern& pattern,  104 : : const DIV\_TH& divisionParam,  105 : : uint32\_t inSFLNo);  106 : :   107 : : inline void prepareNextHorizontalRectangle  108 : : (ParsedPattern& patternDivide,  109 : : uint32\_t divisionLength);  110 : :   111 : : inline void createLeftTriangle\_0x07(uint32\_t x0,   112 : : uint32\_t y0,   113 : : uint32\_t size,   114 : : uint32\_t attribute,  115 : : uint32\_t inSFLNo);  116 : :   117 : : inline void createRightTriangle\_0x07(uint32\_t x0,   118 : : uint32\_t y0,   119 : : uint32\_t size,   120 : : uint32\_t attribute,  121 : : uint32\_t inSFLNo);  122 : :   123 : : inline void createLeftTriangle\_0x08(uint32\_t x0,   124 : : uint32\_t y0,   125 : : uint32\_t size,   126 : : uint32\_t attribute,  127 : : uint32\_t inSFLNo);  128 : :   129 : : inline void createRightTriangle\_0x08(uint32\_t x0, uint32\_t y0,  130 : : uint32\_t size, uint32\_t attribute,  131 : : uint32\_t inSFLNo);  132 : :   133 : : inline void createHorizontalLeftTriangle(ParsedPattern& inPat,  134 : : uint32\_t inSFLNo);  135 : :   136 : : inline void createHorizontalRightTriangle(ParsedPattern& inPat,  137 : : uint32\_t inSFLNo);  138 : :   139 : : inline void sliceVerticalNonLNPlgPattern (const ParsedPattern& pat,  140 : : const DIV\_TH& sliceParam,  141 : : const DIV\_TH& divideParam,  142 : : uint32\_t inSFLNo);  143 : :   144 : 255 : inline void sliceVerticalLNPlgPattern (const ParsedPattern& pat,  145 : : const DIV\_TH& sliceParam,  146 : : const DIV\_TH& divideParam,  147 : : uint32\_t inSFLNo);  148 : 372 :   149 : : inline void prepareNextVerticalSlice (ParsedPattern& patternSlice,   150 : : uint32\_t divisionLength);  151 : :   152 : : inline uint32\_t getLyForFirstNonLNPlgVerticalSlice (const ParsedPattern& pat,  153 : : uint32\_t divisionLength);  154 : :   155 : : inline uint32\_t getLyForNextNonLNPlgVerticalSlice (const ParsedPattern& pat,  156 : : uint32\_t prevLx, uint32\_t thisLx);  157 : :   158 : : inline void divideVerticalPatternSlice (const ParsedPattern& pat,   159 : : const DIV\_TH& divisionParam,  160 : : uint32\_t inSFLNo);  161 : :   162 : : inline void createVerticalRectangle(ParsedPattern& inPat,   163 : : const DIV\_TH& divisionParam,  164 : : uint32\_t inSFLNo);  165 : :   166 : : inline void prepareNextVerticalRectangle (ParsedPattern& patternDivide,  167 : : uint32\_t divisionLength);  168 : :   169 : : inline void createBottomTriangle\_0x05(uint32\_t x0,   170 : : uint32\_t y0,   171 : : uint32\_t size,  172 : : uint32\_t attribute,  173 : : uint32\_t inSFLNo);  174 : :   175 : : inline void createTopTriangle\_0x05(uint32\_t x0,   176 : : uint32\_t y0,   177 : : uint32\_t size,   178 : : uint32\_t attribute,  179 : : uint32\_t inSFLNo);  180 : :   181 : : inline void createBottomTriangle\_0x06(uint32\_t x0,   182 : : uint32\_t y0,   183 : : uint32\_t size,   184 : : uint32\_t attribute,  185 : : uint32\_t inSFLNo);  186 : :   187 : : inline void createTopTriangle\_0x06(uint32\_t x0,   188 : : uint32\_t y0,   189 : : uint32\_t size,   190 : : uint32\_t attribute,  191 : : uint32\_t inSFLNo);  192 : :   193 : : inline void createVerticalBottomTriangle(ParsedPattern& inPat,  194 : : uint32\_t inSFLNo);  195 : :   196 : : inline void  197 : : createVerticalTopTriangle(ParsedPattern& inPat,  198 : : uint32\_t inSFLNo);  199 : :   200 : : inline uint32\_t  201 : : calculateShiftValueForSlice( uint32\_t inSFLNo, uint32\_t MaxShiftNum,  202 : : uint32\_t inShotMax)  203 : 948 : {  204 : 474 : uint32\_t sflId = (mLayers + inSFLNo)->uiSFLID;  205 : 474 : uint32\_t shiftValue = ((sflId - 1) % MaxShiftNum) \* inShotMax/MaxShiftNum;  206 : : return shiftValue;  207 : : }  208 : :   209 : : private:  210 : : //Blocked  211 : : PatternProcessor(const PatternProcessor &);  212 : : PatternProcessor &operator=(const PatternProcessor &);  213 : :   214 : : //Holds the list of all shot patterns generated from this parsed pattern.  215 : : MemInfo mOutputHolder;  216 : :   217 : : TShotDistributor mShotDistributor;  218 : :   219 : : ThreadID mThdID;  220 : :   221 : : //The division parameters for pattern division.  222 : : DivisionParams \*mDivisionParams;  223 : :   224 : : //Division modes  225 : : PatternDivisionMode mMode;  226 : :   227 : : PatternEdgeVectors mEdgeVector;  228 : :   229 : : const DPMANAGER\_MESSAGE\_SFL\_INFO \*mLayers;  230 : :   231 : : std::vector<SimplePattern> spat, spatOut;  232 : : std::vector<PatternPosition> pos, posOut;  233 : :   234 : : uint16\_t mMaxShotShiftCount;  235 : : };  236 : :   237 : : inline   238 : : PatternProcessor::PatternProcessor()  239 : 1142 : :mOutputHolder(),  240 : : mThdID(),   241 : : mDivisionParams(),  242 : : mMode(),  243 : : mEdgeVector(),  244 : : mLayers(ConverterParams::instance()->getLayerParams() ),  245 : : spat(5), spatOut(5), pos(5), posOut(5),  246 : : mMaxShotShiftCount(ConverterParams::instance()  247 : 2410 : ->getStartupMessage()->usMaxShotShiftCount) {  248 : : }   249 : : inline   250 : 1142 : PatternProcessor::~PatternProcessor(){  251 : 1142 : delete mDivisionParams;  252 : : }   253 : :   254 : : //Sets the various parameters used for pattern division  255 : : inline void  256 : 1142 : PatternProcessor::setDivisionParams(DivisionParams \*divParam) {  257 : 1142 : mDivisionParams = divParam;  258 : : }  259 : :   260 : : inline void  261 : 1142 : PatternProcessor::setMode(PatternDivisionMode inMode) {  262 : 1142 : mMode = inMode;  263 : : }  264 : :   265 : : inline void  266 : : PatternProcessor::setEdgeVector(PatternEdgeVectors inEdgeVector) {  267 : : mEdgeVector = inEdgeVector;  268 : : }  269 : :   270 : : inline void  271 : 1142 : PatternProcessor::setThreadID( ThreadID inThreadID ) {  272 : 1142 : mThdID = inThreadID;  273 : : }  274 : :   275 : : /\* Translate the origin of a pattern so that it refers to  276 : : the LL of the "pattern" and NOT to the LL of the "bounding  277 : : box" of the pattern.  278 : : \*/  279 : : inline void   280 : : PatternProcessor::adjustPatternOrigin(ParsedPattern& inPat) const   281 : 8106 : {  282 : 8106 : switch(inPat.getCode())  283 : : {  284 : : //TBD: Insert the case of vertical patterns  285 : : case 0x04:  286 : : case 0x05:  287 : : case 0x0e:  288 : 189 : inPat.setPositionY( inPat.getPositionY() + inPat.getSizeX());  289 : 189 : break;  290 : : case 0x08:  291 : : case 0x0c:  292 : : case 0x10:  293 : 62 : inPat.setPositionX( inPat.getPositionX() + inPat.getSizeY());  294 : 8106 : break;  295 : 8106 : }  296 : : return;  297 : : }  298 : :   299 : : //Converts an input pattern to Shot format.  300 : : inline void  301 : : PatternProcessor::process(const ParsedPattern & inputPattern,  302 : 8106 : uint32\_t inSFLNo) throw(DPL::Error \*) {  303 : 8106 : e9k\_debug("Pattern received at %lld %lld\n",  304 : : inputPattern.getPositionX(), inputPattern.getPositionY());  305 : 8106 : slicePattern(inputPattern, inSFLNo);  306 : : }  307 : :   308 : : //Breaks up the input pattern into several slices along the parallel edges of the pattern.  309 : : inline void  310 : : PatternProcessor::slicePattern(const ParsedPattern & outParsedPattern,  311 : : uint32\_t inSFLNo  312 : 8106 : ) {  313 : : //#ifdef ERROR\_SEED //<<Test\_Id:PatternProcessor\_slicePattern>>  314 : : // mMode = (PatternDivisionMode)-1;  315 : : //#endif  316 [ + + ][ - + ]: 8106 : switch(mMode)  317 : : {  318 : : case eFixedDivisionMode:  319 : 7863 : slicePatternFixedDivisionMode(outParsedPattern, inSFLNo);  320 : 7847 : break;  321 : : case eSlopeDivisionMode:  322 : 243 : slicePatternSlopeDivisionMode(outParsedPattern, inSFLNo);  323 : 243 : break;  324 : : default:  325 : 0 : Logger::instance()->log(LogMessageTable::ePATTERN\_PROCESSOR\_UNSUPPORTED\_DIVISION\_MODE,  326 : : (char\*)\_\_FILE\_\_, (char\*)\_\_FUNCTION\_\_, (int)\_\_LINE\_\_,  327 : : "Unsupported division mode");  328 : :   329 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  330 : 0 : pErr->errorAdd(NULL, (char\*)\_\_FILE\_\_, (char\*)\_\_FUNCTION\_\_, (int)\_\_LINE\_\_  331 : : ,ErrorHandler::eErrMessageString,  332 : : pErr->getErrorMsg((ErrorHandler::eErrMessageString)  333 : : , "Unsupported division mode") );  334 [ - + ]: 16 : break;  335 : : }  336 : : }  337 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  338 : : \* Function Name: slicePatternFixedDivisionMode  339 : : \* Purpose: Breaks up the input pattern into several slices   340 : : \* along the parallel edges of the pattern.  341 : : \* in FIXED\_DIVISION\_MODE  342 : : \* Parameters: pattern: ParsedPattern [IN]  343 : : \* Return Values:  344 : : \* None  345 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  346 : :   347 : : inline void PatternProcessor::slicePatternFixedDivisionMode(const ParsedPattern& inPat,  348 : : uint32\_t inSFLNo  349 : : )  350 : 7863 : {  351 : 7863 : ParsedPattern pat = inPat;  352 : 7863 : adjustPatternOrigin(pat);  353 : : //Obtain the divth for the pattern  354 : 7863 : DIV\_TH sliceParam = mDivisionParams->getDivThForFixedDivisionMode();  355 : 7863 : DIV\_TH divideParam = sliceParam;   356 : : //TODO  357 : : //div\_tbl max value > 4096 then error should ne thrown  358 : :   359 : : //Perf Fix  360 : 7863 : uint32\_t sflId = (mLayers + inSFLNo)->uiSFLID;  361 : 7863 : if (pat.getCode() == 0x11   362 : : && pat.getSizeX() <= sliceParam.max   363 : : && pat.getSizeY() <= sliceParam.max )  364 : : {   365 : : //Directly generate call the fixed mode shot distributor  366 : : //divCase = 1;   367 : 4994 : pat.setCode(0x01);  368 : 4994 : SDDB \*pSDDB = SDDB::instance();  369 : 4994 : pat.createSimplePattern(spat[0], pos[0]);  370 : 4994 : try  371 : : {  372 : 4994 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  373 : : }  374 [ # # ]: 0 : catch(DPL::Error \*pChild)  375 : : {  376 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  377 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  378 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  379 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  380 : : "SDDB::SimplePatternGenerated"));  381 : : }  382 : :   383 : 4994 : try   384 : : {  385 : 4994 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  386 : : }  387 [ # # ]: 0 : catch(DPL::Error \*pChild)  388 : : {  389 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  390 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  391 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  392 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  393 : : "SDDB::SimplePatternShifted"));  394 : : }  395 : 4994 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pat.getAI());  396 : 4994 : return;   397 : : }  398 : : //End Fix  399 : :   400 : 2446 : switch (pat.getCode())  401 : : {  402 : : case 0x00:  403 : : case 0x11:  404 : 423 : sliceHorizontalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  405 : 407 : break;  406 : : case 0x01:  407 : : case 0x02:  408 : : case 0x03:  409 : : case 0x06:  410 : 1963 : sliceVerticalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  411 : 1963 : break;  412 : : case 0x04:  413 : : case 0x05:  414 : : {  415 : 135 : sliceVerticalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  416 : 135 : break;  417 : : }  418 : : case 0x07:  419 : : case 0x09:  420 : : case 0x0a:  421 : : case 0x0b:  422 : 220 : sliceHorizontalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  423 : 220 : break;  424 : : case 0x0c:  425 : : case 0x08:  426 : : {  427 : 22 : sliceHorizontalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  428 : 22 : break;  429 : : }  430 : : case 0x0d:  431 : : case 0x0e:  432 : 50 : if (pat.getSizeY() >= sliceParam.max) //normal parallelograms  433 : : {  434 : 21 : sliceVerticalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  435 : : }  436 : : else //long and narrow parallelograms  437 : : {  438 : 29 : sliceVerticalLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  439 : : }  440 : 29 : break;  441 : : case 0x0f:  442 : : case 0x10:  443 : : {  444 : 56 : if (pat.getSizeX() >= sliceParam.max) //normal parallelograms  445 : : {  446 : 14 : sliceHorizontalNonLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  447 : : }  448 : : else //long and narrow parallelograms  449 : : {  450 : 42 : sliceHorizontalLNPlgPattern(pat, sliceParam, divideParam, inSFLNo);  451 : : }  452 : : }  453 : 42 : break;  454 : : }  455 : :   456 : : }  457 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  458 : : \* Function Name: slicePatternSlopeDivisionMode  459 : : \* Purpose: To slice a parsed pattern and divide its slices  460 : : \* in SLOPE\_DIVISION\_MODE  461 : : \* Parameters: pattern: ParsedPattern [IN]  462 : : \* Return Values:  463 : : \* None  464 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  465 : :   466 : : inline void   467 : : PatternProcessor::slicePatternSlopeDivisionMode(  468 : : const ParsedPattern& pat,  469 : : uint32\_t inSFLNo  470 : : )  471 : 243 : {  472 : 243 : ParsedPattern patternSlice = pat;  473 : 243 : adjustPatternOrigin(patternSlice);  474 : : //Obtain the divth for the pattern  475 : 243 : DIV\_TH sliceParam;  476 : 243 : DIV\_TH divideParam;  477 : :   478 : 243 : switch (pat.getCode())  479 : : {  480 : : case 0x00:  481 : : case 0x11:  482 : 35 : sliceParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eXDirection);  483 : 35 : divideParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eYDirection);   484 : 35 : sliceHorizontalNonLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  485 : 35 : break;  486 : : case 0x01:  487 : : case 