| LCOV - code coverage report |
| --- |
|  |
| | Current view: | [top level](http://docs.google.com/index.html) - [LibVsb/include](http://docs.google.com/index.html) - vsb\_utils.h (source / [functions](http://docs.google.com/vsb_utils.h.func.html)) |  |  | Hit | Total | Coverage | | --- | --- | --- | --- | --- | --- | --- | | Test: | EBM9K Converter Coverage Report |  | Lines: | 163 | 172 | 94.8 % | | Date: | 2012-03-23 |  | Functions: | 5 | 5 | 100.0 % | | Legend: | Lines: hit not hit | Branches: + taken - not taken # not executed |  | Branches: | 57 | 121 | 47.1 % | |  | |
|  |

|  |
| --- |
| Branch data Line data Source code  1 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   2 : : \* Copyrighted 2007 by Interra Systems, USA  3 : : \* All rights reserved   4 : : \*  5 : : \* Date of Creation: Tue Aug 28 16:57:25 IST 2007  6 : : \* File Name: vsb\_utils.h  7 : : \* Purpose:   8 : : \* Author:   9 : : \*  10 : : \* Revision History:   11 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/   12 : :   13 : : /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  14 : : \* Header Include section   15 : : \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  16 : :   17 : : #include "context.h"  18 : : #include "vsb\_types.h"  19 : : #include "limit.h"  20 : : #include "dec\_types.h"  21 : : #include "dec\_math.h"  22 : : #include "Porting.h"  23 : : #include "PH4Flag.h"  24 : : #include "StrictAliasUnion.h"  25 : : #include "ErrorHandler.h"  26 : :   27 : : #ifndef \_VSB\_UTILS\_H\_  28 : : #define \_VSB\_UTILS\_H\_  29 : :   30 : : namespace nft {  31 : : const LocDirection WhichDirection[3]=  32 : : {  33 : : XYDirxn,  34 : : XDirxn,  35 : : YDirxn  36 : : } ;  37 : :   38 : : static const int NumPPFields[3]=  39 : : {  40 : : 2,  41 : : 1,  42 : : 1  43 : : } ;  44 : :   45 : : static const int BitMask[5]=  46 : : {  47 : : 0x00000000,  48 : : 0xFF000000,  49 : : 0xFFFF0000,  50 : : 0xFFFFFF00,  51 : : 0xFFFFFFFF  52 : : } ;  53 : :   54 : : static const int BitMask\_rev[5]=  55 : : {  56 : : 0x00000000,  57 : : 0x000000FF,  58 : : 0x0000FFFF,  59 : : 0x00FFFFFF,  60 : : 0xFFFFFFFF  61 : : } ;  62 : :   63 : : static const int Shift[5]=  64 : : {  65 : : 8 \* 4,  66 : : 8 \* 3,  67 : : 8 \* 2,  68 : : 8 \* 1,  69 : : 8 \* 0  70 : : } ;  71 : :   72 : : /\*  73 : : static const uint64\_t BitMaskPP[5]=  74 : : {  75 : : 0x0000000000000000,  76 : : 0x00000000000000FF,  77 : : 0x000000000000FFFF,  78 : : 0x0000000000FFFFFF,  79 : : 0x00000000FFFFFFFF  80 : : } ;  81 : : \*/  82 : :   83 : : static const int ShiftPP[8]=  84 : : { 8 \* 0, /\* dont care \*/  85 : : 8 \* 7,  86 : : 8 \* 6,  87 : : 8 \* 5,   88 : : 8 \* 4,  89 : : 8 \* 3,  90 : : 8 \* 2,  91 : : 8 \* 1,  92 : : } ;  93 : :   94 : : static const int SizeOfOption[2]=  95 : : {  96 : : 0,  97 : : sizeof(int)   98 : : } ;  99 : :   100 : : static const int SizeOfAP[5][4]=  101 : : {  102 : : {2,2,4,2},  103 : : {3,3,6,4},  104 : : {4,4,8,6},  105 : : {5,5,10,8},  106 : : {6,6,12,10}  107 : : } ;  108 : :   109 : : static const int AlignedSizeOfAP[5][4]=  110 : : {  111 : : {2,2,4,2},  112 : : {4,4,8,4},  113 : : {4,4,8,6},  114 : : {6,6,12,8},  115 : : {6,6,12,10}  116 : : } ;  117 : :   118 : : static const PatternArrangement WhichArrangement[5]=  119 : : {  120 : : NoArrangement,  121 : : Horizontal,  122 : : Vertical,  123 : : Matrix,  124 : : Linear  125 : : } ;  126 : :   127 : :   128 : : static const ArrayType WhichArrayType[2]=  129 : : {  130 : : Term,  131 : : Pitch,  132 : : } ;  133 : :   134 : : typedef union {  135 : : struct {  136 : : uint hi ;  137 : : uint lo ; } split ;  138 : : uint64\_t val ;  139 : : } raw\_data\_t ;  140 : :   141 : : inline unsigned short vsbGetShort(Byte \*buffer)  142 : 848 : {  143 : 424 : unsigned short int nTemp = \*((unsigned short int\*)buffer);  144 : 424 : nTemp = BE2LE\_16(nTemp);  145 : : return(nTemp);  146 : : }  147 : :   148 : : inline uint vsbGetInt(Byte \*buffer)  149 : 17336 : {  150 : 8668 : unsigned int nTemp = \*((unsigned int\*)buffer);  151 : 8668 : nTemp = BE2LE\_32(nTemp);  152 : : return(nTemp);  153 : : }   154 : :   155 : :   156 : : inline ContextType GetContextType(Byte \*PatternHeader)  157 : 4304 : {  158 : 2152 : ContextType ctype = NoContext ;  159 : 2152 : vsbPH1 ph1 ;  160 : 2152 : uint temp ;  161 : 2152 : alias\_PH4\_1 alias\_PH4\_1Val;  162 : 2152 : alias\_PH1 alias\_PH1Val;  163 : :   164 : : //PH4 Handling  165 : 2152 : Byte \*PatternHeaderAfterPH4 = PatternHeader;  166 : :   167 : : //temp = vsbGetInt(PatternHeader) ;  168 : 2152 : alias\_PH4\_1Val.value = vsbGetInt(PatternHeader) ;  169 : 2152 : if( 1 == PH4Flag::instance()->getValue() )  170 : : {  171 : 32 : vsbPH4\_1 ph4;  172 : : //ph4 =\*(vsbPH4\_0 \*)(&temp) ;  173 : 32 : ph4 = alias\_PH4\_1Val.ph4 ;  174 [ + - ][ # # ]: 32 : switch(ph4.b)  [ # # ]  175 : : {  176 : : case 0xb1:  177 : : case 0xb2:  178 : 32 : PatternHeaderAfterPH4 = PatternHeader + sizeof(vsbPH4\_1);  179 : 2152 : break;  180 : : }  181 : : }  182 : :   183 : : //temp = vsbGetInt(PatternHeaderAfterPH4) ;  184 : 2152 : alias\_PH1Val.value = vsbGetInt(PatternHeaderAfterPH4) ;  185 : : //ph1 = \*(vsbPH1 \*)(&temp);   186 : 2152 : ph1 = alias\_PH1Val.ph1;  187 : :   188 [ + + ][ + + ]: 2152 : if(ph1.code1 == 0x81 || ph1.code1 == 0x82)  [ + + ]  189 : 263 : ctype = Array ;  190 [ + + ][ + + ]: 1889 : else if(ph1.code1 >= 0x91)  [ - + ]  191 : 59 : ctype = Composite ;  192 : : else  193 : : {  194 : : //ph1 = (vsbPH1 \*)PatternHeader ;  195 [ + + ][ + + ]: 1830 : if(ph1.isOR2)  [ + + ]  196 : 174 : ctype = Optimize ;  197 : : else  198 : 1656 : ctype = Normal ;  199 : 2152 : }  200 : :   201 : : return ctype ;  202 : : }  203 : :   204 : : inline int vsbNumEPFields(unsigned char PatCode)  205 : 4122 : {  206 [ + - ][ + + : 2061 : switch(PatCode)  - - - ]  [ + - ][ - +   - - - ]  [ + - ][ - +   - - - ]  207 : : {  208 : : case 