

nmpp

Создано системой Doxygen 1.9.5

1 Алфавитный указатель групп	1
1.1 nmpp	1
2 Иерархический список классов	9
2.1 Иерархия классов	9
3 Алфавитный указатель структур данных	13
3.1 Структуры данных	13
4 Список файлов	17
4.1 Файлы	17
5 Группы	21
5.1 FFTFwdInitAlloc	21
5.1.1 Подробное описание	21
5.2 FFTInvInitAlloc	21
5.2.1 Подробное описание	22
5.3 DFT-8	22
5.3.1 Подробное описание	22
5.3.2 Функции	22
5.3.2.1 nmppsDFT8Fwd_32fcr()	23
5.4 FFT-16	23
5.4.1 Подробное описание	23
5.5 FFT-32	24
5.5.1 Подробное описание	24
5.6 FFT-64	25
5.6.1 Подробное описание	25
5.7 FFT-128	25
5.7.1 Подробное описание	26
5.8 FFT-256	26
5.8.1 Подробное описание	26
5.9 FFT-512	27
5.9.1 Подробное описание	27
5.10 FFT-1024	28
5.10.1 Подробное описание	28
5.11 FFT-2048	28
5.11.1 Подробное описание	29
5.12 FFT-4096	29
5.12.1 Подробное описание	29
5.13 IDFT-8	30
5.13.1 Подробное описание	30
5.14 IFFT-16	31
5.14.1 Подробное описание	31
5.15 IFFT-32	31
5.15.1 Подробное описание	32

5.16 IFFT-64	32
5.16.1 Подробное описание	32
5.17 IFFT-128	33
5.17.1 Подробное описание	33
5.18 IFFT-256	34
5.18.1 Подробное описание	34
5.19 IFFT-512	34
5.19.1 Подробное описание	35
5.20 IFFT-1024	35
5.20.1 Подробное описание	35
5.20.2 Функции	35
5.20.2.1 nmppsFFT1024Inv_32fcr()	36
5.21 IFFT-2048	36
5.21.1 Подробное описание	36
5.22 IFFT-4096	37
5.22.1 Подробное описание	37
5.23 FFT-Сомпон	38
5.23.1 Подробное описание	38
5.24 IFFT-Common	38
5.25 nmppsMalloc	39
5.25.1 Подробное описание	40
5.26 nmppsFree	40
5.26.1 Подробное описание	40
5.27 BLASS-LEVEL1	41
5.27.1 Подробное описание	42
5.27.2 Функции	42
5.27.2.1 nmblas_dasum()	42
5.27.2.2 nmblas_daxpy()	43
5.27.2.3 nmblas_dcopy()	43
5.27.2.4 nmblas_ddot()	44
5.27.2.5 nmblas_dnrm2()	45
5.27.2.6 nmblas_drot()	45
5.27.2.7 nmblas_drotg()	46
5.27.2.8 nmblas_drotm()	46
5.27.2.9 nmblas_dscal()	47
5.27.2.10 nmblas_dsdot()	47
5.27.2.11 nmblas_dswap()	48
5.27.2.12 nmblas_dznrm2()	48
5.27.2.13 nmblas_idamax()	49
5.27.2.14 nmblas_isamax()	49
5.27.2.15 nmblas_sasum()	50
5.27.2.16 nmblas_saxpy()	50
5.27.2.17 nmblas_scnrm2()	51

5.27.2.18 nmblas_scopy()	51
5.27.2.19 nmblas_sdot()	52
5.27.2.20 nmblas_sdsdot()	52
5.27.2.21 nmblas_snrm2()	53
5.27.2.22 nmblas_srot()	53
5.27.2.23 nmblas_srotm()	54
5.27.2.24 nmblas_sscal()	54
5.27.2.25 nmblas_sswap()	55
5.28 BLASS-LEVEL2	55
5.28.1 Подробное описание	55
5.28.2 Функции	55
5.28.2.1 nmblas_dgemv()	56
5.28.2.2 nmblas_sgемv()	57
5.28.2.3 nmblas_sger()	58
5.29 BLASS-LEVEL3	58
5.29.1 Подробное описание	59
5.30 nmppcDivC	59
5.30.1 Подробное описание	59
5.31 nmppcProdC	59
5.31.1 Подробное описание	59
5.32 nmppcFixExp32	60
5.32.1 Подробное описание	60
5.33 nmppcFixSinCos32	60
5.33.1 Подробное описание	60
5.34 nmppcFixArcTan32	61
5.34.1 Подробное описание	61
5.35 nmppcDoubleToFix32	61
5.35.1 Подробное описание	61
5.36 nmppcFix32toDouble	62
5.36.1 Подробное описание	62
5.37 nmppcFixSqrt32	62
5.37.1 Подробное описание	62
5.38 nmppcFixMul32	63
5.38.1 Подробное описание	63
5.39 nmppcFixInv32	64
5.39.1 Подробное описание	64
5.40 nmppcTblFixArcSin32	64
5.40.1 Подробное описание	64
5.41 nmppcTblFixArcCos32	65
5.41.1 Подробное описание	65
5.42 nmppcTblFixCos32	65
5.42.1 Подробное описание	66
5.43 nmppcTblFixSin32	66

5.43.1 Подробное описание	66
5.44 nmppcFixDivMod32	66
5.44.1 Подробное описание	67
5.45 nmppcFixSqrt64	67
5.45.1 Подробное описание	67
5.46 nmppcDoubleToFix64	68
5.46.1 Подробное описание	68
5.47 nmppcFix64ToDouble	68
5.47.1 Подробное описание	68
5.48 nmppcFixDiv64	69
5.48.1 Подробное описание	69
5.49 nmppcFixSinCos64	69
5.49.1 Подробное описание	69
5.50 nmppcFixArcTan64	70
5.50.1 Подробное описание	70
5.51 nmppcFix64Exp01	70
5.51.1 Подробное описание	70
5.52 nmppsRand	71
5.52.1 Подробное описание	71
5.53 nmppcSqrt	71
5.53.1 Подробное описание	72
5.54 Инициализация	72
5.54.1 Подробное описание	72
5.55 Integer operations	72
5.55.1 Подробное описание	72
5.56 Fix-point 64	72
5.56.1 Подробное описание	73
5.57 Fix-point 32	73
5.57.1 Подробное описание	74
5.58 Арифметические операции	74
5.58.1 Подробное описание	74
5.59 Функции деинтерлейсинга	74
5.59.1 Подробное описание	74
5.59.2 Функции	74
5.59.2.1 nmppiDeinterlaceBlend()	74
5.59.2.2 nmppiDeinterlaceSplit()	75
5.60 КИХ-фильтрация	75
5.60.1 Подробное описание	75
5.61 Floodfill	76
5.61.1 Подробное описание	76
5.61.2 Функции	76
5.61.2.1 FloodFill8()	77
5.62 Переупорядочивание изображений	81

5.62.1 Подробное описание	81
5.63 Блочное переупорядочивание	81
5.63.1 Подробное описание	82
5.64 nmpipiSplitIntoBlocks	82
5.64.1 Подробное описание	82
5.65 nmpipiMergeFromBlocks	83
5.65.1 Подробное описание	83
5.66 nmpipiRelease	83
5.66.1 Подробное описание	83
5.67 Арифметические действия	84
5.68 Масочная фильтрация	84
5.68.1 Подробное описание	84
5.69 Инициализация и копирование	84
5.70 Изменение размеров	84
5.71 Операции выборки	84
5.72 Функции поддержки	84
5.72.1 Подробное описание	84
5.73 Инициализация и копирование	84
5.73.1 Подробное описание	84
5.74 nmprrmCopyua	84
5.74.1 Подробное описание	85
5.75 MTR_Copyau	85
5.75.1 Подробное описание	86
5.76 MTR_Copy	87
5.76.1 Подробное описание	87
5.77 MTR_fpResolve_Gauss	88
5.77.1 Подробное описание	88
5.78 MTR_fpResolve_PivotGauss	89
5.78.1 Подробное описание	89
5.79 nmprrmLUDecomp	89
5.79.1 Подробное описание	89
5.80 nmprrmLUResolve	90
5.80.1 Подробное описание	90
5.81 nmprrmMul_mm	90
5.81.1 Подробное описание	91
5.82 nmprrmMul_mv	91
5.82.1 Подробное описание	92
5.83 nmprrmMul_mv_8xH	92
5.83.1 Подробное описание	92
5.84 nmprrmMul_mv__AddC	93
5.84.1 Подробное описание	93
5.85 MTR_ProdUnitV	94
5.85.1 Подробное описание	94

5.86 MTR_Malloc	95
5.87 MTR_Free	95
5.87.1 Подробное описание	95
5.88 MTR_Addr	95
5.88.1 Подробное описание	96
5.89 MTR_SetVal	96
5.89.1 Подробное описание	97
5.90 MTR_GetVal	97
5.90.1 Подробное описание	98
5.91 Функции поддержки	98
5.91.1 Подробное описание	99
5.92 Векторно-матричные операции	99
5.92.1 Подробное описание	99
5.93 Обращение матрицы	99
5.93.1 Подробное описание	99
5.94 FFT-256	99
5.94.1 Подробное описание	100
5.94.2 Функции	100
5.94.2.1 FFT_Fwd256()	100
5.95 IFFT-256	101
5.95.1 Подробное описание	101
5.95.2 Функции	101
5.95.2.1 FFT_Inv256()	101
5.95.2.2 FFT_Inv256Set6bit()	103
5.96 FFT-512	103
5.96.1 Подробное описание	103
5.96.2 Функции	103
5.96.2.1 FFT_Fwd512()	104
5.97 IFFT-512	105
5.97.1 Подробное описание	105
5.97.2 Функции	105
5.97.2.1 FFT_Inv512()	105
5.98 FFT-1024	107
5.98.1 Подробное описание	107
5.98.2 Функции	107
5.98.2.1 FFT_Fwd1024()	107
5.99 IFFT-1024	109
5.99.1 Подробное описание	109
5.99.2 Функции	109
5.99.2.1 FFT_Inv1024()	109
5.100 FFT-2048	111
5.100.1 Подробное описание	111
5.100.2 Функции	111

5.100.2.1 FFT_Fwd2048()	111
5.101 IFFT-2048	112
5.101.1 Подробное описание	112
5.101.2 Функции	112
5.101.2.1 FFT_Inv2048()	112
5.102 FFT-4096	114
5.102.1 Подробное описание	114
5.102.2 Функции	114
5.102.2.1 FFT_Fwd4096()	114
5.103 IFFT-4096	115
5.103.1 Подробное описание	115
5.103.2 Функции	115
5.103.2.1 FFT_Inv4096()	116
5.104 FFT-8192	116
5.104.1 Подробное описание	117
5.104.2 Функции	117
5.104.2.1 FFT_Fwd8192()	117
5.105 IFFT-8192	118
5.105.1 Подробное описание	118
5.105.2 Функции	118
5.105.2.1 FFT_Inv8192()	118
5.106 Свертка	119
5.106.1 Подробное описание	119
5.107 Масочная фильтрация	119
5.107.1 Подробное описание	119
5.108 Изменение размеров	119
5.108.1 Подробное описание	120
5.109 Быстрое преобразование Фурье	120
5.109.1 Подробное описание	120
5.110 Быстрое преобразование Фурье	120
5.110.1 Подробное описание	121
5.110.2 Функции	121
5.110.2.1 nmppsFFTFree_32fcr()	122
5.111 nmppsXCorr_32s	122
5.111.1 Подробное описание	123
5.112 nmppcMedian3_32u	123
5.112.1 Подробное описание	124
5.113 КИХ-фильтрация	124
5.113.1 Подробное описание	124
5.114 nmppsFIR_Xs	124
5.114.1 Подробное описание	125
5.115 nmppsFIRInit_Xs	125
5.115.1 Подробное описание	125

5.116 nmppsFIRInitAlloc_Xs	126
5.116.1 Подробное описание	126
5.117 nmppsFIRGetStateSize_Xs	127
5.117.1 Подробное описание	127
5.118 nmppsFIRFree	127
5.118.1 Подробное описание	127
5.119 nmppsResampleDown2	128
5.119.1 Подробное описание	128
5.120 nmppsResampleUp3Down2	128
5.120.1 Подробное описание	129
5.121 nmppsCreateResample	129
5.121.1 Подробное описание	130
5.122 nmppsSetResample	130
5.122.1 Подробное описание	130
5.123 nmppsResample_perf	131
5.123.1 Подробное описание	131
5.124 Типы векторных данных	132
5.124.1 Подробное описание	134
5.124.2 Типы	134
5.124.2.1 nm1	135
5.124.2.2 nm16s	135
5.124.2.3 nm16s15b	135
5.124.2.4 nm16u	135
5.124.2.5 nm16u15b	136
5.124.2.6 nm2s	136
5.124.2.7 nm2u	136
5.124.2.8 nm32s	136
5.124.2.9 nm32s30b	137
5.124.2.10 nm32s31b	137
5.124.2.11 nm32u	137
5.124.2.12 nm32u31b	137
5.124.2.13 nm4s	138
5.124.2.14 nm4u	138
5.124.2.15 nm4u3b	138
5.124.2.16 nm64s	138
5.124.2.17 nm64s63b	139
5.124.2.18 nm64u	139
5.124.2.19 nm8s	139
5.124.2.20 nm8s7b	139
5.124.2.21 nm8u	140
5.124.2.22 nm8u7b	140
5.124.2.23 v16nm16s	140
5.124.2.24 v16nm16u	140

5.124.2.25 v16nm32s	140
5.124.2.26 v16nm32u	141
5.124.2.27 v16nm4b3u	141
5.124.2.28 v16nm4u	141
5.124.2.29 v16nm8s	141
5.124.2.30 v16nm8s7b	141
5.124.2.31 v16nm8u	142
5.124.2.32 v2nm32s	142
5.124.2.33 v2nm32u	142
5.124.2.34 v4nm16s	142
5.124.2.35 v4nm16u	142
5.124.2.36 v4nm32s	143
5.124.2.37 v4nm32u	143
5.124.2.38 v4nm8u	143
5.124.2.39 v8nm16s	143
5.124.2.40 v8nm16u	143
5.124.2.41 v8nm32s	144
5.124.2.42 v8nm32u	144
5.124.2.43 v8nm8s	144
5.124.2.44 v8nm8u	144
5.125 Типы скалярных данных	145
5.125.1 Подробное описание	145
5.125.2 Типы	145
5.125.2.1 int15b	146
5.125.2.2 int16b	146
5.125.2.3 int1b	146
5.125.2.4 int2b	146
5.125.2.5 int30b	146
5.125.2.6 int31b	147
5.125.2.7 int32b	147
5.125.2.8 int3b	147
5.125.2.9 int4b	147
5.125.2.10 int63b	147
5.125.2.11 int64b	148
5.125.2.12 int7b	148
5.125.2.13 int8b	148
5.125.2.14 uint15b	148
5.125.2.15 uint16b	148
5.125.2.16 uint1b	149
5.125.2.17 uint2b	149
5.125.2.18 uint31b	149
5.125.2.19 uint32b	149
5.125.2.20 uint3b	149

5.125.2.21 uint4b	150
5.125.2.22 uint63b	150
5.125.2.23 uint64b	150
5.125.2.24 uint7b	150
5.125.2.25 uint8b	150
5.126 Функции поддержки	151
5.126.1 Подробное описание	151
5.127 Инициализация и копирование	151
5.127.1 Подробное описание	152
5.128 Инициализация и копирование	152
5.128.1 Подробное описание	152
5.129 Арифметические операции	152
5.129.1 Подробное описание	153
5.130 Арифметические операции	153
5.130.1 Подробное описание	154
5.131 Логические и бинарные операции	154
5.131.1 Подробное описание	155
5.132 Операции сравнения	155
5.132.1 Подробное описание	157
5.133 Операции сравнения	157
5.133.1 Подробное описание	157
5.134 Статистические функции	157
5.135 Переупорядочивание и сортировка	157
5.135.1 Подробное описание	158
5.136 Элементарные математические функции	158
5.136.1 Подробное описание	158
5.137 Тригонометрические функции	158
5.137.1 Подробное описание	158
5.138 Функция деления векторов	159
5.138.1 Подробное описание	159
5.139 Экспонента	159
5.139.1 Подробное описание	160
5.140 Натуральный логарифм	160
5.140.1 Подробное описание	160
5.141 Возведение в степень	161
5.141.1 Подробное описание	161
5.142 Вычисление квадратного корня	161
5.142.1 Подробное описание	162
5.143 nmppsAbs	162
5.143.1 Подробное описание	162
5.144 nmppsAbs1	163
5.144.1 Подробное описание	163
5.145 nmppsNeg	164

5.145.1 Подробное описание	164
5.146 nmppsAddC	165
5.146.1 Подробное описание	165
5.147 nmppsAddC_p	166
5.147.1 Подробное описание	166
5.148 nmppsAddC	166
5.148.1 Подробное описание	167
5.149 nmppsAdd	168
5.149.1 Подробное описание	168
5.150 nmppsAdd	169
5.150.1 Подробное описание	169
5.151 nmppsAdd_AddC	170
5.151.1 Подробное описание	170
5.152 nmppsSubC	170
5.152.1 Подробное описание	171
5.153 nmppsSubC	171
5.153.1 Подробное описание	172
5.154 nmppsSubCRev	173
5.154.1 Подробное описание	173
5.155 nmppsSubCRev	174
5.155.1 Подробное описание	174
5.156 nmppsSub	175
5.156.1 Подробное описание	175
5.157 nmppsSub	176
5.157.1 Подробное описание	176
5.158 nmppsAbsDiff	176
5.158.1 Подробное описание	177
5.159 nmppsAbsDiff_f	177
5.159.1 Подробное описание	177
5.160 nmppsAbsDiff1	178
5.160.1 Подробное описание	178
5.161 nmppsMulC	179
5.161.1 Подробное описание	179
5.162 nmppsMulC	180
5.162.1 Подробное описание	180
5.163 nmppsMul_Mul_Add	181
5.163.1 Подробное описание	181
5.164 nmppsMul_Add	181
5.164.1 Подробное описание	182
5.165 nmppsMul_ConjMul_Add	182
5.165.1 Подробное описание	182
5.166 nmppsMulC_AddC	183
5.166.1 Подробное описание	183

5.167 nmppsMulC_AddV_AddC	184
5.167.1 Подробное описание	184
5.168 nmppsMul	184
5.168.1 Подробное описание	185
5.169 nmppsConjMul	185
5.169.1 Подробное описание	185
5.170 nmppsMul_Mul_Sub	186
5.170.1 Подробное описание	186
5.171 nmppsMulC_Add	187
5.171.1 Подробное описание	187
5.172 nmppsMul_AddC	187
5.172.1 Подробное описание	187
5.173 nmppsMul_AddC	188
5.173.1 Подробное описание	188
5.174 nmppsMulC_AddC	189
5.174.1 Подробное описание	189
5.174.2 Функции	190
5.174.2.1 nmppsMulC_AddC_2x32s()	190
5.175 nmppsRShiftC_MulC_AddC	191
5.175.1 Подробное описание	191
5.176 nmppsMulC_AddV_AddC	191
5.176.1 Подробное описание	192
5.177 nmppsSumN	192
5.177.1 Подробное описание	192
5.178 nmppsDivC	193
5.178.1 Подробное описание	193
5.179 nmppsSum	194
5.179.1 Подробное описание	194
5.180 nmppsSum_1	195
5.180.1 Подробное описание	195
5.181 nmppsDotProd_sm	195
5.181.1 Подробное описание	196
5.182 nmppsDotProd	196
5.182.1 Подробное описание	197
5.183 nmppsWeightedSum	197
5.183.1 Подробное описание	198
5.184 nmppsNot_	198
5.184.1 Подробное описание	198
5.185 nmppsAndC	199
5.185.1 Подробное описание	199
5.186 nmppsAnd	200
5.186.1 Подробное описание	200
5.187 nmppsAnd4V_	201

5.187.1 Подробное описание	201
5.188 nmppsAndNotV_	202
5.188.1 Подробное описание	202
5.189 nmppsOrC	202
5.189.1 Подробное описание	203
5.190 nmppsOr	203
5.190.1 Подробное описание	204
5.191 nmppsOr3V_	204
5.191.1 Подробное описание	204
5.192 nmppsOr4V_	205
5.192.1 Подробное описание	205
5.193 nmppsXorC	206
5.193.1 Подробное описание	206
5.194 nmppsXor	207
5.194.1 Подробное описание	207
5.195 nmppsMaskV_	208
5.195.1 Подробное описание	208
5.196 nmppsRShiftC	208
5.196.1 Подробное описание	209
5.197 nmppsRShiftC_	209
5.197.1 Подробное описание	210
5.198 nmppsRShiftC_AddC_	210
5.199 nmppsDisplaceBits	211
5.199.1 Подробное описание	211
5.200 nmppsSet-инициализация	212
5.200.1 Подробное описание	212
5.201 nmppsRandUniform	213
5.201.1 Подробное описание	213
5.201.2 Функции	214
5.201.2.1 nmppsRandUniform_64s()	214
5.202 nmppsRandUniform	214
5.202.1 Подробное описание	214
5.203 nmppsRandUniform_	215
5.203.1 Подробное описание	215
5.204 nmppsRamp_	215
5.204.1 Подробное описание	216
5.205 nmppsConvert	216
5.205.1 Подробное описание	217
5.205.2 Функции	217
5.205.2.1 nmppsConvert_1s2s()	218
5.205.2.2 nmppsConvert_1u2u()	218
5.206 nmppsConvert	218
5.206.1 Подробное описание	218

5.207 nmppsConvert_rounding	219
5.207.1 Подробное описание	219
5.208 nmppsMerge	220
5.208.1 Подробное описание	220
5.209 nmppsConvertRisc	220
5.209.1 Подробное описание	220
5.210 nmppsCopy_	221
5.210.1 Подробное описание	221
5.211 nmppsCopy	222
5.211.1 Подробное описание	222
5.212 nmppsCopyOddToOdd_32f	222
5.212.1 Подробное описание	222
5.213 nmppsCopyEvenToOdd_32f	223
5.213.1 Подробное описание	223
5.214 nmppsCopyOddToEven_32f	223
5.214.1 Подробное описание	223
5.215 nmppsCopyEvenToEven_32f	223
5.215.1 Подробное описание	224
5.216 nmppsCopyRisc	224
5.216.1 Подробное описание	224
5.217 nmppsCopyuca_	224
5.217.1 Подробное описание	225
5.218 nmppsSwap_	225
5.218.1 Подробное описание	225
5.219 nmppsMax_	226
5.219.1 Подробное описание	226
5.219.2 Функции	227
5.219.2.1 nmppsMax_16s15b()	227
5.219.2.2 nmppsMax_32s31b()	227
5.219.2.3 nmppsMax_8s7b()	228
5.220 nmppsMin	228
5.220.1 Подробное описание	228
5.220.2 Функции	229
5.220.2.1 nmppsMin_16s15b()	229
5.220.2.2 nmppsMin_32s31b()	229
5.220.2.3 nmppsMin_8s7b()	230
5.221 nmppsMaxIdx_	230
5.221.1 Подробное описание	230
5.222 nmppsMinIdx_	231
5.222.1 Подробное описание	231
5.223 nmppsMinIdxVN_	232
5.223.1 Подробное описание	233
5.224 nmppsFirstZeroIdx	234

5.224.1 Подробное описание	234
5.225 nmppsFirstNonZeroIdx	235
5.225.1 Подробное описание	235
5.226 nmppsLastZeroIdx	235
5.226.1 Подробное описание	236
5.227 nmppsLastNonZeroIdx	236
5.227.1 Подробное описание	236
5.228 nmppsMinEvery_	237
5.228.1 Подробное описание	237
5.229 nmppsMaxEvery_	238
5.229.1 Подробное описание	238
5.230 nmppsMinCmpLtV_	239
5.230.1 Подробное описание	239
5.231 nmppsCmpLt0	240
5.231.1 Подробное описание	240
5.232 nmppsCmpLteC	241
5.232.1 Подробное описание	241
5.233 nmppsCmpLtC	242
5.233.1 Подробное описание	242
5.234 nmppsCmpEq0(Reduced)	242
5.234.1 Подробное описание	243
5.234.2 Функции	243
5.234.2.1 nmppsCmpEq0_16u15b()	243
5.234.2.2 nmppsCmpEq0_32u31b()	244
5.234.2.3 nmppsCmpEq0_8u7b()	244
5.235 nmppsCmpGt0	244
5.235.1 Подробное описание	244
5.236 nmppsMinMaxEvery_	245
5.236.1 Подробное описание	245
5.237 nmppsClipPowC_	246
5.237.1 Подробное описание	246
5.238 nmppsClipCC_	247
5.238.1 Подробное описание	247
5.239 nmppsThreshold_Lt_Gt_f	248
5.239.1 Подробное описание	248
5.240 nmppsClipRShiftConvert_AddC_	249
5.240.1 Подробное описание	249
5.241 nmppsClipConvert_AddC_	250
5.241.1 Подробное описание	250
5.242 nmppsCmpEqC	251
5.242.1 Подробное описание	251
5.243 nmppsCmpNe0	252
5.243.1 Подробное описание	252

5.244 nmppsCmpEq0	253
5.244.1 Подробное описание	253
5.245 nmppsCmpNeC	254
5.245.1 Подробное описание	254
5.246 nmppsCmpNeC_flag	254
5.246.1 Подробное описание	255
5.247 nmppsCmpLtC	255
5.247.1 Подробное описание	256
5.248 nmppsCmpGtC	256
5.248.1 Подробное описание	257
5.249 nmppsCmpGteC	258
5.249.1 Подробное описание	258
5.250 nmppsCmpEqV_	258
5.250.1 Подробное описание	259
5.251 nmppsCmpNe	259
5.251.1 Подробное описание	260
5.252 nmppsCmpLt	260
5.252.1 Подробное описание	260
5.253 nmppsCmpNeV_	261
5.253.1 Подробное описание	261
5.254 Vec_ClipRShiftConvert_AddC	262
5.254.1 Подробное описание	262
5.255 nmppsAddr_	263
5.255.1 Подробное описание	263
5.256 nmppsSetVal_	264
5.256.1 Подробное описание	265
5.257 nmppsGetVal_	265
5.257.1 Подробное описание	266
5.258 nmppsGetVal_(return)	267
5.258.1 Подробное описание	267
5.259 VEC_QSort	268
5.259.1 Подробное описание	268
5.260 nmppsRemap_	268
5.260.1 Подробное описание	269
5.261 nmppSplitTmp	269
5.261.1 Подробное описание	270
5.262 nmppSplit	270
5.262.1 Подробное описание	270
5.263 nmppMerge	271
5.263.1 Подробное описание	271
5.264 nmppSplit_32fcr	271
5.264.1 Подробное описание	272
5.265 nmppsDecimate	273

5.265.1 Подробное описание	273
5.266 Целочисленные функции	274
5.266.1 Подробное описание	274
5.267 Функции с плавающей точкой	274
5.267.1 Подробное описание	274
5.268 Типы данных	274
5.268.1 Подробное описание	274
5.269 Векторные функции	274
5.269.1 Подробное описание	275
5.270 Векторные функции	275
5.270.1 Подробное описание	275
5.271 Матричные функции	275
5.271.1 Подробное описание	275
5.272 Матричные функции	275
5.272.1 Подробное описание	275
5.273 Функции обработки сигналов	275
5.273.1 Подробное описание	275
5.274 Функции обработки сигналов	275
5.274.1 Подробное описание	276
5.275 Функции обработки изображений	276
5.275.1 Подробное описание	276
5.276 Скалярные функции	276
5.276.1 Подробное описание	276
5.276.1.1 Введение	276
5.277 Базовые регистровые функции библиотеки	278
5.277.1 Подробное описание	278
5.278 implement	278
5.278.1 Подробное описание	278
5.279 Базовые регистровые функции библиотеки	278
5.279.1 Подробное описание	278
5.280 nblas	278
5.280.1 Подробное описание	278
5.281 контроль переполнения	278
5.281.1 Подробное описание	279
5.281.2 Макросы	279
5.281.2.1 GetVec	279
5.281.3 Функции	279
5.281.3.1 operator<<()	280
5.282 Элементарные функции	280
5.282.1 Подробное описание	281
5.282.2 Функции	281
5.282.2.1 vec_Mask()	281
5.283 Элементарные функции	282

5.283.1 Подробное описание	282
5.284 функции взвешенного суммирования	282
5.284.1 Подробное описание	282
5.285 Целевые функции	282
5.285.1 Подробное описание	283
5.286 Vec_0_sub_data	283
5.286.1 Подробное описание	284
5.287 Vec_activate_data	284
5.287.1 Подробное описание	284
5.288 Vec_activate_data_add_0	285
5.288.1 Подробное описание	285
5.289 Vec_activate_data_xor_data	286
5.289.1 Подробное описание	286
5.290 Vec_activate_data_add_ram	287
5.290.1 Подробное описание	287
5.291 	287
5.291.1 Подробное описание	288
5.292 Vec_afifo	288
5.292.1 Подробное описание	289
5.293 Vec_data	289
5.293.1 Подробное описание	289
5.294 Vec_copyEvenToEven_32f	290
5.294.1 Подробное описание	290
5.295 Vec_CopyOddToEven_32f	291
5.295.1 Подробное описание	291
5.296 Vec_data_add_afifo	291
5.296.1 Подробное описание	291
5.297 Vec_data_add_ram	292
5.297.1 Подробное описание	292
5.298 Vec_data_and_ram	293
5.298.1 Подробное описание	293
5.299 Vec_data_or_ram	293
5.299.1 Подробное описание	294
5.300 Vec_data_sub_ram	294
5.300.1 Подробное описание	294
5.301 Vec_data_xor_ram	295
5.301.1 Подробное описание	295
5.302 Vec_FilterCoreRow2	296
5.302.1 Подробное описание	296
5.303 Vec_FilterCoreRow4	296
5.303.1 Подробное описание	296
5.304 Vec_FilterCoreRow8	296
5.304.1 Подробное описание	296

5.305 Vec_And	297
5.305.1 Подробное описание	297
5.306 Vec_Or	297
5.306.1 Подробное описание	297
5.307 Vec_Xor	298
5.307.1 Подробное описание	298
5.308 Vec_Abs	299
5.308.1 Подробное описание	299
5.309 Vec_Add	300
5.309.1 Подробное описание	300
5.310 Vec_ClipExt	301
5.310.1 Подробное описание	301
5.311 Vec_ClipMul2D2W8_AddVr	302
5.311.1 Подробное описание	302
5.312 Vec_ClipMulNDNW2_AddVr	303
5.312.1 Подробное описание	303
5.313 Vec_ClipMulNDNW4_AddVr	304
5.313.1 Подробное описание	304
5.314 Vec_ClipMulNDNW8_AddVr	305
5.314.1 Подробное описание	305
5.315 Vec_IncNeg	306
5.315.1 Подробное описание	306
5.316 Vec_Mul2D2W1_AddVr	307
5.316.1 Подробное описание	307
5.317 Vec_Mul2D2W2_AddVr	308
5.317.1 Подробное описание	308
5.318 Vec_Mul2D2W4_AddVr	309
5.318.1 Подробное описание	309
5.319 Vec_Mul2D2W8_AddVr	310
5.319.1 Подробное описание	310
5.320 Vec_Mul3D3W2_AddVr	311
5.320.1 Подробное описание	311
5.321 Vec_Mul3D3W8_AddVr	312
5.321.1 Подробное описание	312
5.322 Vec_Mul4D4W2_AddVr	314
5.322.1 Подробное описание	314
5.323 Vec_MulVN_AddVN	315
5.323.1 Подробное описание	316
5.324 Vec_Sub	316
5.324.1 Подробное описание	317
5.325 Vec_SubAbs	317
5.325.1 Подробное описание	317
5.326 Vec_SubVN_Abs	318

5.326.1 Подробное описание	319
5.327 Vec_Swap	319
5.327.1 Подробное описание	320
5.328 Vec_MUL_2V4toW8_shift	320
5.328.1 Подробное описание	321
5.329 Vec_MUL_2V8toW16_shift	321
5.329.1 Подробное описание	322
5.330 Vec_not_data	322
5.330.1 Подробное описание	323
5.331 Vec_ram	323
5.331.1 Подробное описание	323
5.332 Vec_ram_sub_data	324
5.332.1 Подробное описание	324
5.333 Vec_vsum_activate_data_0	325
5.333.1 Подробное описание	325
5.334 Vec_vsum_data_0	326
5.334.1 Подробное описание	326
5.335 Vec_vsum_data_afifo	326
5.335.1 Подробное описание	327
5.336 Vec_vsum_data_vr	327
5.336.1 Подробное описание	327
5.337 Vec_vsum_shift_data_0	328
5.337.1 Подробное описание	328
5.338 Vec_vsum_shift_data_vr	329
5.338.1 Подробное описание	329
5.339 Vec_vsum_shift_data_afifo	330
5.339.1 Подробное описание	330
5.340 Vec_CompareMinV	331
5.340.1 Подробное описание	331
5.341 Vec_CompareMaxV	331
5.341.1 Подробное описание	332
5.342 Vec_DupValueInVector8	332
5.342.1 Подробное описание	333
5.343 Vec_DupValueInVector16	333
5.343.1 Подробное описание	333
5.344 Vec_BuildDiagWeights8	334
5.344.1 Подробное описание	334
5.345 Vec_BuildDiagWeights16	334
5.345.1 Подробное описание	334
5.346 Vec_MaxVal_v8nm8s	335
5.346.1 Подробное описание	335
5.347 Vec_MaxVal_v4nm16s	335
5.347.1 Подробное описание	336

5.348 Vec_MaxVal	336
5.348.1 Подробное описание	336
5.349 Vec_MinVal_v8nm8s	337
5.349.1 Подробное описание	337
5.350 Vec_MinVal_v4nm16s	338
5.350.1 Подробное описание	338
5.351 Vec_MinVal	338
5.351.1 Подробное описание	339
5.352 Vec_AccMul1D1W32_AddVr	340
5.352.1 Подробное описание	340
6 Структуры данных	343
6.1 Класс C_2DSubPixelMinPosition	343
6.1.1 Методы	343
6.1.1.1 Find()	343
6.1.1.2 Release()	344
6.2 Класс C_2DTrigSubPixelMinPosition	344
6.2.1 Методы	344
6.2.1.1 Find()	344
6.2.1.2 Release()	345
6.3 Класс C_Allocator32	345
6.4 Шаблон класса C_BoxImg< T >	345
6.5 Шаблон класса C_BoxVec< T >	346
6.6 Класс C_Heap	346
6.6.1 Подробное описание	347
6.6.2 Методы	348
6.6.2.1 AllocateMaxAvail()	348
6.7 Шаблон класса C_Img< T >	348
6.7.1 Подробное описание	348
6.8 Класс C_MultiHeap	349
6.9 Класс C_PlessyCornerDetector	350
6.9.1 Методы	351
6.9.1.1 Allocate()	351
6.9.1.2 DeAllocate()	351
6.9.1.3 FindCorners()	351
6.9.1.4 Release()	351
6.9.1.5 SetThreshold()	352
6.10 Класс C_PlessyCornerDetector_16s	352
6.10.1 Методы	353
6.10.1.1 Allocate()	353
6.10.1.2 CountPlessy()	353
6.10.1.3 DeAllocate()	353
6.10.1.4 SetThreshold()	353

6.11 Класс C_PlessyCornerDetector_32f	353
6.11.1 Методы	354
6.11.1.1 Allocate()	354
6.11.1.2 CountDer()	355
6.11.1.3 CountPlessy()	355
6.11.1.4 DeAllocate()	355
6.11.1.5 SetThreshold()	355
6.12 Шаблон класса C_RingBuffer< T >	356
6.12.1 Методы	356
6.12.1.1 PushRequest()	357
6.13 Шаблон класса C_RingBufferRemote< T >	357
6.14 Класс Cplx_float	358
6.15 Структура ds_struct	358
6.16 Класс EnterHardMode	359
6.17 Класс I_2DSubPixelMinPosition	359
6.18 Класс I_PlessyCornerDetector	359
6.19 Структура int15in16x4	360
6.20 Структура int30in32x2	360
6.21 Структура int31in32x2	360
6.22 Шаблон класса mtr< T >	360
6.23 Структура nm16sc	362
6.24 Класс nmchar	362
6.25 Шаблон класса nmchar1D< N >	363
6.26 Шаблон класса nmchar2D< Y, X >	363
6.27 Шаблон класса nmintpack< T >	363
6.27.1 Подробное описание	364
6.28 Шаблон класса nmmtr< T >	364
6.28.1 Подробное описание	365
6.29 Шаблон класса nmmtrpack< T >	365
6.29.1 Подробное описание	367
6.30 Структура NmppsFFTSpec	367
6.31 Структура NmppsFFTSpec_32fc	367
6.32 Структура NmppsFrame_16s	368
6.33 Структура NmppsFrame_16u	368
6.34 Структура NmppsFrame_32s	368
6.35 Структура NmppsFrame_32u	368
6.36 Структура NmppsFrame_64s	369
6.37 Структура NmppsFrame_64u	369
6.38 Структура NmppsFrame_8s	369
6.39 Структура NmppsFrame_8u	369
6.40 Структура NmppsMallocSpec	370
6.41 Структура NmppsTmpSpec	370
6.42 Структура nmreg	370

6.42.1 Подробное описание	371
6.43 Класс nmshort	371
6.44 Шаблон класса nmshort2D< Y, X >	371
6.45 Шаблон класса nmvecspace< T >	372
6.45.1 Подробное описание	373
6.46 Структура RGB24_nm8u	373
6.46.1 Подробное описание	373
6.47 Структура RGB32_nm10s	374
6.48 Структура RGB32_nm10u	374
6.49 Структура RGB32_nm8s	374
6.50 Структура RGB32_nm8u	374
6.51 Структура RoundShift	375
6.52 Класс RPoint	375
6.52.1 Подробное описание	375
6.53 Структура S_BufferInfo	375
6.53.1 Подробное описание	376
6.54 Структура s_int32x2	376
6.55 Структура s_nm32fc	376
6.56 Структура s_nm32fcr	377
6.57 Структура s_nm32sc	377
6.58 Структура s_nm64sc	377
6.58.1 Поля	377
6.58.1.1 im	377
6.58.1.2 re	377
6.59 Структура S_nmppiFilterKernel	378
6.60 Структура S_nmppiFilterKernel_32s32s	378
6.61 Структура s_v16nm16s	378
6.61.1 Подробное описание	378
6.62 Структура s_v16nm16u	379
6.62.1 Подробное описание	379
6.63 Структура s_v16nm32s	379
6.63.1 Подробное описание	379
6.64 Структура s_v16nm32u	379
6.64.1 Подробное описание	380
6.65 Структура s_v16nm4u	380
6.65.1 Подробное описание	380
6.66 Структура s_v16nm8s	380
6.66.1 Подробное описание	380
6.67 Структура s_v16nm8u	381
6.67.1 Подробное описание	381
6.68 Структура s_v2nm32f	381
6.69 Структура s_v2nm32s	381
6.69.1 Подробное описание	381

6.70 Структура s_v2nm32u	382
6.70.1 Подробное описание	382
6.71 Структура s_v4nm16s	382
6.71.1 Подробное описание	382
6.72 Структура s_v4nm16u	382
6.72.1 Подробное описание	383
6.73 Структура s_v4nm32f	383
6.74 Структура s_v4nm32s	383
6.74.1 Подробное описание	383
6.75 Структура s_v4nm32u	383
6.75.1 Подробное описание	384
6.76 Структура s_v4nm64f	384
6.77 Структура s_v4nm8u	384
6.77.1 Подробное описание	384
6.78 Структура s_v8nm16s	385
6.78.1 Подробное описание	385
6.79 Структура s_v8nm16u	385
6.79.1 Подробное описание	385
6.80 Структура s_v8nm32s	385
6.80.1 Подробное описание	386
6.81 Структура s_v8nm32u	386
6.81.1 Подробное описание	386
6.82 Структура s_v8nm8s	386
6.82.1 Подробное описание	386
6.83 Структура s_v8nm8u	387
6.83.1 Подробное описание	387
6.84 Структура SpecTmp1	387
6.85 Структура spot_struct	387
6.86 Структура tagSegmentInfo	388
6.87 Шаблон класса tcube< T >	388
6.87.1 Подробное описание	389
6.88 Шаблон класса tfixpoint< T, point >	389
6.89 Структура Tmp2BuffSpec	390
6.90 Шаблон класса tmtr< T >	390
6.91 Класс uint16ptr	391
6.92 Класс uint8ptr	391
6.93 Структура v16nm4s	392
6.93.1 Подробное описание	392
6.94 Структура v4nm8s	392
6.94.1 Подробное описание	393
6.95 Шаблон класса vec< T >	393
6.95.1 Подробное описание	394
6.96 Класс Wi_4096_fixed	395

7 Файлы	397
7.1 crtdbg2.h	397
7.2 fft.h	397
7.3 fft.h	397
7.4 fft_32fc.r.h	401
7.5 fftexp.h	402
7.6 macros_fpu.h	404
7.7 malloc32.h	405
7.8 metric.h	408
7.9 minrep.h	408
7.10 multiheap.h	408
7.11 Файл g:/nmpp/include/nmblas.h	418
7.11.1 Подробное описание	422
7.12 nmblas.h	422
7.13 Файл g:/nmpp/include/nmblas/nmblas_sgemm.h	425
7.13.1 Подробное описание	425
7.14 nmblas_sgemm.h	427
7.15 nmchar.h	429
7.16 nmdef.h	435
7.17 cArithmetric.h	436
7.18 cfixpnt32.h	436
7.19 cfixpnt64.h	437
7.20 cInit.h	438
7.21 cInteger.h	439
7.22 nmplc.h	439
7.23 nmplc.h	439
7.24 filter.h	439
7.25 iArithmetics.h	440
7.26 iCellTexture.h	440
7.27 iConvert.h	441
7.28 iDef.h	444
7.29 iDeinterlace.h	445
7.30 iFilter.h	445
7.31 iFiltration.h	447
7.32 iFloodFill.h	448
7.33 iPlessy.h	449
7.34 iPlessyDetector.h	449
7.35 iPrint.h	450
7.36 iReodering.h	451
7.37 iResample.h	451
7.38 iSelect.h	452
7.39 isubpixel2d.h	452
7.40 isubpixel2dimpl.h	453

7.41 iSupport.h	453
7.42 nmpli.h	455
7.43 nmpli.h	455
7.44 warpimg.h	456
7.45 mInit.h	457
7.46 mInverse.h	458
7.47 mMatrixVector.h	459
7.48 mMatrixVectorDev.h	460
7.49 mStat.h	461
7.50 mSupport.h	462
7.51 nmplm.h	464
7.52 nmplm.h	464
7.53 fft_old.h	464
7.54 fftext.h	469
7.55 fftref.h	473
7.56 nmpls.h	473
7.57 nmpls.h	473
7.58 sConvolution.h	474
7.59 sCorrelation.h	474
7.60 sFiltration.h	474
7.61 sfir.h	475
7.62 sResample.h	476
7.63 nmplv.h	477
7.64 nmplv.h	477
7.65 nmtl.h	478
7.66 nmtl.h	478
7.67 nmtl.h	478
7.68 sElementary.h	478
7.69 vArithmetics.h	478
7.70 vArithmeticsDev.h	482
7.71 vBitwise.h	482
7.72 vInit.h	484
7.73 vSelect.h	486
7.74 vStat.h	490
7.75 vSupport.h	491
7.76 vSupport_.h	493
7.77 vTransform.h	495
7.78 nmpp-cpp.h	496
7.79 nmpp.h	496
7.80 nmshort.h	497
7.81 nmtlio.h	501
7.82 tcmplx.h	505
7.83 tcmplx_spec.h	508

7.84 tfixpoint.h	509
7.85 tfixpoint.h	513
7.86 tfixpointmath.h	513
7.87 tmatrix.h	514
7.88 tmatrix_spec.h	523
7.89 tnmcube.h	524
7.90 tnmcvec.h	526
7.91 tnmint.h	529
7.92 tnmmtr.h	533
7.93 tnmmtrpack.h	540
7.94 tnmvec.h	546
7.95 tnmvecpack.h	551
7.96 tvector.h	556
7.97 Файл g:/nmpp/include/nmtype.h	563
7.97.1 Подробное описание	566
7.97.2 Макросы	566
7.97.2.1 NM16Sx4	566
7.97.2.2 NM32Sx2	566
7.97.2.3 VEC_NM16S	567
7.97.2.4 VEC_NM16U	567
7.97.2.5 VEC_NM32S	567
7.97.2.6 VEC_NM4U	567
7.97.2.7 VEC_NM8S	567
7.98 nmtype.h	568
7.99 vCore.h	576
7.100 ownmalloc.h	580
7.101 ringbuffer_.h	580
7.102 ringremote.h	584
7.103 rpc-host.h	589
7.104 rpc-nmc-func.h	591
7.105 rpc-nmc.h	598
7.106 test.h	601
Предметный указатель	603

Глава 1

Алфавитный указатель групп

1.1 nmpp

Полный список групп.