0x02:  488 : : case 0x03:  489 : : case 0x04:  490 : : case 0x05:  491 : : case 0x06:  492 : 102 : sliceParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eYDirection);  493 : 102 : divideParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eXDirection);  494 : 102 : sliceVerticalNonLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  495 : 102 : break;  496 : : case 0x07:  497 : : case 0x08:  498 : : case 0x09:  499 : : case 0x0a:  500 : : case 0x0b:  501 : : case 0x0c:  502 : 60 : sliceParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eXDirection);  503 : 60 : divideParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eYDirection);  504 : 60 : sliceHorizontalNonLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  505 : 60 : break;  506 : : case 0x0d:  507 : : case 0x0e:  508 : 23 : sliceParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eYDirection);  509 : 23 : divideParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eXDirection);  510 : 23 : if (pat.getSizeY() >= sliceParam.max) //normal parallelograms  511 : : {  512 : 10 : sliceVerticalNonLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  513 : : }  514 : : else //long and narrow parallelograms  515 : : {   516 : 13 : sliceVerticalLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  517 : : }  518 : 13 : break;  519 : : case 0x0f:  520 : : case 0x10:  521 : : {  522 : 23 : sliceParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eXDirection);  523 : 23 : divideParam = mDivisionParams->getDivThForSlopeDivisionMode(pat, eYDirection);  524 : 23 : if (pat.getSizeX() >= sliceParam.max) //normal parallelograms  525 : : {  526 : 12 : sliceHorizontalNonLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  527 : : }  528 : : else //long and narrow parallelograms  529 : : {  530 : 11 : sliceHorizontalLNPlgPattern(patternSlice, sliceParam, divideParam, inSFLNo);  531 : : }  532 : : }  533 : 11 : break;  534 : : }   535 : :   536 : : }  537 : : //Slicing: Horizontal Patterns  538 : :   539 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  540 : : \* Function Name: sliceHorizontalNonLNPlgPattern  541 : : \* Purpose: Slice horizontal patterns (except long & narrow   542 : : \* parallelograms)  543 : : \* Parameters:  544 : : \* pat: ParsedPattern [IN]  545 : : \* sliceParam: DIV\_TH [IN] [Division Threshold for slicing]  546 : : \* divideParam: DIV\_TH [IN] [Division Threshold for division]  547 : : \* Return Values:  548 : : \* None  549 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  550 : :   551 : : inline void PatternProcessor::sliceHorizontalNonLNPlgPattern   552 : : (const ParsedPattern& pat,   553 : : const DIV\_TH& sliceParam,   554 : : const DIV\_TH& divideParam,  555 : : uint32\_t inSFLNo  556 : : )  557 : 786 : {  558 : 786 : ParsedPattern patternSlice = pat;  559 : 786 : ShotShiftCalculator shotShift;  560 : 786 : int32\_t divCase;  561 : 786 : uint32\_t lastDivLength;  562 : :   563 : 324 : if (patternSlice.getSizeY() <= sliceParam.max) { divCase = 1; }  564 : 320 : else if (patternSlice.getSizeY() <= static\_cast<uint32\_t>(2\*sliceParam.max) ) { divCase = 2; }  565 : 67 : else if (patternSlice.getSizeY() <= static\_cast<uint32\_t>(3\*sliceParam.max) ) { divCase = 3; }  566 : 75 : else { divCase = 4; }  567 : 786 : int32\_t i;  568 : 786 : uint32\_t divisionLength = 0, firstDivisionLength = 0;   569 [ + + ]: 786 : if (divCase < 4)  570 : : {  571 : : //Fixed division length when no shift is applied  572 : 711 : divisionLength = pat.getSizeY() / divCase;  573 : 711 : uint32\_t sizeY = divisionLength + (pat.getSizeY() % divCase);   574 : 2918 : patternSlice = pat;  575 : : // --------------- --------------- ---------------  576 : : // | 2 | | 3 | | 4 |  577 : : // --------------- --------------- ---------------  578 : : // | ////////// | | | | |  579 : : // | 1 | | 2 | | 3 |  580 : : // --------------- --------------- ---------------  581 : : // | ////////// | | |  582 : : // | 1 | | 2 |  583 : : // --------------- ---------------  584 : : // | ////1///// |  585 : : // ---------------  586 : : // divCase == 1 divCase == 2 divCase == 3  587 : :   588 : :   589 : : //Variable Shift length for the last(no 1) slice  590 : : //firstDivisionLength = calculateShiftValueForSlice(  591 : : // inSFLNo,mMaxShotShiftCount,sliceParam.max);  592 [ + + ]: 711 : if(divCase != 1){   593 : 387 : firstDivisionLength = shotShift.calculateShiftValue(  594 : : mMaxShotShiftCount, pat.getSizeY(), sliceParam.max, inSFLNo);  595 : : }  596 [ + + ]: 387 : if(firstDivisionLength > 0){  597 : : //Handle the last(no 1) slice  598 : 3 : patternSlice.setSizeY( firstDivisionLength );  599 : 3 : patternSlice.setSizeX( getLxForFirstNonLNPlgHorizontalSlice(pat, firstDivisionLength) );  600 : : }else{  601 : : //No shift or shift == 0  602 : 708 : divCase--;  603 : 708 : patternSlice.setSizeX( getLxForFirstNonLNPlgHorizontalSlice(pat, divisionLength) );  604 : 708 : patternSlice.setSizeY( divisionLength );  605 : : }  606 [ + + ][ + + ]: 1154 : for (i=0; i<divCase; i++){  607 : : //Apply division algo for this slice  608 : 457 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  609 [ + + ]: 443 : if( (i == divCase-1) && (firstDivisionLength > 0) ){  610 : : //calculate the origin and dimensions of the slice(no 3)  611 : 3 : prepareNextHorizontalSlice(patternSlice, divisionLength - firstDivisionLength);  612 : : //Adjust the position Y  613 : 3 : patternSlice.setPositionY(patternSlice.getPositionY()   614 : : + firstDivisionLength);  615 : 3 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(  616 : : patternSlice,   617 : : divisionLength-firstDivisionLength,   618 : : patternSlice.getSizeY()) );  619 : 3 : patternSlice.setSizeY( divisionLength-firstDivisionLength );  620 : : }else{  621 : : //calculate the origin and dimensions of the next slice(no 2, i e, n slices)  622 : 440 : prepareNextHorizontalSlice(patternSlice, divisionLength);  623 : : //Adjust the position Y for the 2nd pattern slice  624 [ + + ]: 440 : if( (i == 0) && (firstDivisionLength > 0) ){  625 : 3 : patternSlice.setPositionY(patternSlice.getPositionY()   626 : : - divisionLength + firstDivisionLength);  627 : 3 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(  628 : : patternSlice,   629 : : firstDivisionLength,   630 : : patternSlice.getSizeY()) );  631 : :   632 : : }  633 : : else{  634 [ + + ]: 437 : if(divCase - 1 == i){  635 : 370 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(  636 : : patternSlice,   637 : : divisionLength,   638 : : sizeY) );  639 : : } else {  640 : 67 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(  641 : : patternSlice,   642 : : divisionLength,   643 : : patternSlice.getSizeY()) );  644 : : }  645 : : }  646 [ + + ]: 440 : if(divCase - 1 == i)  647 : 370 : patternSlice.setSizeY(sizeY); //Adjust the size for the Last Slice  648 : : else  649 : 70 : patternSlice.setSizeY(divisionLength);  650 : : }  651 : : }  652 : 697 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  653 : : }  654 : : else  655 : : {  656 : 75 : uint32\_t midPatternLengthY = pat.getSizeY() - (sliceParam.edge<<1); //replaced \*2 by <<1  657 : 75 : uint32\_t midPatternCount = (midPatternLengthY - 1)/sliceParam.max;  658 : 75 : uint32\_t L1 = midPatternLengthY/(midPatternCount +1);  659 : 75 : L1 += ((midPatternCount +1)\*L1 < midPatternLengthY)?1:0;  660 : 75 : uint32\_t L2 = midPatternLengthY - (midPatternCount \* L1);  661 : 75 : uint32\_t L1Count = midPatternLengthY / L1;  662 : 75 : uint32\_t L2Count = (midPatternLengthY - L1Count \* L1 >0)?1:0;  663 : 75 : uint32\_t prevLy;  664 : : //Process the 1st edge  665 : 75 : divisionLength = sliceParam.edge;  666 : 75 : patternSlice.setSizeY( sliceParam.edge );  667 : 75 : patternSlice.setSizeX( getLxForFirstNonLNPlgHorizontalSlice(pat, divisionLength) );  668 : 75 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  669 : :   670 : 73 : prevLy = divisionLength;  671 : : //Variable Shift length for the last(no 1) slice  672 : 73 : firstDivisionLength = calculateShiftValueForSlice(  673 : : inSFLNo,mMaxShotShiftCount,L1);  674 : : //Handle the last(no 1) slice  675 [ + + ]: 73 : if(firstDivisionLength > 0){  676 : : //prepareNextHorizontalSlice(patternSlice, firstDivisionLength);  677 : 11 : prepareNextHorizontalSlice(patternSlice, prevLy); //Fix 2  678 : 11 : patternSlice.setSizeY( firstDivisionLength );  679 : 11 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(patternSlice, prevLy, firstDivisionLength) );  680 : 11 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  681 : 11 : prepareNextHorizontalSlice(patternSlice, firstDivisionLength); //Fix 3  682 : :   683 : : // ---------------  684 : : // | 2ndEDGE |  685 : : // ---------------  686 : : // | 4 |  687 : : // ---------------  688 : : // | |  689 : : // | 3 |  690 : : // ---------------  691 : : // | |  692 : : // | 2 |  693 : : // ---------------  694 : : // | ////1///// |  695 : : // ---------------  696 : : // | 1stEDGE |  697 : : // ---------------  698 : :   699 : : // divCase == 3  700 : :   701 : 11 : prevLy = firstDivisionLength;  702 : : }else{  703 : : //No shift (no 2) slice  704 : 62 : prepareNextHorizontalSlice(patternSlice, divisionLength);  705 : 62 : if(firstDivisionLength > 0){  706 : 73 : patternSlice.setPositionY(patternSlice.getPositionY()   707 : : - divisionLength + firstDivisionLength);  708 : : }  709 : : }  710 : 73 : divisionLength = L1;  711 : 73 : patternSlice.setSizeY( divisionLength );  712 : : //Process the L1 patterns  713 [ + - ][ + + ]: 215 : for (i=0; static\_cast<uint32\_t>(i)<L1Count-1; i++)  714 : : {  715 : 142 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(patternSlice, prevLy, patternSlice.getSizeY()) );  716 : 142 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  717 : 142 : prepareNextHorizontalSlice(patternSlice, divisionLength);  718 : 142 : prevLy = patternSlice.getSizeY();  719 : : }  720 : : //Process the L2 patterns  721 : 73 : divisionLength = L2;  722 : 73 : patternSlice.setSizeY( divisionLength );  723 [ + + ]: 73 : if( L2Count )  724 : : {  725 : 11 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(patternSlice,   726 : : prevLy, patternSlice.getSizeY()) );  727 : 11 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  728 : :   729 : 11 : prepareNextHorizontalSlice(patternSlice, divisionLength);  730 : 11 : prevLy = patternSlice.getSizeY();  731 : : }  732 : :   733 : : //Handle 2nd last pattern (no 4)  734 [ + + ]: 73 : if(firstDivisionLength > 0){  735 : : //Handle when shift != 0  736 : 11 : lastDivLength = L1 - firstDivisionLength;  737 : : }else{  738 : : //Handle when shift == 0  739 : 62 : lastDivLength = L1;  740 : : }  741 : 73 : patternSlice.setSizeY( lastDivLength );  742 : 73 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(patternSlice,   743 : : prevLy, patternSlice.getSizeY()) );  744 : 73 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  745 : :   746 : 73 : prevLy = patternSlice.getSizeY();  747 : : //Process the 2nd edge  748 : 73 : prepareNextHorizontalSlice(patternSlice, prevLy); //Fix 1  749 : 73 : patternSlice.setSizeY( sliceParam.edge );  750 : 73 : patternSlice.setSizeX( getLxForNextNonLNPlgHorizontalSlice(patternSlice,  751 : : prevLy, patternSlice.getSizeY()) );  752 : 786 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);   753 : : }  754 : : }  755 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  756 : : \* Function Name: prepareNextHorizontalSlice  757 : : \* Purpose: Prepare the next slice for processing of a given  758 : : \* horizontal pattern  759 : : \* Parameters:  760 : : \* pat: ParsedPattern [IN]  761 : : \* divisionLength: uint32\_t [IN]  762 : : \* Return Values:  763 : : \* None  764 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  765 : : inline void PatternProcessor::prepareNextHorizontalSlice (ParsedPattern& patternSlice,   766 : : uint32\_t divisionLength  767 : : )  768 : :   769 : 918 : {  770 : 918 : int64\_t divLength = divisionLength;  771 : 918 : patternSlice.setPositionX(patternSlice.getPositionX()   772 : : + (divLength \* mEdgeVector.getEX(patternSlice.getCode())) );  773 : 918 : patternSlice.setPositionY(patternSlice.getPositionY() + divisionLength);  774 : : }  775 : :   776 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  777 : : \* Function Name: sliceHorizontalLNPlgPattern  778 : : \* Purpose: Slice long & narrow horizontal parallelograms  779 : : \* Parameters:  780 : : \* pat: ParsedPattern [IN]  781 : : \* sliceParam: DIV\_TH [IN] [Division Threshold for slicing]  782 : : \* divideParam: DIV\_TH [IN] [Division Threshold for division]  783 : : \* Return Values:  784 : : \* None  785 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  786 : : inline void PatternProcessor::sliceHorizontalLNPlgPattern(const ParsedPattern& pat,   787 : : const DIV\_TH& sliceParam,  788 : : const DIV\_TH& divideParam,  789 : : uint32\_t inSFLNo  790 : : )  791 : :   792 : 53 : {  793 : : /\* PRECONDITION: pat.getSizeX() < MAX \*/  794 : : /\* The intent is to prevent generation of indivisible patterns. \*/  795 : 53 : uint32\_t L1, L2, N;  796 : 53 : L1 = pat.getSizeX();  797 : 53 : N = pat.getSizeY() / L1;  798 : 53 : L2 = pat.getSizeY() % L1;  799 : 53 : uint32\_t L1CntLower = N/2;  800 : 53 : uint32\_t L1CntUpper = L1CntLower + N%2;  801 : :   802 : 53 : ParsedPattern patternSlice = pat;  803 : 53 : patternSlice.setSizeY( L1 );  804 : 53 : uint32\_t i;  805 : :   806 [ + + ]: 113 : for (i=0; i<L1CntLower; i++)  [ + + + + ]  [ + + ]  807 : : {  808 : 60 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  809 : 60 : prepareNextHorizontalSlice(patternSlice, L1);  810 : : }  811 [ + + ][ + + ]: 53 : if (0 != L2)  812 : : {  813 : 3 : patternSlice.setSizeY( L2 );  814 : 3 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  815 : 3 : prepareNextHorizontalSlice(patternSlice, L2);  816 : 3 : patternSlice.setSizeY( L1 );  817 : : }  818 [ + - ][ + + ]: 155 : for (i=0; i<L1CntUpper; i++)  [ + + ][ + + ]  [ # # ][ # # ]  819 : : {  820 : 102 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  821 : 102 : prepareNextHorizontalSlice(patternSlice, L1);  822 : : }   823 : : }  824 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  825 : : \* Function Name: sliceVerticalLNPlgPattern  826 : : \* Purpose: Slice long & narrow vertical parallelograms  827 : : \* Parameters:  828 : : \* pat: ParsedPattern [IN]  829 : : \* sliceParam: DIV\_TH [IN] [Division Threshold for slicing]  830 : : \* divideParam: DIV\_TH [IN] [Division Threshold for division]  831 : : \* Return Values:  832 : : \* None  833 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  834 : :   835 : : /\* Perform slice processing for 0x0d and 0x0e patterns   836 : : \*/  837 : : inline void PatternProcessor::sliceVerticalLNPlgPattern(const ParsedPattern& pat,   838 : : const DIV\_TH& sliceParam,  839 : : const DIV\_TH& divideParam,  840 : : uint32\_t inSFLNo  841 : : )  842 : :   843 : 42 : {  844 : : /\* PRECONDITION: pat.getSizeY() < MAX \*/  845 : : /\* The intent is to prevent generation of indivisible patterns. \*/  846 : 42 : uint32\_t L1, L2, N;  847 : 42 : L1 = pat.getSizeY();  848 : 42 : N = pat.getSizeX() / L1;  849 : 42 : L2 = pat.getSizeX() % L1;  850 : 42 : uint32\_t L1CntLower = N/2;  851 : 42 : uint32\_t L1CntUpper = L1CntLower + N%2;  852 : :   853 : 42 : uint32\_t i;  854 : 42 : ParsedPattern patternSlice = pat;  855 : 42 : patternSlice.setSizeX( L1 );  856 [ + + ]: 58 : for (i=0; i<L1CntLower; i++)  [ - + + + ]  [ - + ]  857 : : {  858 : 16 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  859 : 16 : prepareNextVerticalSlice(patternSlice, L1);  860 : : }  861 [ + + ][ + + ]: 42 : if (0 != L2)  862 : : {  863 : 20 : patternSlice.setSizeX( L2 );  864 : 20 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  865 : 20 : prepareNextVerticalSlice(patternSlice, L2);  866 : 20 : patternSlice.setSizeX( L1 );  867 : : }  868 [ + + ][ + + ]: 92 : for (i=0; i<L1CntUpper; i++)  [ + + ][ + + ]  869 : : {  870 : 50 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  871 : 50 : prepareNextVerticalSlice(patternSlice, L1);  872 : : }  873 : : }  874 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  875 : : \* Function Name: getLxForFirstNonLNPlgHorizontalSlice  876 : : \* Purpose: Get the X-dimension of the first slice for processing   877 : : \* of a given horizontal pattern  878 : : \* Parameters:  879 : : \* pat: ParsedPattern [IN]  880 : : \* divisionLength: uint32\_t [IN]  881 : : \* Return Values:  882 : : \* Lx of the first slice  883 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  884 : :   885 : : inline uint32\_t PatternProcessor::getLxForFirstNonLNPlgHorizontalSlice  886 : : (const ParsedPattern& pat,   887 : : uint32\_t divisionLength)  888 : :   889 : 1110 : {  890 : 786 : uint32\_t base = 0;  891 : 786 : uint32\_t firstLx = 0;  892 : :   893 : : //Note that the mix of uint and int in the   894 : : //foll. is okay since 0 <= firstLx <= base.  895 : 786 : switch (pat.getCode())  896 : : {  897 : : case 0x07:   898 : 31 : base = pat.getSizeX() + (pat.getSizeY()<<1);  899 : 31 : firstLx = base + divisionLength \*  900 : : (-mEdgeVector.getEX(pat.getCode()) +  901 : : mEdgeVector.getKX(pat.getCode()));  902 : 31 : break;  903 : : case 0x09:  904 : : case 0x0a:  905 : 164 : base = pat.getSizeX() + pat.getSizeY();  906 : 164 : firstLx = base + divisionLength \*  907 : : (-mEdgeVector.getEX(pat.getCode()) +  908 : : mEdgeVector.getKX(pat.getCode()));   909 : 164 : break;  910 : : default:  911 : : //No need to do the calculations as.getSizeX() of the first  912 : : //slice is same as that of the parent pattern.  913 : 591 : firstLx = pat.getSizeX();  914 : 786 : break;  915 : 786 : }  916 : : return firstLx;  917 : : }  918 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  919 : : \* Function Name: getLyForFirstNonLNPlgVerticalSlice  920 : : \* Purpose: Get the X-dimension of the first slice for processing  921 : : \* of a given vertical pattern  922 : : \* Parameters:  923 : : \* pat: ParsedPattern [IN]  924 : : \* divisionLength: uint32\_t [IN]  925 : : \* Return Values:  926 : : \* Ly of the first slice  927 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  928 : :   929 : : //Check the following  930 : : inline uint32\_t PatternProcessor::getLyForFirstNonLNPlgVerticalSlice  931 : : (const ParsedPattern& pat,  932 : : uint32\_t divisionLength)  933 : :   934 : 4261 : {  935 : 2231 : uint32\_t base = 0;  936 : 2231 : uint32\_t firstLy = 0;  937 : :   938 : 2231 : switch (pat.getCode())  939 : : {  940 : : case 0x06:  941 : 40 : base = pat.getSizeY() + (pat.getSizeX()<<1);  942 : 40 : firstLy = base + divisionLength \*  943 : : (-mEdgeVector.getEY(pat.getCode()) +  944 : : mEdgeVector.getKY(pat.getCode()));  945 : 40 : break;  946 : : case 0x02:  947 : : case 0x03:  948 : 1408 : base = pat.getSizeX() + pat.getSizeY();  949 : 1408 : firstLy = base + divisionLength \*  950 : : (-mEdgeVector.getEY(pat.getCode()) +  951 : : mEdgeVector.getKY(pat.getCode()));  952 : 1408 : break;  953 : : default:  954 : : //includes {0x01, 0x04, 0x05}  955 : 783 : firstLy = pat.getSizeY();  956 : 2231 : break;  957 : 2231 : }  958 : : return firstLy;  959 : : }  960 : :   961 : :   962 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  963 : : \* Function Name: getLxForNextNonLNPlgHorizontalSlice  964 : : \* Purpose: Get the X-dimension of the next slice for processing  965 : : \* of a given horizontal pattern  966 : : \* Parameters:  967 : : \* pat: ParsedPattern [IN]  968 : : \* prevLy: uint32\_t [IN] [Ly of the previous slice]  969 : : \* nextLy: uint32\_t [IN] [Ly of this slice]  970 : : \* Return Values:  971 : : \* Lx of the next slice  972 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  973 : :   974 : : inline uint32\_t PatternProcessor::getLxForNextNonLNPlgHorizontalSlice  975 : : (const ParsedPattern& pat,   976 : : uint32\_t prevLy,  977 : : uint32\_t thisLy)  978 : 381 : {  979 : : //Note that if the base length is available, it is always possible to calculate  980 : : //the next base length.  981 : : //Now assume there is a function getLx(pat, baselength);  982 : : //getLx(.) = baselength if pat.getCode() is in {0x08, 0x0b, 0x0c}  983 : : // = (baselength + f(pat.getSizeY())) if pat.getCode() is in {0x07, 0x09, 0x0a}  984 : :   985 : 753 : uint32\_t nextLx = pat.getSizeX();  986 : :   987 : 753 : switch (pat.getCode())  988 : : {  989 : : case 0x00:  990 : : case 0x11:  991 : : //nextLx = pat.getSizeX();  992 : 70 : break;  993 : : //Patterns with Lx as TOP  994 : : case 0x07:  995 : : case 0x09:  996 : : case 0x0a:  997 : 70 : nextLx += thisLy\*(-mEdgeVector.getEX(pat.getCode()) +  998 : : mEdgeVector.getKX(pat.getCode()));  999 : 70 : break;  1000 : : case 0x08:  1001 : : case 0x0b:  1002 : : case 0x0c:  1003 : 68 : nextLx += prevLy\*(-mEdgeVector.getEX(pat.getCode()) +  1004 : : mEdgeVector.getKX(pat.getCode()));  1005 : 753 : break;  1006 : 753 : }  1007 : : return nextLx;  1008 : :   1009 : : }  1010 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1011 : : \* Function Name: getLyForNextNonLNPlgVerticalSlice  1012 : : \* Purpose: Get the X-dimension of the next slice for processing  1013 : : \* of a given vertical pattern  1014 : : \* Parameters:  1015 : : \* pat: ParsedPattern [IN]  1016 : : \* prevLx: uint32\_t [IN] [Lx of the previous slice]  1017 : : \* nextLx: uint32\_t [IN] [Lx of this slice]  1018 : : \* Return Values:  1019 : : \* Ly of the next slice  1020 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1021 : :   1022 : : inline uint32\_t PatternProcessor::getLyForNextNonLNPlgVerticalSlice  1023 : : (const ParsedPattern& pat,  1024 : : uint32\_t prevLx,  1025 : : uint32\_t thisLx)  1026 : 175 : {  1027 : 430 : uint32\_t nextLy = pat.getSizeY();  1028 : :   1029 : 430 : switch (pat.getCode())  1030 : : {  1031 : : case 0x02:  1032 : : case 0x03:  1033 : : case 0x06:  1034 : 206 : nextLy += thisLx\*(-mEdgeVector.getEY(pat.getCode()) +  1035 : : mEdgeVector.getKY(pat.getCode()));  1036 : 206 : break;  1037 : : case 0x01:  1038 : : case 0x04:  1039 : : case 0x05:  1040 : 197 : nextLy += prevLx\*(-mEdgeVector.getEY(pat.getCode()) +  1041 : : mEdgeVector.getKY(pat.getCode()));  1042 : 430 : break;  1043 : 430 : }  1044 : : return nextLy;  1045 : : }  1046 : : //Dividing horizontal patterns  1047 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1048 : : \* Function Name: divideHorizontalPatternSlice  1049 : : \* Purpose: Divide a given horizontal pattern slice  1050 : : \* Parameters:  1051 : : \* pat: ParsedPattern [IN]  1052 : : \* divisionParam: DIV\_TH [IN]   1053 : : \* Return Values:  1054 : : \* None  1055 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1056 : :   1057 : : inline void PatternProcessor::divideHorizontalPatternSlice   1058 : : (const ParsedPattern &pat,  1059 : : const DIV\_TH& divTH,  1060 : : uint32\_t inSFLNo  1061 : : )  1062 : :   1063 : 1704 : {  1064 : 1704 : ParsedPattern patternUnderProcess = pat;  1065 : 1704 : switch (patternUnderProcess.getCode())  1066 : : {  1067 : : case 0x00:  1068 : : case 0x11:  1069 : : {  1070 : 1049 : createHorizontalRectangle (patternUnderProcess,   1071 : : divTH, inSFLNo);  1072 : 1033 : break;  1073 : : }  1074 : : case 0x07:  1075 : : case 0x08:  1076 : : case 0x0f:  1077 : : case 0x10:  1078 : : {  1079 : 288 : createHorizontalLeftTriangle (patternUnderProcess, inSFLNo);  1080 : 288 : createHorizontalRectangle (patternUnderProcess,  1081 : : divTH, inSFLNo);  1082 : 288 : createHorizontalRightTriangle (patternUnderProcess, inSFLNo);  1083 : 288 : break;  1084 : : }  1085 : : case 0x09:  1086 : : case 0x0c:  1087 : : {  1088 : 220 : createHorizontalLeftTriangle (patternUnderProcess, inSFLNo);  1089 : 220 : createHorizontalRectangle (patternUnderProcess,  1090 : : divTH, inSFLNo);  1091 : 220 : break;  1092 : : }  1093 : : case 0x0a:  1094 : : case 0x0b:  1095 : : {  1096 : 147 : createHorizontalRectangle (patternUnderProcess,  1097 : : divTH, inSFLNo);  1098 : 147 : createHorizontalRightTriangle (patternUnderProcess, inSFLNo);  1099 : 147 : break;  1100 : : }  1101 : : default:  1102 : 16 : break;  1103 : : }  1104 : : }  1105 : : //Division: Vertical Patterns   1106 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1107 : : \* Function Name: divideVerticalPatternSlice  1108 : : \* Purpose: Divide a given vertical pattern slice  1109 : : \* Parameters:  1110 : : \* pat: ParsedPattern [IN]  1111 : : \* divisionParam: DIV\_TH [IN]  1112 : : \* Return Values:  1113 : : \* None  1114 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1115 : :   1116 : : inline void PatternProcessor::divideVerticalPatternSlice  1117 : : (const ParsedPattern &pat,  1118 : : const DIV\_TH& divTH,  1119 : : uint32\_t inSFLNo  1120 : : )  1121 : :   1122 : 2747 : {  1123 : 2747 : ParsedPattern patternUnderProcess = pat;  1124 : 2747 : switch (patternUnderProcess.getCode())  1125 : : {  1126 : : case 0x01:  1127 : : case 0x02:  1128 : : {  1129 : 1119 : createVerticalRectangle (patternUnderProcess,   1130 : : divTH, inSFLNo);  1131 : 1119 : createVerticalTopTriangle (patternUnderProcess, inSFLNo);  1132 : 1119 : break;  1133 : : }  1134 : : case 0x03:  1135 : : case 0x04:  1136 : : {  1137 : 1353 : createVerticalBottomTriangle (patternUnderProcess, inSFLNo);  1138 : 1353 : createVerticalRectangle (patternUnderProcess,  1139 : : divTH, inSFLNo);  1140 : 1353 : break;  1141 : : }  1142 : : case 0x05:  1143 : : case 0x06:  1144 : : case 0x0d:  1145 : : case 0x0e:  1146 : : {  1147 : 275 : createVerticalBottomTriangle (patternUnderProcess, inSFLNo);  1148 : 275 : createVerticalRectangle (patternUnderProcess,   1149 : : divTH, inSFLNo);   1150 : 275 : createVerticalTopTriangle (patternUnderProcess, inSFLNo);  1151 : 275 : break;  1152 : : }  1153 : : default:  1154 : 0 : break;  1155 : : }   1156 : :   1157 : : }  1158 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1159 : : \* Function Name: createHorizontalRectangle  1160 : : \* Purpose: Extract a horizontal rectangle from a horizontal  1161 : : \* slice  1162 : : \* Parameters:  1163 : : \* pattern: ParsedPattern [IN]  1164 : : \* divisionParam: DIV\_TH [IN]  1165 : : \* Return Values:  1166 : : \* None  1167 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1168 : :   1169 : : /\* This function creates horizontal patterns from a parsed pattern \*/  1170 : : inline void PatternProcessor::createHorizontalRectangle  1171 : : (ParsedPattern& pat,   1172 : : const DIV\_TH& divTH,  1173 : : uint32\_t inSFLNo  1174 : : )  1175 : :   1176 : 1704 : {  1177 : : //Treat parallelograms and non-parallelograms differently  1178 : 1704 : ShotShiftCalculator shotShift;  1179 : 1704 : ParsedPattern patternUnderProcess = pat;  1180 : 1704 : patternUnderProcess.setCode(0x01);  1181 : 1704 : uint32\_t divCase = 0;  1182 : 641 : if (pat.getSizeX() <= divTH.max) { divCase = 1; }  1183 : 639 : else if (pat.getSizeX() <= static\_cast<uint32\_t>(2\*divTH.max) ) { divCase = 2; }  1184 : 174 : else if (pat.getSizeX() <= static\_cast<uint32\_t>(3\*divTH.max) ) { divCase = 3; }  1185 : 250 : else { divCase = 4; }  1186 : 1704 : uint32\_t i;  1187 : 1704 : uint32\_t divisionLength, shiftLength;   1188 : 1704 : ParsedPattern patternDivide = pat;  1189 : 1704 : SDDB \*pSDDB = SDDB::instance();  1190 [ + + ]: 1704 : if (divCase < 4)  1191 : : {  1192 : 1454 : shiftLength = shotShift.calculateShiftValue(  1193 : : mMaxShotShiftCount, pat.getSizeX(), divTH.max, inSFLNo);  1194 : 1454 : divisionLength = pat.getSizeX() / divCase;  1195 : 1454 : patternDivide.setCode(0x01);  1196 : 1454 : patternDivide.setSizeX( divisionLength );  1197 [ + + ][ + + ]: 1454 : switch (divCase)  [ - + ][ - + ]  1198 : : {  1199 : : case 1:  1200 : : {  1201 : 641 : patternDivide.createSimplePattern(spat[0], pos[0]);  1202 : 641 : try  1203 : : {  1204 : 641 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1205 : : }  1206 [ # # ]: 0 : catch(DPL::Error \*pChild)  1207 : : {  1208 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1209 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1210 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1211 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1212 : : "SDDB::SimplePatternGenerated"));  1213 : : }  1214 : :   1215 : 641 : try   1216 : : {  1217 : 641 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1218 : : }  1219 [ # # ]: 0 : catch(DPL::Error \*pChild)  1220 : : {  1221 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1222 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1223 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1224 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1225 : : "SDDB::SimplePatternShifted"));  1226 : : }  1227 : 641 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, patternDivide.getAI());  1228 : 641 : break;   1229 : : }  1230 : : case 2:  1231 : : {  1232 : 639 : if( spat.capacity() < 2 )  1233 : : {  1234 : 0 : spat.reserve( 2 );  1235 : 0 : spatOut.reserve( 3 );  1236 : 0 : pos.reserve( 2 );  1237 : 0 : posOut.reserve( 3 );  1238 : : }  1239 : 639 : patternDivide.createSimplePattern(spat[0], pos[0]);  1240 : 639 : try  1241 : : {  1242 : 639 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1243 : : }  1244 [ # # ]: 0 : catch(DPL::Error \*pChild)  1245 : : {  1246 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1247 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1248 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1249 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1250 : : "SDDB::SimplePatternGenerated"));  1251 : : }  1252 : :   1253 : 639 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1254 : 639 : patternDivide.setSizeX( pat.getSizeX() - divisionLength );  1255 : 639 : patternDivide.createSimplePattern(spat[1], pos[1]);  1256 : 639 : try  1257 : : {  1258 : 639 : pSDDB->SimplePatternGenerated(spat[1],pos[1]);  1259 : : }  1260 [ # # ]: 0 : catch(DPL::Error \*pChild)  1261 : : {  1262 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1263 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1264 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1265 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1266 : : "SDDB::SimplePatternGenerated"));  1267 : : }  1268 : :   1269 : 639 : shotShift.adjustShotShiftAfterHorDivision(spat, pos, spatOut, posOut, 2, shiftLength);  1270 : 639 : if(shotShift.getShiftFlag()){  1271 [ + + ]: 24 : for(uint32\_t i=0; i<3; i++){   1272 : 18 : try   1273 : : {  1274 : 18 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  1275 : : }  1276 [ # # ]: 0 : catch(DPL::Error \*pChild)  1277 : : {  1278 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1279 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1280 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1281 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1282 : : "SDDB::SimplePatternShifted"));  1283 : :   1284 : : }  1285 : :   1286 : 18 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  1287 : : }  1288 : : }else{  1289 [ + + ]: 1871 : for(uint32\_t i=0; i<2; i++){  1290 : 1252 : try  1291 : : {   1292 : 1252 : pSDDB->SimplePatternShifted(spat[i],pos[i]);  1293 : : }  1294 [ # # ]: 0 : catch(DPL::Error \*pChild)  1295 : : {  1296 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1297 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1298 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1299 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1300 : : "SDDB::SimplePatternShifted"));  1301 : : }  1302 : :   1303 : 1252 : mShotDistributor(boost::ref(spat[i]), boost::ref(pos[i]), inSFLNo, mThdID, patternDivide.getAI());  1304 : : }  1305 : : }  1306 : 619 : break;  1307 : : }  1308 : : case 3:  1309 : : {  1310 : 174 : if( spat.capacity() < 3 )  1311 : : {  1312 : 0 : spat.reserve( 3 );  1313 : 0 : spatOut.reserve( 4 );  1314 : 0 : pos.reserve( 3 );  1315 : 0 : posOut.reserve( 4 );  1316 : : }  1317 : 174 : patternDivide.createSimplePattern(spat[0], pos[0]);  1318 : 174 : try  1319 : : {  1320 : 174 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1321 : : }  1322 [ # # ]: 0 : catch(DPL::Error \*pChild)  1323 : : {  1324 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1325 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1326 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1327 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1328 : : "SDDB::SimplePatternGenerated"));  1329 : : }  1330 : :   1331 : 174 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1332 : 174 : patternDivide.setSizeX( divisionLength );  1333 : 174 : patternDivide.createSimplePattern(spat[1], pos[1]);  1334 : 174 : try  1335 : : {  1336 : 174 : pSDDB->SimplePatternGenerated(spat[1],pos[1]);  1337 : : }  1338 [ # # ]: 0 : catch(DPL::Error \*pChild)  1339 : : {  1340 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1341 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1342 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1343 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1344 : : "SDDB::SimplePatternGenerated"));  1345 : : }  1346 : :   1347 : 174 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1348 : 174 : patternDivide.setSizeX( pat.getSizeX() - 2\*divisionLength );  1349 : 174 : patternDivide.createSimplePattern(spat[2], pos[2]);  1350 : 174 : try  1351 : : {  1352 : 174 : pSDDB->SimplePatternGenerated(spat[2],pos[2]);  1353 : : }  1354 [ # # ]: 0 : catch(DPL::Error \*pChild)  1355 : : {  1356 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1357 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1358 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1359 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1360 : : "SDDB::SimplePatternGenerated"));  1361 : : }  1362 : :   1363 : 174 : shotShift.adjustShotShiftAfterHorDivision(spat, pos, spatOut, posOut, 3, shiftLength);  1364 : 174 : if(shotShift.getShiftFlag()){  1365 [ + + ]: 105 : for(uint32\_t i=0; i<4; i++){   1366 : 84 : try  1367 : : {  1368 : 84 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  1369 : : }  1370 [ # # ]: 0 : catch(DPL::Error \*pChild)  1371 : : {  1372 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1373 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1374 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1375 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1376 : : "SDDB::SimplePatternShifted"));  1377 : : }  1378 : 84 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  1379 : : }  1380 : : }else{  1381 [ + + ]: 612 : for(uint32\_t i=0; i<3; i++){  1382 : 459 : try  1383 : : {   1384 : 459 : pSDDB->SimplePatternShifted(spat[i],pos[i]);  1385 : : }  1386 [ # # ]: 0 : catch(DPL::Error \*pChild)  1387 : : {  1388 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1389 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1390 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1391 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1392 : : "SDDB::SimplePatternShifted"));  1393 : : }  1394 : 459 : mShotDistributor(boost::ref(spat[i]), boost::ref(pos[i]), inSFLNo, mThdID, patternDivide.getAI());  1395 : : }  1396 : : }  1397 : : }  1398 : 153 : break;  1399 : : }  1400 : : }  1401 : : else  1402 : : {  1403 : 250 : uint32\_t midPatternLengthX, midPatternCount, L1, L2,  1404 : 250 : L1Count, L2Count;  1405 : 250 : midPatternLengthX = pat.getSizeX() - (divTH.edge<<1);  1406 : 250 : midPatternCount = (midPatternLengthX - 1)/divTH.max;  1407 : 250 : L1 = midPatternLengthX/(midPatternCount + 1);  1408 : 250 : L1 += ((midPatternCount + 1)\*L1 < midPatternLengthX)?1:0;  1409 : 250 : L2 = midPatternLengthX - (midPatternCount \* L1);  1410 : 250 : L1Count = midPatternLengthX / L1;   1411 : 250 : L2Count = (midPatternLengthX - L1Count \* L1 > 0)?1:0;  1412 : :   1413 : 250 : shiftLength = calculateShiftValueForSlice(  1414 : : inSFLNo, mMaxShotShiftCount, L1);  1415 : :   1416 : 250 : size\_t shotCount = 1 + L1Count + L2Count + 2;  1417 : :   1418 : 250 : if( spatOut.capacity() < shotCount )  1419 : : {  1420 : 49 : spat.reserve( shotCount );  1421 : 49 : spatOut.reserve( shotCount );  1422 : 49 : pos.reserve( shotCount );  1423 : 49 : posOut.reserve( shotCount );  1424 : : }  1425 : :   1426 : 250 : uint32\_t noOfPatterns = 0;  1427 : :   1428 : : //Process the 1st edge  1429 : 250 : divisionLength = divTH.edge;  1430 : 250 : patternDivide.setCode(0x01);  1431 : 250 : patternDivide.setSizeX( divTH.edge );  1432 : 250 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1433 : 250 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1434 : 250 : try  1435 : : {  1436 : 250 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1437 : : }  1438 [ # # ]: 0 : catch(DPL::Error \*pChild)  1439 : : {  1440 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1441 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1442 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1443 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1444 : : "SDDB::SimplePatternGenerated"));  1445 : : }  1446 : 250 : noOfPatterns++;  1447 : : //Handle shot shift  1448 [ + + ]: 250 : if(shiftLength > 0){  1449 : 5 : patternDivide.setSizeX( shiftLength );   1450 : 5 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1451 : 5 : try  1452 : : {  1453 : 5 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1454 : : }  1455 [ # # ]: 0 : catch(DPL::Error \*pChild)  1456 : : {  1457 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1458 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1459 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1460 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1461 : : "SDDB::SimplePatternGenerated"));  1462 : : }  1463 : 5 : noOfPatterns++;  1464 : 5 : prepareNextHorizontalRectangle(patternDivide, shiftLength);  1465 : : } else{ // Shift == 0  1466 : : }  1467 : 250 : L1Count--;  1468 : : //Process the L1 patterns  1469 [ + - + + ]: 1229 : for (i=0; i<L1Count; i++)  1470 : : {  1471 : 979 : patternDivide.setSizeX( L1 );   1472 : 979 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1473 : 979 : try  1474 : : {  1475 : 979 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1476 : : }  1477 [ # # ]: 0 : catch(DPL::Error \*pChild)  1478 : : {  1479 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1480 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1481 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1482 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1483 : : "SDDB::SimplePatternGenerated"));  1484 : : }  1485 : 979 : divisionLength = L1;  1486 : 979 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1487 : 979 : noOfPatterns++;  1488 : : }  1489 : : //Process the L2 patterns  1490 [ + + ]: 250 : if(L2Count)  1491 : : {  1492 : 3 : divisionLength = L2;  1493 : 3 : patternDivide.setSizeX(divisionLength);  1494 : 3 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1495 : 3 : try  1496 : : {  1497 : 3 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1498 : : }  1499 [ # # ]: 0 : catch(DPL::Error \*pChild)  1500 : : {  1501 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1502 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1503 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1504 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1505 : : "SDDB::SimplePatternGenerated"));  1506 : : }  1507 : 3 : prepareNextHorizontalRectangle(patternDivide, divisionLength);  1508 : 3 : noOfPatterns++;  1509 : : }  1510 : : // patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1511 : : // try  1512 : : // {  1513 : : // pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1514 : : // }  1515 : : // catch(DPL::Error \*pChild)  1516 : : // {  1517 : : // ErrorHandler \*pErr = ErrorHandler::instance();  1518 : : // pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1519 : : // \_\_LINE\_\_, ErrorHandler::eErrFunction,  1520 : : // pErr->getErrorMsg(ErrorHandler::eErrFunction,  1521 : : // "SDDB::SimplePatternGenerated"));  1522 : : // }  1523 : : // noOfPatterns++;  1524 : :   1525 : : //Handle 2nd last pattern  1526 [ + + ]: 250 : if(shiftLength > 0){//Handle when shift != 0  1527 : 5 : patternDivide.setSizeX( L1 - shiftLength );  1528 : : }else{//Handle when shift == 0  1529 : 245 : patternDivide.setSizeX( L1 );  1530 : : }  1531 : :   1532 : 250 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  1533 : 250 : try  1534 : : {  1535 : 250 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1536 : : }  1537 [ # # ]: 0 : catch(DPL::Error \*pChild)  1538 : : {  1539 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1540 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1541 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1542 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1543 : : "SDDB::SimplePatternGenerated"));  1544 : : }  1545 : 250 : noOfPatterns++;  1546 : :   1547 : : //Process the 2nd edge  1548 [ + + ]: 250 : if(shiftLength > 0){//Handle when shift != 0  1549 : 5 : prepareNextHorizontalRectangle(patternDivide, L1-shiftLength );  1550 : : }else{  1551 : 245 : prepareNextHorizontalRectangle(patternDivide, L1);  1552 : : }  1553 : 250 : patternDivide.setSizeX( divTH.edge );  1554 : 250 : patternDivide.createSimplePattern(  1555 : : spatOut[noOfPatterns], posOut[noOfPatterns]);  1556 : 250 : try  1557 : : {  1558 : 250 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  1559 : : }  1560 [ # # ]: 0 : catch(DPL::Error \*pChild)  1561 : : {  1562 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1563 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1564 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1565 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1566 : : "SDDB::SimplePatternGenerated"));  1567 : : }  1568 : 250 : noOfPatterns++;  1569 [ + - + + ]: 1979 : for(uint32\_t i=0; i<noOfPatterns; i++){  1570 : 1731 : try  1571 : : {   1572 : 1731 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  1573 : : }  1574 [ # # ]: 0 : catch(DPL::Error \*pChild)  1575 : : {  1576 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1577 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1578 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1579 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1580 : : "SDDB::SimplePatternShifted"));  1581 : : }  1582 : :   1583 : 1731 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  1584 : : }  1585 : : }  1586 : : //Now change the coordinate of the original pattern  1587 : 1688 : pat.setPositionX(patternDivide.getPositionX() + patternDivide.getSizeX());   1588 : : }  1589 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1590 : : \* Function Name: prepareNextHorizontalRectangle  1591 : : \* Purpose: Prepare the next horizontal rectangle from a horizontal  1592 : : \* slice  1593 : : \* Parameters:  1594 : : \* pattern: ParsedPattern [IN]  1595 : : \* divisionLength: DIV\_TH [IN]  1596 : : \* Return Values:  1597 : : \* None  1598 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1599 : : inline void   1600 : : PatternProcessor::prepareNextHorizontalRectangle  1601 : : (ParsedPattern& patternDivide,  1602 : : uint32\_t divisionLength)  1603 : :   1604 : 2474 : {  1605 : 2474 : patternDivide.setPositionX(patternDivide.getPositionX() + divisionLength);  1606 : : }  1607 : :   1608 : :   1609 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1610 : : \* Function Name: createLeftTriangle\_0x07  1611 : : \* Purpose: Prepare the left triangle for 0x07 pattern  1612 : : \* Parameters:  1613 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1614 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1615 : : \* size: uint32\_t [IN] [length of the triangle]  1616 : : \* Return Values:  1617 : : \* None  1618 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1619 : :   1620 : : inline void PatternProcessor::createLeftTriangle\_0x07(uint32\_t x0,   1621 : : uint32\_t y0,   1622 : : uint32\_t size,   1623 : : uint32\_t attribute,  1624 : : uint32\_t inSFLNo)  1625 : :   1626 : 319 : {  1627 : 319 : ParsedPattern pattern;  1628 [ + - ]: 319 : if (size != 0)  1629 : : {  1630 : 319 : pattern.setCode(0x02);  1631 : 319 : pattern.setPositionX(x0 + size);  1632 : 319 : pattern.setPositionY(y0 + size);  1633 : 319 : pattern.setSizeX( size );  1634 : 319 : pattern.setSizeY( size );  1635 : 319 : pattern.setAI(attribute);  1636 : 319 : pattern.createSimplePattern(spat[0], pos[0]);  1637 : 319 : SDDB \*pSDDB = SDDB::instance();  1638 : 319 : try  1639 : : {  1640 : 319 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1641 : : }  1642 [ # # ]: 0 : catch(DPL::Error \*pChild)  1643 : : {  1644 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1645 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1646 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1647 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1648 : : "SDDB::SimplePatternGenerated"));  1649 : : }  1650 : 319 : try  1651 : : {   1652 : 319 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1653 : : }  1654 [ # # ]: 0 : catch(DPL::Error \*pChild)  1655 : : {  1656 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1657 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1658 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1659 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1660 : : "SDDB::SimplePatternShifted"));  1661 : : }  1662 : :   1663 : 319 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1664 : : }  1665 : : }  1666 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1667 : : \* Function Name: createRightTriangle\_0x07  1668 : : \* Purpose: Prepare the right triangle for 0x07 pattern  1669 : : \* Parameters:  1670 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1671 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1672 : : \* size: uint32\_t [IN] [length of the triangle]  1673 : : \* Return Values:  1674 : : \* None  1675 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1676 : :   1677 : : inline void PatternProcessor::createRightTriangle\_0x07(uint32\_t x0,   1678 : : uint32\_t y0,   1679 : : uint32\_t size,   1680 : : uint32\_t attribute,  1681 : : uint32\_t inSFLNo)  1682 : :   1683 : 204 : {   1684 : 204 : ParsedPattern pattern;  1685 [ + - ]: 204 : if (size != 0)  1686 : : {  1687 : 204 : pattern.setCode(0x05);  1688 : 204 : pattern.setPositionX(x0);  1689 : 204 : pattern.setPositionY(y0 + size);  1690 : 204 : pattern.setSizeX(size );  1691 : 204 : pattern.setSizeY( size );  1692 : 204 : pattern.setAI(attribute);  1693 : 204 : pattern.createSimplePattern(spat[0], pos[0]);  1694 : 204 : SDDB \*pSDDB = SDDB::instance();  1695 : 204 : try  1696 : : {  1697 : 204 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1698 : : }  1699 [ # # ]: 0 : catch(DPL::Error \*pChild)  1700 : : {  1701 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1702 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1703 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1704 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1705 : : "SDDB::SimplePatternGenerated"));  1706 : : }  1707 : 204 : try  1708 : : {  1709 : 204 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1710 : : }  1711 [ # # ]: 0 : catch(DPL::Error \*pChild)  1712 : : {  1713 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1714 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1715 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1716 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1717 : : "SDDB::SimplePatternShifted"));  1718 : : }  1719 : :   1720 : :   1721 : 204 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1722 : : }  1723 : : }  1724 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1725 : : \* Function Name: createLeftTriangle\_0x08  1726 : : \* Purpose: Prepare the left triangle for 0x08 pattern  1727 : : \* Parameters:  1728 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1729 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1730 : : \* size: uint32\_t [IN] [length of the triangle]  1731 : : \* Return Values:  1732 : : \* None  1733 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1734 : :   1735 : : inline void PatternProcessor::createLeftTriangle\_0x08(uint32\_t x0,   1736 : : uint32\_t y0,   1737 : : uint32\_t size,   1738 : : uint32\_t attribute,  1739 : : uint32\_t inSFLNo)  1740 : :   1741 : 189 : {  1742 : 189 : ParsedPattern pattern;  1743 [ + - + - : 189 : if (size != 0)  + - # # ]  1744 : : {  1745 : 189 : pattern.setCode(0x03);  1746 : 189 : pattern.setPositionX(x0);  1747 : 189 : pattern.setPositionY(y0);  1748 : 189 : pattern.setSizeX( size );  1749 : 189 : pattern.setSizeY( size );  1750 : 189 : pattern.setAI(attribute);  1751 : 189 : pattern.createSimplePattern(spat[0], pos[0]);  1752 : 189 : SDDB \*pSDDB = SDDB::instance();  1753 : 189 : try  1754 : : {  1755 : 189 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1756 : : }  1757 [ # # ][ # # ]: 0 : catch(DPL::Error \*pChild)  [ # # ][ # # ]  1758 : : {  1759 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1760 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1761 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1762 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1763 : : "SDDB::SimplePatternGenerated"));  1764 : : }  1765 : 189 : try  1766 : : {  1767 : 189 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1768 : : }  1769 [ # # ][ # # ]: 0 : catch(DPL::Error \*pChild)  [ # # ][ # # ]  1770 : : {  1771 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1772 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1773 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1774 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1775 : : "SDDB::SimplePatternShifted"));  1776 : : }  1777 : :   1778 : :   1779 : 189 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1780 : : }  1781 : : }  1782 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1783 : : \* Function Name: createRightTriangle\_0x08  1784 : : \* Purpose: Prepare the left triangle for 0x08 pattern  1785 : : \* Parameters:  1786 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1787 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1788 : : \* size: uint32\_t [IN] [length of the triangle]  1789 : : \* Return Values:  1790 : : \* None  1791 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1792 : : inline void PatternProcessor::createRightTriangle\_0x08(uint32\_t x0,   1793 : : uint32\_t y0,   1794 : : uint32\_t size,   1795 : : uint32\_t attribute,  1796 : : uint32\_t inSFLNo)  1797 : :   1798 : 231 : {  1799 : 231 : ParsedPattern pattern;  1800 [ + - ]: 231 : if (size != 0)  1801 : : {  1802 : 231 : pattern.setCode(0x04);  1803 : 231 : pattern.setPositionX(x0);  1804 : 231 : pattern.setPositionY(y0);  1805 : 231 : pattern.setSizeX(size );  1806 : 231 : pattern.setSizeY( size );  1807 : 231 : pattern.setAI(attribute);  1808 : 231 : pattern.createSimplePattern(spat[0], pos[0]);  1809 : 231 : SDDB \*pSDDB = SDDB::instance();  1810 : 231 : try  1811 : : {  1812 : 231 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1813 : : }  1814 [ # # ]: 0 : catch(DPL::Error \*pChild)  1815 : : {  1816 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1817 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1818 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1819 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1820 : : "SDDB::SimplePatternGenerated"));  1821 : : }  1822 : 231 : try  1823 : : {  1824 : 231 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1825 : : }  1826 [ # # ]: 0 : catch(DPL::Error \*pChild)  1827 : : {  1828 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1829 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1830 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1831 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1832 : : "SDDB::SimplePatternShifted"));  1833 : : }  1834 : 231 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1835 : : }  1836 : : }  1837 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1838 : : \* Function Name: createTopTriangle\_0x05  1839 : : \* Purpose: Prepare the top triangle for 0x05 pattern  1840 : : \* Parameters:  1841 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1842 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1843 : : \* size: uint32\_t [IN] [length of the triangle]  1844 : : \* Return Values:  1845 : : \* None  1846 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1847 : : inline void PatternProcessor::createTopTriangle\_0x05(uint32\_t x0,   1848 : : uint32\_t y0,   1849 : : uint32\_t size,   1850 : : uint32\_t attribute,  1851 : : uint32\_t inSFLNo)  1852 : :   1853 : 819 : {  1854 : 819 : ParsedPattern pattern;  1855 [ + - ]: 819 : if (size != 0)  1856 : : {  1857 : 819 : pattern.setCode(0x02);  1858 : 819 : pattern.setPositionX(x0 + size);  1859 : 819 : pattern.setPositionY(y0 + size);  1860 : 819 : pattern.setSizeX( size );  1861 : 819 : pattern.setSizeY( size );  1862 : 819 : pattern.setAI(attribute);  1863 : 819 : pattern.createSimplePattern(spat[0], pos[0]);  1864 : 819 : SDDB \*pSDDB = SDDB::instance();  1865 : 819 : try  1866 : : {  1867 : 819 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1868 : : }  1869 [ # # ]: 0 : catch(DPL::Error \*pChild)  1870 : : {  1871 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1872 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1873 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1874 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1875 : : "SDDB::SimplePatternGenerated"));  1876 : : }  1877 : 819 : try  1878 : : {   1879 : 819 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1880 : : }   1881 [ # # ]: 0 : catch(DPL::Error \*pChild)  1882 : : {  1883 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1884 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1885 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1886 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1887 : : "SDDB::SimplePatternShifted"));  1888 : : }  1889 : 819 