0x31 :  209 : : case 0x32 :  210 : : case 0x33 :  211 : : case 0x34 :  212 : 301 : case 0x35 : return 3 ;  213 : :   214 : : case 0x01 :  215 : : case 0x02 :  216 : : case 0x03 :  217 : : case 0x04 :  218 : : case 0x05 :  219 : : case 0x06 :  220 : : case 0x07 :  221 : : case 0x08 :  222 : : case 0x09 :  223 : : case 0x0a :  224 : : case 0x0b :  225 : : case 0x0c :  226 : : case 0x0d :  227 : : case 0x0e :  228 : : case 0x0f :  229 : : case 0x10 :  230 : : case 0x11 :  231 : : case 0x40 :  232 : : case 0x41 :  233 : : case 0x42 :  234 : 1760 : case 0x43 : return 4 ;  235 : :   236 : : case 0x44 :  237 : 0 : case 0x45 : return 5 ;  238 : :   239 : : case 0x46 :  240 : 0 : case 0x47 : return 6 ;  241 : 0 : default : return 0 ;  242 : : }  243 : : }  244 : :   245 : :   246 : : inline int vsbBufferToEP(Byte \*buffer, const int NumFieldsInEP, const int X0Y0WordLength, const int L1WordLength, const int L2WordLength, EP \*ep)   247 : 11574 : {  248 : :   249 : : #ifndef SOLARIS //LINUX default  250 : :   251 : 5787 : int SkipSize = X0Y0WordLength + X0Y0WordLength + L1WordLength ;  252 : 5787 : unsigned int nTemp =0;  253 : 5787 : const int BitMaskLinux[5]=  254 : : {  255 : : 0x00000000,  256 : : 0xFF000000,  257 : : 0xFFFF0000,  258 : : 0xFFFFFF00,  259 : : 0xFFFFFFFF  260 : 5787 : } ;  261 : :   262 : 5787 : nTemp = \*((unsigned int\*)(buffer-(WORD-X0Y0WordLength)));  263 : 5787 : nTemp = nTemp & BitMaskLinux[X0Y0WordLength];   264 : 5787 : ep->X0 = BE2LE\_32(nTemp);  265 : 5787 : buffer += X0Y0WordLength;   266 : :   267 : 5787 : nTemp = \*((unsigned int\*)(buffer-(WORD-X0Y0WordLength)));  268 : 5787 : nTemp = nTemp & BitMaskLinux[X0Y0WordLength];   269 : 5787 : ep->Y0 = BE2LE\_32(nTemp);  270 : 5787 : buffer += X0Y0WordLength;   271 : :   272 : 5787 : nTemp = \*((unsigned int\*)(buffer-(WORD-L1WordLength)));  273 : 5787 : nTemp = nTemp & BitMaskLinux[L1WordLength];   274 : 5787 : ep->L1 = BE2LE\_32(nTemp);  275 : 5787 : buffer += L1WordLength;  276 : :   277 [ + + ]: 5787 : if (4 == NumFieldsInEP)  278 : : {  279 : 4526 : nTemp = \*((unsigned int\*)(buffer-(WORD-L2WordLength)));  280 : 4526 : nTemp = nTemp & BitMaskLinux[L2WordLength];   281 : 4526 : ep->L2 = BE2LE\_32(nTemp);  282 : 4526 : SkipSize += L2WordLength ;  283 : :   284 : : } else {  285 : 1261 : ep->L2 = ep->L1;  286 : 5787 : }  287 : : return SkipSize ;  288 : :   289 : :   290 : : #else // SOLARIS  291 : : register uint offset ;   292 : : register raw\_data\_t raw\_data ;  293 : : register uint \*temp\_ptr ;  294 : :   295 : : int SkipSize = X0Y0WordLength + X0Y0WordLength + L1WordLength ;  296 : :   297 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/  298 : : temp\_ptr = (uint \*)(buffer - offset) ;  299 : : raw\_data.split.hi = \*temp\_ptr ;  300 : : raw\_data.split.lo = \*(++temp\_ptr) ;  301 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + X0Y0WordLength] ;  302 : : ep->X0 = (raw\_data.split.lo) & BitMask\_rev[X0Y0WordLength] ;  