Целочисленные функции	274
Векторные функции	274
Функции поддержки	151
nmppsMalloc	39
nmppsFree	40
nmppsAddr	263
nmppsSetVal_	264
nmppsGetVal_	265
nmppsGetVal_(return)	267
Инициализация и копирование	151
nmppsSet-инициализация	212
nmppsRandUniform	213
nmppsRandUniform_	215
nmppsRamp_	215
nmppsConvert	216
nmppsCopy_	221
nmppsCopyua_	224
nmppsSwap_	225
Vec_ClipRShiftConvert_AddC	262
Арифметические операции	152
nmppsAbs	162
nmppsAbs1	163
nmppsNeg	164
nmppsAddC	165
nmppsAddC_p	166
nmppsAdd	168
nmppsAdd_AddC	170
nmppsSubC	170
nmppsSubCRev	173
nmppsSub	175
nmppsAbsDiff	176
nmppsAbsDiff_f	177
nmppsAbsDiff1	178
nmppsMulC	179
nmppsMul_AddC	187

nmppsMulC_AddC	189
nmppsRShiftC_MulC_AddC	191
nmppsMulC_AddV_AddC	191
nmppsSumN	192
nmppsDivC	193
nmppsSum	194
nmppsSum_1	195
nmppsDotProd_sm	195
nmppsDotProd	196
nmppsWeightedSum	197
Логические и бинарные операции	154
nmppsNot	198
nmppsAndC	199
nmppsAnd	200
nmppsAnd4V	201
nmppsAndNotV	202
nmppsOrC	202
nmppsOr	203
nmppsOr3V	204
nmppsOr4V	205
nmppsXorC	206
nmppsXor	207
nmppsMaskV	208
nmppsRShiftC	208
nmppsRShiftC_	209
nmppsRShiftC_AddC	210
nmppsDisplaceBits	211
Операции сравнения	155
nmppsMax	226
nmppsMin	228
nmppsMaxIdx	230
nmppsMinIdx	231
nmppsMinIdxVN	232
nmppsFirstZeroIndx	234
nmppsFirstNonZeroIndx	235
nmppsLastZeroIndx	235
nmppsLastNonZeroIndx	236
nmppsMinEvery	237
nmppsMaxEvery	238
nmppsMinCmpLtV	239
nmppsCmpLt0	240
nmppsCmpEq0(Reduced)	242
nmppsCmpGt0	244
nmppsMinMaxEvery	245
nmppsClipPowC	246
nmppsClipCC	247
nmppsThreshold_Lt_Gt_f	248
nmppsClipRShiftConvert_AddC	249
nmppsClipConvert_AddC	250
nmppsCmpEqC	251
nmppsCmpNe0	252
nmppsCmpEq0	253
nmppsCmpNeC	254
nmppsCmpNeC_flag	254
nmppsCmpLtc	255
nmppsCmpGtc	256
nmppsCmpEqV	258
nmppsCmpNe	259

nmppsCmpLt	260
nmppsCmpNeV	261
Статистические функции	157
Переупорядочивание и сортировка	157
VEC_QSort	268
nmppsRemap	268
nmppSplitTmp	269
nmppSplit	270
nmppMerge	271
nmppSplit_32fc	271
nmppsDecimate	273
Матричные функции	275
Инициализация и копирование	84
nmppmCopyu	84
MTR_Copyau	85
MTR_Copy	87
Функции поддержки	98
MTR_Malloc	95
MTR_Free	95
MTR_Addr	95
MTR_SetVal	96
MTR_GetVal	97
Векторно-матричные операции	99
nmppmMul_mm	90
nmppmMul_mv	91
nmppmMul_mv_8xH	92
nmppmMul_mv__AddC	93
MTR_ProdUnitV	94
Функции обработки сигналов	275
Свертка	119
nmppsXCorr_32s	122
Масочная фильтрация	119
nmppcMedian3_32u	123
КИХ-фильтрация	124
nmppsFIR_Xs	124
nmppsFIRInit_Xs	125
nmppsFIRInitAlloc_Xs	126
nmppsFIRGetStateSize_Xs	127
nmppsFIRFree	127
Изменение размеров	119
nmppsResampleDown2	128
nmppsResampleUp3Down2	128
nmppsCreateResample	129
nmppsSetResample	130
nmppsResample_perf	131
Быстрое преобразование Фурье	120
FFT-256	99
IFFT-256	101
FFT-512	103
IFFT-512	105
FFT-1024	107
IFFT-1024	109
FFT-2048	111
IFFT-2048	112
FFT-4096	114
IFFT-4096	115
FFT-8192	116

IFFT-8192	118
Функции обработки изображений	276
Floodfill	76
Переупорядочивание изображений	81
Блочное переупорядочивание	81
nmppiSplitIntoBlocks	82
nmppiMergeFromBlocks	83
Арифметические действия	84
Масочная фильтрация	84
КИХ-фильтрация	75
Инициализация и копирование	84
Изменение размеров	84
Операции выборки	84
Функции поддержки	84
nmppiRelease	83
Скалярные функции	276
Инициализация	72
nmppsRand	71
Integer operations	72
nmppcSqrt	71
Fix-point 64	72
nmppcFixSqrt64	67
nmppcDoubleToFix64	68
nmppcFix64toDouble	68
nmppcFixDiv64	69
nmppcFixSinCos64	69
nmppcFixArcTan64	70
Fix-point 32	73
nmppcFixExp32	60
nmppcFixSinCos32	60
nmppcFixArcTan32	61
nmppcDoubleToFix32	61
nmppcFix32toDouble	62
nmppcFixSqrt32	62
nmppcFixMul32	63
nmppcFixInv32	64
nmppcTblFixArcSin32	64
nmppcTblFixArcCos32	65
nmppcTblFixCos32	65
nmppcTblFixSin32	66
nmppcFixDivMod32	66
nmppcFix64Exp01	70
Арифметические операции	74
nmppcDivC	59
nmppcProdC	59
Функции деинтерлейсинга	74
Базовые регистровые функции библиотеки	278
Элементарные функции	280
Vec_0_sub_data	283
Vec_activate_data	284
Vec_activate_data_add_0	285
Vec_activate_data_xor_data	286
Vec_activate_data_add_ram	287
Vec_afifo	288
Vec_data	289
Vec_data_add_ram	292
Vec_data_and_ram	293

Vec_data_or_ram	293
Vec_data_sub_ram	294
Vec_data_xor_ram	295
Vec_And	297
Vec_Or	297
Vec_Xor	298
Vec_Add	300
Vec_Sub	316
Vec_not_data	322
Vec_ram	323
Vec_ram_sub_data	324
Vec_vsum_activate_data_0	325
функции взвешенного суммирования	282
Vec_ClipMul2D2W8_AddVr	302
Vec_ClipMulNDNW2_AddVr	303
Vec_ClipMulNDNW4_AddVr	304
Vec_ClipMulNDNW8_AddVr	305
Vec_Mul2D2W1_AddVr	307
Vec_Mul2D2W2_AddVr	308
Vec_Mul2D2W4_AddVr	309
Vec_Mul2D2W8_AddVr	310
Vec_Mul3D3W2_AddVr	311
Vec_Mul3D3W8_AddVr	312
Vec_Mul4D4W2_AddVr	314
Vec_MulVN_AddVN	315
Vec_vsum_data_0	326
Vec_vsum_data_vr	327
Vec_vsum_shift_data_0	328
Vec_vsum_shift_data_vr	329
Vec_vsum_shift_data_afifo	330
Целевые функции	282
 	287
Vec_data_add_afifo	291
Vec_FilterCoreRow2	296
Vec_FilterCoreRow4	296
Vec_FilterCoreRow8	296
Vec_Abs	299
Vec_ClipExt	301
Vec_IncNeg	306
Vec_SubAbs	317
Vec_SubVN_Abs	318
Vec_Swap	319
Vec_MUL_2V4toW8_shift	320
Vec_MUL_2V8toW16_shift	321
Vec_vsum_data_afifo	326
Vec_CompareMinV	331
Vec_CompareMaxV	331
Vec_DupValueInVector8	332
Vec_DupValueInVector16	333
Vec_BuildDiagWeights8	334
Vec_BuildDiagWeights16	334
Vec_MaxVal_v8nm8s	335
Vec_MaxVal_v4nm16s	335
Vec_MaxVal	336
Vec_MinVal_v8nm8s	337
Vec_MinVal_v4nm16s	338
Vec_MinVal	338
Vec_AccMul1D1W32_AddVr	340

implement	278
контроль переполнения	278
Функции с плавающей точкой	274
Векторные функции	275
Инициализация и копирование	152
nmppsRandUniform	214
nmppsConvert	218
nmppsConvert_rounding	219
nmppsMerge	220
nmppsConvertRisc	220
nmppsCopy	222
nmppsCopyOddToOdd_32f	222
nmppsCopyEvenToOdd_32f	223
nmppsCopyOddToEven_32f	223
nmppsCopyEvenToEven_32f	223
nmppsCopyRisc	224
Арифметические операции	153
nmppsAddC	166
nmppsAdd	169
nmppsSubC	171
nmppsSubCRev	174
nmppsSub	176
nmppsMulC	180
nmppsMul_Mul_Add	181
nmppsMul_Add	181
nmppsMul_ConjMul_Add	182
nmppsMulC_AddC	183
nmppsMulC_AddV_AddC	184
nmppsMul	184
nmppsConjMul	185
nmppsMul_Mul_Sub	186
nmppsMulC_Add	187
nmppsMul_AddC	188
Операции сравнения	157
nmppsCmpLteC	241
nmppsCmpLtC	242
nmppsCmpGteC	258
nmppsCmpGtC	256
Элементарные математические функции	158
Тригонометрические функции	158
Функция деления векторов	159
Экспонента	159
Натуральный логарифм	160
Возведение в степень	161
Вычисление квадратного корня	161
Матричные функции	275
Обращение матрицы	99
MTR_fpResolve_Gauss	88
MTR_fpResolve_PivotGauss	89
nmppmLUDecomp	89
nmppmLUResolve	90
Функции обработки сигналов	275
Быстрое преобразование Фурье	120
FFT_FwdInitAlloc	21
FFT_InvInitAlloc	21
DFT-8	22
FFT-16	23

FFT-32	24
FFT-64	25
FFT-128	25
FFT-256	26
FFT-512	27
FFT-1024	28
FFT-2048	28
FFT-4096	29
IDFT-8	30
IFFT-16	31
IFFT-32	31
IFFT-64	32
IFFT-128	33
IFFT-256	34
IFFT-512	34
IFFT-1024	35
IFFT-2048	36
IFFT-4096	37
FFT-Common	38
IFFT-Common	38
Базовые регистровые функции библиотеки	278
Элементарные функции	282
Vec_copyEvenToEven_32f	290
Vec_CopyOddToEven_32f	291
nmblas	278
BLASS-LEVEL1	41
BLASS-LEVEL2	55
BLASS-LEVEL3	58
Типы данных	274
Типы векторных данных	132
Типы скалярных данных	145

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

C_Allocator32	345
C_MultiHeap	349
C_BoxImg< T >	345
C_BoxVec< T >	346
C_Heap	346
C_Img< T >	348
C_RingBuffer< T >	356
C_RingBufferRemote< T >	357
Cplx_float	358
ds_struct	358
EnterHardMode	359
I_2DSubPixelMinPosition	359
C_2DSubPixelMinPosition	343
C_2DTrigSubPixelMinPosition	344
I_PlessyCornerDetector	359
C_PlessyCornerDetector	350
C_PlessyCornerDetector_16s	352
C_PlessyCornerDetector_32f	353
int15in16x4	360
int30in32x2	360
int31in32x2	360
mtr< T >	360
nm16sc	362
nmchar	362
nmchar1D< N >	363
nmchar2D< Y, X >	363
nmintpack< T >	363
nmmtr< T >	364
nmmtrpack< T >	365
NmppsFFTSpec	367
NmppsFFTSpec_32fc	367
NmppsFrame_16s	368
NmppsFrame_16u	368
NmppsFrame_32s	368

NmppsFrame_32u	368
NmppsFrame_64s	369
NmppsFrame_64u	369
NmppsFrame_8s	369
NmppsFrame_8u	369
NmppsMallocSpec	370
NmppsTmpSpec	370
nmreg	370
nmshort	371
nmshort2D< Y, X >	371
nmvecpack< T >	372
RGB24_nm8u	373
RGB32_nm10s	374
RGB32_nm10u	374
RGB32_nm8s	374
RGB32_nm8u	374
RoundShift	375
RPoint	375
S_BufferInfo	375
s_int32x2	376
s_nm32fc	376
s_nm32fcr	377
s_nm32sc	377
s_nm64sc	377
S_nmppiFilterKernel	378
S_nmppiFilterKernel_32s32s	378
s_v16nm16s	378
s_v16nm16u	379
s_v16nm32s	379
s_v16nm32u	379
s_v16nm4u	380
s_v16nm8s	380
s_v16nm8u	381
s_v2nm32f	381
s_v2nm32s	381
s_v2nm32u	382
s_v4nm16s	382
s_v4nm16u	382
s_v4nm32f	383
s_v4nm32s	383
s_v4nm32u	383
s_v4nm64f	384
s_v4nm8u	384
s_v8nm16s	385
s_v8nm16u	385
s_v8nm32s	385
s_v8nm32u	386
s_v8nm8s	386
s_v8nm8u	387
SpecTmp1	387
spot_struct	387
tagSegmentInfo	388
tcube< T >	388
tfixpoint< T, point >	389
Tmp2BuffSpec	390
tmtr< T >	390
uint16ptr	391
uint8ptr	391

v16nm4s	392
v4nm8s	392
vec< T >	393
Wi_4096_fixed	395

Глава 3

Алфавитный указатель структур данных

3.1 Структуры данных

Структуры данных с их кратким описанием.

C_2DSubPixelMinPosition	343
C_2DTrigSubPixelMinPosition	344
C_Allocator32	345
C_BoxImg< T >	345
C_BoxVec< T >	346
C_Heap класс - куча	346
C_Img< T >	348
C_MultiHeap	349
C_PlessyCornerDetector	350
C_PlessyCornerDetector_16s	352
C_PlessyCornerDetector_32f	353
C_RingBuffer< T >	356
C_RingBufferRemote< T >	357
Cplx_float	358
ds_struct	358
EnterHardMode	359
I_2DSubPixelMinPosition	359
I_PlessyCornerDetector	359
int15in16x4	360
int30in32x2	360
int31in32x2	360
mtr< T >	360
nm16sc	362
nmchar	362
nmchar1D< N >	363
nmchar2D< Y, X >	363
nmintpack< T >	363
nmmtr< T >	364
nmmtrpack< T >	365
NmppsFFTSpec	367
NmppsFFTSpec_32fcr	367
NmppsFrame_16s	368
NmppsFrame_16u	368
NmppsFrame_32s	368

NmppsFrame_32u	368
NmppsFrame_64s	369
NmppsFrame_64u	369
NmppsFrame_8s	369
NmppsFrame_8u	369
NmppsMallocSpec	370
NmppsTmpSpec	370
nmreg	370
nmshort	371
nmshort2D< Y, X >	371
nmvecpack< T >	372
RGB24_nm8u	373
RGB32_nm10s	374
RGB32_nm10u	374
RGB32_nm8s	374
RGB32_nm8u	374
RoundShift	375
RPoint	375
S_BufferInfo	375
класс буфер - заголовок в начале выделяемой динамической памяти	375
s_int32x2	376
s_nm32fc	376
s_nm32fcr	377
s_nm32sc	377
s_nm64sc	377
S_nmpipiFilterKernel	378
S_nmpipiFilterKernel_32s32s	378
s_v16nm16s	378
s_v16nm16u	379
s_v16nm32s	379
s_v16nm32u	379
s_v16nm4u	380
s_v16nm8s	380
s_v16nm8u	381
s_v2nm32f	381
s_v2nm32s	381
s_v2nm32u	382
s_v4nm16s	382
s_v4nm16u	382
s_v4nm32f	383
s_v4nm32s	383
s_v4nm32u	383
s_v4nm64f	384
s_v4nm8u	384
s_v8nm16s	385
s_v8nm16u	385
s_v8nm32s	385
s_v8nm32u	386
s_v8nm8s	386
s_v8nm8u	387
SpecTmp1	387
spot_struct	387
tagSegmentInfo	388
tcube< T >	388
tfixpoint< T, point >	389
Tmp2BuffSpec	390
tmtt< T >	390
uint16ptr	391

uint8ptr	391
v16nm4s	392
v4nm8s	392
vec< T >	393
Wi_4096_fixed	395

Глава 4

Список файлов

4.1 Файлы

Полный список документированных файлов.

g:/nmpp/include/crtdbg2.h	397
g:/nmpp/include/fft.h	397
g:/nmpp/include/fft_32fcr.h	401
g:/nmpp/include/fftexp.h	402
g:/nmpp/include/macros_fpu.h	404
g:/nmpp/include/malloc32.h	405
g:/nmpp/include/metric.h	408
g:/nmpp/include/minrep.h	408
g:/nmpp/include/multiheap.h	408
g:/nmpp/include/nmblas.h	418
Nmblas (NeuroMatrix Basic Linear Algebra Subroutines)	418
g:/nmpp/include/nmchar.h	429
g:/nmpp/include/nmdef.h	435
g:/nmpp/include/nmplc.h	439
g:/nmpp/include/nmpli.h	455
g:/nmpp/include/nmplm.h	464
g:/nmpp/include/nmpls.h	473
g:/nmpp/include/nmplv.h	477
g:/nmpp/include/nmpp-cpp.h	496
g:/nmpp/include/nmpp.h	496
g:/nmpp/include/nmshort.h	497
g:/nmpp/include/nmtl.h	478
g:/nmpp/include/nmtype.h	563
g:/nmpp/include/ownmalloc.h	580
g:/nmpp/include/ringbuffer_.h	580
g:/nmpp/include/ringremote.h	584
g:/nmpp/include/test.h	601
g:/nmpp/include/tfixpoint.h	513
g:/nmpp/include/nmblas/nmblas_sgemm.h	
Part of nmblas library, sgemm function implementation	425
g:/nmpp/include/nmplc/cArithmetic.h	436
g:/nmpp/include/nmplc/cfixpnt32.h	436
g:/nmpp/include/nmplc/cfixpnt64.h	437
g:/nmpp/include/nmplc/cInit.h	438
g:/nmpp/include/nmplc/cInteger.h	439

g:/nmpp/include/nmplc/ nmplc.h	439
g:/nmpp/include/nmpli/ filter.h	439
g:/nmpp/include/nmpli/ iArithmetics.h	440
g:/nmpp/include/nmpli/ iCellTexture.h	440
g:/nmpp/include/nmpli/ iConvert.h	441
g:/nmpp/include/nmpli/ iDef.h	444
g:/nmpp/include/nmpli/ iDeinterlace.h	445
g:/nmpp/include/nmpli/ iFilter.h	445
g:/nmpp/include/nmpli/ iFiltration.h	447
g:/nmpp/include/nmpli/ iFloodFill.h	448
g:/nmpp/include/nmpli/ iPlessy.h	449
g:/nmpp/include/nmpli/ iPlessyDetector.h	449
g:/nmpp/include/nmpli/ iPrint.h	450
g:/nmpp/include/nmpli/ iReordering.h	451
g:/nmpp/include/nmpli/ iResample.h	451
g:/nmpp/include/nmpli/ iSelect.h	452
g:/nmpp/include/nmpli/ isubpixel2d.h	452
g:/nmpp/include/nmpli/ isubpixel2dimpl.h	453
g:/nmpp/include/nmpli/ iSupport.h	453
g:/nmpp/include/nmpli/ nmpli.h	455
g:/nmpp/include/nmpli/ warpimg.h	456
g:/nmpp/include/nmplm/ mInit.h	457
g:/nmpp/include/nmplm/ mInverse.h	458
g:/nmpp/include/nmplm/ mMatrixVector.h	459
g:/nmpp/include/nmplm/ mMatrixVectorDev.h	460
g:/nmpp/include/nmplm/ mStat.h	461
g:/nmpp/include/nmplm/ mSupport.h	462
g:/nmpp/include/nmplm/ nmplm.h	464
g:/nmpp/include/nmpls/ fft.h	397
g:/nmpp/include/nmpls/ fft_old.h	464
g:/nmpp/include/nmpls/ fftext.h	469
g:/nmpp/include/nmpls/ fftref.h	473
g:/nmpp/include/nmpls/ nmpls.h	473
g:/nmpp/include/nmpls/ sConvolution.h	474
g:/nmpp/include/nmpls/ sCorrelation.h	474
g:/nmpp/include/nmpls/ sFiltration.h	474
g:/nmpp/include/nmpls/ sfir.h	475
g:/nmpp/include/nmpls/ sResample.h	476
g:/nmpp/include/nmplv/ nmplv.h	477
g:/nmpp/include/nmplv/ nmtl.h	478
g:/nmpp/include/nmplv/ sElementary.h	478
g:/nmpp/include/nmplv/ vArithmetics.h	478
g:/nmpp/include/nmplv/ vArithmeticsDev.h	482
g:/nmpp/include/nmplv/ vBitwise.h	482
g:/nmpp/include/nmplv/ vInit.h	484
g:/nmpp/include/nmplv/ vSelect.h	486
g:/nmpp/include/nmplv/ vStat.h	490
g:/nmpp/include/nmplv/ vSupport.h	491
g:/nmpp/include/nmplv/ vSupport_.h	493
g:/nmpp/include/nmplv/ vTransform.h	495
g:/nmpp/include/nmtl/ nmtl.h	478
g:/nmpp/include/nmtl/ nmtlio.h	501
g:/nmpp/include/nmtl/ tcmplx.h	505
g:/nmpp/include/nmtl/ tcmplx_spec.h	508
g:/nmpp/include/nmtl/ tfixedpoint.h	509
g:/nmpp/include/nmtl/ tfixedpointmath.h	513
g:/nmpp/include/nmtl/ tmatrix.h	514
g:/nmpp/include/nmtl/ tmatrix_spec.h	523

g:/nmpp/include/nmtl/ tnmcube.h	524
g:/nmpp/include/nmtl/ tnmcvec.h	526
g:/nmpp/include/nmtl/ tnmint.h	529
g:/nmpp/include/nmtl/ tnmmtr.h	533
g:/nmpp/include/nmtl/ tnmmtrpack.h	540
g:/nmpp/include/nmtl/ tnmvec.h	546
g:/nmpp/include/nmtl/ tnmvecpack.h	551
g:/nmpp/include/nmtl/ tvector.h	556
g:/nmpp/include/nmvcore/ vCore.h	576
g:/nmpp/include/rpc/ rpc-host.h	589
g:/nmpp/include/rpc/ rpc-nmc-func.h	591
g:/nmpp/include/rpc/ rpc-nmc.h	598

Глава 5

Группы

5.1 FFTFwdInitAlloc

Функции инициализации структур коэффициентов, необходимых для вычисления прямого БПФ

Функции

- int nmppsFFT16FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT32FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT64FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT128FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT256FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT512FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT1024FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT2048FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)
- int nmppsFFT4096FwdInitAlloc_32fcr ([NmppsFFTSpec_32fcr](#) **addr)

5.1.1 Подробное описание

Функции инициализации структур коэффициентов, необходимых для вычисления прямого БПФ

Аргументы

in	addr	двойной указатель на структуру коэффициентов
----	------	--

Возвращает

Функции возвращают 0 в случае успешной инициализации и число, отличное от нуля, в случае ошибок

5.2 FFTInvInitAlloc

Функции инициализации структур коэффициентов, необходимых для вычисления обратного БПФ

Функции

- int nmppsFFT16InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT32InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT64InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT128InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT256InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT512InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT1024InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT2048InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)
- int nmppsFFT4096InvInitAlloc_32fcr ([NmppsFFTSPEC_32fcr](#) **addr)

5.2.1 Подробное описание

Функции инициализации структур коэффициентов, необходимых для вычисления обратного БПФ

Аргументы

in	addr	двойной указатель на структуру коэффициентов
----	------	--

Возвращает

Функции возвращают 0 в случае успешной инициализации и число, отличное от нуля, в случае ошибок

5.3 DFT-8

Функция для вычисления прямого ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

Функции

- void [nmppsDFT8Fwd_32fcr](#) (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSPEC_32fcr](#) *spec)

Функция для вычисления прямого ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

5.3.1 Подробное описание

Функция для вычисления прямого ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

5.3.2 Функции

5.3.2.1 nmppsDFT8Fwd_32fcr()

```
void nmppsDFT8Fwd_32fcr (
    const nm32fcr * pSrcVec,
    nm32fcr * pDstVec,
    NmppsFFTSpec_32fcr * spec )
```

Функция для вычисления прямого ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

5.4 FFT-16

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел

Функции

- void nmppsFFT16Fwd_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSpec_32fcr *spec)

5.4.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

5.5 FFT-32

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел

Функции

- void nmppsFFT32Fwd_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.5.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

5.6 FFT-64

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел

Функции

- void nmppsFFT64Fwd_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.6.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

5.7 FFT-128

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел

Функции

- void nmppsFFT128Fwd_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.7.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структурь, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

5.8 FFT-256

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел

Функции

- void nmppsFFT256Fwd_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.8.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Для достижения максимальной производительности (1763 такта) необходимо положить входной вектор в 1-ый банк, выходной вектор в 5-ый банк

5.9 FFT-512

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

Функции

- void nmppsFFT512Fwd_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSpec_32fcr *spec)

5.9.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Для достижения максимальной производительности (3675 такта) необходимо положить входной вектор в 2-ый банк, выходной вектор в 5-ый банк

5.10 FFT-1024

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 1024 комплексных чисел

Функции

- void nmppsFFT1024Fwd_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.10.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 1024 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Для достижения максимальной производительности (8655 такта) необходимо положить входной вектор в 6-ый банк, выходной вектор в 5-ый банк

5.11 FFT-2048

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

Функции

- void nmppsFFT2048Fwd_32fcr (const **nm32fcr** *pSrcVec, **nm32fcr** *pDstVec, **NmppsFFTSpec_32fcr** *spec)

5.11.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Для достижения максимальной производительности (19504 такта) необходимо положить входной вектор в 5-ый банк, выходной вектор в 5-ый банк

5.12 FFT-4096

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

Функции

- void nmppsFFT4096Fwd_32fcr (const **nm32fcr** *pSrcVec, **nm32fcr** *pDstVec, **NmppsFFTSpec_32fcr** *spec)

5.12.1 Подробное описание

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Для достижения максимальной производительности (54258 такта) необходимо положить входной вектор в 5-ый банк, выходной вектор в 5-ый банк

5.13 IDFT-8

Функция для вычисления обратного ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

Функции

- void nmppsDFT8Inv_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSpec_32fcr *spec)

5.13.1 Подробное описание

Функция для вычисления обратного ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.14 IFFT-16

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел

Функции

- void nmppsFFT16Inv_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSPEC_32fcr *spec)

5.14.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.15 IFFT-32

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел

Функции

- void nmppsFFT32Inv_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSPEC_32fcr *spec)

5.15.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структурь, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.16 IFFT-64

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел

Функции

- void nmppsFFT64Inv_32fc (const nm32fc *pSrcVec, nm32fc *pDstVec, NmppsFFTSpec_32fc *spec)

5.16.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.17 IFFT-128

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел

Функции

- void nmppsFFT128Inv_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.17.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.18 IFFT-256

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел

Функции

- void nmppsFFT256Inv_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.18.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Для достижения максимальной производительности (1763 такта) необходимо положить входной вектор в 1-ый банк, выходной вектор в 5-ый банк

5.19 IFFT-512

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

Функции

- void nmppsFFT512Inv_32fcr (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

5.19.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Для достижения максимальной производительности (3675 такта) необходимо положить входной вектор в 2-ый банк, выходной вектор в 5-ый банк

5.20 IFFT-1024

Функции

- void [nmppsFFT1024Inv_32fcr](#) (const [nm32fcr](#) *pSrcVec, [nm32fcr](#) *pDstVec, [NmppsFFTSpec_32fcr](#) *spec)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 1024 комплексных чисел

5.20.1 Подробное описание

5.20.2 Функции

5.20.2.1 nmppsFFT1024Inv_32fcr()

```
void nmppsFFT1024Inv_32fcr (
    const nm32fcr * pSrcVec,
    nm32fcr * pDstVec,
    NmppsFFTSpec_32fcr * spec )
```

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 1024 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Для достижения максимальной производительности (8655 такта) необходимо положить входной вектор в 6-ый банк, выходной вектор в 5-ый банк

5.21 IFFT-2048

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

Функции

- void nmppsFFT2048Inv_32fcr (const nm32fcr *pSrcVec, nm32fcr *pDstVec, NmppsFFTSpec_32fcr *spec)

5.21.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Для достижения максимальной производительности (19504 такта) необходимо положить входной вектор в 5-ый банк, выходной вектор в 5-ый банк

5.22 IFFT-4096

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

Функции

- void nmppsFFT4096Inv_32fcr (const **nm32fcr** *pSrcVec, **nm32fcr** *pDstVec, **NmppsFFTSpec_32fcr** *spec)

5.22.1 Подробное описание

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Для достижения максимальной производительности (54258 такта) необходимо положить входной вектор в 5-ый банк, выходной вектор в 5-ый банк

5.23 FFT-Common

Группы

- [IFFT-Common](#)

Функция для вычисления прямого БПФ с плавающей точкой над вектором длины от 8 до 2048.

5.23.1 Подробное описание

5.24 IFFT-Common

Функция для вычисления прямого БПФ с плавающей точкой над вектором длины от 8 до 2048.

Функция для вычисления прямого БПФ с плавающей точкой над вектором длины от 8 до 2048.

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления прямого БПФ определенного размера
----	------	--

Функция инициализации структуры коэффициентов, необходимых для вычисления прямого БПФ с плавающей точкой над вектором длины от 8 до 2048

Аргументы

in	spec	двойной указатель на структуру коэффициентов
in	order	размерность БПФ, которое нужно вычислить, например, для БПФ256 этот параметр равен 8 (т.к. $2^8 = 256$)

Возвращает

Функция возвращают 0 в случае успешной инициализации и отрицательное число (от -1 и меньше) в случае ошибок

Функция для вычисления обратного БПФ с плавающей точкой над вектором длины от 8 до 2048

Аргументы

in	pSrcVec	входной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор комплексных чисел (на мнимая и действительная части имеют тип float)
-------	---------	--

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

5.25 nmppsMalloc

Распределение памяти для векторов библиотеки.

Функции

- void * nmppsMalloc32 (unsigned sizeInt32)
- **nm64s** * nmppsMalloc_64s (unsigned nSize)
- **nm1** * nmppsMalloc_1 (unsigned nSize)
- **nm2s** * nmppsMalloc_2s (unsigned nSize)
- **nm2u** * nmppsMalloc_2u (unsigned nSize)
- **nm4s** * nmppsMalloc_4s (unsigned nSize)
- **nm4u** * nmppsMalloc_4u (unsigned nSize)
- **nm8u** * nmppsMalloc_8u (unsigned nSize)
- **nm8s** * nmppsMalloc_8s (unsigned nSize)
- **nm16u** * nmppsMalloc_16u (unsigned nSize)
- **nm16s** * nmppsMalloc_16s (unsigned nSize)
- **nm32u** * nmppsMalloc_32u (unsigned nSize)

- `nm32s * nmppsMalloc_32s (unsigned nSize)`
- `nm64u * nmppsMalloc_64u (unsigned nSize)`
- `nm64sc * nmppsMalloc_64sc (unsigned nSize)`
- `nm32sc * nmppsMalloc_32sc (unsigned sizeCmplxInt32)`
- `nm32fc * nmppsMalloc_32fc (unsigned sizeCmplxFLOAT)`
- `nm32fcr * nmppsMalloc_32fcr (unsigned sizeCmplxFLOAT)`
- `float * nmppsMalloc_32f (unsigned sizeFloat)`
- `double * nmppsMalloc_64f (unsigned sizeDouble)`

5.25.1 Подробное описание

Распределение памяти для векторов библиотеки.

Аргументы

<code>nSize</code>	Число элементов в векторе.
<code>hint</code>	Номер банка памяти. Может принимать значения <code>MEM_LOCAL</code> , <code>MEM_GLOBAL</code> .

Заметки

Память, распределенная с помощью функций `nmppsMalloc_` должна освобождаться с помощью функции `nmppsFree()`.

5.26 nmppsFree

Освобождение памяти для векторов.

Функции

- `void nmppsFree (void *ptr)`

5.26.1 Подробное описание

Освобождение памяти для векторов.

Заметки

Данная функция должна вызываться только для векторов, распределенных с помощью функций `nmppsMalloc_`.

5.27 BLASS-LEVEL1

Функции

- double `nmbblas_dasum` (const int N, const double *X, const int INCX)
takes the sum of the absolute values <>
- void `nmbblas_daxpy` (const int N, const double A, const double *X, const int INCX, double *Y, const int INCY)
constant times a vector plus a vector <>
- void `nmbblas_dcopy` (const int N, const double *X, const int INCX, double *Y, const int INCY)
copies a vector, X, to a vector, Y
- double `nmbblas_ddot` (const int N, const double *X, const int INCX, const double *Y, const int INCY)
forms the dot product of two vectors.
- double `nmbblas_dnrm2` (const int N, const double *x, const int INCX)
returns the euclidean norm of a vector via the function name, so that SCNRM2 := sqrt(x**H*x)
- void `nmbblas_drot` (const int N, double *X, const int INCX, double *Y, const int INCY, const double C, const double S)
applies a plane rotation <>
- void `nmbblas_drotg` (double *A, double *B, double *C, double *S)
construct givens plane rotation <>
- void `nmbblas_drotm` (const int N, double *X, const int INCX, double *Y, const int INCY, double *param)
Apply a Given's rotation constructed by DROTMG. <>
- void `nmbblas_dscal` (const int N, const double ALPHA, double *X, const int INCX)
scales a vector by a constant <>
- double `nmbblas_dsdot` (const int N, const float *X, const int INCX, const float *Y, const int INCY)
Compute the inner product of two vectors with double precision accumulation. <>
- void `nmbblas_dswap` (const int N, double *X, const int INCX, double *Y, const int INCY)
interchanges two vectors <>
- double `nmbblas_dznrm2` (const int N, const double *X, const int INCX)
returns the euclidean norm of a vector via the function name, so that DZNRM2 := sqrt(x**H*x) <>
- int `nmbblas_idamax` (const int N, const double *X, const int INCX)
finds the index of the first element having maximum absolute value.