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1890 : : }  1891 : : else  1892 : : {  1893 : : }  1894 : : }  1895 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1896 : : \* Function Name: createBottomTriangle\_0x05   1897 : : \* Purpose: Prepare the bottom triangle for 0x05 pattern  1898 : : \* Parameters:  1899 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]  1900 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1901 : : \* size: uint32\_t [IN] [length of the triangle]  1902 : : \* Return Values:  1903 : : \* None  1904 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1905 : :   1906 : : inline void PatternProcessor::createBottomTriangle\_0x05(uint32\_t x0,   1907 : : uint32\_t y0,   1908 : : uint32\_t size,   1909 : : uint32\_t attribute,  1910 : : uint32\_t inSFLNo)  1911 : :   1912 : 332 : {  1913 : 332 : ParsedPattern pattern;  1914 [ + - ]: 332 : if (size != 0)  1915 : : {  1916 : 332 : pattern.setCode(0x03);  1917 : 332 : pattern.setPositionX(x0 + size);  1918 : 332 : pattern.setPositionY(y0 - size);  1919 : 332 : pattern.setSizeX( size );  1920 : 332 : pattern.setSizeY( size );  1921 : 332 : pattern.setAI(attribute);  1922 : 332 : pattern.createSimplePattern(spat[0], pos[0]);  1923 : 332 : SDDB \*pSDDB = SDDB::instance();  1924 : 332 : try  1925 : : {  1926 : 332 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1927 : : }  1928 [ # # ]: 0 : catch(DPL::Error \*pChild)  1929 : : {  1930 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1931 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1932 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1933 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1934 : : "SDDB::SimplePatternGenerated"));  1935 : : }  1936 : 332 : try  1937 : : {  1938 : 332 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1939 : : }  1940 [ # # ]: 0 : catch(DPL::Error \*pChild)  1941 : : {  1942 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1943 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1944 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1945 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1946 : : "SDDB::SimplePatternShifted"));  1947 : : }  1948 : 332 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  1949 : : }  1950 : : else  1951 : : {  1952 : : }  1953 : : }  1954 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  1955 : : \* Function Name: createTopTriangle\_0x06  1956 : : \* Purpose: Prepare the top triangle for 0x06 pattern  1957 : : \* Parameters:  1958 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]   1959 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  1960 : : \* size: uint32\_t [IN] [length of the triangle]  1961 : : \* Return Values:  1962 : : \* None  1963 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  1964 : :   1965 : : inline void PatternProcessor::createTopTriangle\_0x06(uint32\_t x0,   1966 : : uint32\_t y0,   1967 : : uint32\_t size,   1968 : : uint32\_t attribute,  1969 : : uint32\_t inSFLNo)  1970 : :   1971 : 575 : {  1972 : 575 : ParsedPattern pattern;  1973 [ + - ]: 575 : if (size != 0)  1974 : : {  1975 : 575 : pattern.setCode(0x05);  1976 : 575 : pattern.setPositionX(x0);  1977 : 575 : pattern.setPositionY(y0 + size);  1978 : 575 : pattern.setSizeX(size);  1979 : 575 : pattern.setSizeY( size );  1980 : 575 : pattern.setAI(attribute);  1981 : 575 : pattern.createSimplePattern(spat[0], pos[0]);  1982 : 575 : SDDB \*pSDDB = SDDB::instance();  1983 : 575 : try  1984 : : {  1985 : 575 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  1986 : : }  1987 [ # # ]: 0 : catch(DPL::Error \*pChild)  1988 : : {  1989 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  1990 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  1991 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  1992 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  1993 : : "SDDB::SimplePatternGenerated"));  1994 : : }  1995 : 575 : try  1996 : : {   1997 : 575 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  1998 : : }  1999 [ # # ]: 0 : catch(DPL::Error \*pChild)  2000 : : {  2001 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2002 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2003 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2004 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2005 : : "SDDB::SimplePatternShifted"));  2006 : : }  2007 : 575 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  2008 : : }  2009 : : else  2010 : : {  2011 : : }  2012 : : }  2013 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2014 : : \* Function Name: createBottomTriangle\_0x06  2015 : : \* Purpose: Prepare the bottom triangle for 0x06 pattern  2016 : : \* Parameters:  2017 : : \* x0: uint32\_t [IN] [X-coordinate of the LL of the pattern]   2018 : : \* y0: uint32\_t [IN] [Y-coordinate of the LL of the pattern]  2019 : : \* size: uint32\_t [IN] [length of the triangle]  2020 : : \* Return Values:  2021 : : \* None  2022 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2023 : :   2024 : : inline void PatternProcessor::createBottomTriangle\_0x06(uint32\_t x0,   2025 : : uint32\_t y0,   2026 : : uint32\_t size,   2027 : : uint32\_t attribute,  2028 : : uint32\_t inSFLNo)  2029 : :   2030 : 1296 : {  2031 : 1296 : ParsedPattern pattern;  2032 [ + - # # : 1296 : if (size != 0)  + - + - ]  2033 : : {  2034 : 1296 : pattern.setCode(0x04);  2035 : 1296 : pattern.setPositionX(x0);  2036 : 1296 : pattern.setPositionY(y0);  2037 : 1296 : pattern.setSizeX( size );  2038 : 1296 : pattern.setSizeY( size );  2039 : 1296 : pattern.setAI(attribute);  2040 : 1296 : pattern.createSimplePattern(spat[0], pos[0]);  2041 : 1296 : SDDB \*pSDDB = SDDB::instance();  2042 : 1296 : try  2043 : : {  2044 : 1296 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  2045 : : }  2046 [ # # ][ # # ]: 0 : catch(DPL::Error \*pChild)  [ # # ][ # # ]  2047 : : {  2048 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2049 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2050 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2051 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2052 : : "SDDB::SimplePatternGenerated"));  2053 : : }  2054 : 1296 : try  2055 : : {  2056 : 1296 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  2057 : : }  2058 [ # # ][ # # ]: 0 : catch(DPL::Error \*pChild)  [ # # ][ # # ]  2059 : : {  2060 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2061 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2062 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2063 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2064 : : "SDDB::SimplePatternShifted"));  2065 : : }  2066 : 1296 : mShotDistributor(boost::ref(spat[0]), boost::ref(pos[0]), inSFLNo, mThdID, pattern.getAI());  2067 : : }  2068 : : else  2069 : : {  2070 : : }  2071 : : }  2072 : :   2073 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2074 : : \* Function Name: createHorizontalLeftTriangle  2075 : : \* Purpose: Extract the left triangle for a horizontal pattern  2076 : : \* Parameters:  2077 : : \* inPat: ParsedPattern [IN]  2078 : : \* Return Values:  2079 : : \* None  2080 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2081 : :   2082 : : inline void PatternProcessor::createHorizontalLeftTriangle(ParsedPattern& inPat, uint32\_t inSFLNo)  2083 : :   2084 : 508 : {  2085 : 508 : uint32\_t size = inPat.getSizeY();  2086 : 508 : int64\_t x = inPat.getPositionX();  2087 : 508 : int64\_t y = inPat.getPositionY();  2088 : 508 : uint32\_t attribute = inPat.getAI();  2089 : 508 : switch(inPat.getCode())  2090 : : {   2091 : : case 0x00:  2092 : : case 0x11:  2093 : : case 0x0a:  2094 : : case 0x0b:  2095 : 210 : break;  2096 : :   2097 : : case 0x07:   2098 : : case 0x09:  2099 : 210 : createLeftTriangle\_0x07(x, y, size, attribute, inSFLNo);  2100 : 210 : inPat.setPositionX(inPat.getPositionX() + size);  2101 : 210 : break;  2102 : : case 0x0f:  2103 : 109 : createLeftTriangle\_0x07(x, y, size, attribute, inSFLNo);  2104 : 109 : inPat.setPositionX(inPat.getPositionX() + size);  2105 : 109 : inPat.setSizeX(inPat.getSizeX() - size);  2106 : 109 : break;  2107 : : case 0x08:   2108 : : case 0x0c:  2109 : 83 : createLeftTriangle\_0x08(x, y, size, attribute, inSFLNo);  2110 : 83 : break;   2111 : : case 0x10:  2112 : 106 : createLeftTriangle\_0x08(x, y, size, attribute, inSFLNo);  2113 : 106 : inPat.setSizeX(inPat.getSizeX() - size);  2114 : 508 : break;   2115 : : }  2116 : : }  2117 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2118 : : \* Function Name: createHorizontalRightTriangle  2119 : : \* Purpose: Extract the right triangle for a horizontal pattern  2120 : : \* Parameters:  2121 : : \* inPat: ParsedPattern [IN]  2122 : : \* Return Values:  2123 : : \* None  2124 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2125 : : inline void PatternProcessor::createHorizontalRightTriangle(ParsedPattern& inPat, uint32\_t inSFLNo)  2126 : :   2127 : 435 : {  2128 : 435 : uint32\_t size;  2129 : 435 : int64\_t x, y;  2130 : 435 : x = inPat.getPositionX();  2131 : 435 : y = inPat.getPositionY();  2132 : 435 : size = inPat.getSizeY();  2133 : 435 : uint32\_t attribute = inPat.getAI();  2134 : 435 : switch(inPat.getCode())  2135 : : {  2136 : : case 0x00:  2137 : : case 0x09:  2138 : : case 0x0c:  2139 : : case 0x11:  2140 : 204 : break;  2141 : : case 0x07:   2142 : : case 0x0a:  2143 : : case 0x10:  2144 : 204 : createRightTriangle\_0x07(x, y, size, attribute, inSFLNo);  2145 : 204 : break;  2146 : : case 0x08:   2147 : : case 0x0b:  2148 : : case 0x0f:  2149 : 231 : createRightTriangle\_0x08(x, y, size, attribute, inSFLNo);  2150 : 435 : break;   2151 : : }  2152 : : }  2153 : : //Slicing: Vertical Patterns  2154 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2155 : : \* Function Name: sliceVerticalNonLNPlgPattern  2156 : : \* Purpose: Slice vertical patterns (except long & narrow  2157 : : \* parallelograms)  2158 : : \* Parameters:  2159 : : \* pat: ParsedPattern [IN]  2160 : : \* sliceParam: DIV\_TH [IN] [Division Threshold for slicing]  2161 : : \* divideParam: DIV\_TH [IN] [Division Threshold for division]  2162 : : \* Return Values:  2163 : : \* None  2164 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2165 : :   2166 : : inline void PatternProcessor::sliceVerticalNonLNPlgPattern (const ParsedPattern& pat,  2167 : : const DIV\_TH& sliceParam,  2168 : : const DIV\_TH& divideParam,  2169 : : uint32\_t inSFLNo  2170 : : )  2171 : :   2172 : 2231 : {  2173 : 2231 : ParsedPattern patternSlice = pat;  2174 : 2231 : ShotShiftCalculator shotShift;  2175 : 2231 : int32\_t divCase = 0;  2176 : 2231 : uint32\_t lastDivLength;  2177 : :   2178 : 2030 : if (pat.getSizeX() <= sliceParam.max) { divCase = 1; }  2179 : 67 : else if (pat.getSizeX() <= static\_cast<uint32\_t>(2\*sliceParam.max) ){ divCase = 2; }  2180 : 110 : else if (pat.getSizeX() <= static\_cast<uint32\_t>(3\*sliceParam.max) ){ divCase = 3; }  2181 : 24 : else { divCase = 4; }  2182 : 2231 : int32\_t i;  2183 : 2231 : uint32\_t divisionLength = 0, firstDivisionLength = 0;  2184 [ + + ]: 2231 : if (divCase < 4)  2185 : : {  2186 : :   2187 : 2207 : divisionLength = pat.getSizeX() / divCase;  2188 : 2207 : uint32\_t sizeX = divisionLength + (pat.getSizeX() % divCase);  2189 : 2207 : patternSlice = pat;  2190 : : // ------- --------- ---------------  2191 : : // | 1 | |1 | 2 | | 1 | 2 | 3 |  2192 : : // | | | | | | | | |  2193 : : // ------- --------- ---------------  2194 : : // ------- ---------- ----------------  2195 : : // | 0| 1| |0 |1 | 2| |0 | 1 | 2 | 3|  2196 : : // | | | | | | | | | | | |  2197 : : // ------- ---------- ----------------  2198 : : // divCase == 1 divCase == 2 divCase == 3  2199 : :   2200 : : //Variable Shift length for the last(no 1) slice  2201 [ + + ]: 2207 : if(divCase != 1){   2202 : 177 : firstDivisionLength = shotShift.calculateShiftValue(  2203 : : mMaxShotShiftCount, pat.getSizeX(), sliceParam.max, inSFLNo);  2204 : : }  2205 : : //firstDivisionLength = calculateShiftValueForSlice(  2206 : : // inSFLNo,mMaxShotShiftCount,sliceParam.max);  2207 [ + + ]: 177 : if(firstDivisionLength > 0){  2208 : : //Handle the last(no 1) slice  2209 : 2 : patternSlice.setSizeX(firstDivisionLength);  2210 : 2 : patternSlice.setSizeY(getLyForFirstNonLNPlgVerticalSlice  2211 : : ( pat, firstDivisionLength ));  2212 : : }else{  2213 : : //No shift or shift == o  2214 : 2205 : divCase--;  2215 : 2205 : patternSlice.setSizeX(divisionLength);  2216 : 2205 : patternSlice.setSizeY(getLyForFirstNonLNPlgVerticalSlice  2217 : : ( pat, divisionLength ));  2218 : : }  2219 [ + + ][ + + ]: 2496 : for (i=0; i<divCase; i++)  2220 : : {  2221 : : //Apply division algo for this slice  2222 : 289 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2223 [ + + ]: 289 : if ((i == divCase-1) && (firstDivisionLength > 0) ){  2224 : : //calculate the origin and dimensions of the slice(no 3)  2225 : 2 : prepareNextVerticalSlice(patternSlice, divisionLength - firstDivisionLength );  2226 : : //Adjust the position X  2227 : 2 : patternSlice.setPositionX(patternSlice.getPositionX()   2228 : : + firstDivisionLength);  2229 : 2 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2230 : : ( patternSlice, divisionLength,  2231 : : patternSlice.getSizeX() ) );  2232 : 2 : patternSlice.setSizeX( divisionLength-firstDivisionLength );  2233 : : }else{  2234 : 287 : prepareNextVerticalSlice(patternSlice, divisionLength);  2235 [ + + ]: 287 : if ((i == 0) && (firstDivisionLength > 