303 : : buffer += X0Y0WordLength ;  304 : :   305 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/   306 : : temp\_ptr = (uint \*)(buffer - offset) ;  307 : : raw\_data.split.hi = \*temp\_ptr ;  308 : : raw\_data.split.lo = \*(++temp\_ptr) ;  309 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + X0Y0WordLength] ;  310 : : ep->Y0 = (raw\_data.split.lo) & BitMask\_rev[X0Y0WordLength] ;  311 : : buffer += X0Y0WordLength ;  312 : :   313 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/  314 : : temp\_ptr = (uint \*)(buffer - offset) ;  315 : : raw\_data.split.hi = \*temp\_ptr ;  316 : : raw\_data.split.lo = \*(++temp\_ptr) ;  317 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + L1WordLength] ;  318 : : ep->L1 = (raw\_data.split.lo) & BitMask\_rev[L1WordLength] ;  319 : : buffer += L1WordLength ;  320 : :   321 : : ep->L2 = ep->L1 ;  322 : :   323 : : /\* Consider Removing Loop and using Static States for Performance Reasons \*/  324 : : if (4 == NumFieldsInEP)  325 : : {  326 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/  327 : : temp\_ptr = (uint \*)(buffer - offset) ;  328 : : raw\_data.split.hi = \*temp\_ptr ;  329 : : raw\_data.split.lo = \*(++temp\_ptr) ;  330 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + L2WordLength] ;  331 : : ep->L2 = (raw\_data.split.lo) & BitMask\_rev[L2WordLength] ;  332 : : // buffer += L2WordLength ; /\* not required, as buffer is not used after this point \*/  333 : : SkipSize += L2WordLength ;  334 : : }  335 : : return SkipSize ;  336 : : #endif  337 : : }  338 : :   339 : : inline int vsbBufferToPP(Byte \*buffer, const LocDirection PPDirection, const int X0Y0WordLength, PP \*pp)   340 : 944 : {  341 : :   342 : : #ifndef SOLARIS //LINUX default  343 : 472 : int SkipSize = X0Y0WordLength ;  344 : :   345 : 472 : int NumFieldsInPP = NumPPFields[PPDirection] ;  346 : 472 : unsigned int nTemp =0;  347 : 472 : unsigned int nTemp1 =pp->X0;  348 : 472 : const int BitMaskLinux[5]=  349 : : {  350 : : 0x00000000,  351 : : 0xFF000000,  352 : : 0xFFFF0000,  353 : : 0xFFFFFF00,  354 : : 0xFFFFFFFF  355 : 472 : } ;  356 : :   357 : 472 : nTemp = \*((unsigned int\*)(buffer-(WORD-X0Y0WordLength)));  358 : 472 : nTemp = nTemp & BitMaskLinux[X0Y0WordLength];  359 : 472 : pp->X0 = BE2LE\_32(nTemp);  360 : 472 : buffer += X0Y0WordLength;  361 : :   362 [ + + ][ + - ]: 472 : if (2 == NumFieldsInPP)  [ + + ]  363 : : {  364 : 414 : nTemp = \*((unsigned int\*)(buffer-(WORD-X0Y0WordLength)));  365 : 414 : nTemp = nTemp & BitMaskLinux[X0Y0WordLength];  366 : 414 : pp->Y0 = BE2LE\_32(nTemp);  367 : 414 : SkipSize += X0Y0WordLength ;  368 : :   369 : : }   370 : :   371 [ + + ][ + + ]: 413 : if(YDirxn == PPDirection)  372 : : {  373 : 85 : pp->Y0 = pp->X0;  374 : 26 : pp->X0 = nTemp1;  375 : : }  376 : :   377 : 472 : return SkipSize ;  378 : :   379 : : #else // SOLARIS  380 : :   381 : : register raw\_data\_t raw\_data ;  382 : : register uint \*temp\_ptr ;  383 : : register uint offset ;  384 : : int SkipSize = X0Y0WordLength ;  385 : : int NumFieldsInPP = NumPPFields[PPDirection] ;  386 : : int X0 = pp->X0 ;  387 : :   388 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/  389 : : temp\_ptr = (uint \*)(buffer - offset) ;  390 : : raw\_data.split.hi = \*temp\_ptr ;  391 : : raw\_data.split.lo = \*(++temp\_ptr) ;  392 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + X0Y0WordLength] ;  393 : : pp->X0 = (raw\_data.split.lo) & BitMask\_rev[X0Y0WordLength] ;  394 : : buffer += X0Y0WordLength ;  395 : :   396 : : if (2 == NumFieldsInPP)  397 : : {  398 : : offset = (uint64\_t)buffer & 0x0000000000000003 ; /\* requires last 2 bits only \*/  399 : : temp\_ptr = (uint \*)(buffer - offset) ;  400 : : raw\_data.split.hi = \*temp\_ptr ;  401 : : raw\_data.split.lo = \*(++temp\_ptr) ;  402 : : raw\_data.val = raw\_data.val >> ShiftPP[offset + X0Y0WordLength] ;  403 : : pp->Y0 = (raw\_data.split.lo) & BitMask\_rev[X0Y0WordLength] ;  404 : : // buffer += X0Y0WordLength ; /\* not required, as buffer is not used after this point \*/  405 : : SkipSize += X0Y0WordLength ;  406 : : }  407 : :   408 : : if(YDirxn == PPDirection)  409 : : {  410 : : pp->Y0 = pp->X0 ;  411 : : pp->X0 = X0 ;  412 : : }  413 : :   414 : : return SkipSize ;  415 : : #endif  416 : : }  417 : :   418 : : inline int vsbBufferToValue(Byte \*buffer, const int ValWordLength, const int isAligned, int \*Value)   419 : 570 : {  420 : 285 : int SkipSize = ValWordLength ;  421 : :   422 [ - + ][ - + ]: 186 : if(isAligned)  [ - + ][ - + ]  423 : : {  424 : 0 : SkipSize = (int)ceil((double)(ValWordLength/2.0)) \* 2 ;  425 : : }   426 : :   427 : 285 : \*Value = vsbGetInt(buffer) ;  428 : 285 : \*Value = \*Value & BitMask[ValWordLength] ;  429 : 285 : \*Value = \*Value >> Shift[ValWordLength] ;  430 : :   431 : : return SkipSize ;  432 : : }  433 : :   434 : : inline int sizeofEP(const unsigned char PatCode, const int X0Y0WordLength, const int L1WordLength, const int L2WordLength, const int isAligned)  435 : 450 : {  436 : 225 : int EPSize = (2 \* X0Y0WordLength) ;  437 : :   438 : 225 : int NumFieldsinEP = vsbNumEPFields(PatCode) ;  439 : :   440 : 225 : int DeltaXY = EPSize ;  441 : 225 : int Delta = X0Y0WordLength ;  442 : :   443 [ - + ][ - + ]: 225 : if(isAligned)  [ - + ]  444 : : {  445 : 0 : Delta = (int)ceil((double)(Delta/2.0)) \* 2 ;  446 : : }  447 : :   448 [ - + ][ # # ]: 225 : switch(NumFieldsinEP)  [ # # ][ # # ]  [ # # ][ - + ]  [ # # ][ # # ]  [ # # ][ # # ]  [ - + ][ # # ]  [ # # ][ # # ]  [ # # ]  449 : : {  450 : 0 : case 3 : EPSize += L1WordLength ; break ;  451 : 225 : case 4 : EPSize += L1WordLength + L2WordLength ; break ;  452 : 0 : case 5 : EPSize += L1WordLength + L2WordLength + Delta ; break ;  453 : 0 : case 6 : EPSize += L1WordLength + L2WordLength + DeltaXY ; break ;  454 : 225 : default : break ;  455 : 225 : }  456 : :   457 : : return EPSize ;  458 : :   