- int `nmbblas_isamax` (const int N, const float *X, const int INCX)
finds the index of the first element having maximum absolute value.
- float `nmbblas_sasum` (const int N, const float *X, const int INCX)
takes the sum of the absolute values
- void `nmbblas_saxpy` (int N, const float A, const float *X, const int INCX, float *Y, const int INCY)
constant times a vector plus a vector
- float `nmbblas_scnrm2` (const int N, const float *X, const int INCX)
returns the euclidean norm of a vector via the function name, so that SCNRM2 := sqrt(x**H*x)
- void `nmbblas_scopy` (const int N, const float *X, const int INCX, float *Y, const int INCY)
copies a vector, X, to a vector, Y
- float `nmbblas_sdot` (const int N, const float *X, const int INCX, const float *Y, const int INCY)
SDOT forms the dot product of two vectors.
- float `nmbblas_sdsdot` (const int N, const float B, const float *X, const int INCX, const float *Y, const int INCY)
Compute the inner product of two vectors with double precision accumulation. <>
- float `nmbblas_snrm2` (const int N, const float *X, const int INCX)
SNRM2 returns the euclidean norm of a vector via the function name, so that SNRM2 := sqrt(x'*x).
<>
- void `nmbblas_srot` (const int N, float *X, const int INCX, float *Y, const int INCY, const float C, const float S)
applies a plane rotation <>
- void `nmbblas_srotm` (const int N, float *X, const int INCX, float *Y, const int INCY, float *param)
Apply a Given's rotation constructed by SROTMG. <>
- void `nmbblas_sscal` (const int N, const float A, const float *X, const int INCX)
scales a vector by a constant <>
- void `nmbblas_sswap` (const int N, const float *X, const int INCX, const float *Y, const int INCY)
interchanges two vectors <>

5.27.1 Подробное описание

5.27.2 Функции

5.27.2.1 `nmbblas_dasum()`

```
double nmbblas_dasum (
    const int N,
    const double * X,
    const int INCX )
```

takes the sum of the absolute values <>

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX)
in	INCX	storage spacing between elements of X

Возвращает

the sum of the absolute values.

5.27.2.2 nmbblas_daxpy()

```
void nmbblas_daxpy (
    const int N,
    const double A,
    const double * X,
    const int INCX,
    double * Y,
    const int INCY )
```

constant times a vector plus a vector <>

Аргументы

in	N	number of elements in input vector
in	A	specifies the scalar alpha
in	X	array, dimension (1 + (N - 1)*abs(INCX)
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

5.27.2.3 nmbblas_dc当地()

```
void nmbblas_dc当地 (
    const int N,
    const double * X,
```

```
const int INCX,
double * Y,
const int INCY )
```

copies a vector, X, to a vector, Y

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

5.27.2.4 nmbblas_ddot()

```
double nmbblas_ddot (
    const int N,
    const double * X,
    const int INCX,
    const double * Y,
    const int INCY )
```

forms the dot product of two vectors.

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

Возвращает

Return dot product of two vectors

5.27.2.5 nmbblas_dnrm2()

```
double nmbblas_dnrm2 (
    const int N,
    const double * x,
    const int INCX )
```

returns the euclidean norm of a vector via the function name, so that SCNRM2 := sqrt(x**H*x)

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

Возвращает

euclidean norm

5.27.2.6 nmbblas_drot()

```
void nmbblas_drot (
    const int N,
    double * X,
    const int INCX,
    double * Y,
    const int INCY,
    const double C,
    const double S )
```

applies a plane rotation <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y
in	C	
in	S	

5.27.2.7 nmblas_drotg()

```
void nmblas_drotg (
    double * A,
    double * B,
    double * C,
    double * S )
```

construct givens plane rotation <>

Аргументы

in	A	
in	B	
in	C	
out	S	

5.27.2.8 nmblas_drotm()

```
void nmblas_drotm (
    const int N,
    double * X,
    const int INCX,
    double * Y,
    const int INCY,
    double * param )
```

Apply a Given's rotation constructed by DROTMG. <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y
in	param	array, dimension (5). The rotation values constructed by SROTMG

5.27.2.9 nmblas_dscal()

```
void nmblas_dscal (
    const int N,
    const double ALPHA,
    double * X,
    const int INCX )
```

scales a vector by a constant <>

Аргументы

in	N	number of elements in input vector
in	A	specifies the scalar alpha.
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

5.27.2.10 nmblas_dsdot()

```
double nmblas_dsdot (
    const int N,
    const float * X,
    const int INCX,
    const float * Y,
    const int INCY )
```

Compute the inner product of two vectors with double precision accumulation. <>

Аргументы

in	N	number of elements in input vector
in	B	single precision scalar to be added to inner product
in	X	array, dimension (1 + (N - 1)*abs(INCX)) single precision vector with N elements
in	INCX	storage spacing between elements of X
in	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

Возвращает

Returns D.P. dot product accumulated in D.P., for S.P. SX and SY DSDOT = sum for I = 0 to N-1 of SX(LX+I*INCX) * SY(LY+I*INCY), where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is defined in a similar way using INCY.

5.27.2.11 nmbblas_dswap()

```
void nmbblas_dswap (
    const int N,
    double * X,
    const int INCX,
    double * Y,
    const int INCY )
```

interchanges two vectors <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

Возвращает

5.27.2.12 nmbblas_dznrm2()

```
double nmbblas_dznrm2 (
    const int N,
    const double * X,
    const int INCX )
```

returns the euclidean norm of a vector via the function name, so that DZNRM2 := sqrt(x**H*x) <>

Аргументы

in	N	number of elements in input vector
in	X	COMPLEX*16 array of DIMENSION at least (1 + (m - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

Возвращает

euclidean norm

5.27.2.13 nmbblas_idamax()

```
int nmbblas_idamax (
    const int N,
    const double * X,
    const int INCX )
```

finds the index of the first element having maximum absolute value.

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

Возвращает

index of the first element having maximum absolute value.

5.27.2.14 nmbblas_isamax()

```
int nmbblas_isamax (
    const int N,
    const float * X,
    const int INCX )
```

finds the index of the first element having maximum absolute value.

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

Возвращает

index of the first element having maximum absolute value.

5.27.2.15 nmbblas_sasum()

```
float nmbblas_sasum (
    const int N,
    const float * X,
    const int INCX )
```

takes the sum of the absolute values

Аргументы

in	N	number of elements in input vector
in	A	specifies the scalar alpha
in	X	array, dimension (1 + (N - 1)*abs(INCX))

Возвращает

sum of the absolute values

5.27.2.16 nmbblas_saxpy()

```
void nmbblas_saxpy (
    int N,
    const float A,
    const float * X,
    const int INCX,
    float * Y,
    const int INCY )
```

constant times a vector plus a vector

Аргументы

in	N	number of elements in input vector
in	A	specifies the scalar alpha
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

5.27.2.17 nmblas_scnrm2()

```
float nmblas_scnrm2 (
    const int N,
    const float * X,
    const int INCX )
```

returns the euclidean norm of a vector via the function name, so that $\text{SCNRM2} := \sqrt(\text{x}^*\text{H}\text{x})$

Аргументы

in	N	number of elements in input vector
in	X	array, dimension $(1 + (N - 1) * \text{abs}(INCX))$
in	INCX	storage spacing between elements of X

Возвращает

euclidean norm

5.27.2.18 nmblas_scopy()

```
void nmblas_scopy (
    const int N,
    const float * X,
    const int INCX,
    float * Y,
    const int INCY )
```

copies a vector, X, to a vector, Y

Аргументы

in	N	number of elements in input vector
in	X	array, dimension $(1 + (N - 1) * \text{abs}(INCX))$
in	INCX	storage spacing between elements of X
out	Y	array, dimension $(1 + (N - 1) * \text{abs}(INCY))$
in	INCY	storage spacing between elements of Y

5.27.2.19 nmbblas_sdot()

```
float nmbblas_sdot (
    const int N,
    const float * X,
    const int INCX,
    const float * Y,
    const int INCY )
```

SDOT forms the dot product of two vectors.

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

Возвращает

Return dot product of two vectors

5.27.2.20 nmbblas_sdsdot()

```
float nmbblas_sdsdot (
    const int N,
    const float B,
    const float * X,
    const int INCX,
    const float * Y,
    const int INCY )
```

Compute the inner product of two vectors with double precision accumulation. <>

Аргументы

in	N	number of elements in input vector
in	B	single precision scalar to be added to inner product
in	X	array, dimension (1 + (N - 1)*abs(INCX)) single precision vector with N elements
in	INCX	storage spacing between elements of X
in	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

Возвращает

Returns S.P. result with dot product accumulated in D.P. SDSDOT = SB + sum for I = 0 to N-1 of SX(LX+I*INCX)*SY(LY+I*INCY), where LX = 1 if INCX .GE. 0, else LX = 1+(1-N)*INCX, and LY is defined in a similar way using INCY.

5.27.2.21 nmblas_snrm2()

```
float nmblas_snrm2 (
    const int N,
    const float * X,
    const int INCX )
```

SNRM2 returns the euclidean norm of a vector via the function name, so that SNRM2 := sqrt(x'*x).
 <>

Аргументы

in	N	number of elements in input vector
in	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

Возвращает

euclidean norm

5.27.2.22 nmblas_srot()

```
void nmblas_srot (
    const int N,
    float * X,
    const int INCX,
    float * Y,
    const int INCY,
    const float C,
    const float S )
```

applies a plane rotation <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y
in	C	
in	S	

5.27.2.23 nmblas_srotm()

```
void nmblas_srotm (
    const int N,
    float * X,
    const int INCX,
    float * Y,
    const int INCY,
    float * param )
```

Apply a Given's rotation constructed by SROTMG. <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y
in	param	array, dimension (5). The rotation values constructed by SROTMG

5.27.2.24 nmblas_sscl()

```
void nmblas_sscl (
    const int N,
    const float A,
    const float * X,
    const int INCX )
```

scales a vector by a constant <>

Аргументы

in	N	number of elements in input vector
in	A	specifies the scalar alpha.
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X

5.27.2.25 nmblas_sswap()

```
void nmblas_sswap (
    const int N,
    const float * X,
    const int INCX,
    const float * Y,
    const int INCY )
```

interchanges two vectors <>

Аргументы

in	N	number of elements in input vector
in,out	X	array, dimension (1 + (N - 1)*abs(INCX))
in	INCX	storage spacing between elements of X
in,out	Y	array, dimension (1 + (N - 1)*abs(INCY))
in	INCY	storage spacing between elements of Y

5.28 BLASS-LEVEL2

- void [nmblas_sgenv](#) (const enum nm_trans TRANS, const int M, const int N, const float ALPHA, const float *A, const int LDA, const float *X, const int INCX, const float BETA, float *Y, const int INCY)

performs one of the matrix-vector operations $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$, or $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
- void [nmblas_dgenv](#) (const enum nm_trans TRANS, const int M, const int N, const double ALPHA, const double *A, const int LDA, const double *X, const int INCX, const double BETA, double *Y, const int INCY)

performs one of the matrix-vector operations $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$, or $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
- void [nmblas_sger](#) (const int M, const int N, const float ALPHA, const float *X, const int INCX, const float *Y, const int INCY, float *A, const int LDA)

perform the rank 1 operation $A := \text{alpha} \cdot x \cdot y' + A$, where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

5.28.1 Подробное описание

5.28.2 Функции

5.28.2.1 nmblas_dgemv()

```
void nmblas_dgemv (
    const enum nm_trans TRANS,
    const int M,
    const int N,
    const double ALPHA,
    const double * A,
    const int LDA,
    const double * X,
    const int INCX,
    const double BETA,
    double * Y,
    const int INCY )
```

performs one of the matrix-vector operations $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$, or $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Аргументы

in	TRANS	is CHARACTER*1 On entry, TRANS specifies the operation to be performed as follows:
----	-------	--

TRANS = 'N' or 'n' $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$.

TRANS = 'T' or 't' $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$.

TRANS = 'C' or 'c' $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$.

Аргументы

in	M	specifies the number of rows of the matrix A. M must be at least zero.
in	N	specifies the number of columns of the matrix A. N must be at least zero.
in	ALPHA	LPHA specifies the scalar alpha
in	A	array, dimension (LDA, N) Before entry, the leading m by n part of the array A must contain the matrix of coefficients
in	LDA	specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).
in	X	array, dimension at least (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.
in	INCX	INCX specifies the increment for the elements of X. INCX must not be zero.
in	BETA	specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.
in,out	Y	is REAL array, dimension at least (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
in	INCY	specifies the increment for the elements of Y. INCY must not be zero.

Возвращает

5.28.2.2 nmblas_sgmv()

```
void nmblas_sgmv (
    const enum nm_trans TRANS,
    const int M,
    const int N,
    const float ALPHA,
    const float * A,
    const int LDA,
    const float * X,
    const int INCX,
    const float BETA,
    float * Y,
    const int INCY )
```

performs one of the matrix-vector operations $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$, or $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.

Аргументы

in	TRANS	is CHARACTER*1 On entry, TRANS specifies the operation to be performed as follows:
----	-------	--

TRANS = 'N' or 'n' $y := \text{alpha} \cdot A \cdot x + \text{beta} \cdot y$.

TRANS = 'T' or 't' $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$.

TRANS = 'C' or 'c' $y := \text{alpha} \cdot A^T \cdot x + \text{beta} \cdot y$.

Аргументы

in	M	specifies the number of rows of the matrix A. M must be at least zero.
in	N	specifies the number of columns of the matrix A. N must be at least zero.
in	ALPHA	LPHA specifies the scalar alpha
in	A	array, dimension (LDA, N) Before entry, the leading m by n part of the array A must contain the matrix of coefficients
in	LDA	specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).
in	X	array, dimension at least (1 + (n - 1)*abs(INCX)) when TRANS = 'N' or 'n' and at least (1 + (m - 1)*abs(INCX)) otherwise. Before entry, the incremented array X must contain the vector x.
in	INCX	INCX specifies the increment for the elements of X. INCX must not be zero.
in	BETA	specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input.
in,out	Y	is REAL array, dimension at least (1 + (m - 1)*abs(INCY)) when TRANS = 'N' or 'n' and at least (1 + (n - 1)*abs(INCY)) otherwise. Before entry with BETA non-zero, the incremented array Y must contain the vector y. On exit, Y is overwritten by the updated vector y.
Создано системой Doxygen	INCY	specifies the increment for the elements of Y. INCY must not be zero.

Возвращает

5.28.2.3 nmbblas_sger()

```
void nmbblas_sger (
    const int M,
    const int N,
    const float ALPHA,
    const float * X,
    const int INCX,
    const float * Y,
    const int INCY,
    float * A,
    const int LDA )
```

perform the rank 1 operation $A := \text{alpha} \cdot x \cdot y' + A$, where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

Аргументы

in	M	M is INTEGER On entry, M specifies the number of rows of the matrix A. M must be at least zero.
in	N	N is INTEGER On entry, N specifies the number of columns of the matrix A. N must be at least zero.
in	ALPHA	ALPHA is REAL On entry, ALPHA specifies the scalar alpha.
in	X	X is REAL array, dimension at least (1 + (m - 1)*abs(INCX)). Before entry, the incremented array X must contain the m element vector x.
in	INCX	INCX is INTEGER On entry, INCX specifies the increment for the elements of X. INCX must not be zero.
in	Y	Y is REAL array, dimension at least (1 + (n - 1)*abs(INCY)). Before entry, the incremented array Y must contain the n element vector y.
in	INCY	INCY is INTEGER On entry, INCY specifies the increment for the elements of Y. INCY must not be zero.
in	INCX	INCX specifies the increment for the elements of X. INCX must not be zero.
in,out	A	A is REAL array, dimension (LDA, N) Before entry, the leading m by n part of the array A must contain the matrix of coefficients. On exit, A is overwritten by the updated matrix.
in	LDA	LDA is INTEGER On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. LDA must be at least max(1, m).

Возвращает

5.29 BLASS-LEVEL3

Файлы

- файл [nmbblas_sgemm.h](#)

part of nmblas library, sgemm function implementation

5.29.1 Подробное описание

5.30 nmppcDivC

частное двух комплексных чисел

Функции

- void nmppcDivC ([nm64sc](#) *pnSrcA, [nm64s](#) *pnSrcB, [nm64sc](#) *Dst)

5.30.1 Подробное описание

частное двух комплексных чисел

<>

Аргументы

<code>*pnSrcA</code>	указатель на делимое.
<code>*pnSrcB</code>	указатель на делитель.
<code>*Dst</code>	указатель на частное.

5.31 nmppcProdC

произведение двух комплексных чисел.

Функции

- void nmppcProdC ([nm64sc](#) *pnSrcA, [nm64sc](#) *pnSrcB, [nm64sc](#) *Dst)

5.31.1 Подробное описание

произведение двух комплексных чисел.

Аргументы

<code>*pnSrcA</code>	указатель на первый множитель.
<code>*pnSrcB</code>	указатель на второй множитель.

5.32 nmppcFixExp32

Вычисление вычисления экспоненты числа в формате fixed-point (16.16)

Функции

- int nmppcFixExp32 (int nVal)

5.32.1 Подробное описание

Вычисление вычисления экспоненты числа в формате fixed-point (16.16)

Аргументы

nVal	Входное число с фиксированной точкой в формату (16.16)
------	--

Возвращает

Экспонента числа в формате с фиксированной точкой (16.16)

5.33 nmppcFixSinCos32

Вычисление синуса и косинуса от аргумента в формате fixed-point (16.16)

Функции

- void nmppcFixSinCos32 (int nArg, int *pnSin, int *pnCos)

5.33.1 Подробное описание

Вычисление синуса и косинуса от аргумента в формате fixed-point (16.16)

Аргументы

nArg	Угол в радианах. Угол должен быть в диапазоне от -PI/2 до +PI/2
------	---

Возвращаемые значения

pnSin	указатель на синус
pnCos	указатель на косинус

5.34 nmppcFixArcTan32

Вычисление арктангенса от аргумента в формате fixed-point (16.16)

Функции

- int nmppcFixArcTan32 (int nArg)

5.34.1 Подробное описание

Вычисление арктангенса от аргумента в формате fixed-point (16.16)

Аргументы

nArg	Угол в радианах
------	-----------------

Возвращает

Арктангенс

5.35 nmppcDoubleToFix32

Функция перевода из Fixed-Point (16.16) в Double.

Функции

- int nmppcDoubleToFix32 (double arg)

5.35.1 Подробное описание

Функция перевода из Fixed-Point (16.16) в Double.

Аргументы

arg	Входное число с плавающей точкой
fixpoint	позиция двоичной точки

Возвращает

Число с фиксированной точкой

5.36 nmppcFix32ToDouble

Преобразование 32р. числа с фиксированной точкой (16.16) в число с плавающей точкой типа double.

Функции

- double nmppcFix32ToDouble (int arg)

5.36.1 Подробное описание

Преобразование 32р. числа с фиксированной точкой (16.16) в число с плавающей точкой типа double.

Аргументы

arg	Входное 32р. число в формате с фиксированной точкой (16.16)
-----	---

Возвращает

Число с плавающей точкой

5.37 nmppcFixSqrt32

Вычисление квадратного корня числа в формате fixed-point (16.16)

Функции

- unsigned int nmppcFixSqrt32 (unsigned int nVal)

5.37.1 Подробное описание

Вычисление квадратного корня числа в формате fixed-point (16.16)

Аргументы

nVal	Входное число с фиксированной точкой в формату (16.16)
------	--

Возвращает

Квадратный корень в формате с фиксированной точкой (16.16)

5.38 nmppcFixMul32

Вычисление произведения двух числе в формате fixed-point (16.16)

Функции

- int nmppcFixMul32 (int nX, int nY)
- int nmppcFixDiv32 (int nX, int nY)

5.38.1 Подробное описание

Вычисление произведения двух числе в формате fixed-point (16.16)

Деление двух целых чисел с записью результата в формате fixed-point (16.16)

Аргументы

nX	Первое входное число с фиксированной точкой в формату (16.16)
nY	Второе входное число с фиксированной точкой в формату (16.16)

Возвращает

Произведение в формате с фиксированной точкой (16.16)

Аргументы

nX	Делимое
nY	Делитель

Возвращает

Частное от деления в формате с фиксированной точкой (16.16)

5.39 nmppcFixInv32

Вычисление обратного значения целого числа с записью результата в формате fixed-point.

Функции

- int nmppcFixInv32 (int nVal, int nFixpoint)

5.39.1 Подробное описание

Вычисление обратного значения целого числа с записью результата в формате fixed-point.

$$Res = 2^n Fixpoint / nVal$$

Аргументы

nVal	Делитель
nFixpoint	Позиция бинарной точки в результирующем слове

Возвращает

Частное от деления в формате с фиксированной точкой (16.16)

5.40 nmppcTblFixArcSin32

Вычисление функции arcsin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Функции

- int nmppcTblFixArcSin32 (int nArg)

5.40.1 Подробное описание

Вычисление функции arcsin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Аргументы

nArg	Входное значение.
------	-------------------

Возвращает

Угол в диапазоне от $-\text{PI}/2$ до $+\text{PI}/2$ в формате fixed-point (16.16)

5.41 nmppcTblFixArcCos32

Вычисление функции arccos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Функции

- int nmppcTblFixArcCos32 (int nArg)

5.41.1 Подробное описание

Вычисление функции arccos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Аргументы

nArg	Входное значение.
------	-------------------

Возвращает

Угол в диапазоне от 0 до PI в формате fixed-point (16.16)

5.42 nmppcTblFixCos32

Вычисление функции cos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Функции

- int nmppcTblFixCos32 (int nArg)

5.42.1 Подробное описание

Вычисление функции cos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Аргументы

nArg	Угол в диапазоне от 0 до PI в формате fixed-point (16.16)
------	---

Возвращает

значение cos в формате fixed-point (16.16)

5.43 nmppcTblFixSin32

Вычисление функции sin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Функции

- int nmppcTblFixSin32 (int nArg)

5.43.1 Подробное описание

Вычисление функции sin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

Аргументы

nArg	Угол в диапазоне от -PI/2 до +PI/2 в формате fixed-point (16.16)
------	--

Возвращает

значение sin в формате fixed-point (16.16)

5.44 nmppcFixDivMod32

Вычисление частного и остатка при делении чисел с фиксированной запятой в формате fixed-point (16.16)

Функции

- void nmppcFixDivMod32 (int nDividend, int nDivisor, int *pnQuotient, int *pnReminder)
- void nmppcFixDivPosMod32 (unsigned int nDividend, unsigned int nDivisor, int *pnQuotient, int *pnReminder)

5.44.1 Подробное описание

Вычисление частного и остатка при делении чисел с фиксированной запятой в формате fixed-point (16.16)

Аргументы

nDividend	Делимое в формате fixed-point (16.16)
nDivisor	Делитель в формате fixed-point (16.16)

Возвращаемые значения

pnQuotient	Частное от деления в формате fixed-point (16.16)
pnReminder	Остаток от деления в формате fixed-point (16.16)

5.45 nmppcFixSqrt64

Вычисление квадратного корня числа в формате fixed-point (32.32)

Функции

- unsigned long nmppcFixSqrt64 (unsigned long x)

5.45.1 Подробное описание

Вычисление квадратного корня числа в формате fixed-point (32.32)

Аргументы

x	Входное число с фиксированной точкой в формате (32.32)
---	--

Возвращает

Квадратный корень в формате с фиксированной точкой (32.32)

5.46 nmppcDoubleToFix64

Функция перевода из Fixed-Point 64 в Double.

Функции

- long nmppcDoubleToFix64 (double arg, int fixpoint)

5.46.1 Подробное описание

Функция перевода из Fixed-Point 64 в Double.

Аргументы

arg	Входное число с плавающей точкой
fixpoint	позиция двоичной точки

Возвращает

Число с фиксированной точкой

5.47 nmppcFix64toDouble

Преобразование 64р. числа с фиксированной точкой в число с плавающей точкой типа double.

Функции

- double nmppcFix64toDouble (long arg, int fixpoint)

5.47.1 Подробное описание

Преобразование 64р. числа с фиксированной точкой в число с плавающей точкой типа double.

Аргументы

arg	Входное 64р. число в формате с фиксированной точкой
fixpoint	Позиция двоичной точки

Возвращает

Число с плавающей точкой

5.48 nmppcFixDiv64

Деление двух целых чисел с записью результата в формате fixed-point.

Функции

- void nmppcFixDiv64 (long *nDividend, long *nDivisor, int nFixpoint, long *nQuotient)

5.48.1 Подробное описание

Деление двух целых чисел с записью результата в формате fixed-point.

Аргументы

nDividend	Делимое
nDivisor	Делитель. Делитель должен быть по модулю больше чем делимое.
nFixpoint	Позиция двоичной точки

Возвращаемые значения

nQuotient	Частное в формате числа с фиксированной точкой
-----------	--

5.49 nmppcFixSinCos64

Вычисление синуса и косинуса от аргумента в формате fixed-point (32.32)

Функции

- void nmppcFixSinCos64 (long nArg, long *pnSin, long *pnCos)

5.49.1 Подробное описание

Вычисление синуса и косинуса от аргумента в формате fixed-point (32.32)

Аргументы

nArg	Угол в радианах. Угол должен быть в диапазоне от -PI/2 до +PI/2
------	---

Возвращаемые значения

pnSin	указатель на синус
pnCos	указатель на косинус

5.50 nmppcFixArcTan64

Вычисление арктангенса от аргумента в формате fixed-point (32.32)

Функции

- long nmppcFixArcTan64 (long nArg)

5.50.1 Подробное описание

Вычисление арктангенса от аргумента в формате fixed-point (32.32)

Аргументы

nArg	Угол в радианах
------	-----------------

\retrun Арктангенс

5.51 nmppcFix64Exp01

Вычисление вычисления экспоненты числа в формате fixed-point (4.60)

Функции

- long nmppcFix64Exp01 (long nArg)

5.51.1 Подробное описание

Вычисление вычисления экспоненты числа в формате fixed-point (4.60)

Аргументы

nVal	Входное число с фиксированной точкой в формате (4.60)
------	---

Возвращает

Экспонента числа в формате с фиксированной точкой (4.60)

5.52 nmppsRand

Генерация случайного числа с равномерным распределением.

Функции

- int nmppcRandMinMaxDiv (int nMin, int nMax, int nDivisible)
- int nmppcRandMinMax (int nMin, int nMax)
- int nmppcRand ()

5.52.1 Подробное описание

Генерация случайного числа с равномерным распределением.

Аргументы

nMin	Минимальное возможное значение случайного числа.
nMax	Максимальное возможное значение случайного числа.
nDivisible	Значение, которому будет кратно случайное число.

Возвращает

int Случайное число в диапазоне либо [nMin, nMax]. Для функции без параметров данный диапазон [-2^31; 2^31-1].

5.53 nmppcSqrt

Вычисление квадратного корня

Функции

- `unsigned int nmppcSqrt_64u (unsigned long long x)`

5.53.1 Подробное описание

Вычисление квадратного корня

Аргументы

<code>x</code>	Входное число
----------------	---------------

Возвращает

Квадратный корень

5.54 Инициализация

Группы

- [nmppsRand](#)

Генерация случайного числа с равномерным распределением.

5.54.1 Подробное описание

5.55 Integer operations

Группы

- [nmppcSqrt](#)

Вычисление квадратного корня

5.55.1 Подробное описание

5.56 Fix-point 64

Группы

- [nmppcFixSqrt64](#)

Вычисление квадратного корня числа в формате fixed-point (32.32)

- [nmppcDoubleToFix64](#)

Функция перевода из Fixed-Point 64 в Double.

- [nmppcFix64.ToDouble](#)

Преобразование 64р. числа с фиксированной точкой в число с плавающей точкой типа double.

- [nmppcFixDiv64](#)

Деление двух целых чисел с записью результата в формате fixed-point.

- [nmppcFixSinCos64](#)

Вычисление синуса и косинуса от аргумента в формате fixed-point (32.32)

- [nmppcFixArcTan64](#)

Вычисление арктангенса от аргумента в формате fixed-point (32.32)

5.56.1 Подробное описание

5.57 Fix-point 32

Группы

- [nmppcFixExp32](#)

Вычисление вычисления экспоненты числа в формате fixed-point (16.16)

- [nmppcFixSinCos32](#)

Вычисление синуса и косинуса от аргумента в формате fixed-point (16.16)

- [nmppcFixArcTan32](#)

Вычисление арктангенса от аргумента в формате fixed-point (16.16)

- [nmppcDoubleToFix32](#)

Функция перевода из Fixed-Point (16.16) в Double.

- [nmppcFix32.ToDouble](#)

Преобразование 32р. числа с фиксированной точкой (16.16) в число с плавающей точкой типа double.

- [nmppcFixSqrt32](#)

Вычисление квадратного корня числа в формате fixed-point (16.16)

- [nmppcFixMul32](#)

Вычисление произведения двух числе в формате fixed-point (16.16)

- [nmppcFixInv32](#)

Вычисление обратного значения целого числа с записью результата в формате fixed-point.

- [nmppcTblFixArcSin32](#)

Вычисление функции arcsin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

- [nmppcTblFixArcCos32](#)

Вычисление функции arccos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

- [nmppcTblFixCos32](#)

Вычисление функции cos по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

- [nmppcTblFixSin32](#)

Вычисление функции sin по таблице. Входные и выходные значения задаются в формате fixed-point (16.16)

- [nmppcFixDivMod32](#)

Вычисление частного и остатка при делении чисел с фиксированной запятой в формате fixed-point (16.16)

- [nmppcFix64Exp01](#)

Вычисление вычисления экспоненты числа в формате fixed-point (4.60)

5.57.1 Подробное описание

5.58 Арифметические операции

Группы

- [nmppcDivC](#)
частное двух комплексных чисел
- [nmppcProdC](#)
произведение двух комплексных чисел.

5.58.1 Подробное описание

5.59 Функции деинтерлейсинга

Функции

- void [nmppiDeinterlaceSplit](#) (nm8u *pSrcImg, int nSrcWidth, int nSrcHeight, nm8u *pDstEven, nm8u *pDstOdd)
- void [nmppiDeinterlaceBlend](#) (nm8u *pSrcEven, nm8u *pSrcOdd, int nSrcWidth, int nSrcHeight, nm8u *pDst)

5.59.1 Подробное описание

5.59.2 Функции

5.59.2.1 nmppiDeinterlaceBlend()

```
void nmppiDeinterlaceBlend (
    nm8u * pSrcEven,
    nm8u * pSrcOdd,
    int nSrcWidth,
    int nSrcHeight,
    nm8u * pDst )
```

Функция объединяет два полукарда в один полный кадр.

Аргументы

pSrcEven	указатель на четный полукард.
nSrcOdd	указатель на нечетный полукард.
nSrcWidth	ширина исходных полукардов в пикселях.
pSrcHeight	высота исходных полукардов в пикселях.
pDst	указатель на буфер результирующего кадра

Возвращает

void

5.59.2.2 nmppiDeinterlaceSplit()

```
void nmppiDeinterlaceSplit (
    nm8u * pSrcImg,
    int nSrcWidth,
    int nSrcHeight,
    nm8u * pDstEven,
    nm8u * pDstOdd )
```

Функция разделяет кадр на два полукаадра.

Аргументы

pSrcImg	указатель на исходный кадр.
nSrcWidth	ширина исходного кадра в пикселях.
nSrcHeight	высота исходного кадра в пикселях.
pDstEven	указатель на буфер четного полукаадра
pDstOdd	указатель на буфер нечетного полукаадра

Возвращает

void

5.60 КИХ-фильтрация

Двумерная КИХ фильтрация

Переменные

- nm32s * S_nmppiFilterKernel::pDispArray
- nm32s * S_nmppiFilterKernel::pWeightMatrix
- int S_nmppiFilterKernel::nKerWidth
- int S_nmppiFilterKernel::nKerHeight
- nm32s * S_nmppiFilterKernel_32s32s::pDispArray
- nm32s * S_nmppiFilterKernel_32s32s::pWeightMatrix
- int S_nmppiFilterKernel_32s32s::nKerWidth
- int S_nmppiFilterKernel_32s32s::nKerHeight

5.60.1 Подробное описание

Двумерная КИХ фильтрация

<>

5.61 Floodfill

Исполняет разделение бинарной картинки на односвязные области. Пример вызова: no=VL_ ← FloodFill32b(pSrcImage, Tetr,Image, pTmpBuff, nSrcWidth, nSrcHeight);

Функции

- int nmppiFloodFill (unsigned int *pSrcImage, SegmentInfo *pSegmentInfo, unsigned int *pSegmentImage, int nWidth, int nHeight, unsigned int *pTmpBuff)
- int **FloodFill8** (void *src, void *dst, int nWidth, int nHeight, spot_struct *spot, int lenSpot, unsigned *pixels, int mSpot, int dtFull, int dtSpot, int lDiag, int lDropSpot, ds_struct *dropSpot, int nPxlMin, int nPxlMax, int dXYmin, int dXYmax)

Функция FloodFill8 выполняет поиск пятен (сегментов, односвязных областей) во входной 8-битной матрице (изображения в градациях серого от 0 до 255), и строит такие же пятна в выходной матрице, заполняя их одним и тем же значением (цветом, соответствующим номеру пятна).

5.61.1 Подробное описание

Исполняет разделение бинарной картинки на односвязные области. Пример вызова: no=VL_ ← FloodFill32b(pSrcImage, Tetr,Image, pTmpBuff, nSrcWidth, nSrcHeight);

Аргументы

pSrcImage	Входное изображение
pSegmentInfo	массив структур, где содержатся минимальные и максимальные координаты прямоугольника, описывающего сегментированную область.
nWidth	ширина изображения
nHeight	высота изображения
pTmpBuff	Временный массив. Его размер должен быть $2*nWidth*nHeight$

Возвращаемые значения

pSegmentImage	Результирующее "изображение", где все точки одного сегмента имеют одинаковое значение (1,...)
---------------	---

Возвращает

Число сегментов на изображении

Заметки

Массив pSegmentImage должен быть обнулен. Функция изменяет массив pSrcImage.

5.61.2 Функции

5.61.2.1 FloodFill8()

```
int FloodFill8 (
    void * src,
    void * dst,
    int nWidth,
    int nHeight,
    spot_struct * spot,
    int lenSpot,
    unsigned * pixels,
    int mSpot,
    int dtFull,
    int dtSpot,
    int lDiag,
    int lDropSpot,
    ds_struct * dropSpot,
    int nPx1Min,
    int nPx1Max,
    int dXYmin,
    int dXYmax )
```

Функция FloodFill8 выполняет поиск пятен (сегментов, односвязных областей) во входной 8-битной матрице (изображения в градациях серого от 0 до 255), и строит такие же пятна в выходной матрице, заполняя их одним и тем же значением (цветом, соответствующим номеру пятна).

Аргументы

src	входная 8-битная матрица размером nHeight x nWidth. Внимание. Входная матрица src модифицируется: <ul style="list-style-type: none"> в целях оптимизации программы обнуляются верхняя и нижняя строки, и крайний левый и крайний правый столбцы (границы матрицы), в процессе обработки найденные пятна обнуляются.
dst	результатирующая 8-битная матрица того же размера nHeight x nWidth. Внимание. Перед обращением к функции выходная матрица dst должна быть обнулена.
nWidth	ширина входной (и выходной) матрицы в 8-битных элементах. Ограничения. Ширина матрицы должна быть кратна 32-битным словам: nWidth - положительное, $(nWidth \& 3) = 0$.
nHeight	высота входной (и выходной) матрицы. Ограничения. Высота матрицы не должна превышать 1080 строк: nHeight - положительное, $(nHeight \leq 1080)$. Это ограничение можно обойти задав в начале файла FloodFill8.asm вместо 1080 нужное значение константы: const NHEIGHT = 1080;
spot	массив, содержащий обобщенную информацию о найденных пятнах – минимально 6 параметров формата int для каждого пятна: Замечание1. Чтобы гарантированно избежать переполнения массива spot в общем случае (когда характер матрицы, количество, размер пятен неизвестны, и не задан параметр mSpot>0), размер массива spot в 32-битных словах должен быть: $(nSpotMax + 1 + [(nSpotMax-1)/255]) * lenSpot$, $nSpotMax = [(nHeight-2)*(nWidth-2)/2 + 0.5]$, при lDiag=0, $= [(nHeight-2)/2+0.5] * [(nWidth-2)/2+0.5]$, при lDiag=1, квадратные скобки [...] обозначают целую часть числа, напомним: 2 строки и 2 столбца входной матрицы обнулены. Замечание2. Если задан параметр mSpot>0, то массив spot достаточно определить для mSpot пятен (mSpot задается с учетом служебного и фиктивных пятен).

Аргументы

lenSpot	размер пятна в 32-битных словах, позволяет задать размер пятна более 6 слов, резервируя место для дополнительных параметров пятна дозвычисляемых другими функциями (например, угол наклона пятна к горизонтальной оси...). Замечание. Если параметр lenSpot задан некорректно $\text{lenSpot} < 6$, то ему присваивается минимальное возможное значение $\text{lenSpot}=6$.
pixels	массив координат пикселов, принадлежащих найденным пятнам, каждый элемент массива состоит из одного 32 битного слова, включающего в себя координаты пикселя в матрице src (или dst): <code>unsigned int ij;</code> биты [31..16] - i - номер строки, биты [15..0] - j - номер столбца. (Отсюда ограничение: nHeight, nWidth не могут быть больше $2^{*}16-1 = 65535$). Замечание. В общем случае (когда заранее неизвестно о количестве и размерах пятен в матрице), чтобы гарантированно избежать переполнения массива pixels, его размер должен быть $(\text{nHeight}-2)*(\text{nWidth}-2)$ 32-битных слов (что почти в 4 раза больше входной 8-битной матрицы, напомним: 2 строки и 2 солбца входной матрицы обнулены).
mSpot	число пятен, ограничивающее поиск. Если задано $\text{mSpot}>0$, то после обработки очередного пятна проверяется: достигло ли количество найденных пятен nSpot заданного mSpot ($\text{mSpot} \leq \text{nSpot}$)? Если «Да», то происходит (принудительный) выход из программы с записью в <code>spot[0].Ymin=2</code> . Если задано $\text{mSpot}=0$, то проверок на количество найденных пятен не производится. Замечание1. Если задать $\text{mSpot} \leq 255$, то пятна будут однозначно соответствовать своим номерам, то есть не будет пятен с повторяющимися номерами. Замечание2. mSpot следует задавать с учетом служебного пятна (с номером 0) и фиктивных пятен (с номерами 256,512,768...), то есть, если поиск надо ограничить нахождением первых $\text{mSpotReal}>0$ пятен, то $\text{mSpot} = 1 + \text{mSpotReal} + [(mSpotReal-1)/255]$ квадратные скобки [...] обозначают целую часть числа
dtFull	время, выделенное на работу программы (в тактах процессора). Если задано $\text{dtFull}>0$, то после обработки очередного пятна (и в конце каждой строки) проверяется: осталось ли время на обработку следующего пятна $\text{dtRest} < \text{dtSpot}$? ($\text{dtRest}=\text{dtFull}-(T-T_0)$, T – текущее время, T_0 – начало работы программы). Если «Нет», то происходит выход из программы с записью в <code>spot[0].Ymin=1</code> . Если задано $\text{dtFull}=0$, то проверок на достаточность времени на обработку следующего пятна не производится. Замечание1. Задание $\text{dtFull}>0$ маленьким может значительно сократить время работы программы, однако, не гарантирует корректного завершения программы (например, когда обрабатываемое пятно занимает всю оставшуюся часть матрицы). Замечание2. При выборе значения dtFull следует учитывать время обработки типового пятна dtSpot.
dtSpot	время обработки типового пятна (в тактах процессора). Используется совместно с параметром dtFull при оценке достаточного времени на обработку следующего пятна. Ограничение. Если задано $\text{dtFull}>0$, то dtSpot должно быть меньше него: dtSpot - неотрицательное, $(\text{dtFull}>0) \&\& (\text{dtSpot}<\text{dtFull})$.