0) ){  2236 : 2 : patternSlice.setPositionX(patternSlice.getPositionX()   2237 : : - divisionLength + firstDivisionLength);  2238 : 2 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2239 : : ( patternSlice, firstDivisionLength,  2240 : : patternSlice.getSizeX() ) );  2241 : : }  2242 : : else{  2243 [ + + ]: 285 : if(divCase - 1 == i) {//Adjust the size of the Last slice  2244 : 175 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2245 : : ( patternSlice, divisionLength,  2246 : : sizeX ) );  2247 : : }else {  2248 : 110 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2249 : : ( patternSlice, divisionLength,  2250 : : patternSlice.getSizeX() ) );  2251 : : }  2252 : : }  2253 [ + + ]: 287 : if(divCase - 1 == i) //Adjust the size of the Last slice  2254 : 175 : patternSlice.setSizeX(sizeX);  2255 : : else   2256 : 112 : patternSlice.setSizeX(divisionLength);  2257 : : }  2258 : : }  2259 : 2207 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2260 : : }  2261 : : else  2262 : : {  2263 : 24 : uint32\_t midPatternLengthX = pat.getSizeX() - (sliceParam.edge<<1); //replaced \*2 by <<1  2264 : 24 : uint32\_t midPatternCount = (midPatternLengthX - 1)/sliceParam.max;  2265 : 24 : uint32\_t L1 = midPatternLengthX/(midPatternCount + 1);  2266 : 24 : L1 += ((midPatternCount +1)\*L1 < midPatternLengthX)?1:0;  2267 : 24 : uint32\_t L2 = midPatternLengthX - (midPatternCount \* L1);  2268 : 24 : uint32\_t L1Count = midPatternLengthX / L1;   2269 : 24 : uint32\_t L2Count = (midPatternLengthX - L1Count \* L1 >0)?1:0;  2270 : 24 : uint32\_t prevLx;  2271 : : //Process the 1st edge  2272 : 24 : uint32\_t divisionLength = sliceParam.edge;  2273 : 24 : patternSlice.setSizeX(sliceParam.edge);  2274 : 24 : patternSlice.setSizeY(getLyForFirstNonLNPlgVerticalSlice  2275 : : ( pat, divisionLength ));  2276 : 24 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2277 : : // ---------------------------  2278 : : // |//| 1| 2| 3 | 4 | 5 |//|  2279 : : // |//| | | | | |//|  2280 : : // ---------------------------  2281 : :   2282 : : // ----------------------------  2283 : : // |//| 0| 1| 2| 3 | 4 | 5|//|  2284 : : // |//| | | | | | |//|  2285 : : // ----------------------------  2286 : :   2287 : 24 : prevLx = divisionLength;  2288 : : //Variable Shift length for the last(no 1) slice  2289 : 24 : firstDivisionLength = calculateShiftValueForSlice(  2290 : : inSFLNo,mMaxShotShiftCount,L1);  2291 : : // firstDivisionLength = shotShift.calculateShiftValue(  2292 : : // mMaxShotShiftCount, L1, sliceParam.max, inSFLNo);  2293 [ - + ]: 24 : if(firstDivisionLength > 0){  2294 : : //Handle the last(no 1) slice  2295 : 0 : prepareNextVerticalSlice(patternSlice, prevLx); //Fix 2  2296 : 0 : patternSlice.setSizeX( firstDivisionLength );  2297 : 0 : patternSlice.setSizeY( getLxForNextNonLNPlgHorizontalSlice(patternSlice, prevLx, firstDivisionLength) );  2298 : 0 : divideHorizontalPatternSlice(patternSlice, divideParam, inSFLNo);  2299 : 0 : prepareNextVerticalSlice(patternSlice, firstDivisionLength); //Fix 3  2300 : :   2301 : 0 : prevLx = firstDivisionLength;  2302 : : }else{  2303 : : //No shift (no 2) slice  2304 : 24 : prepareNextVerticalSlice(patternSlice, divisionLength);  2305 : 24 : if(firstDivisionLength > 0){  2306 : 24 : patternSlice.setPositionX(patternSlice.getPositionX()   2307 : : - divisionLength + firstDivisionLength);  2308 : : }  2309 : : }  2310 : 24 : divisionLength = L1;  2311 : 24 : patternSlice.setSizeX( divisionLength );  2312 : : //Process the L1 patterns  2313 [ + - ][ + + ]: 116 : for (i=0; static\_cast<uint32\_t>(i)<L1Count-1; i++)  2314 : : {  2315 : 92 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2316 : : ( patternSlice,   2317 : : prevLx,  2318 : : patternSlice.getSizeX() ) );  2319 : 92 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2320 : 92 : prepareNextVerticalSlice(patternSlice, divisionLength);  2321 : 92 : prevLx = patternSlice.getSizeX();  2322 : : }  2323 : : //Process the L2 patterns  2324 : 24 : divisionLength = L2;  2325 : 24 : patternSlice.setSizeX(divisionLength);  2326 [ + + ]: 24 : if(L2Count)  2327 : : {  2328 : 1 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2329 : : ( patternSlice,  2330 : : prevLx,  2331 : : patternSlice.getSizeX() ));  2332 : 1 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2333 : 1 : prepareNextVerticalSlice(patternSlice, divisionLength);  2334 : 1 : prevLx = patternSlice.getSizeX();  2335 : : }  2336 : : //Handle 2nd last pattern   2337 [ - + ]: 24 : if(firstDivisionLength > 0){  2338 : : //Handle when shift != 0  2339 : 0 : lastDivLength = L1-firstDivisionLength;  2340 : : }else{  2341 : : //Handle when shift == 0  2342 : 24 : lastDivLength = L1;  2343 : : }  2344 : 24 : patternSlice.setSizeX( lastDivLength );  2345 : 24 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2346 : : ( patternSlice,  2347 : : prevLx,  2348 : : patternSlice.getSizeX() ));  2349 : 24 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2350 : :   2351 : 24 : prevLx = patternSlice.getSizeX();  2352 : : //Process the 2nd edge  2353 : : //prepareNextVerticalSlice(patternSlice, sliceParam.edge);  2354 : 24 : prepareNextVerticalSlice(patternSlice, prevLx); //Fix 1  2355 : 24 : patternSlice.setSizeX(sliceParam.edge);  2356 : 24 : patternSlice.setSizeY(getLyForNextNonLNPlgVerticalSlice  2357 : : ( patternSlice,  2358 : : prevLx,  2359 : : patternSlice.getSizeX() ));  2360 : 2231 : divideVerticalPatternSlice(patternSlice, divideParam, inSFLNo);  2361 : : }  2362 : : }  2363 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2364 : : \* Function Name: createVerticalRectangle  2365 : : \* Purpose: Extract a vertical rectangle from a vertical  2366 : : \* slice  2367 : : \* Parameters:  2368 : : \* pattern: ParsedPattern [IN]  2369 : : \* divisionParam: DIV\_TH [IN]  2370 : : \* Return Values:  2371 : : \* None  2372 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2373 : :   2374 : : inline void PatternProcessor::createVerticalRectangle  2375 : : (ParsedPattern& pat,   2376 : : const DIV\_TH& divTH,  2377 : : uint32\_t inSFLNo  2378 : : )  2379 : :   2380 : 2747 : {  2381 : 2747 : ParsedPattern patternUnderProcess = pat;  2382 : 2747 : patternUnderProcess.setCode(0x01);  2383 : 2747 : ShotShiftCalculator shotShift;  2384 : 2747 : uint32\_t divCase = 0;  2385 : 2404 : if (pat.getSizeY() <= divTH.max) { divCase = 1; }  2386 : 173 : else if (pat.getSizeY() <= static\_cast<uint32\_t>(2\*divTH.max) ){ divCase = 2; }  2387 : 43 : else if (pat.getSizeY() <= static\_cast<uint32\_t>(3\*divTH.max) ){ divCase = 3; }  2388 : 127 : else { divCase = 4; }  2389 : 2747 : uint32\_t i;  2390 : 2747 : uint32\_t divisionLength, shiftLength;  2391 : 2747 : ParsedPattern patternDivide = pat;  2392 : 2747 : SDDB \*pSDDB = SDDB::instance();  2393 : :   2394 [ + + ]: 2747 : if (divCase < 4)  2395 : : {  2396 : 2620 : shiftLength = shotShift.calculateShiftValue(  2397 : : mMaxShotShiftCount, pat.getSizeY(), divTH.max, inSFLNo);  2398 : 2620 : divisionLength = pat.getSizeY() / divCase;  2399 : 2620 : patternDivide.setCode(0x01);  2400 : 2620 : patternDivide.setSizeY( divisionLength );  2401 [ + + ][ + + ]: 2620 : switch (divCase)  [ - + ][ - + ]  2402 : : {  2403 : : case 1:  2404 : : {  2405 : 2404 : patternDivide.createSimplePattern(spat[0],pos[0]);  2406 : 2404 : try  2407 : : {  2408 : 2404 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  2409 : : }  2410 [ # # ]: 0 : catch(DPL::Error \*pChild)  2411 : : {  2412 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2413 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2414 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2415 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2416 : : "SDDB::SimplePatternGenerated"));  2417 : : }  2418 : 2404 : try  2419 : : {  2420 : 2404 : pSDDB->SimplePatternShifted(spat[0],pos[0]);  2421 : : }  2422 [ # # ]: 0 : catch(DPL::Error \*pChild)  2423 : : {  2424 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2425 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2426 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2427 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2428 : : "SDDB::SimplePatternShifted"));  2429 : : }  2430 : :   2431 : 2404 : mShotDistributor(boost::ref(spat[0]),boost::ref(pos[0]), inSFLNo, mThdID, patternDivide.getAI());  2432 : 2404 : break;  2433 : : }  2434 : : case 2:  2435 : : {  2436 : 173 : if( spat.capacity() < 2 )  2437 : : {  2438 : 0 : spat.reserve( 2 );  2439 : 0 : spatOut.reserve( 3 );  2440 : 0 : pos.reserve( 2 );  2441 : 0 : posOut.reserve( 3 );  2442 : : }  2443 : 173 : patternDivide.createSimplePattern(spat[0],pos[0]);  2444 : 173 : try  2445 : : {  2446 : 173 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  2447 : : }  2448 [ # # ]: 0 : catch(DPL::Error \*pChild)  2449 : : {  2450 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2451 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2452 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2453 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2454 : : "SDDB::SimplePatternGenerated"));  2455 : : }  2456 : :   2457 : 173 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2458 : 173 : patternDivide.setSizeY( pat.getSizeY() - divisionLength );  2459 : 173 : patternDivide.createSimplePattern(spat[1], pos[1]);  2460 : 173 : try  2461 : : {  2462 : 173 : pSDDB->SimplePatternGenerated(spat[1],pos[1]);  2463 : : }  2464 [ # # ]: 0 : catch(DPL::Error \*pChild)  2465 : : {  2466 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2467 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2468 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2469 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2470 : : "SDDB::SimplePatternGenerated"));  2471 : : }  2472 : :   2473 : 173 : shotShift.adjustShotShiftAfterVerDivision(spat, pos, spatOut, posOut, 2, shiftLength);  2474 : 173 : if(shotShift.getShiftFlag()){ //When shiftLength == 0, not shifting is applied  2475 [ + + ]: 4 : for(uint32\_t i=0; i<3; i++){  2476 : 3 : try  2477 : : {  2478 : 3 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  2479 : : }  2480 [ # # ]: 0 : catch(DPL::Error \*pChild)  2481 : : {  2482 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2483 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2484 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2485 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2486 : : "SDDB::SimplePatternShifted"));  2487 : : }  2488 : 3 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  2489 : : }  2490 : : } else {  2491 [ + + ]: 516 : for(uint32\_t i=0; i<2; i++){  2492 : 344 : try  2493 : : {  2494 : 344 : pSDDB->SimplePatternShifted(spat[i],pos[i]);  2495 : : }  2496 [ # # ]: 0 : catch(DPL::Error \*pChild)  2497 : : {  2498 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2499 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2500 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2501 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2502 : : "SDDB::SimplePatternShifted"));  2503 : : }  2504 : 344 : mShotDistributor(boost::ref(spat[i]), boost::ref(pos[i]), inSFLNo, mThdID, patternDivide.getAI());  2505 : : }  2506 : : }  2507 : 172 : break;  2508 : : }  2509 : : case 3:  2510 : : {  2511 : 43 : if( spat.capacity() < 3 )  2512 : : {  2513 : 0 : spat.reserve( 3 );  2514 : 0 : spatOut.reserve( 4 );  2515 : 0 : pos.reserve( 3 );  2516 : 0 : posOut.reserve( 4 );  2517 : : }  2518 : 43 : patternDivide.createSimplePattern(spat[0], pos[0]);  2519 : 43 : try  2520 : : {  2521 : 43 : pSDDB->SimplePatternGenerated(spat[0],pos[0]);  2522 : : }  2523 [ # # ]: 0 : catch(DPL::Error \*pChild)  2524 : : {  2525 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2526 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2527 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2528 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2529 : : "SDDB::SimplePatternGenerated"));  2530 : : }  2531 : :   2532 : 43 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2533 : 43 : patternDivide.setSizeY( divisionLength );  2534 : 43 : patternDivide.createSimplePattern(spat[1], pos[1]);  2535 : 43 : try  2536 : : {  2537 : 43 : pSDDB->SimplePatternGenerated(spat[1],pos[1]);  2538 : : }  2539 [ # # ]: 0 : catch(DPL::Error \*pChild)  2540 : : {  2541 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2542 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2543 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2544 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2545 : : "SDDB::SimplePatternGenerated"));  2546 : : }  2547 : :   2548 : 43 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2549 : 43 : patternDivide.setSizeY( pat.getSizeY() - 2\*divisionLength );  2550 : 43 : patternDivide.createSimplePattern(spat[2], pos[2]);  2551 : 43 : try  2552 : : {  2553 : 43 : pSDDB->SimplePatternGenerated(spat[2],pos[2]);  2554 : : }  2555 [ # # ]: 0 : catch(DPL::Error \*pChild)  2556 : : {  2557 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2558 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2559 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2560 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2561 : : "SDDB::SimplePatternGenerated"));  2562 : : }  2563 : :   2564 : 43 : shotShift.adjustShotShiftAfterVerDivision(spat, pos, spatOut, posOut, 3, shiftLength);  2565 : 43 : if(shotShift.getShiftFlag()){ //When shiftLength == 0, not shifting is applied  2566 [ + + ]: 10 : for(uint32\_t