459 : : }  460 : :   461 : : inline int sizeofNR(Byte \*buffer, const int X0Y0WordLength, const int isAligned)  462 : 79 : {  463 : 79 : int NRSize = 0 ;  464 : 79 : alias\_PH1 alias\_PH1Val;  465 : :   466 [ + - ]: 79 : if(buffer)  467 : : {  468 : : //vsbPH1 \*ph1 = (vsbPH1 \*)(buffer);  469 : 79 : uint temp ;  470 : : //temp = vsbGetInt(buffer);  471 : 79 : alias\_PH1Val.value = vsbGetInt(buffer) ;  472 : : //vsbPH1 ph1 = \*(vsbPH1 \*)&temp;  473 : 79 : vsbPH1 ph1 = alias\_PH1Val.ph1;  474 : 79 : int NumEP = ph1.NumEP ;  475 : : //int NumFieldsInEP = vsbNumEPFields(ph1.code1) ;  476 : 79 : int L1WordLength = ph1.L1WordLength + 1 ;  477 : 79 : int L2WordLength = ph1.L2WordLength + 1 ;  478 : 79 : int EPSize = sizeofEP(ph1.code1, X0Y0WordLength, L1WordLength, L2WordLength, isAligned) ;   479 : :   480 : 79 : NRSize = sizeof(vsbPH1) + SizeOfOption[ph1.isOptionAdded] + (EPSize \* NumEP);  481 : : }  482 : :   483 : 79 : return NRSize ;  484 : : }   485 : :   486 : : inline int sizeofOR(Byte \*buffer, const int X0Y0WordLength, const int isAligned)  487 : 72 : {  488 : 72 : int ORSize = 0 ;  489 : 72 : alias\_PH1 alias\_PH1Val;  490 : :   491 [ + - ]: 72 : if(buffer)  492 : : {  493 : : // vsbPH1 \*ph1 = (vsbPH1 \*)(buffer);  494 : 72 : uint temp ;  495 : : //temp = vsbGetInt(buffer);  496 : 72 : alias\_PH1Val.value = vsbGetInt(buffer);  497 : : //vsbPH1 ph1 = \*(vsbPH1 \*)&temp;  498 : 72 : vsbPH1 ph1 = alias\_PH1Val.ph1;  499 : : //int NumFieldsInEP = vsbNumEPFields(ph1.code1) ;  500 : 72 : int L1WordLength = ph1.L1WordLength + 1 ;  501 : 72 : int L2WordLength = ph1.L2WordLength + 1 ;  502 : 72 : int EPSize = sizeofEP(ph1.code1, X0Y0WordLength, L1WordLength, L2WordLength, isAligned) ;   503 : :   504 : 72 : int PPSize = NumPPFields[WhichDirection[ph1.PPDirection]] \* X0Y0WordLength ;  505 : :   506 : 72 : ORSize = sizeof(vsbPH1) + SizeOfOption[ph1.isOptionAdded] ;  507 : :   508 : 72 : buffer += ORSize ;  509 : :   510 [ + - ][ - + ]: 72 : for(unsigned int i=0; i < ph1.NumEP; i++)  511 : : {  512 : 72 : short int NumPP = vsbGetShort(buffer);   513 : 72 : int OR1Size = sizeof(short int) + EPSize + ((NumPP-1) \* PPSize) ;  514 : 72 : ORSize += OR1Size ;  515 : 72 : buffer += OR1Size ;  516 : : }  517 : : }  518 : :   519 : 72 : return ORSize ;  520 : : }  521 : :   522 : : inline int sizeofAR(Byte \*buffer, const int X0Y0WordLength, const int isAligned)  523 : 74 : {  524 : 74 : int ARSize = 0 ;  525 : 74 : alias\_PH2 alias\_PH2Val;  526 : :   527 [ + - ]: 74 : if(buffer)  528 : : {  529 : : // vsbPH2 \*ph2 = (vsbPH2 \*)(buffer);  530 : 74 : uint temp ;  531 : : //temp = vsbGetInt(buffer);  532 : 74 : alias\_PH2Val.value = vsbGetInt(buffer);  533 : : //vsbPH2 ph2 = \*(vsbPH2 \*)&temp;  534 : 74 : vsbPH2 ph2 = alias\_PH2Val.ph2;  535 : : //int NumFieldsInEP = vsbNumEPFields(ph2.code1) ;  536 : 74 : int L1WordLength = ph2.L1WordLength + 1 ;  537 : 74 : int L2WordLength = ph2.L2WordLength + 1 ;  538 : 74 : int EPSize = sizeofEP(ph2.code1, X0Y0WordLength, L1WordLength, L2WordLength, isAligned) ;   539 : 74 : PatternArrangement parrange = WhichArrangement[ph2.codea] ;  540 : 74 : int APSize = SizeOfAP[X0Y0WordLength][parrange] ;  541 : :   542 [ - + ]: 74 : if(isAligned)  543 : : {  544 : 0 : APSize = AlignedSizeOfAP[X0Y0WordLength][parrange] ;  545 : : }  546 : :   547 : 74 : ARSize = sizeof(vsbPH2) + SizeOfOption[ph2.isOptionAdded] + APSize + EPSize;  548 : : }  549 : :   550 : 74 : return ARSize ;  551 : : }  552 : :   553 : : inline int itod(EP \*ep, vsbEP \*vep)  554 : 1004 : {  555 : 502 : vep->PatCode = ep->PatCode ;  556 : 502 : vep->X0 = ep->X0 ;  557 : 502 : vep->Y0 = ep->Y0 ;  558 [ + - ]: 502 : vep->L1 = ep->L1 ;  559 [ + - ]: 502 : vep->L2 = ep->L2 ;  560 : : // vep->DLx = ep->DLx ;  561 : : // vep->DLy = ep->DLy ;  562 : :   563 : : return 0 ;  564 : : }   565 : :   566 : : inline int dtoi(vsbEP \*vep, EP \*ep)  567 : 842 : {  568 : 421 : ep->PatCode = vep->PatCode ;   569 : 421 : ep->X0 = RoundINT(vep->X0) ;   570 : 421 : ep->Y0 = RoundINT(vep->Y0) ;   571 : 421 : ep->L1 = RoundINT(vep->L1) ;   572 : 842 : ep->L2 = RoundINT(vep->L2) ;   573 : : // ep->DLx = RoundINT(vep->DLx) ;   574 : : // ep->DLy = RoundINT(vep->DLy) ;   575 : :   576 : : return 0 ;  577 : : }   578 : :   579 : : inline int itod(PP \*pp, vsbPP \*vpp)  580 : : {  581 : : vpp->X0 = pp->X0 ;  582 : : vpp->Y0 = pp->Y0 ;  583 : :   584 : : return 0 ;  585 : : }   586 : :   587 : : inline int dtoi(vsbPP \*vpp, PP \*pp)  588 : 900 : {  589 : 450 : pp->X0 = RoundINT(vpp->X0) ;   590 : 900 : pp->Y0 = RoundINT(vpp->Y0) ;   591 : :   592 : : return 0 ;  593 : : }   594 : :   595 : : inline void checkPatternCode(uint PatCode)  596 : 1868 : {  597 : : //#ifdef ERROR\_SEED //<<Test\_Id:vsb\_utils\_PatCode\_Set\_To\_0x40>>  598 : : // PatCode=0x40;  599 : : //#endif  600 [ + + ][ - + ]: 1868 : switch(PatCode)  601 : : {  602 : : case 0x01 :  603 : : case 0x02 :  604 : : case 0x03 :  605 : : case 0x04 :  606 : : case 0x0d :  607 : : case 0x0e :  608 : : case 0x09 :  609 : : case 0x0a :  610 : : case 0x0b :  611 : : case 0x0c :  612 : : case 0x0f :  613 : : case 0x10 :  614 : : case 0x05 :  615 : : case 0x06 :  616 : : case 0x07 :  617 : : case 0x08 :  618 : : case 0x11 :  619 : : case 0x31 :  620 : : case 0x32 :  621 : : case 0x33 :  622 : : case 0x34 :  623 : : case 0x35 :  624 : 32 : break;  625 : : default :  626 : 32 : char msg[1024];  627 : 32 : sprintf(msg,"Invalid Pattern code 0x%x\n",PatCode);  628 : :   629 : 32 : ErrorHandler \*pErr = ErrorHandler::instance();  630 : 32 : pErr->errorAdd(NULL, (char\*)\_\_FILE\_\_, (char\*)\_\_FUNCTION\_\_, (int)\_\_LINE\_\_  631 : : ,ErrorHandler::eErrMessageString, pErr->getErrorMsg(ErrorHandler::eErrMessageString,msg));  632 : : return;  633 : : }  634 : : }  635 : :   636 : : } //namespace nft  637 : : #endif /\* \_VSB\_UTILS\_H\_ \*/ |

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