Аргументы

lDiag	<p>параметр, определяющий связность пятна по диагоналям:</p> <ul style="list-style-type: none"> • lDiag=0 - элементы являются соседними, если они граничат или по вертикали, или по горизонтали (по ходам ладьи, но не по ходам слона), • lDiag=1 - элементы являются соседними, если они граничат или по вертикали, или по горизонтали, или по диагоналям (по ходам ферзя). <p>Замечание. Как показывает практика, результат обработки видеоизображений существенно не зависит от параметра lDiag, но при lDiag=1 значительно увеличивает время работы функции. Если нет особой необходимости, рекомендуется задавать lDiag=0.</p>
lDropSpot	<p>параметр, задающий необходимость отбраковки найденных пятен по условиям НЕвхождения числа пикселов или размеров пятна в заданные диапазоны (см. ниже параметры nPxImin, nPxImax, dXYmin, dXYmax), статистика по отбракованным пятнам ведется в массиве dropSpot.</p> <p>lDropSpot=0 – найденные пятна НЕ отбраковываются, то есть все найденные пятна учитываются в массивах spot, pixels и отмечаются в выходной матрице dst, lDropSpot=1 – пятна с числом пикселов или размеров пятна не попадающими в заданные диапазоны отбраковываются, то есть не учитываются в массивах spot, pixels и не отмечаются в выходной матрице dst, статистика по отбракованным пятнам ведется в массиве dropSpot.</p> <p>Замечание. Задание параметра lDropSpot=1 приводит к увеличению работы функции и требует выделения памяти для массива dropSpot. Если нет особой необходимости, задавайте lDropSpot=0.</p>
dropSpot	<p>– массив, в который дополнительно собирается статистика по отбракованным пятнам: на каждый из 4 признаков отбраковки: nPxIMin, nPxImax, dXYmin, dXYmax, - отведено по 3 параметра формата int: dropSpot[4*3] +0 +1 +2</p> <ul style="list-style-type: none"> • 0 – dropSpot(nPxIMin): [nnSpot, nnPxl, dttSpot] • 3 - dropSpot(nPxImax): [nnSpot, nnPxl, dttSpot] • 6 - dropSpot(dXYmin) : [nnSpot, nnPxl, dttSpot] • 9 - dropSpot(dXYmax) : [nnSpot, nnPxl, dttSpot] <p>Перед обращением к функции массив dropSpot (размером 12 int) должен быть обнулен.</p> <p>При ldropSpot=0 статистика не ведется и массив dropSpot не требуется, то есть вместо него можно задать фиктивный параметр, например, 0.</p>
nPxIMin	<p>минимальное число пикселов в пятне,</p> <p>при lDropSpot=1: пятно отбраковывается, если число его пикселов меньше nPxIMin,</p> <p>при lDropSpot=0 проверок не производится, параметр можно задать 0.</p>
nPxImax	<p>максимальное число пикселов в пятне,</p> <ul style="list-style-type: none"> • при lDropSpot=1: пятно отбраковывается, если число его пикселов больше nPxImax, • при lDropSpot=0 проверок не производится, параметр можно задать 0.

Аргументы

dXYmin	<p>минимальная протяженность пятна по X или по Y в пикселях,</p> <ul style="list-style-type: none"> • при lDropSpot=1: пятно отбраковывается, если его протяженность меньше dXYMax, • при lDropSpot=0 проверок не производится, но параметр используется для определения шага поиска по строкам входной матрицы src (и начальной строки поиска): $dI = \max(1, dXYmin)$ <p>Замечание. Задание $dXYmin > 1$ приводит к просмотру входной матрицы src через каждые $dXYmin$ строк, что может значительно сократить время работы функции, так как пятна меньшего размера, попавшие между строк поиска "обходятся" (однако, не гарантирует этого, например, когда пятно занимает всю матрицу). Поэтому, независимо от параметра lDropSpot надо задавать $dXYmin$, по возможности, большим.</p>
dXYmax	<p>максимальная протяженность пятна по X или по Y в пикселях,</p> <ul style="list-style-type: none"> • при lDropSpot=1: пятно отбраковывается, если его протяженность больше dXYMax, • при lDropSpot=0 проверок не производится, параметр можно задать 0. <p>Ограничения: nPxlMin, nPxlMax, dXYmin, dXYmax - неотрицательные, ($nPxlMin \leq nPxlMax$), ($dXYmin \leq dXYmax$).</p>

Возвращает

Функция возвращает целое число:

- либо положительное целое, тогда это - nSpot - число найденных пятен,
- либо отрицательное целое, тогда это - nError - код ошибки.

Число найденных пятен nSpot включает в себя также служебное и фиктивные пятна (с номерами 0,256,512,768...). Таким образом, число реальных пятен равно nSpotReal = nSpot - 1 - [nSpot/256]

Если не найдено ни одного реального пятна, то nSpot=1 – одно служебное пятно.

Код ошибки возвращается, если в начале работы функции при проверке входных данных обнаружится недопустимая комбинация параметров: тогда происходит (принудительный) выход из программы с кодом ошибки:

```

dI = max(1,dXYmin);
if ( (nHeight<=0) || (nHeight < dI +2) || (1080<nHeight) ) return -1;
if ( (nWidth <=0) || (nWidth < dI +2) ||((nWidth & 3)!=0)) return -2;
if ( (dtSpot <0) || ((dtFull != 0) && (dtFull < dtSpot)) ) return -3;
if ( (nPxlMin <0) || (nPxlMax < nPxlMin) ) return -4;
if ( (dXYmin <0) || (dXYmax < dXYmin) ) return -5;
if ( mSpot <0) return -6;

```

При нормальном выходе из функции (без ошибок) в параметре Ymin служебного пятна сообщается дополнительный признак завершения программы nExitCode:

- spot[0].Ymin=0 – вся матрица src просмотрена, найдены все пятна, естественный выход из программы.
- spot[0].Ymin=1 – принудительный выход из программы по исчерпанию заданного времени работы программы dtFull>0.
- spot[0].Ymin=2 – принудительный выход из программы по нахождению заданного количества пятен mSpot>0.

Заметки

Предельные случаи

Следует выделить предельные случаи:

- одно сплошное пятно на всю матрицу - максимальное время работы функции и максимальный размер массива pixels (но минимальный размер массива spot),
- «шахматная доска» - однопиксельные пятна, расположенные по диагоналям - максимальный размер массива spot.

Алгоритм

В процессе поиска координаты всех пикселов каждого пятна сохраняются в массиве pixels:

- пиксели каждого пятна занимают непрерывную область в массиве pixels,
- параметр spot[i].noPxl i-го пятна в массиве spot указывает на начальный пиксел (i+1)-го пятна. В начале работы программы устанавливается указатель на начальный пиксел следующего пятна noPxl = 0.

1. Цикл поиск пятна (осуществляется построчным попиксельным проходом матрицы src), если очередной пиксел нулевой, то переход к следующему пикселу иначе (встретился ненулевой пиксел)

2. начало обработки (очередного) пятна:

- засечь время начала обработки пятна tSpot0=clock();
- устанавливаются указатели начала и конца очереди пикселов пятна в массиве pixels: noPxl0 = noPxl1 = noPxl.
- пикセル в матрице src обнуляется,
- координаты пикселя заносятся в массив pixels, указатель noPxl1 увеличивается на 1
- 3. while-цикл обработки пятна:
- выбирается пикセル noPxl0 из массива pixels, указатель noPxl0 увеличивается на 1
- просматриваются все его соседние элементы (с учетом параметра lDiag) координаты ненулевых пикселов записываются в массив pixels
- пиксели в матрице src обнуляются
- если указатели noPxl0, noPxl1 не равны (не все пиксели пятна и их соседи просмотрены), то переход на начало while-цикла // пятно найдено! noPxl0=noPxl1 - указывают на начальный пикSEL //следующего пятна в массиве pixels.

Вычисление минимального прямоугольника (Xmin, Ymin, Xmax, Ymax), объемлющего пятно

Если задан параметр lDropSpot=1, то проверка условий отбраковки пятна по признакам nPxlMin, nPxlMax, dXYmin, dXYmax если пятно отбраковано, то формирование записи массива dropSpot по соответствующему признаку, переход на поиск следующего пятна (noPxl не изменяется).

Меняется указатель noPxl=noPxl1.

Формирование параметров данного пятна в массиве spot[i] = (Xmin, Ymin, Xmax, Ymax, noPxl, 0, 0, 0)

Если задан параметр вычисления угла наклона пятна (lAngleSpot=1), то вычисление codeAngle, lBadSpot и запись в spot[i] =

(., ., ., ., codeAngle, lBadSpot, .)

Запись dtSpot=clock()-tSpot0 в spot[i] = (., ., ., ., ., ., dtSpot).

Выбор следующего пикселя, переход на начало цикла поиска очередного пятна (1).

4. Конец программы.

5.62 Переупорядочивание изображений

Группы

- **Блочное переупорядочивание**

Блочное перупорядочивание изображений.

5.62.1 Подробное описание

5.63 Блочное переупорядочивание

Блочное перупорядочивание изображений.

Группы

- nmppiSplitIntoBlocks

Преобразует изображение в последовательность квадратных блоков.

- nmppiMergeFromBlocks

Объединяет последовательность квадратных блоков в изображение.

5.63.1 Подробное описание

Блочное перупорядочивание изображений.

5.64 nmppiSplitIntoBlocks

Преобразует изображение в последовательность квадратных блоков.

Функции

- void nmppiSplitIntoBlocks8x8 (nm8s *pSrcImg, nm8s *pDstBlockSeq, int nWidth, int nHeight)

5.64.1 Подробное описание

Преобразует изображение в последовательность квадратных блоков.

Аргументы

pSrcImg	Исходное изображение Выходная последовательность блоков Ширина исходного изображения в пикселях. nWidth =[8,16,24...] Высота исходного изображения в пикселях. nHeight =[8,16,24...] Исходное изображение имеет вид
---------	---

[A00]	[A01]	[A02]	[A03]	[A04]	[A05]	[A06]	[A07]	[B00]	[B01]	[..]	[B07]	[C00]	[..]	[H07]
[A10]	[A11]	[A12]	[A13]	[A14]	[A15]	[A16]	[A17]	[B10]	[B11]	[..]	[B17]	[C10]	[..]	[H17]
[A20]	[A21]	[A22]	[A23]	[A24]	[A25]	[A26]	[A27]	[B20]	[B21]	[..]	[B27]	[C20]	[..]	[H27]
[A30]	[A31]	[A32]	[A33]	[A34]	[A35]	[A36]	[A37]	[B30]	[B31]	[..]	[B37]	[C30]	[..]	[H37]

.....

[A70]	[A71]	[A72]	[A73]	[A74]	[A75]	[A76]	[A77]	[B70]	[B71]	[..]	[B77]	[C70]	[..]	[H77]
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	------	-------	-------	------	-------

.....

[I00]	[I01]	[I02]	[I03]	[I04]	[I05]	[I06]	[I07]	[J00]	[J01]	[.....]				
[I10]	[I11]	[I12]	[I13]	[I14]	[I15]	[I16]	[I17]	[J00]	[I09]	[.....]				

.....Z77

Выходная последовательность блоков имеет вид

[A00|A01|A02|...|A06|A07|A20|A21|...|A77|B00|B01|B02|...|B07|B10|...|H77|
[I00|I01|I02|...|I06|I07|I10|I11|...|I77|J00|J01|J02|.....|Z77|

5.65 nmppiMergeFromBlocks

Объединяет последовательность квадратных блоков в изображение.

Функции

- void nmppiMergeFromBlocks8x8 ([nm8s](#) *pSrcBlockSeq, [nm8s](#) *pDstImg, int nWidth, int nHeight)

5.65.1 Подробное описание

Объединяет последовательность квадратных блоков в изображение.

Аргументы

<p><code>pSrcBlockSeq</code></p>	<p>Входная последовательность блоков . Результатирующее изображение Ширина исходного изображения в пикселях. <code>nWidth=[8,16,24..]</code> Высота исходного изображения в пикселях. <code>nHeight=[8,16,24..]</code> Входная последовательность блоков имеет вид</p> <p style="text-align: center;"><code>[A00 A01 A02 ... A06 A07 A20 A21 ... A77 B00 B01 B02 ... B07 B10 ... H77] [I00 I01 I02 ... I06 I07 I10 I11 ... I77 J00 J01 J02 Z77]</code></p> <p>Результатирующее изображение имеет вид</p>
----------------------------------	--

`[A00|A01|A02|A03|A04|A05|A06|A07|B00|B01|..|B07|C00|....H07]
[A10|A11|A12|A13|A14|A15|A16|A17|B10|B11|..|B17|C10|....H17]
[A20|A21|A22|A23|A24|A25|A26|A27|B20|B21|..|B27|C20|....H27]
[A30|A31|A32|A33|A34|A35|A36|A37|B30|B31|..|B37|C30|....H37]
.....
[A70|A71|A72|A73|A74|A75|A76|A77|B70|B71|..|B77|C70|....H77]
.....
[I00|I01|I02|I03|I04|I05|I06|I07|J00|J01|.....]
[I10|I11|I12|I13|I14|I15|I16|I17|J00|I09|.....]
.....Z77]`

5.66 nmppiRelease

Освобождение блоков памяти, выделенных функциями `nmppiCreate***`.

Функции

- `_ _INLINE_ _ void nmppiReleaseObject (nm64s *pKernel)`

5.66.1 Подробное описание

Освобождение блоков памяти, выделенных функциями `nmppiCreate***`.

5.67 Арифметические действия

5.68 Масочная фильтрация

Группы

- [КИХ-фильтрация](#)

Двумерная КИХ фильтрация

5.68.1 Подробное описание

5.69 Инициализация и копирование

5.70 Изменение размеров

5.71 Операции выборки

5.72 Функции поддержки

Группы

- [nmppiRelease](#)

Освобождение блоков памяти, выделенных функциями nmppiCreate***.

5.72.1 Подробное описание

5.73 Инициализация и копирование

Группы

- [nmppmCopyua](#)

Копирование подматрицы с невыровненной по границам 64- разрядного слова позиции в выровненную.

- [MTR_Copyau](#)

Копирование подматрицы с выровненной по границе 64-х разрядных слов позиции в невыровненную позицию.

- [MTR_Copy](#)

Функции копирования прямоугольных областей памяти между двумерными массивами.

5.73.1 Подробное описание

5.74 nmppmCopyua

Копирование подматрицы с невыровненной по границам 64- разрядного слова позиции в выровненную.

Функции

- void nmppmCopyua_8s ([nm8s](#) *pSrcMtr, int nSrcStride, int nSrcOffset, [nm8s](#) *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopyua_16s ([nm16s](#) *pSrcMtr, int nSrcStride, int nSrcOffset, [nm16s](#) *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopyua_32s ([nm32s](#) *pSrcMtr, int nSrcStride, int nSrcOffset, [nm32s](#) *pDstMtr, int nDstStride, int nHeight, int nWidth)

5.74.1 Подробное описание

Копирование подматрицы с невыровненной по границам 64- разрядного слова позиции в выровненную.

Аргументы

pSrcMtr	Исходная матрица.
nSrcStride	Ширина исходной матрицы в элементах.
nSrcOffset	Смещение в элементах от начала матрицы-источника.
nDstStride	Ширина результирующей матрицы в элементах.
nHeight	Число строк подматрицы.
nWidth	Число столбцов подматрицы.

Возвращаемые значения

pDstMtr	Результирующая матрица.
---------	-------------------------

Возвращает

void

Restrictions:

- Входная и выходная матрицы не должны перекрываться в памяти
- Число столбцов копируемой подматрицы nWidth должна быть кратно числу элементов в 64-х разрядном слове.

5.75 MTR_Copyau

Копирование подматрицы с выровненной по границе 64-х разрядных слов позиции в невыровненную позицию.

Функции

- void nmppmCopyau_8s ([nm8s](#) *pSrcMtr, int nSrcStride, [nm8s](#) *pDstMtr, int nDstStride, int nDstOffset, int nHeight, int nWidth)
- void nmppmCopyau_16s ([nm16s](#) *pSrcMtr, int nSrcStride, [nm16s](#) *pDstMtr, int nDstStride, int nDstOffset, int nHeight, int nWidth)
- void nmppmCopyau_32s ([nm32s](#) *pSrcMtr, int nSrcStride, [nm32s](#) *pDstMtr, int nDstStride, int nDstOffset, int nHeight, int nWidth)

5.75.1 Подробное описание

Копирование подматрицы с выровненной по границе 64-х разрядных слов позиции в невыровненную позицию.

Аргументы

pSrcMtr	Исходная матрица.
nSrcStride	Ширина исходной матрицы в элементах.

Возвращаемые значения

pDstMtr	Результирующая матрица.
---------	-------------------------

Аргументы

nDstStride	Ширина результирующей матрицы в элементах.
nDstOffset	Индекс столбца (в элементах) куда производится вставка.
nHeight	Число строк подматрицы.
nWidth	Число столбцов подматрицы.

Возвращает

void

Restrictions:

- Входная и выходная матрицы не должны перекрываться в памяти
- Число столбцов копируемой подматрицы nWidth должна быть кратной 64-битам.

5.76 MTR_Copy

Функции копирования прямоугольных областей памяти между двумерными массивами.

Функции

- void nmppmCopy_1 (**nm1** *pSrcMtr, int nSrcStride, **nm1** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_2s (**nm2s** *pSrcMtr, int nSrcStride, **nm2s** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_4s (**nm4s** *pSrcMtr, int nSrcStride, **nm4s** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_8s (**nm8s** *pSrcMtr, int nSrcStride, **nm8s** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_16s (**nm16s** *pSrcMtr, int nSrcStride, **nm16s** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_32s (**nm32s** *pSrcMtr, int nSrcStride, **nm32s** *pDstMtr, int nDstStride, int nHeight, int nWidth)
- void nmppmCopy_64s (**nm64s** *pSrcMtr, int nSrcStride, **nm64s** *pDstMtr, int nDstStride, int nHeight, int nWidth)

5.76.1 Подробное описание

Функции копирования прямоугольных областей памяти между двумерными массивами.

Аргументы

pSrcMtr	Исходная матрица.
nSrcStride	Ширина исходной матрицы в элементах.
nDstStride	Ширина результирующей матрицы в элементах.
nHeight	Число строк подматрицы.
nWidth	Число столбцов подматрицы.

Возвращаемые значения

pDstMtr	Результирующая матрица.
---------	-------------------------

Возвращает

void

Restrictions:

Выходная матрица не может быть перезаписана (т.е. помещена непосредственно на место входной)

5.77 MTR_fpResolve_Gauss

Решение системы линейных уравнений методом Гаусса.

Функции

- void MTR_fpResolve_Gauss (double *pSrcMtrA, double *pSrcVecB, double *pDstVecX, int n← Size)

5.77.1 Подробное описание

Решение системы линейных уравнений методом Гаусса.

$$A \cdot x = b$$

Аргументы

pSrcMtrA	Квадратная матрица уравнения A.
pSrcVecB	Свободный вектор.
nSize	Размер матрицы.

Возвращаемые значения

pDstVecX	Вектор решения x.
----------	-------------------

Возвращает

void

Restrictions:

Выбор ведущего элемента не производится - возможно деление на нуль.

Заметки

Функция в данный момент не оптимизирована.

5.78 MTR_fpResolve_PivotGauss

Решение системы линейных уравнений методом Гаусса с выбором ведущего элемента.

Функции

- void MTR_fpResolve_PivotGauss (double *pSrcMtrAB, double *pDstVecX, int nSize)

5.78.1 Подробное описание

Решение системы линейных уравнений методом Гаусса с выбором ведущего элемента.

$$A \cdot X = B$$

Аргументы

pSrcMtrAB	Матрица уравнения A, дополненная справа вектором B.
nSize	Высота матрицы.

Возвращаемые значения

pDstVecX	Вектор решения X.
----------	-------------------

Возвращает

void

5.79 nmppmLUDecomp

LU-разложение матрицы

Функции

- void nmppmLUDecomp_64f (const double *A, double *L, double *U, int N)

5.79.1 Подробное описание

LU-разложение матрицы

Аргументы

in	A	Матрица A
out	L	Левая диагональная матрица L
out	U	верхняя диагональная матрица U
in	N	Размерность матрицы

5.80 nmppmLUResolve

Решение системы линейных уравнений из LU-разложения $L*y = b$; $U*x = y$.

Функции

- void nmppmLUResolve_64f (const double *L, const double *U, const double *b, double *x, double *y, int N)

5.80.1 Подробное описание

Решение системы линейных уравнений из LU-разложения $L*y = b$; $U*x = y$.

Аргументы

in	L	Левая диагональная матрица L
in	U	верхняя диагональная матрица U
in	b	вектор b
out	x	вектор x
out	y	вектор y
in	N	размер

More details

5.81 nmppmMul_mm

Умножение матрицы на матрицу.

Функции

- void nmppmMul_mm_2s64s ([nm2s](#) *pSrcMtr1, int nHeight1, int nWidth1, [nm64s](#) *pSrcMtr2, [nm64s](#) *pDstMtr, int nWidth2)
- void nmppmMul_mm_4s64s ([nm4s](#) *pSrcMtr1, int nHeight1, int nWidth1, [nm64s](#) *pSrcMtr2, [nm64s](#) *pDstMtr, int nWidth2)
- void nmppmMul_mm_8s8s ([nm8s](#) *pSrcMtr1, int nHeight1, int nWidth1, [nm8s](#) *pSrcMtr2, [nm8s](#) *pDstMtr, int nWidth2)

- void nmppmMul_mm_8s16s (**nm8s** *pSrcMtr1, int nHeight1, int nWidth1, **nm16s** *pSrcMtr2, **nm16s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_8s32s (**nm8s** *pSrcMtr1, int nHeight1, int nWidth1, **nm32s** *pSrcMtr2, **nm32s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_8s64s (**nm8s** *pSrcMtr1, int nHeight1, int nWidth1, **nm64s** *pSrcMtr2, **nm64s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_16s16s (**nm16s** *pSrcMtr1, int nHeight1, int nWidth1, **nm16s** *pSrcMtr2, **nm16s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_16s32s (**nm16s** *pSrcMtr1, int nHeight1, int nWidth1, **nm32s** *pSrcMtr2, **nm32s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_16s64s (**nm16s** *pSrcMtr1, int nHeight1, int nWidth1, **nm64s** *pSrcMtr2, **nm64s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_32s32s (**nm32s** *pSrcMtr1, int nHeight1, int nWidth1, **nm32s** *pSrcMtr2, **nm32s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_32s64s (**nm32s** *pSrcMtr1, int nHeight1, int nWidth1, **nm64s** *pSrcMtr2, **nm64s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_64s64s (**nm64s** *pSrcMtr1, int nHeight1, int nWidth1, **nm64s** *pSrcMtr2, **nm64s** *pDstMtr, int nWidth2)
- void nmppmMul_mm_32f (float *pSrcMtr1, int nHeight1, int nStride1, float *pSrcMtr2, int nWidth1, int nStride2, float *pDstMtr, int nWidth2, int nStrideDst, int bPlusDst)
- void nmppmMul_mt_32f (float *pSrcMtr1, int nHeight1, int nStride1, float *pSrcMtr2, int nWidth1, int nStride2, float *pDstMtr, int nWidth2, int nStrideDst, int bPlusDst)

5.81.1 Подробное описание

Умножение матрицы на матрицу.

Аргументы

pSrcMtr1	Исходная матрица.
pSrcMtr2	Матрица-множитель.
nHeight1	Число строк исходной матрицы.
nWidth1	Число столбцов исходной матрицы.
nWidth2	Число столбцов матрицы множителя.
nStride	Адресное смещение между строками матрицы.
bPlusDst	Флаг - указание прибавить к произведению прежнее значение Dst

Возвращаемые значения

pDstMtr	Результирующая матрица.
---------	-------------------------

Возвращает

void

5.82 nmppmMul_mv_

Умножение матрицы на вектор.

Функции

- void nmppmMul_mv_8s64s (**nm8s** *pSrcMtr, **nm64s** *pSrcVec, **nm64s** *pDstVec, int nHeight, int nWidth)
- void nmppmMul_mv_16s64s (**nm16s** *pSrcMtr, **nm64s** *pSrcVec, **nm64s** *pDstVec, int nHeight, int nWidth)
- void nmppmMul_mv_32s64s (**nm32s** *pSrcMtr, **nm64s** *pSrcVec, **nm64s** *pDstVec, int nHeight, int nWidth)

5.82.1 Подробное описание

Умножение матрицы на вектор.

Аргументы

pSrcMtr	Исходная матрица.
pSrcVec	Вектор-множитель.
pSrcVec8	Вектор-множитель размерности 8.
nHeight	Число строк исходной матрицы.
nWidth	Число столбцов исходной матрицы.

Возвращаемые значения

pDstVec	Результирующий вектор.
----------------	------------------------

Возвращает

void

5.83 nmppmMul_mv_8xH

Умножение матрицы на вектор.

Функции

- void nmppmMul_mv_8s16s_8xH (**v8nm8s** *pSrcMtr, **v8nm16s** *pSrcVec, **nm16s** *pDstVec, int nHeight)
- void nmppmMul_mv_16s16s_8xH (**v8nm16s** *pSrcMtr, **v8nm16s** *pSrcVec, **nm16s** *pDstVec, int nHeight)

5.83.1 Подробное описание

Умножение матрицы на вектор.

Аргументы

pSrcMtr	Исходная матрица.
pSrcVec	Вектор-множитель.
pSrcVec8	Вектор-множитель размерности 8.
nHeight	Число строк исходной матрицы.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.84 nmppmMul_mv__AddC

Умножение матрицы на вектор с добавлением константы.

Функции

- void nmppmMul_mv__AddC ([v2nm32s](#) *pSrcMtr, [v2nm32s](#) *pnSrcVec, int nAddVal, [nm32s](#) *p← Dst Vec, int nHeight)

5.84.1 Подробное описание

Умножение матрицы на вектор с добавлением константы.

Аргументы

pSrcMtr	Исходная матрица.
pSrcVec	Вектор-множитель.
pSrcVec	Вектор-множитель размерности 2.
nAddVal	Константа.
nHeight	Число строк исходной матрицы. nHeight = [0, 2, 4, ...].
nWidth	Число столбцов исходной матрицы.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.85 MTR_ProdUnitV

Умножение матрицы на единичный вектор.

Функции

- void MTR_ProdUnitV_16s_4xH ([v4nm16s](#) *pSrcMtr, [nm16s](#) *pDstVec, int nHeight)
- void MTR_ProdUnitV_16s_16xH ([v16nm8s](#) *pSrcMtr, [nm16s](#) *pDstVec, int nHeight)

5.85.1 Подробное описание

Умножение матрицы на единичный вектор.

$$pDstVec(i) = \sum_{j=0}^{w-1} pSrcMtr(i, j)$$

Данная функция эквивалентна суммированию столбцов матрицы. Ширины матрицы, для которых имеется реализация данной функции указываются в ее названии.

Аргументы

pSrcMtr	Матрица.
nHeight	Число строк матрицы. nHeight=[128,256,...]

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.86 MTR_Malloc

Распределение памяти для матриц библиотеки.

Начало и конец распределяемой памяти выравнивается на начало 64-х разрядного слова.

Распределение памяти для матриц библиотеки.

Начало и конец распределяемой памяти выравнивается на начало 64-х разрядного слова.

Аргументы

nHeight	Число строк в матрице.
nWidth	Число столбцов в матрице.
hint	Номер банка памяти. Может принимать значения MEM_LOCAL, MEM_GLOBAL.

Заметки

Память, распределенная с помощью функций MTR_Malloc должна освобождаться с помощью функции MTR_Free.

5.87 MTR_Free

Освобождение памяти для матриц.

Функции

- __INLINE__ void MTR_Free (void *ptr)

5.87.1 Подробное описание

Освобождение памяти для матриц.

Заметки

Данная функция должна вызываться только для векторов, распределенных с помощью функций MTR_Malloc.

5.88 MTR_Addr

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

Функции

- `__INLINE__ nm1 * MTR_Addr_1 (nm1 *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm2s * MTR_Addr_2s (nm2s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm4s * MTR_Addr_4s (nm4s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm8s * MTR_Addr_8s (nm8s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm16s * MTR_Addr_16s (nm16s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm32s * MTR_Addr_32s (nm32s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm64s * MTR_Addr_64s (nm64s *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm2u * MTR_Addr_2u (nm2u *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm4u * MTR_Addr_4u (nm4u *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm8u * MTR_Addr_8u (nm8u *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm16u * MTR_Addr_16u (nm16u *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm32u * MTR_Addr_32u (nm32u *pMTR, int nWidth, int nY, int nX)`
- `__INLINE__ nm64u * MTR_Addr_64u (nm64u *pMTR, int nWidth, int nY, int nX)`

5.88.1 Подробное описание

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

Аргументы

pMtr	Входная матрица.
nWidth	Ширина таблицы в элементах.
nY	Номер строки.
nX	Номер колонки.

Возвращает

Адрес ячейки памяти.

Заметки

Для ускорения работы на РС возможно использование макроса ADDR(ptr, index), который раскрывается на РС как (ptr+index), а на NM как вызов функции MTR_Addr.

5.89 MTR_SetVal

Записываете число в элемент матрицы.

Функции

- `__INLINE__ void MTR_SetVal_1 (nm1 *pMtr, int nWidth, int nY, int nX, int1b nVal)`
- `__INLINE__ void MTR_SetVal_2s (nm2s *pMtr, int nWidth, int nY, int nX, int2b nVal)`
- `__INLINE__ void MTR_SetVal_4s (nm4s *pMtr, int nWidth, int nY, int nX, int4b nVal)`
- `__INLINE__ void MTR_SetVal_8s (nm8s *pMtr, int nWidth, int nY, int nX, int8b nVal)`
- `__INLINE__ void MTR_SetVal_16s (nm16s *pMtr, int nWidth, int nY, int nX, int16b nVal)`
- `__INLINE__ void MTR_SetVal_32s (nm32s *pMtr, int nWidth, int nY, int nX, int32b nVal)`
- `__INLINE__ void MTR_SetVal_64s (nm64s *pMtr, int nWidth, int nY, int nX, int64b nVal)`
- `__INLINE__ void MTR_SetVal_2u (nm2u *pMtr, int nWidth, int nY, int nX, uint2b nVal)`
- `__INLINE__ void MTR_SetVal_4u (nm4u *pMtr, int nWidth, int nY, int nX, uint4b nVal)`
- `__INLINE__ void MTR_SetVal_8u (nm8u *pMtr, int nWidth, int nY, int nX, uint8b nVal)`
- `__INLINE__ void MTR_SetVal_16u (nm16u *pMtr, int nWidth, int nY, int nX, uint16b nVal)`
- `__INLINE__ void MTR_SetVal_32u (nm32u *pMtr, int nWidth, int nY, int nX, uint32b nVal)`
- `__INLINE__ void MTR_SetVal_64u (nm64u *pMtr, int nWidth, int nY, int nX, uint64b nVal)`

5.89.1 Подробное описание

Записывает число в элемент матрицы.

$$pMtr[nY][nX] = nVal$$

Аргументы

pMtr	Матрица.
nWidth	Ширина матрицы в элементах
nY	Номер строки
nX	Номер столбца
nVal	Значение элемента

Возвращает

`void`

5.90 MTR_SetVal

Считывает значение элемента матрицы.

Функции

- `__INLINE__ void MTR_SetVal_1 (nm1 *pMtr, int nWidth, int nY, int nX, int1b *nVal)`
- `__INLINE__ void MTR_SetVal_2s (nm2s *pMtr, int nWidth, int nY, int nX, int2b *nVal)`
- `__INLINE__ void MTR_SetVal_4s (nm4s *pMtr, int nWidth, int nY, int nX, int4b *nVal)`

- `__INLINE__ void MTR_GetVal_8s (nm8s *pMtr, int nWidth, int nY, int nX, int8b *nVal)`
- `__INLINE__ void MTR_GetVal_16s (nm16s *pMtr, int nWidth, int nY, int nX, int16b *nVal)`
- `__INLINE__ void MTR_GetVal_32s (nm32s *pMtr, int nWidth, int nY, int nX, int32b *nVal)`
- `__INLINE__ void MTR_GetVal_64s (nm64s *pMtr, int nWidth, int nY, int nX, int64b *nVal)`
- `__INLINE__ void MTR_GetVal_2u (nm2u *pMtr, int nWidth, int nY, int nX, uint2b *nVal)`
- `__INLINE__ void MTR_GetVal_4u (nm4u *pMtr, int nWidth, int nY, int nX, uint4b *nVal)`
- `__INLINE__ void MTR_GetVal_8u (nm8u *pMtr, int nWidth, int nY, int nX, uint8b *nVal)`
- `__INLINE__ void MTR_GetVal_16u (nm16u *pMtr, int nWidth, int nY, int nX, uint16b *nVal)`
- `__INLINE__ void MTR_GetVal_32u (nm32u *pMtr, int nWidth, int nY, int nX, uint32b *nVal)`
- `__INLINE__ void MTR_GetVal_64u (nm64u *pMtr, int nWidth, int nY, int nX, uint64b *nVal)`

5.90.1 Подробное описание

Считывает значение элемента матрицы.

$$nVal = pMtr[nY][nX]$$

Аргументы

<code>pMtr</code>	Матрица.
<code>nWidth</code>	Ширина матрицы в элементах
<code>nY</code>	Номер строки
<code>nX</code>	Номер столбца

Возвращаемые значения

<code>nVal</code>	Значение элемента
-------------------	-------------------

Возвращает

`void`

5.91 Функции поддержки

Группы

- [MTR_Malloc](#)

Распределение памяти для матриц библиотеки.

Начало и конец распределяемой памяти выравнивается на начало 64-х разрядного слова.

- [MTR_Free](#)

Освобождение памяти для матриц.

- [MTR_Addr](#)

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

- [MTR_SetVal](#)

Записывает число в элемент матрицы.

- [MTR_GetVal](#)

Считывает значение элемента матрицы.

5.91.1 Подробное описание

5.92 Векторно-матричные операции

Группы

- [nmpprmMul_mm](#)

Умножение матрицы на матрицу.

- [nmpprmMul_mv_](#)

Умножение матрицы на вектор.

- [nmpprmMul_mv_8xH](#)

Умножение матрицы на вектор.

- [nmpprmMul_mv__AddC](#)

Умножение матрицы на вектор с добавлением константы.

- [MTR_ProdUnitV](#)

Умножение матрицы на единичный вектор.

5.92.1 Подробное описание

5.93 Обращение матрицы

Группы

- [MTR_fpResolve_Gauss](#)

Решение системы линейных уравнений методом Гаусса.

- [MTR_fpResolve_PivotGauss](#)

Решение системы линейных уравнений методом Гаусса с выбором ведущего элемента.

- [nmpprmLUDecomp](#)

LU-разложение матрицы

- [nmpprmLUResolve](#)

Решение системы линейных уравнений из LU-разложения $L*y = b; U*x = y$.

5.93.1 Подробное описание

5.94 FFT-256

Функции

- [void FFT_Fwd256Set6bit \(\)](#)

Устанавливает 6-битную точность вычислений

- [void FFT_Fwd256Set7bit \(\)](#)

Устанавливает 7-битную точность вычислений

- [void FFT_Fwd256 \(nm32sc *GSrcBuffer, nm32sc *LDstBuffer, void *LBuffer, void *GBuffer, int ShiftR=-1\)](#)

Прямое быстрое преобразование Фурье-256.

5.94.1 Подробное описание

5.94.2 Функции

5.94.2.1 FFT_Fwd256()

```
void FFT_Fwd256 (
    nm32sc * GSrcBuffer,
    nm32sc * LDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR = -1 )
```

Прямое быстрое преобразование Фурье-256.

\~russian Функция выполняет дискретное комплексное 256-точечное преобразование Фурье на базе алгоритма БПФ по основанию 16-16

Аргументы

in	GSrcBuffer	Входной массив размером 256 64-р. слов
out	LDstBuffer	Результирующий массив размером 256*3 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 256*3 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 256*2 64-р. слов
in	ShiftR	Коэффициент нормализации, выполняет арифметический сдвиг результирующего массива на ShiftR бит вправо для получения нормализованного массива LDstBuffer. При передаче значения по умолчанию (-1) ShiftR автоматически принимается равным 14 если ранее установлена точность 7-бит функцией FFT_Fwd256Set7bit() и 12 - если ранее установлена точность 6-бит функцией FFT_Fwd256Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer	LDstBuffer	LBuffer	GBuffer	ShiftR	clocks
L	L	L	L	-1	21.54
L	L	L	G	-1	16.60
L	L	G	L	-1	22.17
L	L	G	G	-1	20.98

L	L	L	L	-1	21.54
L	L	L	G	-1	16.60
L	L	G	L	-1	22.17
L	L	G	G	-1	20.98

L	G	L	L	-1	20.53
L	G	L	G	-1	17.56
L	G	G	L	-1	21.17
L	G	G	G	-1	21.94
G	L	L	L	-1	20.57
G	L	L	G	-1	15.64
G	L	G	L	-1	21.21
G	L	G	G	-1	20.02
G	G	L	L	-1	19.57
G	G	L	G	-1	16.59
G	G	G	L	-1	20.20
G	G	G	G	-1	20.97
L	L	L	L	0	21.51
L	L	L	G	0	16.58
L	L	G	L	0	22.15
L	L	G	G	0	20.96
L	G	L	L	0	20.51
L	G	L	G	0	17.53
L	G	G	L	0	21.14
L	G	G	G	0	21.91
G	L	L	L	0	20.55
G	L	L	G	0	15.62
G	L	G	L	0	21.19
G	L	G	G	0	20.00
G	G	L	L	0	19.54
G	G	L	G	0	16.57
G	G	G	L	0	20.18
G	G	G	G	0	20.95

5.95 IFFT-256

Функции

- void **FFT_Inv256Set6bit ()**
Устанавливает 6-битную точность вычислений
- void **FFT_Inv256Set7bit ()**
Устанавливает 7-битную точность вычислений
- void **FFT_Inv256 (nm32sc *GSrcBuffer, nm32sc *GDstBuffer, void *LBuffer, void *GBuffer, int ShiftR1=8, int ShiftR2=-1)**
Обратное быстрое преобразование Фурье. ОБПФ-256.