i=0; i<4; i++){  2567 : 8 : try  2568 : : {  2569 : 8 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  2570 : : }  2571 [ # # ]: 0 : catch(DPL::Error \*pChild)  2572 : : {  2573 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2574 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2575 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2576 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2577 : : "SDDB::SimplePatternShifted"));  2578 : : }  2579 : :   2580 : 8 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  2581 : : }  2582 : : }else{  2583 [ + + ]: 164 : for(uint32\_t i=0; i<3; i++){  2584 : 123 : try  2585 : : {  2586 : 123 : pSDDB->SimplePatternShifted(spat[i],pos[i]);  2587 : : }  2588 [ # # ]: 0 : catch(DPL::Error \*pChild)  2589 : : {  2590 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2591 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2592 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2593 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2594 : : "SDDB::SimplePatternShifted"));  2595 : : }  2596 : :   2597 : 123 : mShotDistributor(boost::ref(spat[i]), pos[i], inSFLNo, mThdID, patternDivide.getAI());  2598 : : }  2599 : : }  2600 : 41 : break;  2601 : : }  2602 : : }  2603 : : }  2604 : : else  2605 : : {  2606 : 127 : uint32\_t midPatternLengthY, midPatternCount, L1, L2,  2607 : 127 : L1Count, L2Count;  2608 : 127 : midPatternLengthY = pat.getSizeY() - (divTH.edge<<1);  2609 : 127 : midPatternCount = (midPatternLengthY - 1)/divTH.max;  2610 : 127 : L1 = midPatternLengthY/(midPatternCount + 1);  2611 : 127 : L1 += ((midPatternCount +1)\*L1 < midPatternLengthY)?1:0;  2612 : 127 : L2 = midPatternLengthY - (midPatternCount \* L1);  2613 : 127 : L1Count = midPatternLengthY / L1;   2614 : 127 : L2Count = (midPatternLengthY - L1Count \* L1 >0)?1:0;  2615 : :   2616 : 127 : shiftLength = calculateShiftValueForSlice(  2617 : : inSFLNo, mMaxShotShiftCount, L1);  2618 : :   2619 : 127 : size\_t shotCount = 1 + L1Count + L2Count + 2;  2620 : :   2621 : 127 : if( spatOut.capacity() < shotCount )  2622 : : {  2623 : 54 : spat.reserve( shotCount );  2624 : 54 : spatOut.reserve( shotCount );  2625 : 54 : pos.reserve( shotCount );  2626 : 54 : posOut.reserve( shotCount );  2627 : : }  2628 : 127 : uint32\_t noOfPatterns =0;  2629 : : //Process the 1st edge  2630 : 127 : divisionLength = divTH.edge;  2631 : 127 : patternDivide.setCode(0x01);  2632 : 127 : patternDivide.setSizeY( divTH.edge );  2633 : 127 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  2634 : 127 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2635 : 127 : try  2636 : : {  2637 : 127 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  2638 : : }  2639 [ # # ]: 0 : catch(DPL::Error \*pChild)  2640 : : {  2641 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2642 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2643 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2644 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2645 : : "SDDB::SimplePatternGenerated"));  2646 : : }  2647 : 127 : noOfPatterns++;  2648 : : //Handle shot shift  2649 [ + + ]: 127 : if(shiftLength > 0){  2650 : 3 : patternDivide.setSizeY( shiftLength ); //Fix 4  2651 : 3 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  2652 : 3 : try  2653 : : {  2654 : 3 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  2655 : : }  2656 [ # # ]: 0 : catch(DPL::Error \*pChild)  2657 : : {  2658 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2659 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2660 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2661 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2662 : : "SDDB::SimplePatternGenerated"));  2663 : : }  2664 : 3 : noOfPatterns++;  2665 : 3 : prepareNextVerticalRectangle(patternDivide, shiftLength);  2666 : : } else{ // Shift == 0  2667 : : }  2668 : 127 : L1Count--;  2669 : : //Process the L1 patterns  2670 [ + + + + ]: 1442 : for (i=0; i<L1Count; i++)  2671 : : {  2672 : 1315 : patternDivide.setSizeY( L1 ); //Fix 4  2673 : 1315 : patternDivide.createSimplePattern(spatOut[noOfPatterns],posOut[noOfPatterns]);  2674 : 1315 : try  2675 : : {  2676 : 1315 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  2677 : : }  2678 [ # # ]: 0 : catch(DPL::Error \*pChild)  2679 : : {  2680 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2681 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2682 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2683 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2684 : : "SDDB::SimplePatternGenerated"));  2685 : : }  2686 : 1315 : divisionLength = L1;  2687 : 1315 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2688 : 1315 : noOfPatterns++;  2689 : : }  2690 : : //Process the L2 patterns  2691 [ + + ]: 127 : if (L2Count)  2692 : : {  2693 : 14 : divisionLength = L2;  2694 : 14 : patternDivide.setSizeY( divisionLength );  2695 : 14 : patternDivide.createSimplePattern(  2696 : : spatOut[noOfPatterns], posOut[noOfPatterns]);  2697 : 14 : try  2698 : : {  2699 : 14 : pSDDB->SimplePatternGenerated(  2700 : : spatOut[noOfPatterns],posOut[noOfPatterns]);  2701 : : }  2702 [ # # ]: 0 : catch(DPL::Error \*pChild)  2703 : : {  2704 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2705 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2706 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2707 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2708 : : "SDDB::SimplePatternGenerated"));  2709 : : }  2710 : 14 : prepareNextVerticalRectangle(patternDivide, divisionLength);  2711 : 14 : noOfPatterns++;  2712 : : }  2713 : : // if( !shotShift )  2714 : : // {  2715 : : // patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  2716 : : // try  2717 : : // {  2718 : : // pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  2719 : : // }  2720 : : // catch(DPL::Error \*pChild)  2721 : : // {  2722 : : // ErrorHandler \*pErr = ErrorHandler::instance();  2723 : : // pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2724 : : // \_\_LINE\_\_, ErrorHandler::eErrFunction,  2725 : : // pErr->getErrorMsg(ErrorHandler::eErrFunction,  2726 : : // "SDDB::SimplePatternGenerated"));  2727 : : // }  2728 : : // }  2729 : :   2730 : : // noOfPatterns++;  2731 : :   2732 : : //Handle last pattern  2733 [ + + ]: 127 : if(shiftLength > 0){//Handle when shift != 0  2734 : 3 : patternDivide.setSizeY( L1-shiftLength );  2735 : : }else{//Handle when shift == 0  2736 : 124 : patternDivide.setSizeY( L1 );  2737 : : }  2738 : 127 : patternDivide.createSimplePattern(spatOut[noOfPatterns], posOut[noOfPatterns]);  2739 : 127 : try  2740 : : {  2741 : 127 : pSDDB->SimplePatternGenerated(spatOut[noOfPatterns],posOut[noOfPatterns]);  2742 : : }  2743 [ # # ]: 0 : catch(DPL::Error \*pChild)  2744 : : {  2745 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2746 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2747 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2748 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2749 : : "SDDB::SimplePatternGenerated"));  2750 : : }  2751 : 127 : noOfPatterns++;  2752 : :   2753 : : //Process the 2nd edge  2754 [ + + ]: 127 : if(shiftLength > 0){//Handle when shift != 0  2755 : 3 : prepareNextVerticalRectangle(patternDivide, L1-shiftLength );  2756 : : }else{  2757 : 124 : prepareNextVerticalRectangle(patternDivide, L1);  2758 : : }  2759 : 127 : patternDivide.setSizeY( divTH.edge );  2760 : 127 : patternDivide.createSimplePattern(  2761 : : spatOut[noOfPatterns], posOut[noOfPatterns]);  2762 : 127 : try  2763 : : {  2764 : 127 : pSDDB->SimplePatternGenerated(  2765 : : spatOut[noOfPatterns],posOut[noOfPatterns]);  2766 : : }  2767 [ # # ]: 0 : catch(DPL::Error \*pChild)  2768 : : {  2769 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2770 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2771 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2772 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2773 : : "SDDB::SimplePatternGenerated"));  2774 : : }  2775 : 127 : noOfPatterns++;  2776 : :   2777 [ + - + + ]: 1840 : for(uint32\_t i=0; i<noOfPatterns; i++){  2778 : 1713 : try  2779 : : {  2780 : 1713 : pSDDB->SimplePatternShifted(spatOut[i],posOut[i]);  2781 : : }  2782 [ # # ]: 0 : catch(DPL::Error \*pChild)  2783 : : {  2784 : 0 : ErrorHandler \*pErr = ErrorHandler::instance();  2785 : 0 : pErr->errorAdd(pChild, \_\_FILE\_\_, \_\_FUNCTION\_\_,  2786 : : \_\_LINE\_\_, ErrorHandler::eErrFunction,  2787 : : pErr->getErrorMsg(ErrorHandler::eErrFunction,  2788 : : "SDDB::SimplePatternShifted"));  2789 : : }  2790 : 1713 : mShotDistributor(boost::ref(spatOut[i]), boost::ref(posOut[i]), inSFLNo, mThdID, patternDivide.getAI());  2791 : : }  2792 : : }  2793 : : //Now change the coordinate of the original pattern  2794 : 2747 : pat.setPositionY( patternDivide.getPositionY() + patternDivide.getSizeY());   2795 : : }  2796 : :   2797 : :   2798 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2799 : : \* Function Name: createVerticalBottomTriangle  2800 : : \* Purpose: Extract the bottom triangle for a vertical   2801 : : \* pattern  2802 : : \* Parameters:  2803 : : \* inPat: ParsedPattern [IN]  2804 : : \* Return Values:  2805 : : \* None  2806 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2807 : :   2808 : : inline void PatternProcessor::createVerticalBottomTriangle(ParsedPattern& inPat, uint32\_t inSFLNo)  2809 : :   2810 : 1628 : {  2811 : 1628 : uint32\_t size = inPat.getSizeX();  2812 : 1628 : int64\_t x = inPat.getPositionX();  2813 : 1628 : int64\_t y = inPat.getPositionY();  2814 : 1628 : uint32\_t attribute = inPat.getAI();  2815 : 1628 : switch(inPat.getCode())  2816 : : {  2817 : : case 0x01:  2818 : : case 0x02:  2819 : 1212 : break;  2820 : :   2821 : : case 0x03:  2822 : : case 0x06:   2823 : 1212 : createBottomTriangle\_0x06(x, y, size, attribute, inSFLNo);  2824 : 1212 : inPat.setPositionY(inPat.getPositionY() + size);  2825 : 1212 : break;  2826 : : case 0x04:  2827 : : case 0x05:  2828 : 272 : createBottomTriangle\_0x05(x, y, size, attribute, inSFLNo);  2829 : 272 : break;  2830 : : case 0x0d:  2831 : 84 : createBottomTriangle\_0x06(x, y, size, attribute, inSFLNo);  2832 : 84 : inPat.setPositionY(inPat.getPositionY() + size);  2833 : 84 : inPat.setSizeY( inPat.getSizeY() - size);  2834 : 84 : break;  2835 : : case 0x0e:  2836 : 60 : createBottomTriangle\_0x05(x, y, size, attribute, inSFLNo);  2837 : 60 : inPat.setSizeY( inPat.getSizeY() - size);  2838 : 1628 : break;   2839 : : }  2840 : : }  2841 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2842 : : \* Function Name: createVerticalTopTriangle  2843 : : \* Purpose: Extract the top triangle for a vertical  2844 : : \* pattern  2845 : : \* Parameters:  2846 : : \* inPat: ParsedPattern [IN]  2847 : : \* Return Values:  2848 : : \* None  2849 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2850 : : inline void PatternProcessor::createVerticalTopTriangle(ParsedPattern& inPat, uint32\_t inSFLNo)   2851 : :   2852 : 1394 : {  2853 : 1394 : uint32\_t size = inPat.getSizeX();  2854 : 1394 : int64\_t x = inPat.getPositionX();  2855 : 1394 : int64\_t y = inPat.getPositionY();  2856 : 1394 : uint32\_t attribute = inPat.getAI();  2857 : 1394 : switch(inPat.getCode())  2858 : : {  2859 : : case 0x01:  2860 : : case 0x05:  2861 : : case 0x0d:  2862 : 819 : createTopTriangle\_0x05(x, y, size, attribute, inSFLNo);  2863 : 819 : break;  2864 : : case 0x02:  2865 : : case 0x06:  2866 : : case 0x0e:  2867 : 575 : createTopTriangle\_0x06(x, y, size, attribute, inSFLNo);  2868 : 1394 : break;  2869 : : case 0x03:  2870 : : case 0x04:  2871 : 1394 : break;  2872 : : }  2873 : : }  2874 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2875 : : \* Function Name: prepareNextVerticalRectangle  2876 : : \* Purpose: Prepare the next vertical rectangle from a   2877 : : \* vertical slice  2878 : : \* Parameters:  2879 : : \* pattern: ParsedPattern [IN]  2880 : : \* divisionLength: DIV\_TH [IN]  2881 : : \* Return Values:  2882 : : \* None  2883 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2884 : : inline void PatternProcessor::prepareNextVerticalRectangle  2885 : : (ParsedPattern& patternDivide,  2886 : : uint32\_t divisionLength)  2887 : :   2888 : 1845 : {  2889 : 1845 : patternDivide.setPositionY(patternDivide.getPositionY() + divisionLength);   2890 : : }  2891 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  2892 : : \* Function Name: prepareNextVerticalSlice  2893 : : \* Purpose: Prepare the next slice for processing of a given  2894 : : \* vertical pattern  2895 : : \* Parameters:  2896 : : \* pat: ParsedPattern [IN]  2897 : : \* divisionLength: uint32\_t [IN]  2898 : : \* Return Values:  2899 : : \* None  2900 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  2901 : : inline void PatternProcessor::prepareNextVerticalSlice (ParsedPattern& patternSlice,   2902 : : uint32\_t divisionLength)  2903 : 516 : {  2904 : 516 : patternSlice.setPositionY(patternSlice.getPositionY() + divisionLength \*   2905 : : mEdgeVector.getEY(patternSlice.getCode()) );   2906 : 516 : patternSlice.setPositionX(patternSlice.getPositionX() + divisionLength);  2907 : : }   2908 : :   2909 : 1205 : inline void PatternProcessor::registerDistributor(const TShotDistributor &func){  2910 : 1205 : mShotDistributor = func;  2911 : : }  2912 : :   2913 : : }; // namespace nft  2914 : :   2915 : : #endif |

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