5.95.1 Подробное описание

5.95.2 Функции

5.95.2.1 FFT_Inv256()

```
void FFT_Inv256 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR1 = 8,
    int ShiftR2 = -1 )
```

Обратное быстрое преобразование Фурье. ОБПФ-256.

Функция выполняет обратное дискретное комплексное 256-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 16-16.

Аргументы

in	GSrcBuffer	Входной массив размером 256 64-р. слов
out	GDstBuffer	Результирующий массив размером 256 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 256*3 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 256*2 64-р. слов
in	ShiftR1	Промежуточный сдвиг результатов на ShiftR1 бит вправо (первая нормализация). Необходимо для предотвращения переполнения. По умолчанию равен 8
in	ShiftR2	Заключительный сдвиг результатов на ShiftR2 бит вправо (вторая нормализация) в конце вычисления обратного БПФ. По умолчанию ShiftR2 принимается равным 14 при установленной точности 7-бит с помощью функции FFT_Inv256Set7bit() и 12 - при точности 6-бит, установленной с помощью функции FFT_Inv256Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer| GDstBuffer| LBuffer | GBuffer | ShiftR1 | ShiftR2 | clocks

L	L	L	L	8	-1	26.76
L	L	L	L	0	-1	26.76
L	L	L	G	8	-1	19.07
L	L	L	G	0	-1	19.07
L	L	G	L	8	-1	22.87
L	L	G	L	0	-1	22.87
L	L	G	G	8	-1	20.77
L	L	G	G	0	-1	20.77
L	G	L	L	8	-1	25.75
L	G	L	L	0	-1	25.75
L	G	L	G	8	-1	18.06
L	G	L	G	0	-1	18.06
L	G	G	L	8	-1	23.82
L	G	G	L	0	-1	23.82
L	G	G	G	8	-1	21.72
L	G	G	G	0	-1	21.72
G	L	L	L	8	-1	25.80
G	L	L	L	0	-1	25.80
G	L	L	G	8	-1	18.10
G	L	L	G	0	-1	18.10
G	L	G	L	8	-1	21.91
G	L	G	L	0	-1	21.91
G	L	G	G	8	-1	19.80
G	L	G	G	0	-1	19.80
G	G	L	L	8	-1	24.79
G	G	L	L	0	-1	24.79
G	G	L	G	8	-1	17.09
G	G	L	G	0	-1	17.09
G	G	G	L	8	-1	22.86
G	G	G	L	0	-1	22.86
G	G	G	G	8	-1	20.76
G	G	G	G	0	-1	20.76

L	L	L	L	8	0	26.74
L	L	L	G	0	0	26.74
L	L	L	G	8	0	19.04
L	L	L	G	0	0	19.04
L	L	G	L	8	0	22.85
L	L	G	L	0	0	22.85
L	L	G	G	8	0	20.75
L	L	G	G	0	0	20.75
L	G	L	L	8	0	25.73
L	G	L	L	0	0	25.73
L	G	L	G	8	0	18.03
L	G	L	G	0	0	18.03
L	G	G	L	8	0	23.80
L	G	G	L	0	0	23.80
L	G	G	G	8	0	21.70
L	G	G	G	0	0	21.70
G	L	L	L	8	0	25.77
G	L	L	L	0	0	25.77
G	L	L	G	8	0	18.08
G	L	L	G	0	0	18.08
G	L	G	L	8	0	21.88
G	L	G	L	0	0	21.88
G	L	G	G	8	0	19.78
G	L	G	G	0	0	19.78
G	G	L	L	8	0	24.76
G	G	L	L	0	0	24.76
G	G	L	G	8	0	17.07
G	G	L	G	0	0	17.07
G	G	G	L	8	0	22.83
G	G	G	L	0	0	22.83
G	G	G	G	8	0	20.73
G	G	G	G	0	0	20.73

5.95.2.2 FFT_Inv256Set6bit()

```
void FFT_Inv256Set6bit ()
```

Устанавливает 6-битную точность вычислений

5.96 FFT-512

Функции

- void FFT_Fwd512Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Fwd512Set7bit ()

Устанавливает 7-битную точность вычислений
- void **FFT_Fwd512** (**nm32sc** *GSrcBuffer, **nm32sc** *GDstBuffer, void *LBuffer, void *GBuffer, int ShiftR=-1)

Прямое быстрое преобразование Фурье-512.

5.96.1 Подробное описание

5.96.2 Функции

5.96.2.1 FFT_Fwd512()

```
void FFT_Fwd512 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR = -1 )
```

Прямое быстрое преобразование Фурье-512.

Функция выполняет дискретное комплексное 512-точечное преобразование Фурье на базе алгоритма БПФ по основанию 2-16-16

Аргументы

in	GSrcBuffer	Входной массив размером 512 64-р. слов
out	GDstBuffer	Результирующий массив размером 512 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 512*2 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 512*3 64-р. слов
in	ShiftR	Коэффициент нормализации, выполняет арифметический сдвиг результирующего массива на ShiftR бит вправо для получения нормализованного массива LDstBuffer. При передаче значения по умолчанию (-1) ShiftR автоматически принимается равным 14 если ранее установлена точность 7-бит функцией FFT_Fwd512Set7bit() и 12 - если ранее установлена точность 6-бит функцией FFT_Fwd512Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer	GDstBuffer	LBuffer	GBuffer	ShiftR	clocks
L	L	L	L	-1	24.12
L	L	L	G	-1	19.29
L	L	G	L	-1	22.81
L	L	G	G	-1	21.62
L	G	L	L	-1	23.10
L	G	L	G	-1	18.27
L	G	G	L	-1	23.77
L	G	G	G	-1	22.58
G	L	L	L	-1	23.06
G	L	L	G	-1	18.23
G	L	G	L	-1	23.79
G	L	G	G	-1	22.60
G	G	L	L	-1	22.04
G	G	L	G	-1	17.21
G	G	G	L	-1	24.75
G	G	G	G	-1	23.56

L		L		L		0		24.11
L		L		G		0		19.28
L		L		G		0		22.80
L		L		G		0		21.61
L		G		L		0		23.09
L		G		L		0		18.26
L		G		G		0		23.76
L		G		G		0		22.57
G		L		L		0		23.05
G		L		L		0		18.22
G		L		G		0		23.78
G		L		G		0		22.59
G		G		L		0		22.03
G		G		L		0		17.20
G		G		G		0		24.74
G		G		G		0		23.55

5.97 IFFT-512

Функции

- void FFT_Inv512Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Inv512Set7bit ()

Устанавливает 7-битную точность вычислений
- void FFT_Inv512 (nm32sc *GSrcBuffer, nm32sc *LDstBuffer, void *LBuffer, void *GBuffer, int ShiftR1=9, int ShiftR2=-1)

Обратное быстрое преобразование Фурье. ОБПФ-512.

5.97.1 Подробное описание

5.97.2 Функции

5.97.2.1 FFT_Inv512()

```
void FFT_Inv512 (
    nm32sc * GSrcBuffer,
    nm32sc * LDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR1 = 9,
    int ShiftR2 = -1 )
```

Обратное быстрое преобразование Фурье. ОБПФ-512.

Функция выполняет обратное дискретное комплексное 512-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 2-16-16.

Аргументы

in	GSrcBuffer	Входной массив размером 512 64-р. слов
out	LDstBuffer	Результирующий массив размером 512 64-р. слов

Аргументы

in	LBuffer	Временный массив на локальной шине (Local Bus) размером 512*3 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 512*3 64-р. слов
in	ShiftR1	Промежуточный сдвиг результатов на ShiftR1 бит вправо (первая нормализация). Необходимо для предотвращения переполнения. По умолчанию равен 9
in	ShiftR2	Заключительный сдвиг результатов на ShiftR2 бит вправо (вторая нормализация) в конце вычисления обратного БПФ. По умолчанию ShiftR2 принимается равным 14 при установленной точности 7-бит с помощью функции FFT_Inv512Set7bit() и 12 - при точности 6-бит, установленной с помощью функции FFT_Inv512Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer| LDstBuffer| LBuffer | GBuffer | ShiftR1 | ShiftR2 | clocks

L	L	L	L	9	-1	23.41
L	L	L	L	0	-1	23.40
L	L	L	G	9	-1	19.39
L	L	L	G	0	-1	19.38
L	L	G	L	9	-1	24.86
L	L	G	L	0	-1	24.86
L	L	G	G	9	-1	26.47
L	L	G	G	0	-1	26.46
L	G	L	L	9	-1	22.39
L	G	L	L	0	-1	22.38
L	G	L	G	9	-1	20.35
L	G	L	G	0	-1	20.34
L	G	G	L	9	-1	23.84
L	G	G	L	0	-1	23.84
L	G	G	G	9	-1	27.43
L	G	G	G	0	-1	27.42
G	L	L	L	9	-1	22.35
G	L	L	L	0	-1	22.34
G	L	L	G	9	-1	18.34
G	L	L	G	0	-1	18.33
G	L	G	L	9	-1	25.84
G	L	G	L	0	-1	25.84
G	L	G	G	9	-1	27.45
G	L	G	G	0	-1	27.44
G	G	L	L	9	-1	21.33
G	G	L	L	0	-1	21.33
G	G	L	G	9	-1	19.30
G	G	L	G	0	-1	19.29
G	G	G	L	9	-1	24.83
G	G	G	L	0	-1	24.82
G	G	G	G	9	-1	28.41
G	G	G	G	0	-1	28.41
L	L	L	L	9	0	23.40
L	L	L	L	0	0	23.39
L	L	L	G	9	0	19.38

L	L	L	G	0	0	19.37
L	L	G	L	9	0	24.85
L	L	G	L	0	0	24.85
L	L	G	G	9	0	26.45
L	L	G	G	0	0	26.45
L	G	L	L	9	0	22.38
L	G	L	L	0	0	22.37
L	G	L	G	9	0	20.34
L	G	L	G	0	0	20.33
L	G	G	L	9	0	23.83
L	G	G	L	0	0	23.83
L	G	G	G	9	0	27.41
L	G	G	G	0	0	27.41
G	L	L	L	9	0	22.34
G	L	L	L	0	0	22.33
G	L	L	G	9	0	18.33
G	L	L	G	0	0	18.32
G	L	G	L	9	0	25.83
G	L	G	L	0	0	25.83
G	L	G	G	9	0	27.43
G	L	G	G	0	0	27.43
G	G	L	L	9	0	21.32
G	G	L	L	0	0	21.31
G	G	L	G	9	0	19.29
G	G	L	G	0	0	19.28
G	G	G	L	9	0	24.81
G	G	G	L	0	0	24.81
G	G	G	G	9	0	28.40
G	G	G	G	0	0	28.39

5.98 FFT-1024

Функции

- void FFT_Fwd1024Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Fwd1024Set7bit ()

Устанавливает 7-битную точность вычислений
- void FFT_Fwd1024 (nm32sc *GSrcBuffer, nm32sc *LDstBuffer, void *LBuffer, void *GBuffer, int ShiftR=-1)

Прямое быстрое преобразование Фурье-1024.

5.98.1 Подробное описание

5.98.2 Функции

5.98.2.1 FFT_Fwd1024()

```
void FFT_Fwd1024 (
    nm32sc * GSrcBuffer,
    nm32sc * LDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR = -1 )
```

Прямое быстрое преобразование Фурье-1024.

Функция выполняет дискретное комплексное 1024-точечное преобразование Фурье на базе алгоритма БПФ по основанию 2-32-16

Аргументы

in	GSrcBuffer	Входной массив размером 1024 64-р. слов
out	LDstBuffer	Результирующий массив размером 1024 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 1024*3 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 1024*3 64-р. слов
in	ShiftR	Коэффициент нормализации, выполняет арифметический сдвиг результирующего массива на ShiftR бит вправо для получения нормализованного массива LDstBuffer. При передаче значения по умолчанию (-1) ShiftR автоматически принимается равным 14 если ранее установлена точность 7-бит функцией FFT_Fwd1024Set7bit() и 12 - если ранее установлена точность 6-бит функцией FFT_Fwd1024Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer| LDstBuffer| LBuffer | GBuffer | Shift R | clocks

L	L	L	L	-1	22.55
L	L	L	G	-1	20.52
L	L	G	L	-1	26.08
L	L	G	G	-1	25.93
L	G	L	L	-1	21.53
L	G	L	G	-1	21.48
L	G	G	L	-1	25.06
L	G	G	G	-1	26.89
G	L	L	L	-1	21.55
G	L	L	G	-1	19.51
G	L	G	L	-1	27.03
G	L	G	G	-1	26.87
G	G	L	L	-1	20.52
G	G	L	G	-1	20.48
G	G	G	L	-1	26.00
G	G	G	G	-1	27.84
L	L	L	L	0	22.55
L	L	L	G	0	20.51
L	L	G	L	0	26.08
L	L	G	G	0	25.92
L	G	L	L	0	21.52
L	G	L	G	0	21.48
L	G	G	L	0	25.05
L	G	G	G	0	26.89
G	L	L	L	0	21.54
G	L	L	G	0	19.51
G	L	G	L	0	27.02
G	L	G	G	0	26.87
G	G	L	L	0	20.52
G	G	L	G	0	20.47
G	G	G	L	0	26.00
G	G	G	G	0	27.83

5.99 IFFT-1024

Функции

- void FFT_Inv1024Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Inv1024Set7bit ()

Устанавливает 7-битную точность вычислений
- void **FFT_Inv1024** (**nm32sc** *GSrcBuffer, **nm32sc** *GDstBuffer, void *LBuffer, void *GBuffer, int ShiftR1=10, int ShiftR2=-1)

Обратное быстрое преобразование Фурье. ОБПФ-1024.

5.99.1 Подробное описание

5.99.2 Функции

5.99.2.1 FFT_Inv1024()

```
void FFT_Inv1024 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR1 = 10,
    int ShiftR2 = -1 )
```

Обратное быстрое преобразование Фурье. ОБПФ-1024.

Функция выполняет обратное дискретное комплексное 1024-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 2-16-16.

Аргументы

in	GSrcBuffer	Входной массив размером 1024 64-р. слов
out	GDstBuffer	Результирующий массив размером 1024 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 1024*3 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 1024*3 64-р. слов
in	ShiftR1	Промежуточный сдвиг результатов на ShiftR1 бит вправо (первая нормализация). Необходимо для предотвращения переполнения. По умолчанию равен 10
in	ShiftR2	Заключительный сдвиг результатов на ShiftR2 бит вправо (вторая нормализация) в конце вычисления обратного БПФ. По умолчанию ShiftR2 принимается равным 14 при установленной точности 7-бит с помощью функции FFT_Inv1024Set7bit() и 12 - при точности 6-бит, установленной с помощью функции FFT_Inv1024Set6bit() .

Возвращает

```
void
```

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer	GDstBuffer	LBuffer	GBuffer	ShiftR1	ShiftR2	clocks
------------	------------	---------	---------	---------	---------	--------

L	L	L	L	10	-1	28.82
L	L	L	L	0	-1	28.82
L	L	L	G	10	-1	23.90
L	L	L	G	0	-1	23.90
L	L	G	L	10	-1	26.60
L	L	G	L	0	-1	26.60
L	L	G	G	10	-1	25.55
L	L	G	G	0	-1	25.55
L	G	L	L	10	-1	27.80
L	G	L	L	0	-1	27.80
L	G	L	G	10	-1	22.88
L	G	L	G	0	-1	22.88
L	G	G	L	10	-1	27.57
L	G	G	L	0	-1	27.57
L	G	G	G	10	-1	26.52
L	G	G	G	0	-1	26.52
G	L	L	L	10	-1	27.81
G	L	L	L	0	-1	27.81
G	L	L	G	10	-1	22.89
G	L	L	G	0	-1	22.89
G	L	G	L	10	-1	27.55
G	L	G	L	0	-1	27.55
G	L	G	G	10	-1	26.50
G	L	G	G	0	-1	26.50
G	G	L	L	10	-1	26.79
G	G	L	L	0	-1	26.79
G	G	L	G	10	-1	21.87
G	G	L	G	0	-1	21.87
G	G	G	L	10	-1	28.51
G	G	G	L	0	-1	28.51
G	G	G	G	10	-1	27.46
G	G	G	G	0	-1	27.46
L	L	L	L	10	0	28.82
L	L	L	L	0	0	28.82
L	L	L	G	10	0	23.90
L	L	L	G	0	0	23.90
L	L	G	L	10	0	26.60
L	L	G	L	0	0	26.60
L	L	G	G	10	0	25.54
L	L	G	G	0	0	25.54
L	G	L	L	10	0	27.79
L	G	L	L	0	0	27.79
L	G	L	G	10	0	22.87
L	G	L	G	0	0	22.87
L	G	G	L	10	0	27.56
L	G	G	L	0	0	27.56
L	G	G	G	10	0	26.51
L	G	G	G	0	0	26.51
G	L	L	L	10	0	27.81
G	L	L	L	0	0	27.81
G	L	L	G	10	0	22.89
G	L	L	G	0	0	22.89
G	L	G	L	10	0	27.54
G	L	G	L	0	0	27.54
G	L	G	G	10	0	26.49
G	L	G	G	0	0	26.49
G	G	L	L	10	0	26.78

G		G		L		L		0		0		26.78
G		G		L		G		10		0		21.86
G		G		L		G		0		0		21.86
G		G		G		L		10		0		28.51
G		G		G		L		0		0		28.51
G		G		G		G		10		0		27.46
G		G		G		G		0		0		27.46

5.100 FFT-2048

Функции

- void FFT_Fwd2048Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Fwd2048Set7bit ()

Устанавливает 7-битную точность вычислений
- void [FFT_Fwd2048](#) ([nm32sc](#) *GSrcBuffer, [nm32sc](#) *GDstBuffer, void *LBuffer, int ShiftR=-1)

Прямое быстрое преобразование Фурье-2048.

5.100.1 Подробное описание

5.100.2 Функции

5.100.2.1 FFT_Fwd2048()

```
void FFT_Fwd2048 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    int ShiftR = -1 )
```

Прямое быстрое преобразование Фурье-2048.

Функция выполняет дискретное комплексное 2048-точечное преобразование Фурье на базе алгоритма БПФ по основанию 2-32-32

Аргументы

in	GSrcBuffer	Входной массив размером 2048 64-р. слов
out	GDstBuffer	Результирующий массив размером 2048 64-р. слов
in	LBuffer	Временный массив на локальнойшине (Local Bus) размером 2048*4 64-р. слов
in	ShiftR	Коэффициент нормализации, выполняет арифметический сдвиг результирующего массива на ShiftR бит вправо для получения нормализованного массива LDstBuffer. При передаче значения по умолчанию (-1) ShiftR автоматически принимается равным 14 если ранее установлена точность 7-бит функцией FFT_Fwd2048Set7bit() и 12 - если ранее установлена точность 6-бит функцией FFT_Fwd2048Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer	GDstBuffer	LBuffer	ShiftR	clocks
L	L	L	-1	26.41
L	L	G	-1	28.97
L	G	L	-1	25.38
L	G	G	-1	29.93
G	L	L	-1	25.38
G	L	G	-1	29.93
G	G	L	-1	24.36
G	G	G	-1	30.89
L	L	L	0	26.40
L	L	G	0	28.96
L	G	L	0	25.38
L	G	G	0	29.93
G	L	L	0	25.38
G	L	G	0	29.93
G	G	L	0	24.35
G	G	G	0	30.89

GSrcBuffer	GDstBuffer	LBuffer	ShiftR	clocks
L	L	L	-1	26.41
L	L	G	-1	28.97
L	G	L	-1	25.38
L	G	G	-1	29.93
G	L	L	-1	25.38
G	L	G	-1	29.93
G	G	L	-1	24.36
G	G	G	-1	30.89
L	L	L	0	26.40
L	L	G	0	28.96
L	G	L	0	25.38
L	G	G	0	29.93
G	L	L	0	25.38
G	L	G	0	29.93
G	G	L	0	24.35
G	G	G	0	30.89

5.101 IFFT-2048

Функции

- void FFT_Inv2048Set6bit ()

Устанавливает 6-битную точность вычислений
- void FFT_Inv2048Set7bit ()

Устанавливает 7-битную точность вычислений
- void **FFT_Inv2048** (**nm32sc** *GSrcBuffer, **nm32sc** *LDstBuffer, void *LBuffer, void *GBuffer, int ShiftR1=11, int ShiftR2=-1)

Обратное быстрое преобразование Фурье. ОБПФ-2048.

5.101.1 Подробное описание

5.101.2 Функции

5.101.2.1 FFT_Inv2048()

```
void FFT_Inv2048 (
    nm32sc * GSrcBuffer,
    nm32sc * LDstBuffer,
    void * LBuffer,
    void * GBuffer,
    int ShiftR1 = 11,
    int ShiftR2 = -1 )
```

Обратное быстрое преобразование Фурье. ОБПФ-2048.

Функция выполняет обратное дискретное комплексное 2048-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 2-32-32.

Аргументы

in	GSrcBuffer	Входной массив размером 2048 64-р. слов
out	LDstBuffer	Результирующий массив размером 2048 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 2048*4 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 2048*4 64-р. слов
in	ShiftR1	Промежуточный сдвиг результатов на ShiftR1 бит вправо (первая нормализация). Необходимо для предотвращения переполнения. По умолчанию равен 11
in	ShiftR2	Заключительный сдвиг результатов на ShiftR2 бит вправо (вторая нормализация) в конце вычисления обратного БПФ. По умолчанию ShiftR2 принимается равным 14 при установленной точности 7-бит с помощью функции FFT_Inv2048Set7bit() и 12 - при точности 6-бит, установленной с помощью функции FFT_Inv2048Set6bit() .

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными)

\perf

GSrcBuffer| LDstBuffer| LBuffer | GBuffer | ShiftR1 | ShiftR2 | clocks

L	L	L	L	11	-1	30.58
L	L	L	L	0	-1	30.58
L	L	L	G	11	-1	26.52
L	L	L	G	0	-1	26.52
L	L	G	L	11	-1	31.33
L	L	G	L	0	-1	31.33
L	L	G	G	11	-1	29.26
L	L	G	G	0	-1	29.26
L	G	L	L	11	-1	29.56
L	G	L	L	0	-1	29.55
L	G	L	G	11	-1	27.48
L	G	L	G	0	-1	27.48
L	G	G	L	11	-1	30.30
L	G	G	L	0	-1	30.30
L	G	G	G	11	-1	30.22
L	G	G	G	0	-1	30.22
G	L	L	L	11	-1	29.56
G	L	L	L	0	-1	29.56
G	L	L	G	11	-1	25.49
G	L	L	G	0	-1	25.49
G	L	G	L	11	-1	32.29
G	L	G	L	0	-1	32.29
G	L	G	G	11	-1	30.22
G	L	G	G	0	-1	30.22
G	G	L	L	11	-1	28.53
G	G	L	L	0	-1	28.53
G	G	L	G	11	-1	26.46
G	G	L	G	0	-1	26.46
G	G	G	L	11	-1	31.26
G	G	G	L	0	-1	31.26
G	G	G	G	11	-1	31.19
G	G	G	G	0	-1	31.18

L		L		L		L		11		0		30.58
L		L		L		L		0		0		30.58
L		L		L		G		11		0		26.51
L		L		L		G		0		0		26.51
L		L		G		L		11		0		31.33
L		L		G		L		0		0		31.32
L		L		G		G		11		0		29.25
L		L		G		G		0		0		29.25
L		G		L		L		11		0		29.55
L		G		L		L		0		0		29.55
L		G		L		G		11		0		27.48
L		G		L		G		0		0		27.48
L		G		G		L		11		0		30.30
L		G		G		L		0		0		30.30
L		G		G		G		11		0		30.22
L		G		G		G		0		0		30.22
G		L		L		L		11		0		29.56
G		L		L		L		0		0		29.56
G		L		L		G		11		0		25.49
G		L		L		G		0		0		25.49
G		L		G		L		11		0		32.29
G		L		G		L		0		0		32.29
G		L		G		G		11		0		30.22
G		L		G		G		0		0		30.22
G		G		L		L		11		0		28.53
G		G		L		L		0		0		28.53
G		G		L		G		11		0		26.46
G		G		L		G		0		0		26.46
G		G		G		L		11		0		31.26
G		G		G		L		0		0		31.26
G		G		G		G		11		0		31.18
G		G		G		G		0		0		31.18

5.102 FFT-4096

Функции

- void **FFT_Fwd4096** (**nm32sc** *GSrcBuffer, **nm32sc** *GDstBuffer, **void** *LBuffer, **void** *GBuffer)
Прямое быстрое преобразование Фурье-4096.

5.102.1 Подробное описание

5.102.2 Функции

5.102.2.1 FFT_Fwd4096()

```
void FFT_Fwd4096 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer )
```

Прямое быстрое преобразование Фурье-4096.

Функция выполняет дискретное комплексное 4096-точечное преобразование Фурье на базе алгоритма БПФ по основанию 16-16-16

Аргументы

in	GSrcBuffer	Входной массив размером 4096 64-р. слов
out	GDstBuffer	Результирующий массив размером 4096 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 4096*2 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 4096*3 64-р. слов

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными) Диапазон входных данных: -4096..4096

\perf

GSrcBuffer | GDstBuffer | LBuffer | GBuffer | Clocks

L	L	L	L	38.25
L	L	L	G	26.82
L	L	G	L	32.21
L	L	G	G	30.74
L	G	L	L	37.22
L	G	L	G	25.79
L	G	G	L	33.17
L	G	G	G	31.71
G	L	L	L	37.26
G	L	L	G	25.83
G	L	G	L	31.21
G	L	G	G	29.75
G	G	L	L	36.23
G	G	L	G	24.80
G	G	G	L	32.18
G	G	G	G	30.71

5.103 IFFT-4096

Функции

- void [FFT_Inv4096](#) ([nm32sc](#) *GSrcBuffer, [nm32sc](#) *GDstBuffer, void *LBuffer, void *GBuffer)
Обратное быстрое преобразование Фурье. ОБПФ-4096.

5.103.1 Подробное описание

5.103.2 Функции

5.103.2.1 FFT_Inv4096()

```
void FFT_Inv4096 (
    nm32sc * GSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer )
```

Обратное быстрое преобразование Фурье. ОБПФ-4096.

Функция выполняет обратное дискретное комплексное 4096-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 16-16-16.

Аргументы

in	GSrcBuffer	Входной массив размером 4096 64-р. слов
out	GDstBuffer	Результирующий массив размером 4096 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 4096*2 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 4096*3 64-р. слов

Возвращает

```
void
```

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными) Диапазон входных данных: -4096..4096

\perf

GSrcBuffer | GDstBuffer | LBuffer | GBuffer | Clocks

L	L	L	L	38.25
L	L	L	G	26.82
L	L	G	L	32.21
L	L	G	G	30.74
L	G	L	L	37.22
L	G	L	G	25.79
L	G	G	L	33.17
L	G	G	G	31.71
G	L	L	L	37.26
G	L	L	G	25.83
G	L	G	L	31.21
G	L	G	G	29.75
G	G	L	L	36.23
G	G	L	G	24.80
G	G	G	L	32.18
G	G	G	G	30.71

5.104 FFT-8192

Функции

- void [FFT_Fwd8192](#) (nm32sc *LSrcBuffer, nm32sc *GdstBuffer, void *LBuffer, void *GBuffer)
- Прямое быстрое преобразование Фурье-8192.

5.104.1 Подробное описание

5.104.2 Функции

5.104.2.1 FFT_Fwd8192()

```
void FFT_Fwd8192 (
    nm32sc * LSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer )
```

Прямое быстрое преобразование Фурье-8192.

Функция выполняет дискретное комплексное 8192-точечное преобразование Фурье на базе алгоритма БПФ по основанию 2-16-16-16

Аргументы

in	LSrcBuffer	Входной массив размером 8192 64-р. слов
out	GDstBuffer	Результирующий массив размером 8192 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 8192 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 8192*3 64-р. слов

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными) Диапазон входных данных: -2048..2048

\perf

LSrcBuffer | GDstBuffer | LBuffer | GBuffer | Clocks

L	L	L	L	40.70
L	L	L	G	28.89
L	L	G	L	35.55
L	L	G	G	31.88
L	G	L	L	39.67
L	G	L	G	27.86
L	G	G	L	36.52
L	G	G	G	32.85
G	L	L	L	40.17
G	L	L	G	29.40
G	L	G	L	35.02
G	L	G	G	32.39
G	G	L	L	39.14
G	G	L	G	28.37
G	G	G	L	35.99
G	G	G	G	33.36

5.105 IFFT-8192

Функции

- void [FFT_Inv8192](#) (`nm32sc *LSrcBuffer, nm32sc *GDstBuffer, void *LBuffer, void *GBuffer`)

Обратное быстрое преобразование Фурье. ОБПФ-8192.

5.105.1 Подробное описание

5.105.2 Функции

5.105.2.1 FFT_Inv8192()

```
void FFT_Inv8192 (
    nm32sc * LSrcBuffer,
    nm32sc * GDstBuffer,
    void * LBuffer,
    void * GBuffer )
```

Обратное быстрое преобразование Фурье. ОБПФ-8192.

Функция выполняет обратное дискретное комплексное 8192-точечное быстрое преобразование Фурье на базе алгоритма ОБПФ по онованию 2-16-16-16.

Аргументы

in	LSrcBuffer	Входной массив размером 8192 64-р. слов
out	GDstBuffer	Результирующий массив размером 8192 64-р. слов
in	LBuffer	Временный массив на локальной шине (Local Bus) размером 8192 64-р. слов
in	GBuffer	Временный массив на глобальной шине (Global Bus) размером 8192*3 64-р. слов

Возвращает

void

Заметки

Использование inplace параметров не допускается (все указатели должны быть разными) Диапазон входных данных: -2048..2048

\perf

GSrcBuffer | GDstBuffer | LBuffer | GBuffer | Clocks

L		L		L		L		40.70
L		L		L		G		28.89

L		L		G		L		35.55
L		L		G		G		31.88
L		G		L		L		39.67
L		G		L		G		27.86
L		G		G		L		36.52
L		G		G		G		32.85
G		L		L		L		40.17
G		L		L		G		29.40
G		L		G		L		35.02
G		L		G		G		32.39
G		G		L		L		39.14
G		G		L		G		28.37
G		G		G		L		35.99
G		G		G		G		33.36

5.106 Свертка

Группы

- [nmppsXCorr_32s](#)

Свертка двух векторов.

5.106.1 Подробное описание

5.107 Масочная фильтрация

Группы

- [nmppcMedian3_32u](#)

Вычисление медианы трех чисел

- [КИХ-фильтрация](#)

Одномерная КИХ-фильтрация.

5.107.1 Подробное описание

5.108 Изменение размеров

Группы

- [nmppsResampleDown2](#)

Уменьшение числа отсчетов в двое.

- [nmppsResampleUp3Down2](#)

Передискретизация сигнала в 3/2 раза

Передискретизация сигнала осуществляется методом Polyphase:

- [nmppsCreateResample](#)

Создание ядра для функции передискретизации nmppsResample().

Функции выделяют память и инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

- [nmppsSetResample](#)

Создание ядра для функции передискретизации nmppsResample().

Функции инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

- [nmppsResample_perf](#)

Функции для оценки производительности функций фильтрации nmppsResample()

Функция эквивалентно следующим псевдоинструкциям:

5.108.1 Подробное описание

5.109 Быстрое преобразование Фурье

Группы

- [FFT-256](#)
- [IFFT-256](#)
- [FFT-512](#)
- [IFFT-512](#)
- [FFT-1024](#)
- [IFFT-1024](#)
- [FFT-2048](#)
- [IFFT-2048](#)
- [FFT-4096](#)
- [IFFT-4096](#)
- [FFT-8192](#)
- [IFFT-8192](#)

5.109.1 Подробное описание

5.110 Быстрое преобразование Фурье

Группы

- [FFTFwdInitAlloc](#)
Функции инициализации структур коэффициентов, необходимых для вычисления прямого БПФ
- [FFTInvInitAlloc](#)
Функции инициализации структур коэффициентов, необходимых для вычисления обратного БПФ
- [DFT-8](#)
Функция для вычисления прямого ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел
- [FFT-16](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел
- [FFT-32](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел
- [FFT-64](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел
- [FFT-128](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел
- [FFT-256](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел
- [FFT-512](#)
Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

- [FFT-1024](#)

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 1024 комплексных чисел

- [FFT-2048](#)

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

- [FFT-4096](#)

Функция для вычисления прямого БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

- [IDFT-8](#)

Функция для вычисления обратного ДПФ с плавающей точкой над вектором, состоящим из 8 комплексных чисел

- [IFFT-16](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 16 комплексных чисел

- [IFFT-32](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 32 комплексных чисел

- [IFFT-64](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 64 комплексных чисел

- [IFFT-128](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 128 комплексных чисел

- [IFFT-256](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 256 комплексных чисел

- [IFFT-512](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 512 комплексных чисел

- [IFFT-1024](#)

- [IFFT-2048](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 2048 комплексных чисел

- [IFFT-4096](#)

Функция для вычисления обратного БПФ с плавающей точкой над вектором, состоящим из 4096 комплексных чисел

- [FFT-Common](#)

Функции

- int [nmppsFFTFree_32fcr](#) (NmppsFFTSPEC_32fcr *spec)

Функция инициализации структуры коэффициентов, необходимых для вычисления обратного БПФ с плавающей точкой над вектором длины от 8 до 2048.

5.110.1 Подробное описание

5.110.2 Функции

5.110.2.1 nmppsFFTFree_32fcr()

```
int nmppsFFTFree_32fcr (
    NmppsFFTSPEC_32fcr * spec )
```

Функция инициализации структуры коэффициентов, необходимых для вычисления обратного БПФ с плавающей точкой над вектором длины от 8 до 2048.

Аргументы

in	spec	двойной указатель на структуру коэффициентов
in	order	размерность БПФ, которое нужно вычислить, например, для БПФ256 этот параметр равен 8 (т.к. $2^8 = 256$)

Возвращает

Функция возвращают 0 в случае успешной инициализации и отрицательное число (от -1 и меньше) в случае ошибок

Функция освобождает память, выделенную под коэффициенты, необходимые для вычисления БПФ определенного размера

Аргументы

in	spec	структура, содержащая необходимые коэффициенты, для вычисления обратного БПФ определенного размера
----	------	--

Возвращает

Функция возвращают 0 в случае успешной инициализации и отрицательное число (от -1 и меньше) в случае ошибок

5.111 nmppsXCorr_32s

Свертка двух векторов.

Функции

- void nmppsXCorr_32s_32s (**nm32s** *pSrcVec, int nSrcVecSize, **nm32s** *pKernel, int nKernelSize, **nm32s** *pDstVec, void *pTmpBuf)
- void nmppsXCorr_32s_16s32s (**nm16s** *pSrcVec, int nSrcVecSize, **nm32s** *pKernel, int nKernelSize, **nm32s** *pDstVec, void *pTmpBuf)
- void nmppsXCorr_32s_8s32s (**nm8s** *pSrcVec, int nSrcVecSize, **nm32s** *pKernel, int nKernelSize, **nm32s** *pDstVec, void *pTmpBuf)

5.111.1 Подробное описание

Свертка двух векторов.

$$DstVec_i = \sum_{j=0}^{nKernelSize-1} pSrcVec[i+j] \cdot pKernel[j]$$

$$i = \overline{0 \dots nSrcVecSize - nKernelSize + 1}$$

Аргументы

pSrcVec	Входной вектор.
pKernel	Вектор коэффициентов окна свертки.
pTmpBuf	Указатель на временный буффер размера 2*nKernelSize + 32 32-битных слов;

2*nKernelSize + 32 32-bit words;

Аргументы

nKernelSize	Размер окна свертки [1,2,3,4...nSrcVecSize-1].
nSrcVecSize	Размер входного вектора в элементах .Размер кратен 8,4 или 2 согласно типу данных.

Возвращаемые значения

pDstVec	Результирующий вектор, размером nSrcVecSize-nKernelSize+1. после которого могут записываться еще до 7 незначащих 32р слова.
---------	---

Заметки

По сути функции осуществляют фильтрацию данных окном свертки.

5.112 nmppcMedian3_32u

Вычисление медианы трех чисел

Функции

- int nmppcMedian3_32u (int a, int b, int c)
- uint32b nmppcMedian3_32u (uint32b a, uint32b b, uint32b c)

5.112.1 Подробное описание

Вычисление медианы трех чисел

Аргументы

a	Первое число
b	Второе число
c	Третье число

Возвращает

Медианное значение

5.113 КИХ-фильтрация

Одномерная КИХ-фильтрация.

Группы

- [nmppsFIR_Xs](#)
Одномерная КИХ-фильтрация
- [nmppsFIRInit_Xs](#)
Инициализация функции одномерной фильтрации
- [nmppsFIRInitAlloc_Xs](#)
Выделение и инициализация служебной структуры для функции одномерной фильтрации
- [nmppsFIRGetSize_Xs](#)
Возвращает размер памяти (в 32р.- словах) необходимый для хранения служебной структуры
- [nmppsFIRFree](#)
освобождает структуру pState в куче

5.113.1 Подробное описание

Одномерная КИХ-фильтрация.

5.114 nmppsFIR_Xs

Одномерная КИХ-фильтрация

Функции

- void nmppsFIR_8s ([nm8s](#) *pSrc, [nm8s](#) *pDst, int srcSize, NmppsFIRState *pState)
- void nmppsFIR_8s16s ([nm8s](#) *pSrc, [nm16s](#) *pDst, int srcSize, NmppsFIRState *pState)
- void nmppsFIR_8s32s ([nm8s](#) *pSrc, [nm32s](#) *pDst, int srcSize, NmppsFIRState *pState)
- void nmppsFIR_16s ([nm16s](#) *pSrc, [nm16s](#) *pDst, int srcSize, NmppsFIRState *pState)
- void nmppsFIR_16s32s ([nm16s](#) *pSrc, [nm32s](#) *pDst, int srcSize, NmppsFIRState *pState)
- void nmppsFIR_32s ([nm32s](#) *pSrc, [nm32s](#) *pDst, int srcSize, NmppsFIRState *pState)

5.114.1 Подробное описание

Одномерная КИХ-фильтрация

Аргументы

in	pSrc	Входной вектор
in	srcSize	Размер входного вектора в элементах. Размер вектора должен быть кратен количеству элементов в 64-р. слове.
out	pDst	Результирующий вектор
in	NmppsFIRState	Служебная структура, содержащая весовые коэффициенты фильтра во внутреннем формате.

Заметки

Инициализация служебной структуры производится соответствующей функцией `nmppsFIRInit_Xs()` или `nmppsFIRInitAlloc_Xs()`. \perf Максимальная производительность достигается при размещении pSrc, pDst и pPstate в разных банках памяти .

5.115 nmppsFIRInit_Xs

Инициализация функции одномерной фильтрации

Функции

- int nmppsFIRInit_8s (NmppsFIRState *pState, int *pTaps, int tapsLen)
- int nmppsFIRInit_8s16s (NmppsFIRState *pState, int *pTaps, int tapsLen)
- int nmppsFIRInit_8s32s (NmppsFIRState *pState, int *pTaps, int tapsLen)
- int nmppsFIRInit_16s (NmppsFIRState *pState, int *pTaps, int tapsLen)
- int nmppsFIRInit_16s32s (NmppsFIRState *pState, int *pTaps, int tapsLen)
- int nmppsFIRInit_32s (NmppsFIRState *pState, int *pTaps, int tapsLen)

5.115.1 Подробное описание

Инициализация функции одномерной фильтрации

Функция преобразует таблицу весовых коэффициентов окна фильтра во внутренний формат

Аргументы

in	pTaps	Указатель на коэффициенты фильтра
in	tapsLen	Размер окна фильтра. tapsLen=[3,5,7,9...]
out	pState	Указатель на служебную структуру, содержащую весовые коэффициенты фильтра во внутреннем формате. Размер памяти (в 32р.- словах) необходимый для хранения служебной структуры можно получить с помощью функции nmppsFIRGetStateSize_Xs

Возвращает

Размер проинициализированной структуры pState в 32р. словах

5.116 nmppsFIRInitAlloc_Xs

Выделение и инициализация служебной структуры для функции одномерной фильтрации

Функции

- int nmppsFIRInitAlloc_8s (NmppsFIRState **ppState, int *pTaps, int tapsLen)
- int nmppsFIRInitAlloc_8s16s (NmppsFIRState **ppState, int *pTaps, int tapsLen)
- int nmppsFIRInitAlloc_8s32s (NmppsFIRState **ppState, int *pTaps, int tapsLen)
- int nmppsFIRInitAlloc_16s (NmppsFIRState **ppState, int *pTaps, int tapsLen)
- int nmppsFIRInitAlloc_16s32s (NmppsFIRState **ppState, int *pTaps, int tapsLen)
- int nmppsFIRInitAlloc_32s (NmppsFIRState **ppState, int *pTaps, int tapsLen)

5.116.1 Подробное описание

Выделение и инициализация служебной структуры для функции одномерной фильтрации

Функция выделяет структуру в куче и преобразует таблицу весовых коэффициентов окна фильтра во внутренний формат

Аргументы

in	pTaps	Указатель на коэффициенты фильтра
in	tapsLen	Размер окна фильтра. nWeights=[3,5,7,9...]

Возвращаемые значения

[out]	ppState Возвращает указатель на служебную структуру, содержащую весовые коэффициенты фильтра во внутреннем формате
-------	--

Возвращает

Размер проинициализированной структуры pState в 32р. словах

5.117 nmppsFIRGetSize_Xs

Возвращает размер памяти (в 32р.-словах) необходимый для хранения служебной структуры

Функции

- int nmppsFIRGetSize_8s (int tapsLen)
- int nmppsFIRGetSize_8s16s (int tapsLen)
- int nmppsFIRGetSize_8s32s (int tapsLen)
- int nmppsFIRGetSize_16s (int tapsLen)
- int nmppsFIRGetSize_16s32s (int tapsLen)
- int nmppsFIRGetSize_32s (int tapsLen)

5.117.1 Подробное описание

Возвращает размер памяти (в 32р.-словах) необходимый для хранения служебной структуры

Аргументы

in	tapsLen	Размер окна фильтра. tapsLen=[3,5,7,9,...]
----	---------	--

Возвращает

Возвращает размер памяти (в 32р.-словах), необходимый для хранения служебной структуры NmppsFIRState

5.118 nmppsFIRFree

освобождает структуру pState в куче

Функции

- void nmppsFIRFree (NmppsFIRState *pState)

5.118.1 Подробное описание

освобождает структуру pState в куче

Аргументы

in	pState	указатель на служебную структуру NmppsFIRState
----	--------	--

5.119 nmppsResampleDown2

Уменьшение числа отсчетов в двое.

Функции

- void nmppsResampleDown2_8u8u ([nm8u7b](#) *pSrcVec, [nm8u7b](#) *pDstVec, int nSrcVecSize, [nm64s](#) *pKernel)
- void nmppsResampleDown2_16u16u ([nm16u15b](#) *pSrcVec, [nm16u15b](#) *pDstVec, int nSrcVecSize, [nm64s](#) *pKernel)

5.119.1 Подробное описание

Уменьшение числа отсчетов в двое.

$$pDstVec = \frac{1}{2} (pSrcVec(2 * x) + pSrcVec(2 * x + 1))$$

Аргументы

pSrcVec	Входной сигнал.
nSize	Размер массива входных данных.

Возвращаемые значения

pDstVec	Результирующий сигнал.
---------	------------------------

Заметки

Для того чтобы избежать переполнения при усреднении, динамический диапазон исходного изображения должен принадлежать диапазону, определенному типом.

5.120 nmppsResampleUp3Down2

Передискретизации сигнала в 3/2 раза

Передискретизации сигнала осуществляется методом Polyphase:

.

Функции

- void nmppsResampleUp3Down2_8s16s ([nm8s](#) *pSrcVec, [nm16s](#) *pDstVec, int nSrcVecSize, [nm64s](#) *pKernel)

5.120.1 Подробное описание

Передискретизации сигнала в 3/2 раза

Передискретизации сигнала осуществляется методом Polyphase:

\~

- Между отсчетами входного сигнала вставляется по 2 нуля
- Полученный сигнал пропускается через фильтр ФНЧ. Длина фильтра 17
- Из полученного сигнала выбирается каждый 2 отсчет

Аргументы

<code>pSrcVec</code>	Входной вектор. Элементы вектора - целые числа со знаком.
<code>nSrcVecSize</code>	Размер входного вектора.

Возвращаемые значения

<code>pDstVec</code>	Результирующий вектор. Элементы вектора возвращаются в формате fixed-point: [12.4] (целая часть-12 бит, дробная -4бита)
----------------------	---

Возвращает

`void`

5.121 nmppsCreateResample

Создание ядра для функции передискретизации `nmppsResample()`.

Функции выделяют память и инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

Функции

- void nmppsCreateResampleUp3Down2_8s16s ([nm64s](#) **pKernel, int nHint)
- void nmppsCreateResampleDown2_8u8u ([nm64s](#) **pKernel, int nHint)
- void nmppsCreateResampleDown2_16u16u ([nm64s](#) **pKernel, int nHint)

5.121.1 Подробное описание

Создание ядра для функции передискретизации nmppsResample().

Функции выделяют память и инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

Аргументы

nHint	Определяет память (Local или Global) в которой создается служебная структура. nHint=[MEM_LOCAL, MEM_GLOBAL].
-------	---

Возвращаемые значения

pKernel	Указатель на служебную структуру, содержащую весовые коэффициенты фильтра во внутреннем формате.
---------	--

Заметки

Используется перед вызовом функции nmppsFilter.

5.122 nmppsSetResample

Создание ядра для функции передискретизации nmppsResample().

Функции инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

Функции

- int nmppsSetResampleUp3Down2_8s16s ([nm64s](#) *pKernel)
- int nmppsSetResampleDown2_8u8u ([nm64s](#) *pKernel)
- int nmppsSetResampleDown2_16u16u ([nm64s](#) *pKernel)

5.122.1 Подробное описание

Создание ядра для функции передискретизации nmppsResample().

Функции инициализируют таблицы весовых коэффициентов для использования в функциях передискретизации.

Аргументы

pKernel	Указатель на служебную структуру, содержащую весовые коэффициенты фильтра во внутреннем формате.
---------	--

Возвращает

Размер проинициализированной структуры pKernel в 32р. словах

Заметки

Используется перед вызовом функции nmppsFilter.

Используется перед вызовом функции nmppsResample().

5.123 nmppsResample_perf

Функции для оценки производительности функций фильтрации nmppsResample()

Функция эквивалентно следующим псевдоинструкциям:

Функции

- void nmppsResampleUp3Down2_perf ([nm8s](#) *pSrcVec, [nm16s](#) *pDstVec, int nSrcVecSize, [nm64s](#) *pKernel)
- void nmppsResampleDown2_perf_8u ([nm8u7b](#) *pSrcVec, [nm8u7b](#) *pDstVec, int nSrcVecSize, [nm64s](#) *pKernel)
- void nmppsResampleDown2_perf_16u ([nm16u15b](#) *pSrcVec, [nm16u15b](#) *pDstVec, int nSrcVec← Size, [nm64s](#) *pKernel)

5.123.1 Подробное описание

Функции для оценки производительности функций фильтрации nmppsResample()

Функция эквивалентно следующим псевдоинструкциям:

```
void nmppsResample***_perf(nm16u15b* pSrcVec, nm16u15b* pDstVec, int nSrcVecSize, nm64s* pKernel);
{
    nmppsSetResample***_***(pKernel);
    clock_t t0,t1;
    t0=clock();
    nmppsResample***(pSrcVec, pDstVec, nSize,pKernel);
    t1=clock();
    exit(t1-t0);
}
```

\~

Возвращаемые значения

pKernel	Указатель на служебную структуру, содержащую весовые коэффициенты фильтра во внутреннем формате.
-------------------------	--

Аргументы

pSrcVec	Входной вектор.
-------------------------	-----------------

Аргументы

nSrcVecSize	Размер входного вектора в элементах.
-------------	--------------------------------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Заметки

Инициализация служебной структуры производится соответствующей функцией nmppsSetFilter() и nmppsCreateFilter().

5.124 Типы векторных данных

Структуры данных

- struct v16nm4s
- struct v4nm8s
- struct s_v8nm8s
- struct s_v16nm8s
- struct s_v4nm16s
- struct s_v8nm16s
- struct s_v16nm16s
- struct s_v2nm32s
- struct s_v4nm32s
- struct s_v8nm32s
- struct s_v16nm32s
- struct s_v16nm4u
- struct s_v4nm8u
- struct s_v8nm8u
- struct s_v16nm8u
- struct s_v4nm16u
- struct s_v8nm16u
- struct s_v16nm16u
- struct s_v2nm32u
- struct s_v4nm32u
- struct s_v8nm32u
- struct s_v16nm32u

Определения типов

- `typedef int nm1`
- `typedef void nm2s`
- `typedef void nm4s`
- `typedef char nm8s`
- `typedef nm8s nm8s7b`
- `typedef short nm16s`
- `typedef nm16s nm16s15b`
- `typedef int nm32s`
- `typedef int nm32s31b`
- `typedef int nm32s30b`
- `typedef long long nm64s`
- `typedef nm64s nm64s63b`
- `typedef void nm2u`
- `typedef void nm4u`
- `typedef nm4u nm4u3b`
- `typedef unsigned char nm8u`
- `typedef nm8u nm8u7b`
- `typedef unsigned short nm16u`
- `typedef nm16u nm16u15b`
- `typedef unsigned int nm32u`
- `typedef unsigned int nm32u31b`
- `typedef unsigned long long nm64u`
- `typedef struct s_v8nm8s v8nm8s`
- `typedef struct s_v16nm8s v16nm8s`
- `typedef struct s_v4nm16s v4nm16s`
- `typedef struct s_v8nm16s v8nm16s`
- `typedef struct s_v16nm16s v16nm16s`
- `typedef struct s_v2nm32s v2nm32s`
- `typedef struct s_v4nm32s v4nm32s`
- `typedef struct s_v8nm32s v8nm32s`
- `typedef struct s_v16nm32s v16nm32s`
- `typedef v16nm8s v16nm8s7b`
- `typedef struct s_v16nm4u v16nm4u`
- `typedef struct s_v4nm8u v4nm8u`
- `typedef struct s_v8nm8u v8nm8u`
- `typedef struct s_v16nm8u v16nm8u`
- `typedef struct s_v4nm16u v4nm16u`
- `typedef struct s_v8nm16u v8nm16u`
- `typedef struct s_v16nm16u v16nm16u`
- `typedef struct s_v2nm32u v2nm32u`
- `typedef struct s_v4nm32u v4nm32u`
- `typedef struct s_v8nm32u v8nm32u`
- `typedef struct s_v16nm32u v16nm32u`
- `typedef v16nm4u v16nm4b3u`

5.124.1 Подробное описание

В данном разделе описываются типы векторных данных с которыми могут работать функции библиотеки, задействующие векторный узел. Также рассматриваются соглашения о передаче параметров.

Поскольку векторный узел работает с данными произвольной разрядности, упакованными в 64-разрядные слова, то это накладывает следующие ограничения на работу с массивами данных и их типами:

1. Указатель на векторные данные всегда является четным адресом. Т.е. выровнен в памяти по границе 64р. слов.
2. Размер массива, передаваемый на вход функций, как правило, исчисляется в отдельных элементах, составляющих этот массив.

Кратность этого размера по умолчанию определяется кол-вом чисел, упакованных в 64р. слово.

Например:

для nm8s кратность-8
для nm16s кратность-4
для nm32s кратность-2
для nm64s кратность-1

Если в описании указаны другие условия кратности, как например [32,64,96,128...], то это означает, что допустимые размеры могут только из этого диапазона с кратностью 32.

3. Типы nm8s , nm16s, nm32s... хоть и созданы для обозначания разрядности упакованных данных, но с точки зрения C++ таковыми не являются , так как определяются через typedef как производные от стандартных типов char, short и int, которые все три в свою очередь являются 32-разрядными типами в архитектуре NeuroMatrix. Поэтому эти векторные типы можно использовать только с оператором * (nm8s*,nm16s*,...). Операции же sizeof() к массивам этих типов будут выполняться некорректно.

Расшифровка мнемоники в названии типа:

1. Префикс nm - означает что данные являются векторными ,упакованными в 64р слова (nm8s,nm8u,nm16s....).
2. Разрядность данных указывается после префикса nm (nm8s,nm8u - байтовые массивы, nm16s,nm16u - 16р. массивы).
3. суффикс s или u означает знаковый или беззнаковый тип данных.
4. Для работы некоторых функций во избежании переполнения требуется суженный диапазон возможных значений, чем позволяет разрядность. Такие данные имеют суффикс в виде кол-ва значащих бит в слове и символом b. (nm8s7b)

5.124.2 Типы

5.124.2.1 nm1

```
typedef int nm1
```

```
\~
```

Большинство функций библиотеки получают и возвращают массивы упакованных чисел. Обращение к элементам данных массивов должно производится с помощью функций доступа к элементам Getval() and Setval().

```
\~
```

Тип характеризует векторные данные как массив одноразрядных чисел.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 64.

Диапазон значений:

```
[-1, 0]
```

5.124.2.2 nm16s

```
typedef short nm16s
```

Тип характеризует векторные данные как массив 16-ти разрядных чисел со знаком.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 4.

Диапазон значений:

```
[-215, ..., +215 - 1].
```

5.124.2.3 nm16s15b

```
typedef nm16s nm16s15b
```

Тип характеризует векторные данные как массив 16-ти разрядных чисел со знаком с ограниченным диапазоном принимаемых значений.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 4.

Диапазон значений:

```
[-214, ..., +214 - 1]
```

5.124.2.4 nm16u

```
typedef unsigned short nm16u
```

Тип характеризует векторные данные как массив 16-ти разрядных чисел без знака.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 4.

Диапазон значений:

```
[0, ..., 216 - 1].
```

5.124.2.5 nm16u15b

`typedef nm16u nm16u15b`

Тип характеризует векторные данные как массив 16-ти разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 4.

Диапазон значений:

$$[-2^{14}, \dots, +2^{14} - 1].$$

5.124.2.6 nm2s

`typedef void nm2s`

Тип характеризует векторные данные как массив 2-х разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 32.

Диапазон значений:

$$[-2^1, \dots, +2^1 - 1] = [-2, \dots, +1]$$

5.124.2.7 nm2u

`typedef void nm2u`

Тип характеризует векторные данные как массив 2-х разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 32.

Диапазон значений:

$$[0, \dots, +2^2 - 1] = [0, \dots, 3]$$

5.124.2.8 nm32s

`typedef int nm32s`

Тип характеризует векторные данные как массив 32-х разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 2.

Диапазон значений:

$$[-2^{31}, \dots, +2^{31} - 1].$$

5.124.2.9 nm32s30b

```
typedef int nm32s30b
```

Тип характеризует векторные данные как массив 32-х разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 2.

Диапазон значений:

$$[-2^{29}, \dots, 2^{29} - 1].$$

5.124.2.10 nm32s31b

```
typedef int nm32s31b
```

Тип характеризует векторные данные как массив 32-х разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 2.

Диапазон значений:

$$[-2^{30}, \dots, 2^{30} - 1].$$

5.124.2.11 nm32u

```
typedef unsigned int nm32u
```

Тип характеризует векторные данные как массив 32-х разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 2.

Диапазон значений

$$[0, \dots, 2^{32} - 1].$$

5.124.2.12 nm32u31b

```
typedef unsigned int nm32u31b
```

Тип характеризует векторные данные как массив 32-х разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 2.

Диапазон значений

$$[0, \dots, 2^{31} - 1].$$

5.124.2.13 nm4s

`typedef void nm4s`

Тип характеризует векторные данные как массив 4-х разрядных чисел со знаком.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 16.

Диапазон значений:

$$[-2^3, \dots, +2^3 - 1] = [-8, \dots, +7]$$

5.124.2.14 nm4u

`typedef void nm4u`

Тип характеризует векторные данные как массив 4-х разрядных чисел без знака.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 16.

Диапазон значений:

$$[0, \dots, +2^4 - 1] = [0, \dots, 15]$$

5.124.2.15 nm4u3b

`typedef nm4u nm4u3b`

Тип характеризует векторные данные как массив 4-х разрядных чисел без знака.

Начальный адрес массива должен быть выровнен по границе 64р слова.

Предполагается , что размер массива данного типа как минимум кратен 16.

Диапазон значений:

$$[0, \dots, +2^3 - 1] = [0, \dots, 7]$$

5.124.2.16 nm64s

`typedef long long nm64s`

Тип характеризует векторные данные как массив 64-х разрядных чисел со знаком.

Начальный адрес массива должен быть выровнен по границе 64р слова.

По умолчанию размер массива произвольный .

Диапазон значений:

$$[-2^{63}, \dots, +2^{63} - 1]$$

5.124.2.17 nm64s63b

```
typedef nm64s nm64s63b
```

Тип характеризует векторные данные как массив 64-х разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.
По умолчанию размер массива произвольный .

Диапазон значений:

$$[-2^{62}, \dots, +2^{62} - 1]$$

5.124.2.18 nm64u

```
typedef unsigned long long nm64u
```

Тип характеризует векторные данные как массив 64-х разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.
По умолчанию размер массива произвольный .

Диапазон значений

$$[0, \dots, 2^{64} - 1].$$

5.124.2.19 nm8s

```
typedef char nm8s
```

Тип характеризует векторные данные как массив 8-ми разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 8.

Диапазон значений:

$$[-2^7, \dots, +2^7 - 1] = [-128, \dots, +127]$$

5.124.2.20 nm8s7b

```
typedef nm8s nm8s7b
```

Тип характеризует векторные данные как массив 8-ми разрядных чисел со знаком.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 8.

Диапазон значений:

$$[-2^6, \dots, +2^6 - 1] = [-64, \dots, +63]$$

5.124.2.21 nm8u

```
typedef unsigned char nm8u
```

Тип характеризует векторные данные как массив 8-ми разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 8.

Диапазон значений:

$$[0, \dots, +2^8 - 1] = [0, \dots, 255]$$

5.124.2.22 nm8u7b

```
typedef nm8u nm8u7b
```

Тип характеризует векторные данные как массив 8-ми разрядных чисел без знака.
Начальный адрес массива должен быть выровнен по границе 64р слова.
Предполагается , что размер массива данного типа как минимум кратен 8.

Диапазон значений:

$$[0, \dots, +2^7 - 1] = [0, \dots, 127]$$

5.124.2.23 v16nm16s

```
typedef struct s_v16nm16s v16nm16s
```

\~

Тип векторной структуры, состоящей из 16-ти 16р. чисел со знаком.

5.124.2.24 v16nm16u

```
typedef struct s_v16nm16u v16nm16u
```

\~

Тип векторной структуры, состоящей из 16-ти 16р. чисел без знака.

5.124.2.25 v16nm32s

```
typedef struct s_v16nm32s v16nm32s
```

\~

Тип векторной структуры, состоящей из 16-ти 32р. чисел со знаком.

5.124.2.26 v16nm32u

```
typedef struct s_v16nm32u v16nm32u
```

```
\~
```

Тип векторной структуры, состоящей из 16-ти 32р. чисел без знака.

5.124.2.27 v16nm4b3u

```
typedef v16nm4u v16nm4b3u
```

```
\~
```

Тип векторной структуры, состоящей из 16-ти 32р. чисел со знаком.

Диапазон значений элементов структуры:

$[0, \dots, 7]$

5.124.2.28 v16nm4u

```
typedef struct s_v16nm4u v16nm4u
```

Тип векторной структуры, состоящей из 16-ти 4-р. чисел без знака.

5.124.2.29 v16nm8s

```
typedef struct s_v16nm8s v16nm8s
```

```
\~
```

Тип векторной структуры, состоящей из 16-ти 8р. чисел со знаком.

5.124.2.30 v16nm8s7b

```
typedef v16nm8s v16nm8s7b
```

```
\~
```

Тип векторной структуры, состоящей из 16-ти 32р. чисел со знаком.

Диапазон значений элементов структуры:

$[-64, \dots, +63]$

5.124.2.31 v16nm8u

```
typedef struct s_v16nm8u v16nm8u
```

```
\~
```

Тип векторной структуры, состоящей из 16-ти 8р. чисел без знака.

5.124.2.32 v2nm32s

```
typedef struct s_v2nm32s v2nm32s
```

```
\~
```

Тип векторной структуры, состоящей из 2-х 32р. чисел со знаком.

5.124.2.33 v2nm32u

```
typedef struct s_v2nm32u v2nm32u
```

```
\~
```

Тип векторной структуры, состоящей из 2-х 32р. чисел без знака.

5.124.2.34 v4nm16s

```
typedef struct s_v4nm16s v4nm16s
```

```
\~
```

Тип векторной структуры, состоящей из 4-х 16р. чисел со знаком.

5.124.2.35 v4nm16u

```
typedef struct s_v4nm16u v4nm16u
```

```
\~
```

Тип векторной структуры, состоящей из 4-х 16р. чисел без знака.

5.124.2.36 v4nm32s

```
typedef struct s_v4nm32s v4nm32s
```

```
\~
```

Тип векторной структуры, состоящей из 4-х 32р. чисел со знаком.

5.124.2.37 v4nm32u

```
typedef struct s_v4nm32u v4nm32u
```

```
\~
```

Тип векторной структуры, состоящей из 4-х 32р. чисел без знака.

5.124.2.38 v4nm8u

```
typedef struct s_v4nm8u v4nm8u
```

```
\~
```

Тип векторной структуры, состоящей из 4-х 8р. чисел без знака.

5.124.2.39 v8nm16s

```
typedef struct s_v8nm16s v8nm16s
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 16р. чисел со знаком.

5.124.2.40 v8nm16u

```
typedef struct s_v8nm16u v8nm16u
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 16р. чисел без знака.

5.124.2.41 v8nm32s

```
typedef struct s_v8nm32s v8nm32s
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 32р. чисел со знаком.

5.124.2.42 v8nm32u

```
typedef struct s_v8nm32u v8nm32u
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 32р. чисел без знака.

5.124.2.43 v8nm8s

```
typedef struct s_v8nm8s v8nm8s
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 8р. чисел со знаком.

5.124.2.44 v8nm8u

```
typedef struct s_v8nm8u v8nm8u
```

```
\~
```

Тип векторной структуры, состоящей из 8-ми 8р. чисел без знака.

5.125 Типы скалярных данных

Определения типов

- `typedef int int1b`
- `typedef int int2b`
- `typedef int int3b`
- `typedef int int4b`
- `typedef int int7b`
- `typedef int int8b`
- `typedef int int15b`
- `typedef int int16b`
- `typedef int int30b`
- `typedef int int31b`
- `typedef int int32b`
- `typedef INT64 int63b`
- `typedef INT64 int64b`
- `typedef unsigned int uint1b`
- `typedef unsigned int uint2b`
- `typedef unsigned int uint3b`
- `typedef unsigned int uint4b`
- `typedef unsigned int uint7b`
- `typedef unsigned int uint8b`
- `typedef unsigned int uint15b`
- `typedef unsigned int uint16b`
- `typedef unsigned int uint31b`
- `typedef unsigned int uint32b`
- `typedef UINT64 uint63b`
- `typedef nm64u uint64b`

5.125.1 Подробное описание

Назначением данной библиотеки является предоставление базовых операций по обработке одномерных массивов (векторов) для процессоров семейства NeroMatrix.

В состав библиотеки входят логические и арифметические функции, операции сравнения, инициализации, копирования, преобразования разрядностей и т.п. Библиотека предназначена для быстрой разработки эффективных пользовательских программ как на языке высокого уровня(C++), так и на языке ассемблера с помощью прилагаемой библиотеки ядра низкоуровневых функций. Функции библиотеки имеют C++ интерфейс.

Большинство функций библиотеки реализованы на языке ассемблера с использованием векторных инструкций и оптимизированы под архитектуру процессоров семейства NeroMatrix. Для удобства разработки прикладных программ библиотека содержит аналогичные реализации функций для процессоров серии x86, выполненных на языке C++. Данные реализации позволяют выполнять написанные с использованием данной библиотеки прикладные программы на персональном компьютере.

5.125.2 Типы

5.125.2.1 int15b

typedef int **int15b**

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{14}, \dots, +2^{14} - 1]$$

5.125.2.2 int16b

typedef int **int16b**

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{15}, \dots, +2^{15} - 1]$$

5.125.2.3 int1b

typedef int **int1b**

Тип для 32-разрядных скалярных переменных с ограниченным диапазоном значений.

Диапазон значений:

$$[-1, 0]$$

5.125.2.4 int2b

typedef int **int2b**

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^1, \dots, +2^1 - 1] = [-2, \dots, +1]$$

5.125.2.5 int30b

typedef int **int30b**

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{29}, \dots, +2^{29} - 1]$$

5.125.2.6 int31b

typedef int [int31b](#)

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{30}, \dots, +2^{30} - 1]$$

5.125.2.7 int32b

typedef int [int32b](#)

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{31}, \dots, +2^{31} - 1]$$

5.125.2.8 int3b

typedef int [int3b](#)

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^2, \dots, +2^2 - 1] = [-4, \dots, +3]$$

5.125.2.9 int4b

typedef int [int4b](#)

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^3, \dots, +2^3 - 1] = [-8, \dots, +7]$$

5.125.2.10 int63b

typedef INT64 [int63b](#)

Тип для 64-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{62}, \dots, +2^{62} - 1]$$

5.125.2.11 int64b

typedef INT64 int64b

Тип для 64-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^{63}, \dots, +2^{63} - 1]$$

5.125.2.12 int7b

typedef int int7b

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[-2^6, \dots, +2^6 - 1] = [-64, \dots, +63]$$

5.125.2.13 int8b

typedef int int8b

Тип для 32-разрядных скалярных переменных с ограниченным диапазоном значений.

Диапазон значений:

$$[-2^7, \dots, +2^7 - 1] = [-128, \dots, +127]$$

5.125.2.14 uint15b

typedef unsigned int uint15b

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{15} - 1]$$

5.125.2.15 uint16b

typedef unsigned int uint16b

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{16} - 1]$$

5.125.2.16 uint1b

```
typedef unsigned int uint1b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, 1] = [0, 1]$$

5.125.2.17 uint2b

```
typedef unsigned int uint2b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^2 - 1] = [0, \dots, 3]$$

5.125.2.18 uint31b

```
typedef unsigned int uint31b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{31} - 1]$$

5.125.2.19 uint32b

```
typedef unsigned int uint32b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{32} - 1]$$

5.125.2.20 uint3b

```
typedef unsigned int uint3b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^3 - 1] = [0, \dots, 7]$$

5.125.2.21 uint4b

```
typedef unsigned int uint4b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^4 - 1] = [0, \dots, 15]$$

5.125.2.22 uint63b

```
typedef UINT64 uint63b
```

Тип для 64-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{63} - 1]$$

5.125.2.23 uint64b

```
typedef nm64u uint64b
```

Тип для 64-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^{64} - 1]$$

5.125.2.24 uint7b

```
typedef unsigned int uint7b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^7 - 1] = [0, \dots, 127]$$

5.125.2.25 uint8b

```
typedef unsigned int uint8b
```

Тип для 32-разрядных скалярных переменных с ограниченным допустимым диапазоном значений.

Диапазон значений:

$$[0, \dots, 2^8 - 1] = [0, \dots, 255]$$

5.126 Функции поддержки

Группы

- **nmppsMAlloc**

Распределение памяти для векторов библиотеки.

- **nmppsFree**

Освобождение памяти для векторов.

- **nmppsAddr_**

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

- **nmppsSetVal_**

Модификация элемента вектора.

- **nmppsGetVal_**

Извлекает значение элемента вектора.

- **nmppsGetVal_(return)**

Извлекает значение элемента вектора.

5.126.1 Подробное описание

5.127 Инициализация и копирование

Группы

- **nmppsSet-инициализация**

Функция инициализации элементов массива постоянным значением.

- **nmppsRandUniform**

Инициализация массива случайными числами.

- **nmppsRandUniform_**

Генерация случайного числа с равномерным распределением.

- **nmppsRamp_**

Инициализация массива элементами арифметической прогрессии.

- **nmppsConvert**

Изменение разрядности элементов вектора.

Преобразование знаковых данных к меньшей разрядности осуществляется отбрасыванием старших битов. Преобразование знаковых данных к большей разрядности осуществляется с распространением влево старшего (знакового) бита. Преобразование беззнаковых данных к большей разрядности осуществляется добавлением слева старших нулевых битов.

- **nmppsCopy_**

Копирование вектора.

- **nmppsCopyua_**

Копирование вектора с невыровненной байтовой позиции в выровненную.

- **nmppsSwap_**

Перестановка двух векторов.

- **Vec_ClipRShiftConvert_AddC**

Изменение разрядности элементов вектора с клиппированием

Преобразование 32р знаковых данных к восьмиразрядным осуществляется вырезанием битов 16..31., затем клиппированием и вырезанием восьми младших бит реультата. При клиппировании используется глобальная константа типа nm64s с именем _F1CR_16_to_x. По умолчанию, там лежит ff00_ff00_ff00_ff00 соответствующая диапазону результата [-255..256]. К каждому полученному 64р слову результата прибавляется значение глобальной переменной _VR_16_to_x, которая по умолчанию равна нулю.

5.127.1 Подробное описание

5.128 Инициализация и копирование

Группы

- [nmppsRandUniform](#)
Инициализация массива случайными числами.
- [nmppsConvert](#)
Изменение разрядности элементов вектора. Входной вектор.
- [nmppsConvert_rounding](#)
Функции конвертации вектора чисел с плавающей точкой одинарной точности в вектор целых 32-битных чисел с разной степенью округления
- [nmppsMerge](#)
Функция соединяет 2 массива 32-х чисел с плавающей точкой (float) в один. Результирующий массив { pSrcVec1[0], pSrcVec2[0], pSrcVec1[1], pSrcVec2[1], pSrcVec1[2], pSrcVec2[2] и т.д. }.
- [nmppsConvertRisc](#)
Функции изменения разрядности чисел входного вектора
- [nmppsCopy](#)
Копирование массива чисел с плавающей точкой одинарной точности (с любого на любой адрес)
- [nmppsCopyOddToOdd_32f](#)
Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на нечетный адрес)
- [nmppsCopyEvenToOdd_32f](#)
Копирование массива чисел с плавающей точкой одинарной точности (с четного на нечетный адрес)
- [nmppsCopyOddToEven_32f](#)
Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на четный адрес)
- [nmppsCopyEvenToEven_32f](#)
Копирование массива чисел с плавающей точкой одинарной точности (с четного на четный адрес)
- [nmppsCopyRisc](#)
Копирование массива

5.128.1 Подробное описание

5.129 Арифметические операции

Группы

- [nmppsAbs](#)
Вычисление абсолютных значений для элементов вектора.
- [nmppsAbs1](#)
Функция логического вычисления модулей элементов вектора.
- [nmppsNeg](#)
Изменение знака элементов вектора на противоположный.
- [nmppsAddC](#)
Добавление к вектору константы.

- [nmppsAddC_p](#)
Добавление к вектору константы.
- [nmppsAdd](#)
Сложение двух векторов.
- [nmppsAdd_AddC](#)
Сложение двух векторов с прибавлением константы.
- [nmppsSubC](#)
Вычитание константы из вектора.
- [nmppsSubCRev](#)
Вычитание константы из вектора с переменой знака элементов вектора.
- [nmppsSub](#)
Вычитание двух векторов.
- [nmppsAbsDiff](#)
Вычисление вектора модулей разности элементов двух векторов.
- [nmppsAbsDiff_f](#)
Вычисление вектора модулей разности элементов двух векторов.
- [nmppsAbsDiff1](#)
Функция логического вычисления модулей разностей элементов двух векторов.
- [nmppsMulC](#)
Умножение вектора на константу.
- [nmppsMulC_AddC](#)
Поэлементное умножение векторов с прибавлением константы.
- [nmppsMulC_MulC](#)
Умножение вектора на константу с прибавлением константы.
- [nmppsRShiftC_MulC_AddC](#)
Умножение вектора на константу с прибавлением вектора и константы.
- [nmppsMulC_AddV_AddC](#)
Умножение вектора на константу с прибавлением вектора и константы.
- [nmppsSumN](#)
Сложение нескольких векторов.
- [nmppsDivC](#)
Деление вектора на константу.
- [nmppsSum](#)
Возвращает сумму всех элементов вектора.
- [nmppsSum_1](#)
Возвращает сумму всех элементов вектора.
- [nmppsDotProd_sm](#)
Скалярное умножение двух векторов.
- [nmppsDotProd](#)
Скалярное умножение двух векторов.
- [nmppsWeightedSum](#)
Поэлементное взвешенное суммирование элементов двух векторов

5.129.1 Подробное описание

5.130 Арифметические операции

Группы

- [nmppsAddC](#)

- Добавление к вектору константы.
- [nmppsAdd](#)
Сложение двух векторов.
- [nmppsSubC](#)
Вычитание константы из вектора.
- [nmppsSubCRev](#)
Вычитание вектора из константы.
- [nmppsSub](#)
Вычитание двух векторов.
- [nmppsMulC](#)
Умножение вектора на константу.
- [nmppsMul_Mul_Add](#)
Умножение двух пар векторов с последующим суммированием результатов ($pDstVec = pSrcVec1 * pSrcVec2 + pSrcVec3 * pSrcVec4$)
- [nmppsMul_Add](#)
Умножение с накоплением
- [nmppsMul_ConjMul_Add](#)
Умножение пары векторов (вектор $pSrcVec4$ комплексно-сопряженный) с последующим суммированием результата ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] + pSrcVec3[i] * \text{Conj}(pSrcVec4[i])$)
- [nmppsMulC_AddC](#)
Умножает вектор на констану с прибавлением другой константы ($pDstVec[i] = nMulC * pSrcVec[i] + nAddC$)
- [nmppsMulC_AddV_AddC](#)
Умножение вектора на константу с прибавлением вектора и константы ($pDstVec[i] = nMulC * pSrcVec[i] + pVecAdd[i] + nAddC$)
- [nmppsMul](#)
Умножение векторов ($pDstVec[i] = pSrcVec1[i] * pSrcVec1[i]$)
- [nmppsConjMul](#)
Умножение вектора на комплексно-сопряженный вектор ($pDstVec[i] = pSrcVec1[i] * \text{Conj}(pSrcVec1[i])$)
- [nmppsMul_Mul_Sub](#)
Умножение пары векторов с последующим вычитанием результатов ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] - pSrcVec3[i] * pSrcVec4[i]$)
- [nmppsMulC_Add](#)
Умножение вектора на константу с накоплением ($pDstVec[i] = pSrcVec1[i] * C + pSrcVec2[i]$)
- [nmppsMul_AddC](#)
Поэлементное умножение векторов с прибавлением константы.

5.130.1 Подробное описание

5.131 Логические и бинарные операции

Группы

- [nmppsNot_](#)
Функция логического "НЕ".
- [nmppsAndC](#)
Функция логического "И" между вектором и константой.
- [nmppsAnd](#)
Функция логического "И" между двумя векторами.

- [nmppsAnd4V_](#)
Функция логического "И" между четырьмя векторами.
- [nmppsAndNotV_](#)
Функция логического "И-НЕ" между двумя векторами.
- [nmppsOrC](#)
Функция логического "ИЛИ" между вектором и константой.
- [nmppsOr](#)
Функция логического "ИЛИ" между двумя векторами.
- [nmppsOr3V_](#)
Функция логического "ИЛИ" между четырьмя векторами.
- [nmppsOr4V_](#)
Функция логического "ИЛИ" между четырьмя векторами.
- [nmppsXorC](#)
Функция логического "Исключающего ИЛИ" между вектором и константой.
- [nmppsXor](#)
Функция логического "Исключающего ИЛИ" между двумя векторами.
- [nmppsMaskV_](#)
Функция логического ИЛИ с предварительным маскированием двух векторов.
- [nmppsRShiftC](#)
Операция арифметического сдвига вправо.
- [nmppsRShiftC_AddC_](#)
Операция логического сдвига.
- [nmppsDisplaceBits](#)
Непрерывное смещение битов внутри бинарного массива в сторону конца массива
Функция смещает биты внутри бинарного массива на несколько позиций (nBits) в сторону конца массива. Внутри 64р. слова младшие биты сдвигаются на старшие позиции того же слова, а старшие биты перемещаются в младшие позиции следующего 64р. слова. Освободившееся место в первом 64р. слове заполняется старшими битами 64р. слова с адреса pnBits. Сдвинутые биты сохраняются в массиве pDst. Пример сдвига на 8 бит :

5.131.1 Подробное описание

5.132 Операции сравнения

Группы

- [nmppsMax_](#)
Поиск значения максимального элемента вектора.
- [nmppsMin](#)
Поиск значения минимального элемента вектора.
- [nmppsMaxIndx_](#)
Поиск значения максимального элемента вектора и его положения (положений) в векторе.
- [nmppsMinIndx_](#)
Поиск значения минимального элемента вектора и его положения (положений) в векторе.
- [nmppsMinIndxVN_](#)
Поиск значения минимального элемента вектора длины N и его положения в векторе.
- [nmppsFirstZeroIndx](#)
Поиск позиции первого нулевого элемента в векторе .

- [nmppsFirstNonZeroIndx](#)

Поиск позиции первого ненулевого элемента в векторе .

- [nmppsLastZeroIndx](#)

Поиск позиции последнего нулевого элемента в векторе .

- [nmppsLastNonZeroIndx](#)

Поиск позиции последнего ненулевого элемента в векторе .

- [nmppsMinEvery_](#)

Поэлементный минимум из двух векторов.

- [nmppsMaxEvery_](#)

Поэлементный максимум из двух векторов.

- [nmppsMinCmpLtV_](#)

Поэлементный минимум из двух векторов.

- [nmppsCmpLt0](#)

Сравнивает элементы массива на меньше нуля.

- [nmppsCmpEq0\(Reduced\)](#)

Сравнивает элементы массива неполной разрядности на признак равенства нулю.

- [nmppsCmpGt0](#)

Сравнивает элементы массива на больше нуля.

- [nmppsMinMaxEvery_](#)

Поэлементное сравнение двух векторов.

- [nmppsClipPowC_](#)

Функция насыщения.

- [nmppsClipCC_](#)

Функция насыщения с произвольными порогами.

- [nmppsThreshold_Lt_Gt_f](#)

Пороговая функция с нижним и верхним порогами

- [nmppsClipRShiftConvert_AddC_](#)

Сокращение разрядности данных с предварительной их обработкой.

- [nmppsClipConvert_AddC_](#)

Сокращение разрядности данных с предварительной их обработкой.

- [nmppsCmpEqC](#)

Сравнивает элементы массива на признак равенства константе.

- [nmppsCmpNe0](#)

Сравнивает элементы массива на признак неравенства нулю.

- [nmppsCmpEq0](#)

Сравнивает элементы массива на признак равенства нулю.

- [nmppsCmpNeC](#)

Сравнивает элементы массива на признак неравенства константе.

- [nmppsCmpNeC_flag](#)

Сравнивает элементы массива на признак неравенства константе.

- [nmppsCmpLtC](#)

Функция сравнения элементов массива с константой (меньше)

- [nmppsCmpGtC](#)

Функция сравнения элементов массива с константой (больше)

- [nmppsCmpEqV_](#)

Поэлементное сравнение элементов (неполной разрядности) двух векторов на признак равенства.

- [nmppsCmpNe](#)

Сравнивает элементы массива на признак неравенства константе.

- [nmppsCmpLt](#)

Функция сравнения элементов массива с константой (меньше)

- [nmppsCmpNeV_](#)

Поэлементное сравнение элементов двух векторов на признак неравенства.

5.132.1 Подробное описание

5.133 Операции сравнения

Группы

- [nmppsCmpLteC](#)

Функция сравнения элементов массива с константой (меньше или равно)

- [nmppsCmpLtC](#)

Функция сравнения элементов массива с константой (меньше или равно)

- [nmppsCmpGteC](#)

Функция сравнения элементов массива с константой (больше или равно)

- [nmppsCmpGtC](#)

Функция сравнения элементов массива с константой (больше)

5.133.1 Подробное описание

5.134 Статистические функции

5.135 Переупорядочивание и сортировка

Группы

- [VEC_QSort](#)

Сортировка массива по убыванию.

- [nmppsRemap_](#)

Переупорядочивание элементов вектора по таблице.

- [nmppSplitTmp](#)

Расщепляет массив на два, группируя по четным и нечетным элементам

- [nmppSplit](#)

Расщепляет массив на два массива, группируя по четным и нечетным элементам

- [nmppMerge](#)

Собирает массив из двух, чередуя элементы из каждого. Функция обратная nmppsSplit.

- [nmppSplit_32fc](#)

Расщепляет массив на два, группируя по четным и нечетным элементам

- [nmppsDecimate](#)

Делает выборку элементов из массива с некоторым шагом

5.135.1 Подробное описание

5.136 Элементарные математические функции

Группы

- Тригонометрические функции

Функция, вычисляющая синус/косинус над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

- Функция деления векторов

Функция, вычисляющая x/y (`pSrcVec1 / pSrcVec2`), где x и y (делимые числа и делители) это вектора чисел с плавающей точкой одинарной

- Экспонента

Функция, вычисляющая экспоненту над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

- Натуральный логарифм

Функция, вычисляющая натуральный логарифм над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

- Возведение в степень

Функция возводит в степень (`Deg`) каждый элемент вектора чисел с плавающей точкой двойной точности (64f)

- Вычисление квадратного корня

Функция вычисляет квадратный корень от каждого элемента входного вектора чисел с плавающей точкой

5.136.1 Подробное описание

5.137 Тригонометрические функции

Функция, вычисляющая синус/косинус над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Функции

- `void nmppsSin_32f (const nm32f *pSrcVec, nm32f *pDstVec, int nSize)`
- `void nmppsCos_32f (const nm32f *pSrcVec, nm32f *pDstVec, int nSize)`

5.137.1 Подробное описание

Функция, вычисляющая синус/косинус над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Аргументы

in	<code>pSrcVec</code>	входной вектор чисел с плавающей точкой
----	----------------------	---

Возвращаемые значения

[out]	pDstVec выходной вектор чисел с плавающей точкой
-------	--

Аргументы

in	nSize	число элементов в векторе
----	-------	---------------------------

Функции совместимы ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.138 Функция деления векторов

Функция, вычисляющая x/y ($pSrcVec1 / pSrcVec2$), где x и y (делимые числа и делители) это вектора чисел с плавающей точкой одинарной

Функции

- void nmppsDiv_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize)

5.138.1 Подробное описание

Функция, вычисляющая x/y ($pSrcVec1 / pSrcVec2$), где x и y (делимые числа и делители) это вектора чисел с плавающей точкой одинарной

Аргументы

in	pSrcVec1	входной вектор делимых чисел с плавающей точкой одинарной точности
in	pSrcVec2	входной вектор делителей с плавающей точкой одинарной точности

Возвращаемые значения

[out]	pDstVec выходной вектор частных с плавающей точкой одинарной точности
-------	---

Аргументы

in	nSize	число элементов в векторе
----	-------	---------------------------

Функция совместима ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.139 Экспонента

Функция, вычисляющая экспоненту над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Функции

- void nmppsExp_32f (const nm32f *pSrcVec, nm32f *pDstVec, int nSize)
- void nmppsExp_64f (const nm64f *pSrcVec, nm64f *pDstVec, int nSize)

5.139.1 Подробное описание

Функция, вычисляющая экспоненту над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Аргументы

in	pSrcVec	входной вектор чисел с плавающей точкой одинарной точности
----	---------	--

Возвращаемые значения

[out]	pDstVec	выходной вектор чисел с плавающей точкой одинарной точности
-------	---------	---

Аргументы

in	nSize	число элементов в векторе
----	-------	---------------------------

Функция совместима ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.140 Натуральный логарифм

Функция, вычисляющая натуральный логарифм над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Функции

- void nmppsLn_32f (const nm32f *pSrcVec, nm32f *pDstVec, int nSize)
- void nmppsLn_64f (const nm64f *pSrcVec, nm64f *pDstVec, int nSize)

5.140.1 Подробное описание

Функция, вычисляющая натуральный логарифм над каждым элементом вектора чисел с плавающей точкой одинарной точности (32f)

Аргументы

in	pSrcVec	входной вектор чисел с плавающей точкой одинарной точности
----	---------	--

Возвращаемые значения

[out]	pDstVec выходной вектор чисел с плавающей точкой одинарной точности
-------	---

Аргументы

in	nSize	число элементов в векторе
----	-------	---------------------------

Функция совместима ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.141 Возведение в степень

Функция возводит в степень (Deg) каждый элемент вектора чисел с плавающей точкой двойной точности (64f)

Функции

- void nmppsPowx_64f (const nm64f *pSrcVec, nm64f *pDstVec, **nm32u** Deg, int nSize)

5.141.1 Подробное описание

Функция возводит в степень (Deg) каждый элемент вектора чисел с плавающей точкой двойной точности (64f)

Аргументы

in	pSrcVec	входной вектор чисел с плавающей точкой двойной точности
----	---------	--

Возвращаемые значения

[out]	pDstVec выходной вектор чисел с плавающей точкой двойной точности
-------	---

Аргументы

in	Deg	степень, в которую возводится каждый элемент входного вектора
in	nSize	число элементов в векторе

Функция совместима ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.142 Вычисление квадратного корня

Функция вычисляет квадратный корень от каждого элемента входного вектора чисел с плавающей точкой

Функции

- void nmppsSqrt_32f (const nm32f *pSrcVec, nm32f *pDstVec, int nSize)

5.142.1 Подробное описание

Функция вычисляет квадратный корень от каждого элемента входного вектора чисел с плавающей точкой

Аргументы

in	pSrcVec	входной вектор чисел с плавающей точкой
----	---------	---

Возвращаемые значения

[out]	pDstVec	выходной вектор чисел с плавающей точкой
-------	---------	--

Аргументы

in	nSize	число элементов в векторе
----	-------	---------------------------

Функция совместима ТОЛЬКО с NMC-GCC-SDK и входит в состав библиотеки libnmpp-nmc4f.a

5.143 nmppsAbs

Вычисление абсолютных значений для элементов вектора.

Функции

- void nmppsAbs_4s (const nm4s *pSrcVec, nm4s *pDstVec, int nSize)
- void nmppsAbs_8s (const nm8s *pSrcVec, nm8s *pDstVec, int nSize)
- void nmppsAbs_16s (const nm16s *pSrcVec, nm16s *pDstVec, int nSize)
- void nmppsAbs_32s (const nm32s *pSrcVec, nm32s *pDstVec, int nSize)
- void nmppsAbs_64s (const nm64s *pSrcVec, nm64s *pDstVec, int nSize)

5.143.1 Подробное описание

Вычисление абсолютных значений для элементов вектора.

$$pDstVec[i] = abs\{pSrcVec[i]\},$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Restrictions:

Значения элементов вектора не должны быть равны минимальному значению для соответствующего типа (т.е. -128, -2¹⁵ и т.д). В противном случае, абсолютное значение для таких элементов вычисляется неверно, давая на выходе то же самое число.

5.144 nmppsAbs1

Функция логического вычисления модулей элементов вектора.

Функции

- void nmppsAbs1_4s (const **nm4s** *pSrcVec, **nm4s** *pDstVec, int nSize)
- void nmppsAbs1_8s (const **nm8s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsAbs1_16s (const **nm16s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsAbs1_32s (const **nm32s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsAbs1_64s (const **nm64s** *pSrcVec, **nm64s** *pDstVec, int nSize)

5.144.1 Подробное описание

Функция логического вычисления модулей элементов вектора.

$$pDstVec[i] = \begin{cases} pSrcVec[i], & \text{if } pSrcVec[i] \geq 0 \\ -pSrcVec[i] - 1, & \text{if } pSrcVec[i] < 0 \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.145 nmppsNeg

Изменение знака элементов вектора на противоположный.

Функции

- void nmppsNeg_8s (const **nm8s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsNeg_16s (const **nm16s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsNeg_32s (const **nm32s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsNeg_64s (const **nm64s** *pSrcVec, **nm64s** *pDstVec, int nSize)

5.145.1 Подробное описание

Изменение знака элементов вектора на противоположный.

$$pDstVec[i] = -pDstVec[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.146 nmppsAddC

Добавление к вектору константы.

Функции

- void nmppsAddC_8s (const **nm8s** *pSrcVec, **int8b** nVal, **nm8s** *pDstVec, int nSize)
- void nmppsAddC_16s (const **nm16s** *pSrcVec, **int16b** nVal, **nm16s** *pDstVec, int nSize)
- void nmppsAddC_32s (const **nm32s** *pSrcVec, **int32b** nVal, **nm32s** *pDstVec, int nSize)
- void nmppsAddC_64s (const **nm64s** *pSrcVec, **int64b** nVal, **nm64s** *pDstVec, int nSize)

5.146.1 Подробное описание

Добавление к вектору константы.

$$pDstVec[i] = pSrcVec[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.147 nmppsAddC_p

Добавление к вектору константы.

Функции

- void nmppsAddC_p64s (const **nm64s** *pSrcVec, **int64b** *pnVal, **nm64s** *pDstVec, int nSize)

5.147.1 Подробное описание

Добавление к вектору константы.

$$pDstVec[i] = pSrcVec[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
pnVal	Указатель на добавляемую константу.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.148 nmppsAddC

Добавление к вектору константы.

Функции

- void nmppsAddC_32fcr (const **nm32fcr** *pSrcVec, **nm32fcr** *pDstVec, float nVal, int nSize)
- void nmppsAddC_32f (const nm32f *pSrcVec, nm32f *pDstVec, float C, int nSize)

5.148.1 Подробное описание

Добавление к вектору константы.

Сложение вектора с константой.

$$pDstVec[i] = pSrcVec[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

$$pDstVec[i] = pSrcVec[i] + nVal$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
---------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

C	Складываемая константа.
nSize	Размер векторов в элементах.

Возвращает

void

5.149 nmppsAdd

Сложение двух векторов.

Функции

- void nmppsAdd_4s (const **nm4s** *pSrcVec1, const **nm4s** *pSrcVec2, **nm4s** *pDstVec, int nSize)
- void nmppsAdd_8s (const **nm8s** *pSrcVec1, const **nm8s** *pSrcVec2, **nm8s** *pDstVec, int nSize)
- void nmppsAdd_16s (const **nm16s** *pSrcVec1, const **nm16s** *pSrcVec2, **nm16s** *pDstVec, int nSize)
- void nmppsAdd_32s (const **nm32s** *pSrcVec1, const **nm32s** *pSrcVec2, **nm32s** *pDstVec, int nSize)
- void nmppsAdd_64s (const **nm64s** *pSrcVec1, const **nm64s** *pSrcVec2, **nm64s** *pDstVec, int nSize)

5.149.1 Подробное описание

Сложение двух векторов.

$$pDstVec[i] = pSrcVec1[i] + pSrcVec2[i],$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер вектора в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.150 nmppsAdd

Сложение двух векторов.

Функции

- void nmppsAdd_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize)

5.150.1 Подробное описание

Сложение двух векторов.

$$pDstVec[i] = pSrcVec1[i] + pSrcVec2[i],$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер вектора в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.151 nmppsAdd_AddC

Сложение двух векторов с прибавлением константы.

Функции

- void nmppsAdd_AddC_32s ([nm32s](#) *pSrcVec1, [nm32s](#) *pSrcVec2, int nVal, [nm32s](#) *pDstVec, int nSize)

5.151.1 Подробное описание

Сложение двух векторов с прибавлением константы.

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nVal	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

$$pDstVec[i] = pSrcVec1[i] + pSrcVec2[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

5.152 nmppsSubC

Вычитание константы из вектора.

Функции

- void nmppsSubC_4s (const **nm4s** *pSrcVec, **int4b** nVal, **nm4s** *pDstVec, int nSize)
- void nmppsSubC_8s (const **nm8s** *pSrcVec, **int8b** nVal, **nm8s** *pDstVec, int nSize)
- void nmppsSubC_16s (const **nm16s** *pSrcVec, **int16b** nVal, **nm16s** *pDstVec, int nSize)
- void nmppsSubC_32s (const **nm32s** *pSrcVec, **int32b** nVal, **nm32s** *pDstVec, int nSize)
- void nmppsSubC_64s (const **nm64s** *pSrcVec, **int64b** nVal, **nm64s** *pDstVec, int nSize)

5.152.1 Подробное описание

Вычитание константы из вектора.

$$pDstVec[i] = pSrcVec[i] - nVal$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Вычитаемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
----------------	------------------------

Возвращает

void

5.153 nmppsSubC

Вычитание константы из вектора.

Функции

- void nmppsSubC_32f (const **nm32f** *pSrcVec, **nm32f** *pDstVec, float C, int nSize)

5.153.1 Подробное описание

Вычитание константы из вектора.

$$pDstVec[i] = pSrcVec[i] - nVal$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
---------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

C	Вычитаемая константа.
nSize	Размер векторов в элементах.

Возвращает

void

5.154 nmppsSubCRev

Вычитание константы из вектора с переменой знака элементов вектора.

Функции

- void nmppsSubCRev_8s (const **nm8s** *pSrcVec, **int8b** nVal, **nm8s** *pDstVec, int nSize)
- void nmppsSubCRev_16s (const **nm16s** *pSrcVec, **int16b** nVal, **nm16s** *pDstVec, int nSize)
- void nmppsSubCRev_32s (const **nm32s** *pSrcVec, **int32b** nVal, **nm32s** *pDstVec, int nSize)
- void nmppsSubCRev_64s (const **nm64s** *pSrcVec, **int64b** nVal, **nm64s** *pDstVec, int nSize)

5.154.1 Подробное описание

Вычитание константы из вектора с переменой знака элементов вектора.

$$pDstVec[i] = nVal - pSrcVec[i],$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.155 nmppsSubCRev

Вычитание вектора из константы.

Функции

- void nmppsSubCRev_32f (const nm32f *pSrcVec, nm32f *pDstVec, float C, int nSize)

5.155.1 Подробное описание

Вычитание вектора из константы.

$$pDstVec[i] = C - pSrcVec[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
---------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

C	Вычитаемая константа.
nSize	Размер векторов в элементах.

Возвращает

void

5.156 nmppsSub

Вычитание двух векторов.

Функции

- void nmppsSub_4s (const [nm4s](#) *pSrcVec1, [nm4s](#) *pSrcVec2, [nm4s](#) *pDstVec, int nSize)
- void nmppsSub_8s (const [nm8s](#) *pSrcVec1, [nm8s](#) *pSrcVec2, [nm8s](#) *pDstVec, int nSize)
- void nmppsSub_16s (const [nm16s](#) *pSrcVec1, [nm16s](#) *pSrcVec2, [nm16s](#) *pDstVec, int nSize)
- void nmppsSub_32s (const [nm32s](#) *pSrcVec1, [nm32s](#) *pSrcVec2, [nm32s](#) *pDstVec, int nSize)
- void nmppsSub_64s (const [nm64s](#) *pSrcVec1, [nm64s](#) *pSrcVec2, [nm64s](#) *pDstVec, int nSize)

5.156.1 Подробное описание

Вычитание двух векторов.

$$pDstVec[i] = pSrcVec1[i] - pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Уменьшаемый вектор.
pSrcVec2	Вычитаемый вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.157 nmppsSub

Вычитание двух векторов.

Функции

- void nmppsSub_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize)

5.157.1 Подробное описание

Вычитание двух векторов.

$$pDstVec[i] = pSrcVec1[i] - pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Уменьшаемый вектор.
pSrcVec2	Вычитаемый вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.158 nmppsAbsDiff

Вычисление вектора модулей разности элементов двух векторов.

Функции

- void nmppsAbsDiff_8s (const nm8s *pSrcVec1, nm8s *pSrcVec2, nm8s *pDstVec, int nSize)
- void nmppsAbsDiff_16s (const nm16s *pSrcVec1, nm16s *pSrcVec2, nm16s *pDstVec, int nSize)
- void nmppsAbsDiff_32s (const nm32s *pSrcVec1, nm32s *pSrcVec2, nm32s *pDstVec, int nSize)
- void nmppsAbsDiff_64s (const nm64s *pSrcVec1, nm64s *pSrcVec2, nm64s *pDstVec, int nSize)

5.158.1 Подробное описание

Вычисление вектора модулей разности элементов двух векторов.

$$pDstVec[i] = \text{abs}\{pSrcVec1[i] - pSrcVec2[i]\},$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Входной вектор.
pSrcVec2	Вычитаемый вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Restrictions:

Разность элементов векторов не должна быть равна минимальному значению для соответствующего типа (т.е. -128, -2^{15} и т.д). В противном случае, абсолютное значение для таких элементов вычисляется не верно, давая на выходе то же самое число.

5.159 nmppsAbsDiff_f

Вычисление вектора модулей разности элементов двух векторов.

Функции

- void nmppsAbsDiff_32f (const nm32f *pSrcVec1, nm32f *pSrcVec2, nm32f *pDstVec, int nSize)

5.159.1 Подробное описание

Вычисление вектора модулей разности элементов двух векторов.

$$pDstVec[i] = \text{abs}\{pSrcVec1[i] - pSrcVec2[i]\},$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Входной вектор.
pSrcVec2	Вычитаемый вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.160 nmppsAbsDiff1

Функция логического вычисления модулей разностей элементов двух векторов.

Функции

- void nmppsAbsDiff1_8s ([nm8s](#) *pSrcVec1, [nm8s](#) *pSrcVec2, [nm8s](#) *pDstVec, int nSize)

5.160.1 Подробное описание

Функция логического вычисления модулей разностей элементов двух векторов.

$$pDstVec[i] = \begin{cases} pSrcVec1[i] - pSrcVec2[i], & \text{if } pSrcVec1[i] - pSrcVec2[i] \geq 0 \\ pSrcVec1[i] - pSrcVec2[i] - 1, & \text{if } pSrcVec1[i] - pSrcVec2[i] < 0 \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Входной вектор.
pSrcVec2	Вычитаемый вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Restrictions:

Разность элементов векторов не должна быть равна минимальному значению для соответствующего типа (т.е. -128, -2^15 и т.д). В противном случае, абсолютное значение для таких элементов вычисляется не верно, давая на выходе то же самое число.

5.161 nmppsMulC

Умножение вектора на константу.

Функции

- void nmppsMulC_8s (const [nm8s](#) *pSrcVec, [int8b](#) nVal, [nm8s](#) *pDstVec, int nSize)
- void nmppsMulC_8s16s (const [nm8s](#) *pSrcVec, [int16b](#) nVal, [nm16s](#) *pDstVec, int nSize)
- void nmppsMulC_16s (const [nm16s](#) *pSrcVec, [int16b](#) nVal, [nm16s](#) *pDstVec, int nSize)
- void nmppsMulC_16s32s (const [nm16s](#) *pSrcVec, [int32b](#) nVal, [nm32s](#) *pDstVec, int nSize)
- void nmppsMulC_32s (const [nm32s](#) *pSrcVec, [int32b](#) nVal, [nm32s](#) *pDstVec, int nSize)
- void nmppsMulC_32s64s (const [nm32s](#) *pSrcVec, [int64b](#) nVal, [nm64s](#) *pDstVec, int nSize)
- void nmppsMulC_64s (const [nm64s](#) *pSrcVec, [int64b](#) nVal, [nm64s](#) *pDstVec, int nSize)
- void nmppsMulC_2s16s (const [nm2s](#) *pSrcVec, [int16b](#) nVal, [nm16s](#) *pDstVec, int nSize)

5.161.1 Подробное описание

Умножение вектора на константу.

$$pDstVec[i] = nVal \cdot pSrcVec[i],$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Константа-множитель.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.162 nmppsMulC

Умножение вектора на константу.

Функции

- void nmppsMulC_32f (const nm32f *pSrcVec, nm32f *pDstVec, float C, int nSize)

5.162.1 Подробное описание

Умножение вектора на константу.

Аргументы

pSrcVec	Входной вектор.
nVal	Константа-множитель.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.163 nmppsMul_Mul_Add

Умножение двух пар векторов с последующим суммированием результатов ($pDstVec = pSrcVec1 * pSrcVec2 + pSrcVec3 * pSrcVec4$)

Функции

- void nmppsMul_Mul_Add_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, const nm32f *pSrcVec3, const nm32f *pSrcVec4, nm32f *pDstVec, int nSize)
- void nmppsMul_Mul_Add_32fcr (const nm32fcr *pSrcVec1, const nm32fcr *pSrcVec2, const nm32fcr *pSrcVec3, const nm32fcr *pSrcVec4, nm32fcr *pDstVec, int nSize)
- void nmppsMul_Mul_Add_64f (const nm64f *pSrcVec1, const nm64f *pSrcVec2, const nm64f *pSrcVec3, const nm64f *pSrcVec4, nm64f *pDstVec, int nSize)

5.163.1 Подробное описание

Умножение двух пар векторов с последующим суммированием результатов ($pDstVec = pSrcVec1 * pSrcVec2 + pSrcVec3 * pSrcVec4$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
pSrcVec3	Входной вектор3
pSrcVec4	Входной вектор4
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.164 nmppsMul_Add

Умножение с накоплением

Функции

- void nmppsMul_Add_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, const nm32f *pSrcVecAdd, nm32f *pDstVec, int nSize)
- void nmppsMul_Add_64f (const nm64f *pSrcVec1, const nm64f *pSrcVec2, const nm64f *pSrcVecAdd, nm64f *pDstVec, int nSize)
- void nmppsMul_Add_32fcr (const nm32fcr *pSrcVec1, const nm32fcr *pSrcVec2, const nm32fcr *pSrcVecAdd, nm32fcr *pDstVec, int nSize)

5.164.1 Подробное описание

Умножение с накоплением

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
pSrcVecAdd	Входной вектор
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.165 nmppsMul_ConjMul_Add

Умножение пары векторов (вектор pSrcVec4 комплексно-сопряженный) с последующим суммированием результата ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] + pSrcVec3[i] * \text{Conj}(pSrcVec4[i])$)

Функции

- void nmppsMul_ConjMul_Add_32fcr (const nm32fcr *pSrcVec1, const nm32fcr *pSrcVec2, const nm32fcr *pSrcVec3, const nm32fcr *pSrcVec4, nm32fcr *pDstVec, int nSize)

5.165.1 Подробное описание

Умножение пары векторов (вектор pSrcVec4 комплексно-сопряженный) с последующим суммированием результата ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] + pSrcVec3[i] * \text{Conj}(pSrcVec4[i])$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
pSrcVec3	Входной вектор3
pSrcVec4	Входной вектор4
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.166 nmppsMulC_AddC

Умножает вектор на константу с прибавлением другой константы ($pDstVec[i] = nMulC * pSrcVec[i] + nAddC$)

Функции

- void nmppsMulC_AddC_32f (const nm32f *pSrcVec, float nMulC, float nAddC, nm32f *pDstVec, int nSize)

5.166.1 Подробное описание

Умножает вектор на константу с прибавлением другой константы ($pDstVec[i] = nMulC * pSrcVec[i] + nAddC$)

Аргументы

pSrcVec	Входной вектор
nMulC	Константа
nAddC	Константа
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.167 nmppsMulC_AddV_AddC

Умножение вектора на константу с прибавлением вектора и константы ($pDstVec[i] = nMulC * pSrcVec[i] + pVecAdd[i] + nAddC$)

Функции

- void nmppsMulC_AddV_AddC_32f (nm32f *pSrcVec, float nMulC, nm32f *pVecAdd, float nAddC, nm32f *pDstVec, int nSize)

5.167.1 Подробное описание

Умножение вектора на константу с прибавлением вектора и константы ($pDstVec[i] = nMulC * pSrcVec[i] + pVecAdd[i] + nAddC$)

Аргументы

pSrcVec	Входной вектор
nMulC	Константа
pVecAdd	Входной вектор
nAddC	Константа
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.168 nmppsMul

Умножение векторов ($pDstVec[i] = pSrcVec1[i] * pSrcVec1[i]$)

Функции

- void nmppsMul_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize)
- void nmppsMul_32fcr (const nm32fcr *pSrcVec1, const nm32fcr *pSrcVec2, nm32fcr *pDstVec, int nSize)

5.168.1 Подробное описание

Умножение векторов ($pDstVec[i] = pSrcVec1[i] * pSrcVec1[i]$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.169 nmppsConjMul

Умножение вектора на комплексно-сопряженный вектор ($pDstVec[i] = pSrcVec1[i] * \text{Conj}(pSrcVec1[i])$)

Функции

- void nmppsConjMul_32fcr (const nm32fcr *pSrcVec1, const nm32fcr *pSrcVec2, nm32fcr *pDstVec, int nSize)

5.169.1 Подробное описание

Умножение вектора на комплексно-сопряженный вектор ($pDstVec[i] = pSrcVec1[i] * \text{Conj}(pSrcVec1[i])$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.170 nmppsMul_Mul_Sub

Умножение пары векторов с последующим вычитанием результатов ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] - pSrcVec3[i] * pSrcVec4[i]$)

Функции

- void nmppsMul_Mul_Sub_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, const nm32f *pSrcVec3, const nm32f *pSrcVec4, nm32f *pDstVec, int nSize)

5.170.1 Подробное описание

Умножение пары векторов с последующим вычитанием результатов ($pDstVec[i] = pSrcVec1[i] * pSrcVec2[i] - pSrcVec3[i] * pSrcVec4[i]$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
pSrcVec3	Входной вектор1
pSrcVec4	Входной вектор4
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.171 nmppsMulC_Add

Умножение вектора на константу с накоплением ($pDstVec[i] = pSrcVec1[i] * C + pSrcVec2[i]$)

Функции

- void nmppsMulC_AddV_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, float C, int nSize)

5.171.1 Подробное описание

Умножение вектора на константу с накоплением ($pDstVec[i] = pSrcVec1[i] * C + pSrcVec2[i]$)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.172 nmppsMul_AddC

Поэлементное умножение векторов с прибавлением константы.

Функции

- void nmppsMul_AddC_64s (const nm64s *pSrcVec1, const nm64s *pSrcVec2, const nm64s *pnVal, nm64s *pDstVec, int nSize)

5.172.1 Подробное описание

Поэлементное умножение векторов с прибавлением константы.

$$pDstVec[i] = pSrcVec1[i] \cdot pSrcVec2[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Входной вектор.
pSrcVec2	Входной вектор.
pnVal	указаель на константу-инкремент.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.173 nmppsMul_AddC

Поэлементное умножение векторов с прибавлением константы.

Функции

- void nmppsMul_AddC_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, float nValueAddC, nm32f *pDstVec, int nSize)

5.173.1 Подробное описание

Поэлементное умножение векторов с прибавлением константы.

$$pDstVec[i] = pSrcVec1[i] \cdot pSrcVec2[i] + nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Входной вектор.
pSrcVec2	Входной вектор.
nValueAddC	указаель на константу-инкремент.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.174 nmppsMulC_AddC

Умножение вектора на константу с прибавлением константы.

Функции

- void nmppsMulC_AddC_32s (const **nm32s** *pSrcVec, int nMulVal, int nAddVal, **nm32s** *pDstVec, int nSize)
- void **nmppsMulC_AddC_2x32s** (**int32x2** *dataSparseSrc, **int32x2** *mulArg, **int32x2** *addArg, **int32x2** *dataSparseDst, int size, int stepSparseSrc, int stepSparseDst)
Sparse vector by constant multiplication with addition of constant.

5.174.1 Подробное описание

Умножение вектора на константу с прибавлением константы.

$$pDstVec[i] = nMulVal \cdot pSrcVec[i] + nAddVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nMulVal	Константа-множитель.
nAddVal	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.174.2 Функции

5.174.2.1 nmppsMulC_AddC_2x32s()

```
void nmppsMulC_AddC_2x32s (
    int32x2 * dataSparseSrc,
    int32x2 * mulArg,
    int32x2 * addArg,
    int32x2 * dataSparseDst,
    int size,
    int stepSparseSrc,
    int stepSparseDst )
```

Sparse vector by constant multiplication with addition of constant.

$$\text{dataSparseDst}[i \cdot \text{stepSparseDst}][k] = \text{dataSparseSrc}[i \cdot \text{stepSparseSrc}][k] \cdot \text{mulArg}[k] + \text{addArg}[k], \\ i = \overline{0 \dots size - 1}; k = \overline{0 \dots K - 1},$$

where K is value of intWxK type

Аргументы

in	dataSparseSrc	Input sparse vector of 64-bit packed words
in	mulArg	Packed 64-bit word with values to multiply
in	addArg	Packed 64-bit word with values to add
in	dataSparseDst	Ouput sparse vector of 64-bit packed words
in	size	actual amount of 64-bit packed words in sparse vector to be processed
in	stepSparseSrc	64-bit step between input packed words in memory . By default=1 means that input vector is continuous
in	stepSparseDst	64-bit step between output packed words in memory. By default=1 means that output vector is continuous

Возвращает

void

5.175 nmppsRShiftC_MulC_AddC

Функции

- void nmppsRShiftC_MulC_AddC_2x32s (**int32x2** *dataSparseSrc, **int32x2** *pshiftArg, **int32x2** *mulArg, **int32x2** *addArg, **int32x2** *dataSparseDst, int size, int stepSparseSrc, int stepSparseDst)

5.175.1 Подробное описание

Sparse vector by constant multiplication with addition of constant.

$dataSparseDst[i \cdot stepSparseDst][k] = (dataSparseSrc[i \cdot stepSparseSrc][k] >> pshiftArg[k]) \cdot mulArg[k] + addArg[k],$

$$i = \overline{0 \dots size - 1}; k = \overline{0 \dots K - 1},$$

where K is value of intWxK type

Аргументы

in	dataSparseSrc	Input sparse vector of 64-bit packed words
in	pshiftArg	Packed 64-bit word of values for preshifting of input data = [2,4,6,8...28,30]
in	mulArg	Packed 64-bit word with values to multiply
in	addArg	Packed 64-bit word with values to add
in	dataSparseDst	Ouput sparse vector of 64-bit packed words
in	size	actual amount of 64-bit packed words in sparse vector to be processed
in	stepSparseSrc	64-bit step between input packed words in memory . By default=1 means that input vector is continuous
in	stepSparseDst	64-bit step between output packed words in memory. By default=1 means that output vector is continuous

Возвращает

void

5.176 nmppsMulC_AddV_AddC

Умножение вектора на константу с прибавлением вектора и константы.

Функции

- void nmppsMulC_AddV_AddC_32s (**nm32s** *pSrcVec1, int nMulVal, **nm32s** *pSrcVec2, int n← AddVal, **nm32s** *pDstVec, int nSize)

5.176.1 Подробное описание

Умножение вектора на константу с прибавлением вектора и константы.

$$pDstVec[i] = nMulVal \cdot pSrcVec1[i] + pSrcVec2[i] + nAddVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
nMulVal	Константа-множитель.
pSrcVec2	Второй входной вектор.
nAddVal	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.177 nmppsSumN

Сложение нескольких векторов.

Функции

- void nmppsSumN_8s16s ([nm8s](#) **ppSrcVec, [nm16s](#) *pDstVec, int nSize, int nNumberOfVectors)
- void nmppsSumN_16s ([nm16s](#) **ppSrcVec, [nm16s](#) *pDstVec, int nSize, int nNumberOfVectors)

5.177.1 Подробное описание

Сложение нескольких векторов.

$$pDstVec[i] = \sum_{j=0}^{(nNumberOfVectors-1)} ppSrcVec(j)(i)$$

Аргументы

ppSrcVec	Массив указателей на суммируемые вектора.
nNumberOfVectors	Число суммируемых векторов.
nSize	Размер векторов в элементах =[32*PACK]

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.178 nmppsDivC

Деление вектора на константу.

Функции

- void nmppsDivC_32s ([nm32s](#) *pSrcVec, int nDivisor, [nm32s](#) *pDstVec, int nSize, void *pTmpBuf1, void *pTmpBuf2)

5.178.1 Подробное описание

Деление вектора на константу.

$$pDstVec[i] = \frac{pSrcVec[i]}{Divisor},$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nDivisor	Константа-делитель.
nSize	Размер входного вектора в элементах.
pTmpBuf1	Временный массив размером nSize 64-х разрядных слов.
pTmpBuf2	Временный массив размером nSize 64-х разрядных слов.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Restrictions:

- Допустимые значения для элементов входного вектора лежат в диапазоне [-4095,...,4095];
- Допустимые значения для делителя лежат в диапазоне [1,2,..145].

Заметки

Для корректного использования in-place параметров следует учитывать порядок получения промежуточных результатов:

the order of obtaining intermediate results:

pSrcVec => pTmpBuf1 (1cpl:L<=>G) - 1/x Multiplying (in-place is supported)

pTmpBuf1 => pTmpBuf2 (1cpl:G<=>L) - Scaling down (in-place is supported)

pTmpBuf2 => pDstVec (2cpl:L<=>G) - Result correction (in-place is supported)

Примеры использования in-place параметров:

nmppsDiv_(L0,G0,10240,3,G0,L0);

nmppsDiv_(L0,L0,10240,3,L0,L0);

5.179 nmppsSum

Возвращает сумму всех элементов вектора.

Функции

- void nmppsSum_8s (const nm8s *pSrcVec, int nSize, int32b *pnRes)
- void nmppsSum_16s (const nm16s *pSrcVec, int nSize, int64b *pnRes)
- void nmppsSum_32s (const nm32s *pSrcVec, int nSize, int64b *pnRes)
- void nmppsSum_64s (const nm64s *pSrcVec, int nSize, int64b *pnRes)

5.179.1 Подробное описание

Возвращает сумму всех элементов вектора.

$$return = \sum_{i=0}^{(nSize-1)} pSrcVec[i]$$

Аргументы

pSrcVec	Входной вектор.
pTmpBuf	Временный массив размера nSize 64-х разрядных слов.
nSize	Размер векторов в элементах.

Возвращает

Сумма элементов вектора.

5.180 nmppsSum_1

Возвращает сумму всех элементов вектора.

Функции

- void nmppsSum_1 (const **nm1** *pSrcVec, int nSize, **int32b** *pnRes, void *pTmpBuf)

5.180.1 Подробное описание

Возвращает сумму всех элементов вектора.

$$return = \sum_{i=0}^{(nSize-1)} pSrcVec[i]$$

Аргументы

pSrcVec	Входной вектор.
pTmpBuf	Временный массив размера nSize 64-х разрядных слов.
nSize	Размер векторов в элементах.

Возвращает

Сумма элементов вектора.

5.181 nmppsDotProd_sm

Скалярное умножение двух векторов.

Функции

- int nmppsDotProd_8s8sm (const **nm8s** *pSrcVec1, const **nm8s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)
- int nmppsDotProd_8s16sm (const **nm8s** *pSrcVec1, const **nm16s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)
- int nmppsDotProd_8s32sm (const **nm8s** *pSrcVec1, const **nm32s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)
- int nmppsDotProd_16s16sm (const **nm16s** *pSrcVec1, const **nm16s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)
- int nmppsDotProd_16s32sm (const **nm16s** *pSrcVec1, const **nm32s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)
- int nmppsDotProd_32s32sm (const **nm32s** *pSrcVec1, const **nm32s** *pSrcVec2, int nSize, **int64b** *pnRes, **nm64s** *tmp)

5.181.1 Подробное описание

Скалярное умножение двух векторов.

$$nRes = \sum_{i=0}^{nSize-1} pSrcVec1[i] \cdot pSrcVec2[i]$$

Аргументы

pSrcVec1	Первый вектор.
pSrcVec2	Второй вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pnRes	Указатель на результирующее значение.
--------------	---------------------------------------

Возвращает

pTmpBuff Временный массив из nSize элементов.
void

5.182 nmppsDotProd

Скалярное умножение двух векторов.

Функции

- void nmppsDotProd_8s64s (const **nm8s** *pSrcVec1, const **nm64s** *pSrcVec2, int nSize, **int64b** *pnRes)
- void nmppsDotProd_16s64s (const **nm16s** *pSrcVec1, const **nm64s** *pSrcVec2, int nSize, **int64b** *pnRes)
- void nmppsDotProd_32s64s (const **nm32s** *pSrcVec1, const **nm64s** *pSrcVec2, int nSize, **int64b** *pnRes)
- void nmppsDotProd_64s64s (const **nm64s** *pSrcVec1, const **nm64s** *pSrcVec2, int nSize, **int64b** *pnRes)

5.182.1 Подробное описание

Скалярное умножение двух векторов.

$$nRes = \sum_{i=0}^{nSize-1} pSrcVec1[i] \cdot pSrcVec2[i]$$

Аргументы

pSrcVec1	Первый вектор.
pSrcVec2	Второй вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pnRes	Указатель на результирующее значение.
--------------	---------------------------------------

Возвращает

pTmpBuff Временный массив из nSize элементов.
void

5.183 nmppsWeightedSum

Поэлементное взвешенное суммирование элементов двух векторов

Функции

- void nmppsWeightedSum_8s16s (**nm8s** *pSrcVec1, int nW1, **nm8s** *pSrcVec2, int nW2, **nm16s** *pDstVec, int nSize)
- void nmppsWeightedSum_16s32s (**nm16s** *pSrcVec1, int nW1, **nm16s** *pSrcVec2, int nW2, **nm32s** *pDstVec, int nSize)
- void nmppsWeightedSum_32s64s (**nm32s** *pSrcVec1, **nm64s** nW1, **nm32s** *pSrcVec2, **nm64s** nW2, **nm64s** *pDstVec, int nSize)

5.183.1 Подробное описание

Поэлементное взвешенное суммирование элементов двух векторов

$$pDstVec[i] = nW1 \cdot pSrcVec1[i] + nW2 \cdot pSrcVec2[i],$$

$$= \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	1-ый входной вектор.
nW1	1-ый весовой коэффициент
pSrcVec2	2-ой входной вектор.
nW2	2-ой весовой коэффициент
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.184 nmppsNot_

Функция логического "НЕ".

Функции

- void nmppsNot_2u (const **nm2u** *pSrcVec, **nm2u** *pDstVec, int nSize)
- void nmppsNot_4u (const **nm4u** *pSrcVec, **nm4u** *pDstVec, int nSize)
- void nmppsNot_8u (const **nm8u** *pSrcVec, **nm8u** *pDstVec, int nSize)
- void nmppsNot_16u (const **nm16u** *pSrcVec, **nm16u** *pDstVec, int nSize)
- void nmppsNot_32u (const **nm32u** *pSrcVec, **nm32u** *pDstVec, int nSize)
- void nmppsNot_64u (const **nm64u** *pSrcVec, **nm64u** *pDstVec, int nSize)

5.184.1 Подробное описание

Функция логического "НЕ".

$$pDstVec[i] = \overline{pSrcVec[i]},$$

$$i = \overline{0 \dots nSize - 1}$$

Функция изменяет значения всех битов входного вектора на противоположные.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.185 nmppsAndC

Функция логического "И" между вектором и константой.

Функции

- void nmppsAndC_4u (const **nm4u** *pSrcVec, **uint4b** nVal, **nm4u** *pDstVec, int nSize)
- void nmppsAndC_8u (const **nm8u** *pSrcVec, **uint8b** nVal, **nm8u** *pDstVec, int nSize)
- void nmppsAndC_16u (const **nm16u** *pSrcVec, **uint16b** nVal, **nm16u** *pDstVec, int nSize)
- void nmppsAndC_32u (const **nm32u** *pSrcVec, **uint32b** nVal, **nm32u** *pDstVec, int nSize)
- void nmppsAndC_64u (const **nm64u** *pSrcVec, **uint64b** nVal, **nm64u** *pDstVec, int nSize)

5.185.1 Подробное описание

Функция логического "И" между вектором и константой.

$$pDstVec[i] = pSrcVec[i] \wedge nVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.186 nmppsAnd

Функция логического "И" между двумя векторами.

Функции

- void nmppsAnd_1 (const **nm1** *pSrcVec1, const **nm1** *pSrcVec2, **nm1** *pDstVec, int nSize)
- void nmppsAnd_2u (const **nm2u** *pSrcVec1, const **nm2u** *pSrcVec2, **nm2u** *pDstVec, int nSize)
- void nmppsAnd_4u (const **nm4u** *pSrcVec1, const **nm4u** *pSrcVec2, **nm4u** *pDstVec, int nSize)
- void nmppsAnd_8u (const **nm8u** *pSrcVec1, const **nm8u** *pSrcVec2, **nm8u** *pDstVec, int nSize)
- void nmppsAnd_16u (const **nm16u** *pSrcVec1, const **nm16u** *pSrcVec2, **nm16u** *pDstVec, int nSize)
- void nmppsAnd_32u (const **nm32u** *pSrcVec1, const **nm32u** *pSrcVec2, **nm32u** *pDstVec, int nSize)
- void nmppsAnd_64u (const **nm64u** *pSrcVec1, const **nm64u** *pSrcVec2, **nm64u** *pDstVec, int nSize)

5.186.1 Подробное описание

Функция логического "И" между двумя векторами.

$$pDstVec[i] = pSrcVec1[i] \wedge pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.187 nmppsAnd4V_

Функция логического "И" между четырьмя векторами.

Функции

- void nmppsAnd4V_64u (**nm64u** *pSrcVec1, **nm64u** *pSrcVec2, **nm64u** *pSrcVec3, **nm64u** *pSrcVec4, **nm64u** *pDstVec, int nSize)

5.187.1 Подробное описание

Функция логического "И" между четырьмя векторами.

$$pDstVec[i] = pSrcVec1[i] \wedge pSrcVec2[i] \wedge pSrcVec3[i] \wedge pSrcVec4[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
pSrcVec3	Третий входной вектор.
pSrcVec4	Четвертый входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.188 nmppsAndNotV_

Функция логического "И-НЕ" между двумя векторами.

Функции

- void nmppsAndNotV_64u ([nm64u *pSrcVec1](#), [nm64u *pSrcVec2](#), [nm64u *pDstVec](#), int nSize)

5.188.1 Подробное описание

Функция логического "И-НЕ" между двумя векторами.

$$pDstVec[i] = pSrcVec1[i] \wedge \neg pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

<code>pSrcVec1</code>	Первый входной вектор.
<code>pSrcVec2</code>	Второй входной вектор.
<code>nSize</code>	Размер векторов в элементах.

Возвращаемые значения

<code>pDstVec</code>	Результирующий вектор.
----------------------	------------------------

Возвращает

void

5.189 nmppsOrC

Функция логического "ИЛИ" между вектором и константой.

Функции

- void nmppsOrC_8u (const **nm8u** *pSrcVec, **uint8b** nVal, **nm8u** *pDstVec, int nSize)
- void nmppsOrC_16u (const **nm16u** *pSrcVec, **uint16b** nVal, **nm16u** *pDstVec, int nSize)
- void nmppsOrC_32u (const **nm32u** *pSrcVec, **uint32b** nVal, **nm32u** *pDstVec, int nSize)
- void nmppsOrC_64u (const **nm64u** *pSrcVec, **uint64b** nVal, **nm64u** *pDstVec, int nSize)

5.189.1 Подробное описание

Функция логического "ИЛИ" между вектором и константой.

$$pDstVec[i] = pSrcVec[i] \vee nVal$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.190 nmppsOr

Функция логического "ИЛИ" между двумя векторами.

Функции

- void nmppsOr_1 (const **nm1** *pSrcVec1, const **nm1** *pSrcVec2, **nm1** *pDstVec, int nSize)
- void nmppsOr_2u (const **nm2u** *pSrcVec1, const **nm2u** *pSrcVec2, **nm2u** *pDstVec, int nSize)
- void nmppsOr_4u (const **nm4u** *pSrcVec1, const **nm4u** *pSrcVec2, **nm4u** *pDstVec, int nSize)
- void nmppsOr_8u (const **nm8u** *pSrcVec1, const **nm8u** *pSrcVec2, **nm8u** *pDstVec, int nSize)
- void nmppsOr_16u (const **nm16u** *pSrcVec1, const **nm16u** *pSrcVec2, **nm16u** *pDstVec, int nSize)
- void nmppsOr_32u (const **nm32u** *pSrcVec1, const **nm32u** *pSrcVec2, **nm32u** *pDstVec, int nSize)
- void nmppsOr_64u (const **nm64u** *pSrcVec1, const **nm64u** *pSrcVec2, **nm64u** *pDstVec, int nSize)

5.190.1 Подробное описание

Функция логического "ИЛИ" между двумя векторами.

$$pDstVec[i] = pSrcVec1[i] \vee pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.191 nmppsOr3V_

Функция логического "ИЛИ" между четырьмя векторами.

Функции

- void nmppsOr3V_64u ([nm64u](#) *pSrcVec1, [nm64u](#) *pSrcVec2, [nm64u](#) *pSrcVec3, [nm64u](#) *pDstVec, int nSize)

5.191.1 Подробное описание

Функция логического "ИЛИ" между четырьмя векторами.

$$pDstVec[i] = pSrcVec1[i] \vee pSrcVec2[i] \vee pSrcVec3[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
pSrcVec3	Третий входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.192 nmppsOr4V_

Функция логического "ИЛИ" между четырьмя векторами.

Функции

- void nmppsOr4V_64u ([nm64u](#) *pSrcVec1, [nm64u](#) *pSrcVec2, [nm64u](#) *pSrcVec3, [nm64u](#) *pSrcVec4, [nm64u](#) *pDstVec, int nSize)

5.192.1 Подробное описание

Функция логического "ИЛИ" между четырьмя векторами.

$$pDstVec[i] = pSrcVec1[i] \vee pSrcVec2[i] \vee pSrcVec3[i] \vee pSrcVec4[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
pSrcVec3	Третий входной вектор.
pSrcVec4	Четвертый входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.193 nmppsXorC

Функция логического "Исключающего ИЛИ" между вектором и константой.

Функции

- void nmppsXorC_8u (const **nm8u** *pSrcVec, **uint8b** nVal, **nm8u** *pDstVec, int nSize)
- void nmppsXorC_16u (const **nm16u** *pSrcVec, **uint16b** nVal, **nm16u** *pDstVec, int nSize)
- void nmppsXorC_32u (const **nm32u** *pSrcVec, **uint32b** nVal, **nm32u** *pDstVec, int nSize)
- void nmppsXorC_64u (const **nm64u** *pSrcVec, **uint64b** nVal, **nm64u** *pDstVec, int nSize)

5.193.1 Подробное описание

Функция логического "Исключающего ИЛИ" между вектором и константой.

$$pDstVec[i] = pSrcVec[i] \vee nVal$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nVal	Константа.
pnVal	Указатель на константу.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.194 nmppsXor

Функция логического "Исключающего ИЛИ" между двумя векторами.

Функции

- void nmppsXor_4u (const **nm4u** *pSrcVec1, const **nm4u** *pSrcVec2, **nm4u** *pDstVec, int nSize)
- void nmppsXor_8u (const **nm8u** *pSrcVec1, const **nm8u** *pSrcVec2, **nm8u** *pDstVec, int nSize)
- void nmppsXor_16u (const **nm16u** *pSrcVec1, const **nm16u** *pSrcVec2, **nm16u** *pDstVec, int nSize)
- void nmppsXor_32u (const **nm32u** *pSrcVec1, const **nm32u** *pSrcVec2, **nm32u** *pDstVec, int nSize)
- void nmppsXor_64u (const **nm64u** *pSrcVec1, const **nm64u** *pSrcVec2, **nm64u** *pDstVec, int nSize)

5.194.1 Подробное описание

Функция логического "Исключающего ИЛИ" между двумя векторами.

$$pDstVec[i] = pSrcVec1[i] \vee pSrcVec2[i]$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.195 nmppsMaskV_

Функция логического ИЛИ с предварительным маскированием двух векторов.

Функции

- void nmppsMaskV_64u (**nm64u** *pSrcVec1, **nm64u** *pSrcVec2, **nm64u** *pMaskVec, **nm64u** *p← Dst Vec, int nSize)

5.195.1 Подробное описание

Функция логического ИЛИ с предварительным маскированием двух векторов.

$$pDstVec[i] = (pSrcVec1[i] \quad and \quad pMaskVec[i]) \quad or \quad (pSrcVec2[i] \quad and \quad \overline{pMaskVec[i]})$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
pMaskVec	Вектор маски.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.196 nmppsRShiftC

Операция арифметического сдвига вправо.

Функции

- void nmppsRShiftC_8s (const **nm8s** *pSrcVec, int nShift, **nm8s** *pDstVec, int nSize)
- void nmppsRShiftC_16s (const **nm16s** *pSrcVec, int nShift, **nm16s** *pDstVec, int nSize)
- void nmppsRShiftC_32s (const **nm32s** *pSrcVec, int nShift, **nm32s** *pDstVec, int nSize)
- void nmppsRShiftC_64s (const **nm64s** *pSrcVec, int nShift, **nm64s** *pDstVec, int nSize)

5.196.1 Подробное описание

Операция арифметического сдвига вправо.

$$pDstVec[i] = pSrcVec[i] >> nShift,$$

$$i = \overline{0 \dots nSize - 1}$$

Функции реализуют операции арифметического сдвига вправо битов элементов вектора. Освободившиеся биты заполняются знаковым битом - старшим битом.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер вектора в элементах.
nShift	Параметр определяет на сколько позиций нужно сдвинуть биты элементов вектора.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.197 nmppsRShiftC_

Операция логического сдвига.

Функции

- void nmppsRShiftC_4u (const **nm4u** *pSrcVec, int nShift, **nm4u** *pDstVec, int nSize)
- void nmppsRShiftC_8u (const **nm8u** *pSrcVec, int nShift, **nm8u** *pDstVec, int nSize)
- void nmppsRShiftC_16u (const **nm16u** *pSrcVec, int nShift, **nm16u** *pDstVec, int nSize)
- void nmppsRShiftC_32u (const **nm32u** *pSrcVec, int nShift, **nm32u** *pDstVec, int nSize)
- void nmppsRShiftC_64u (const **nm64u** *pSrcVec, int nShift, **nm64u** *pDstVec, int nSize)

5.197.1 Подробное описание

Операция логического сдвига.

$$pDstVec[i] = pSrcVec[i] >> nShift,$$

$$i = \overline{0 \dots nSize - 1}$$

Функции реализуют операции логического сдвига вправо битов элементов вектора. Сдвиг осуществляется на число бит, указанных в соответствующем операнде. Освободившиеся биты заполняются нулями.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.
nShift	Определяет на сколько позиций нужно сдвинуть биты элемента вектора.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.198 nmppsRShiftC_AddC_

Операция логического сдвига.

Операция логического сдвига.

$$pDstVec[i] = (pSrcVec[i] >> nShift) + nAddVal,$$

$$i = \overline{0 \dots nSize - 1}$$

Функции реализуют операции логического сдвига вправо битов элементов вектора с прибавлением константы. Сдвиг осуществляется на число бит, указанных в соответствующем операнде. Освободившиеся биты заполняются нулями.

Аргументы

pSrcVec	Входной вектор.
nAddVal	Константа для суммирования.
nSize	Размер векторов в элементах.
nShift	Определяет на сколько позиций нужно сдвинуть биты элемента вектора.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.199 nmppsDisplaceBits

Непрерывное смещение битов внутри бинарного массива в сторону конца массива

Функция смещает биты внутри бинарного массива на несколько позиций (nBits) в сторону конца массива. Внутри 64р. слова младшие биты сдвигаются на старшие позиции того же слова, а старшие биты перемещаются в младшие позиции следующего 64р. слова. Освободившееся место в первом 64р. слове заполняется старшими битами 64р. слова с адреса pnBits. Сдвинутые биты сохраняются в массиве pDst. Пример сдвига на 8 бит :

Функции

- void nmppsFwdShiftBitstream (const nm64u *pSrcVec, nm64u *pDstVec, nm64u *pnBits, int nBits, int nSize)

5.199.1 Подробное описание

Непрерывное смещение битов внутри бинарного массива в сторону конца массива

Функция смещает биты внутри бинарного массива на несколько позиций (nBits) в сторону конца массива. Внутри 64р. слова младшие биты сдвигаются на старшие позиции того же слова, а старшие биты перемещаются в младшие позиции следующего 64р. слова. Освободившееся место в первом 64р. слове заполняется старшими битами 64р. слова с адреса pnBits. Сдвинутые биты сохраняются в массиве pDst. Пример сдвига на 8 бит :

```
pnBits =[AB000000000000]
pSrcVec=[0807060504030201][FF0F0E0D0C0B0A09]
pDstVec=[07060504030201AB][0F0E0D0C0B0A0908]
```

Последние 8 бит массива pDstVec будут потеряны. Если же указатель pBits установить на последнее 64р. слово в результате получится циклическое перемещение бит.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в 64р. элементах.
pnBits	Указатель на 64р-слово, старшие биты которого записываются на освобождающуюся при сдвиге младшую часть первого 64р. слова
nBits	Кол-во позиций на которое происходит смещение бит :nBits=[2,4,6....62].

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.200 nmppsSet-инициализация

Функция инициализации элементов массива постоянным значением.

Функции

- void nmppsSet_8s (**int8b** val, **nm8s** *pDst, int len)
- void nmppsSet_16s (**int16b** val, **nm16s** *pDst, int len)
- void nmppsSet_32s (**int32b** val, **nm32s** *pDst, int len)
- void nmppsSet_64s (**int64b** val, **nm64s** *pDst, int len)
- void nmppsSet_64sc (**int64sc** val, **nm64sc** *pDst, int len)
- **_ _ INLINE_ _** void nmppsSet_8u (**uint8b** val, **nm8u** *pDst, int len)
- **_ _ INLINE_ _** void nmppsSet_16u (**uint16b** val, **nm16u** *pDst, int len)
- **_ _ INLINE_ _** void nmppsSet_32u (**uint32b** val, **nm32u** *pDst, int len)
- **_ _ INLINE_ _** void nmppsSet_64u (**uint64b** val, **nm64u** *pDst, int len)

5.200.1 Подробное описание

Функция инициализации элементов массива постоянным значением.

$$pDst[i] = val,$$

$$i = \overline{0 \dots len - 1}$$

Аргументы

len	Размер вектора в элементах.
val	Константа. Диапазон значений val должен соответствовать типу результирующего вектора.

Возвращаемые значения

pDst	Результирующий вектор.
------	------------------------

Возвращает

void

5.201 nmppsRandUniform

Инициализация массива случайными числами.

Функции

- void [nmppsRandUniform_64s](#) ([nm64s](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_8s ([nm8s](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_16s ([nm16s](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_32s ([nm32s](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_8u ([nm8u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_16u ([nm16u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_32u ([nm32u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsRandUniform_64u ([nm64u](#) *pDstVec, int nSize)

5.201.1 Подробное описание

Инициализация массива случайными числами.

Аргументы

nSize	Размер вектора.
-------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.201.2 Функции

5.201.2.1 nmppsRandUniform_64s()

```
void nmppsRandUniform_64s (
    nm64s * pDstVec,
    int nSize )
```

/ru Инициализация массива 32-разрядными случайными числами. /en Random initialization of 32-bit buffer /~

5.202 nmppsRandUniform

Инициализация массива случайными числами.

Функции

- void nmppsRandUniform_64f (nm64f *pDstVec, int nSize, double low, double hi)
- void nmppsRand_32f (nm32f *pDstVec, int nSize, float low, float hi)
- void nmppsRandUniform_32f_integer (nm32f *pDstVec, int nSize, int hi, int low)

5.202.1 Подробное описание

Инициализация массива случайными числами.

/ru Инициализация массива 64-разрядными случайными числами. /en Random initialization of 64-bit buffer /~

Аргументы

nSize	Размер вектора.
-------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.203 nmppsRandUniform_

Генерация случайного числа с равномерным распределением.

Функции

- int nmppsRandUniform2_32s (int nMin, int nMax, int nDivisible)
- int nmppsRandUniform3_32s (int nMin, int nMax)
- int nmppsRandUniform ()

5.203.1 Подробное описание

Генерация случайного числа с равномерным распределением.

Аргументы

nMin	Минимальное возможное значение случайного числа.
nMax	Максимальное возможное значение случайного числа.
nDivisible	Значение, которому будет кратно случайное число.

Возвращает

int Случайное число в диапазоне либо [nMin, nMax]. Для функции без параметров данный диапазон [-2^31; 2^31-1].

5.204 nmppsRamp_

Инициализация массива элементами арифметической прогрессии.

Функции

- void nmppsRamp_8s (**nm8s** *pVec, **int8b** nOffset, **int8b** nSlope, int nSize)
- void nmppsRamp_16s (**nm16s** *pVec, **int16b** nOffset, **int16b** nSlope, int nSize)
- void nmppsRamp_32s (**nm32s** *pVec, **int32b** nOffset, **int32b** nSlope, int nSize)
- void nmppsRamp_64s (**nm64s** *pVec, **int64b** nOffset, **int64b** nSlope, int nSize)

5.204.1 Подробное описание

Инициализация массива элементами арифметической прогрессии.

$$pVec[i] = nOffset + nSlope \cdot i$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

nOffset	Первый член арифметической прогрессии.
nSlope	Разность арифметической прогрессии.
nSize	Размер вектора.

Возвращаемые значения

pVec	Результирующий массив.
------	------------------------

Возвращает

void

5.205 nmppsConvert

Изменение разрядности элементов вектора.

Преобразование знаковых данных к меньшей разрядности осуществляется отбрасыванием старших битов. Преобразование знаковых данных к большей разрядности осуществляется с распространением влево старшего (знакового) бита. Преобразование беззнаковых данных к большей разрядности осуществляется добавлением слева старших нулевых битов.

Функции

- void [nmppsConvert_1s2s](#) (const **nm1** *pSrcVec, **nm2s** *pDstVec, int nSize)
- void [nmppsConvert_1u2u](#) (const **nm1** *pSrcVec, **nm2u** *pDstVec, int nSize)
- void [nmppsConvert_1u4u](#) (const **nm1** *pSrcVec, **nm4u** *pDstVec, int nSize)
- void [nmppsConvert_2s1s](#) (const **nm2s** *pSrcVec, **nm1** *pDstVec, int nSize)
- void [nmppsConvert_2s4s](#) (const **nm2s** *pSrcVec, **nm4s** *pDstVec, int nSize)
- void [nmppsConvert_2u4u](#) (const **nm2u** *pSrcVec, **nm4u** *pDstVec, int nSize)
- void [nmppsConvert_4s1s](#) (const **nm4s** *pSrcVec, **nm1** *pDstVec, int nSize)
- void [nmppsConvert_4s2s](#) (const **nm4s** *pSrcVec, **nm2s** *pDstVec, int nSize)
- void [nmppsConvert_4s8s](#) (const **nm4s** *pSrcVec, **nm8s** *pDstVec, int nSize)

- void nmppsConvert_4u8u (const **nm4u** *pSrcVec, **nm8u** *pDstVec, int nSize)
- void nmppsConvert_8s4s (const **nm8s** *pSrcVec, **nm4s** *pDstVec, int nSize)
- void nmppsConvert_8s16s (const **nm8s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsConvert_8s32s (const **nm8s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsConvert_8s64s (const **nm8s** *pSrcVec, **nm64s** *pDstVec, int nSize)
- void nmppsConvert_8u16u (const **nm8u** *pSrcVec, **nm16u** *pDstVec, int nSize)
- void nmppsConvert_8u32u (const **nm8u** *pSrcVec, **nm32u** *pDstVec, int nSize)
- void nmppsConvert_8u64u (const **nm8u** *pSrcVec, **nm64u** *pDstVec, int nSize)
- void nmppsConvert_16s4s (const **nm16s** *pSrcVec, **nm4s** *pDstVec, int nSize)
- void nmppsConvert_16s8s (const **nm16s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsConvert_16s32s (const **nm16s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsConvert_16s64s (const **nm16s** *pSrcVec, **nm64s** *pDstVec, int nSize)
- void nmppsConvert_16u32u (const **nm16u** *pSrcVec, **nm32u** *pDstVec, int nSize)
- void nmppsConvert_16u64u (const **nm16u** *pSrcVec, **nm64u** *pDstVec, int nSize)
- void nmppsConvert_32s8s (const **nm32s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsConvert_32s16s (const **nm32s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsConvert_32s64s (const **nm32s** *pSrcVec, **nm64s** *pDstVec, int nSize)
- void nmppsConvert_32u64u (const **nm32u** *pSrcVec, **nm64u** *pDstVec, int nSize)
- void nmppsConvert_64s32s (const **nm64s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsConvert_64s16s (const **nm64s** *pSrcVec, **nm16s** *pDstVec, int nSize)

5.205.1 Подробное описание

Изменение разрядности элементов вектора.

Преобразование знаковых данных к меньшей разрядности осуществляется отбрасыванием старших битов. Преобразование знаковых данных к большей разрядности осуществляется с распространением влево старшего (знакового) бита. Преобразование беззнаковых данных к большей разрядности осуществляется добавлением слева старших нулевых битов.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
----------------	------------------------

Возвращает

void

5.205.2 Функции

5.205.2.1 nmppsConvert_1s2s()

```
void nmppsConvert_1s2s (
    const nm1 * pSrcVec,
    nm2s * pDstVec,
    int nSize )
```

Restrictions: nSize =[32*64,32*64*2,32*64*3,...]

5.205.2.2 nmppsConvert_1u2u()

```
void nmppsConvert_1u2u (
    const nm1 * pSrcVec,
    nm2u * pDstVec,
    int nSize )
```

Restrictions: nSize =[32*64,32*64*2,32*64*3,...]

5.206 nmppsConvert

Изменение разрядности элементов вектора. Входной вектор.

Функции

- void nmppsConvert_32u32fcr (const **nm32u** *pSrcVec, **nm32fcr** *pDstVec, int nSize)
- void nmppsConvert_32s32fcr (const **nm32s** *pSrcVec, **nm32fcr** *pDstVec, int nSize)
- void nmppsConvert_32f32fcr (const **nm32f** *pSrcVec, **nm32fcr** *pDstVec, int nSize)
- void nmppsConvert_32sc32fcr (const **nm32sc** *pSrcVec, **nm32fcr** *pDstVec, int nSize)
- void nmppsConvert_32s32f (const **nm32s** *pSrcVec, **nm32f** *pDstVec, int nSize)
- void nmppsConvert_32f64f (const **nm32f** *pSrcVec, **nm64f** *pDstVec, int nSize)
- void nmppsConvert_64f32f (const **nm64f** *pSrcVec, **nm32f** *pDstVec, int nSize)

5.206.1 Подробное описание

Изменение разрядности элементов вектора. Входной вектор.

Аргументы

nSize	Размер векторов в элементах.
-------	------------------------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Функции выполняются на сопроцессоре (NM6407) с плавающей точкой с использованием преупаковщика данных

5.207 nmppsConvert_rounding

Функции конвертации вектора чисел с плавающей точкой одинарной точности в вектор целых 32-битных чисел с разной степенью округления

Функции

- void nmppsConvert_32f32s_rounding (const nm32f *pSrcVec, **nm32s** *pDstVec, int scale, int n←Size)
- void nmppsConvert_32f32f_ceiling (const nm32f *pSrcVec, nm32f *pDstVec, int scale, int nSize)
- void nmppsConvert_32f32s_ceiling (const nm32f *pSrcVec, **nm32s** *pDstVec, int scale, int nSize)
- void nmppsConvert_32f32s_floor (const nm32f *pSrcVec, **nm32s** *pDstVec, int scale, int nSize)
- void nmppsConvert_32f32s_truncate (const nm32f *pSrcVec, **nm32s** *pDstVec, int scale, int nSize)

5.207.1 Подробное описание

Функции конвертации вектора чисел с плавающей точкой одинарной точности в вектор целых 32-битных чисел с разной степенью округления

Аргументы

pSrcVec	указатель на входной вектор
pDstVec	указатель на выходной вектор
scale	степень двойки (умножение каждого элемента выходного вектора на 2^{scale}), может быть отрицательной
nSize	число элементов во входном векторе (может быть только четным)

Функция выполняется на сопроцессоре (процессор 1879ВМ6Я) с плавающей точкой с использова-

нием переупаковщика данных

`nmppsConvert_32f32s_rounding` округляет все дробные числа из `pSrcVec` до ближайших целых (например, 1.5 будет округлено до 2, 1.7 до 2, а 1.4 до 1)

`nmppsConvert_32f32s_ceiling` округляет все дробные числа из `pSrcVec` к большему целому (например, 0=>0 , 0.1=>1 , 0.9=>1 , 1=>1 , 1.1=>2 1.5=>2, 1.7=> 2, а -1.1 => -1)

5.208 nmppsMerge

Функция соединяет 2 массива 32-х чисел с плавающей точкой (float) в один. Результирующий массив { `pSrcVec1[0]`, `pSrcVec2[0]`, `pSrcVec1[1]`, `pSrcVec2[1]`, `pSrcVec1[2]`, `pSrcVec2[2]` и т.д. }.

Функции

- `void nmppsMerge_32f (const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize)`

5.208.1 Подробное описание

Функция соединяет 2 массива 32-х чисел с плавающей точкой (float) в один. Результирующий массив { `pSrcVec1[0]`, `pSrcVec2[0]`, `pSrcVec1[1]`, `pSrcVec2[1]`, `pSrcVec1[2]`, `pSrcVec2[2]` и т.д. }.

Аргументы

<code>pSrcVec1</code>	указатель на входной массив чисел float
<code>pSrcVec2</code>	указатель на входной массив чисел float
<code>pDstVec</code>	указатель на выходной массив чисел float
<code>nSize</code>	количество элементов в массиве <code>pSrcVec</code> (в массиве <code>pDstVec</code> элементов будет в 2 раза больше)

5.209 nmppsConvertRisc

Функции изменения разрядности чисел входного вектора

Функции

- `void nmppsConvertRisc_8u32u (const nm8u *pSrcVec, nm32u *pDstVec, int nSize)`
- `void nmppsConvertRisc_32u8u (const nm32u *pSrcVec, nm8u *pDstVec, int nSize)`

5.209.1 Подробное описание

Функции изменения разрядности чисел входного вектора

Аргументы

pSrcVec	указатель на входной вектор
pDstVec	указатель на выходной вектор
nSize	число элементов во входном векторе (должно быть кратно 8 и не может быть меньше 8)

Функция выполняется на RISC-процессоре

5.210 nmppsCopy_

Копирование вектора.

Функции

- void nmppsCopy_2s (const [nm2s](#) *pSrcVec, [nm2s](#) *pDstVec, int nSize)
- void nmppsCopy_4s (const [nm4s](#) *pSrcVec, [nm4s](#) *pDstVec, int nSize)
- void nmppsCopy_8s (const [nm8s](#) *pSrcVec, [nm8s](#) *pDstVec, int nSize)
- void nmppsCopy_16s (const [nm16s](#) *pSrcVec, [nm16s](#) *pDstVec, int nSize)
- void nmppsCopy_32s (const [nm32s](#) *pSrcVec, [nm32s](#) *pDstVec, int nSize)
- void nmppsCopy_64s (const [nm64s](#) *pSrcVec, [nm64s](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_4u (const [nm4u](#) *pSrcVec, [nm4u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_8u (const [nm8u](#) *pSrcVec, [nm8u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_16u (const [nm16u](#) *pSrcVec, [nm16u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_32u (const [nm32u](#) *pSrcVec, [nm32u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_64u (const [nm64u](#) *pSrcVec, [nm64u](#) *pDstVec, int nSize)
- [__INLINE__](#) void nmppsCopy_64sc (const [nm64sc](#) *pSrcVec, [nm64sc](#) *pDstVec, int nSize)

5.210.1 Подробное описание

Копирование вектора.

$$pDstVec[i] = pSrcVec[i],$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.211 nmppsCopy

Копирование массива чисел с плавающей точкой одинарной точности (с любого на любой адрес)

Функции

- void nmppsCopy_32f (const float *pSrcVec, float *pDstVec, int size)

5.211.1 Подробное описание

Копирование массива чисел с плавающей точкой одинарной точности (с любого на любой адрес)

Аргументы

pSrcVec	указатель на входной массив чисел (адрес может быть как четным, так и нечетным)
pDstVec	указатель на выходной массив чисел (адрес может быть как четным, так и нечетным)
size	количество элементов в массиве pSrcVec (может быть любым неотрицательным, кроме 0)

5.212 nmppsCopyOddToOdd_32f

Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на нечетный адрес)

Функции

- void nmppsCopyOddToOdd_32f (const float *pSrcVec, float *pDstVec, int size)

5.212.1 Подробное описание

Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на нечетный адрес)

Аргументы

pSrcVec	указатель на входной массив чисел (адрес может быть только нечетным)
pDstVec	указатель на выходной массив чисел (адрес может быть только нечетным)
size	количество элементов в массиве pSrcVec (может быть любым неотрицательным, кроме 0)

5.213 nmppsCopyEvenToOdd_32f

Копирование массива чисел с плавающей точкой одинарной точности (с четного на нечетный адрес)

Функции

- void nmppsCopyEvenToOdd_32f (const float *pSrcVec, float *pDstVec, int size)

5.213.1 Подробное описание

Копирование массива чисел с плавающей точкой одинарной точности (с четного на нечетный адрес)

Аргументы

pSrcVec	указатель на входной массив чисел (адрес может быть только четным)
pDstVec	указатель на выходной массив чисел (адрес может быть только нечетным)
size	количество элементов в массиве pSrcVec (может быть любым неотрицательным, кроме 0)

5.214 nmppsCopyOddToEven_32f

Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на четный адрес)

Функции

- void nmppsCopyOddToEven_32f (const float *pSrcVec, float *pDstVec, int size)

5.214.1 Подробное описание

Копирование массива чисел с плавающей точкой одинарной точности (с нечетного на четный адрес)

Аргументы

pSrcVec	указатель на входной массив чисел (адрес может быть только нечетным)
pDstVec	указатель на выходной массив чисел (адрес может быть только четным)
size	количество элементов в массиве pSrcVec (может быть любым неотрицательным, кроме 0)

5.215 nmppsCopyEvenToEven_32f

Копирование массива чисел с плавающей точкой одинарной точности (с четного на четный адрес)

Функции

- void nmppsCopyEvenToEven_32f (const float *pSrcVec, float *pDstVec, int size)

5.215.1 Подробное описание

Копирование массива чисел с плавающей точкой одинарной точности (с четного на четный адрес)

Аргументы

pSrcVec	указатель на входной массив чисел (адрес может быть только четным)
pDstVec	указатель на выходной массив чисел (адрес может быть только четным)
size	количество элементов в массиве pSrcVec (может быть любым неотрицательным)

5.216 nmppsCopyRisc

Копирование массива

Функции

- void nmppsCopyRisc_32f (const float *pSrcVec, float *pDstVec, int nSize)

5.216.1 Подробное описание

Копирование массива

Аргументы

pSrcVec	указатель на входной массив чисел
pDstVec	указатель на выходной массив чисел
size	количество элементов в массиве pSrcVec

5.217 nmppsCopyua_

Копирование вектора с невыровненной байтовой позиции в выровненную.

Функции

- void nmppsCopyua_8s (const **nm8s** *pSrcVec, int nSrcOffset, **nm8s** *pDstVec, int nSize)

5.217.1 Подробное описание

Копирование вектора с невыровненной байтовой позиции в выровненную.

$$pDstVec[i] = pSrcVec[nSrcOffset + i]$$

$$i = \overline{0 \dots nSize - 1}$$

Позиция байта считается выровненной если она совпадает с границей 64р. слов в памяти.

Аргументы

pSrcVec	Входной вектор.
pDstVec	Результирующий вектор.
nSrcOffset	Смещение в элементах относительно начала вектора. nSrcOffset Может принимать любое значение.
nSize	Кол-во копируемых элементов.

Возвращает

void

5.218 nmppsSwap_

Перестановка двух векторов.

Функции

- void nmppsSwap_64s ([nm64s](#) *pSrcVec1, [nm64s](#) *pSrcVec2, int nSize)

5.218.1 Подробное описание

Перестановка двух векторов.

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер векторов в элементах.

Возвращает

void

5.219 nmppsMax_

Поиск значения максимального элемента вектора.

Функции

- void `nmppsMax_8s7b` (const `nm8s7b` *pSrcVec, int nSize, `int8b` *n.MaxValue)
- void `nmppsMax_16s15b` (const `nm16s15b` *pSrcVec, int nSize, `int16b` *n.MaxValue)
- void `nmppsMax_32s31b` (const `nm32s31b` *pSrcVec, int nSize, int *n.MaxValue)
- void `nmppsMax_64s63b` (const `nm64s63b` *pSrcVec, int nSize, `int64b` *n.MaxValue)
- int `nmppsMax_8sm` (const `nm8s` *srcVec, int size, `int8b` *maxValue, `nm16s` *tmp)
- int `nmppsMax_16sm` (const `nm16s` *srcVec, int size, `int16b` *maxValue, `nm32s` *tmp)
- int `nmppsMax_32sm` (const `nm32s` *srcVec, int size, `int32b` *maxValue, `nm64s` *tmp)

5.219.1 Подробное описание

Поиск значения максимального элемента вектора.

$$n.MaxValue = \max_i (pSrcVec[i])$$

Аргументы

<code>pSrcVec</code>	Входной вектор.
<code>nSize</code>	Размер вектора в элементах.

Возвращаемые значения

<code>n.MaxValue</code>	значение максимального элемент вектора.
-------------------------	---

Возвращает

void

Restrictions:

Ограничения на параметры приводятся в описании каждой из функций.

5.219.2 Функции

5.219.2.1 nmppsMax_16s15b()

```
void nmppsMax_16s15b (
    const nm16s15b * pSrcVec,
    int nSize,
    int16b * n.MaxValue )
```

Restrictions:

Максимальный и минимальный элементы массива должны отличаться не более чем на $2^{15}-1$.
Примеры допустимых диапазонов входных чисел:

5.219.2.2 nmppsMax_32s31b()

```
void nmppsMax_32s31b (
    const nm32s31b * pSrcVec,
    int nSize,
    int * n.MaxValue )
```

Restrictions:

Максимальный и минимальный элементы массива должны отличаться не более чем на $2^{31}-1$.
Примеры допустимых диапазонов входных чисел:
[00000000h..7FFFFFFFh]=[0..+2^31-1] [FFFFFFFFFFh..7FFFFFFEh]=[-1..+2^31-2] [C0000000h..3FFFFFFFh]=[-2^30..+2^30-1] [80000000h..00000000h]=[-2^31..0]

5.219.2.3 nmppsMax_8s7b()

```
void nmppsMax_8s7b (
    const nm8s7b * pSrcVec,
    int nSize,
    int8b * n.MaxValue )
```

Restrictions:

Физический размер вектора должен быть кратен блоку из 32-х 64р. слов

Максимальный и минимальный элементы массива должны отличаться не более чем на 127.

Примеры допустимых диапазонов входных чисел:

[00h..7Fh]=[0..+127] [FFh..7Eh]=[-1..+126] [C0h..3Fh]=[-64..+63] [80h..00h]=[-128..0]

5.220 nmppsMin

Поиск значения минимального элемента вектора.

Функции

- void nmppsMin_8s7b (const nm8s7b *pSrcVec, int nSize, int8b *n.MinValue)
- void nmppsMin_16s15b (const nm16s15b *pSrcVec, int nSize, int16b *n.MinValue)
- void nmppsMin_32s31b (const nm32s31b *pSrcVec, int nSize, int *n.MinValue)
- void nmppsMin_64s63b (const nm64s63b *pSrcVec, int nSize, int64b *n.MinValue)
- int nmppsMin_8sm (const nm8s *srcVec, int size, int8b *minValue, nm16s *tmp)
- int nmppsMin_16sm (const nm16s *srcVec, int size, int16b *minValue, nm32s *tmp)
- int nmppsMin_32sm (const nm32s *srcVec, int size, int32b *minValue, nm64s *tmp)

5.220.1 Подробное описание

Поиск значения минимального элемента вектора.

$$n.MinValue = \min_i (pSrcVec[i])$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер вектора в элементах.

Возвращаемые значения

n.MinValue	значение минимального элемент.
------------	--------------------------------

Возвращает

void

Restrictions:

Ограничения на параметры приводятся в описании каждой из функций.

5.220.2 Функции

5.220.2.1 nmppsMin_16s15b()

```
void nmppsMin_16s15b (
    const nm16s15b * pSrcVec,
    int nSize,
    int16b * nMinValue )
```

Restrictions:

Максимальный и минимальный элементы массива должны отличаться не более чем на 2^{15-1} .

Примеры допустимых диапазонов входных чисел:

[0000h..7FFFh]=[0..+32767] [FFFFh..7FFEh]=[-1..+32766] [C000h..3FFFh]=[-16384..+16383]
[8000h..0000h]=-32768..0]

5.220.2.2 nmppsMin_32s31b()

```
void nmppsMin_32s31b (
    const nm32s31b * pSrcVec,
    int nSize,
    int * nMinValue )
```

Restrictions:

Максимальный и минимальный элементы массива должны отличаться не более чем на 2^{31-1} .

Примеры допустимых диапазонов входных чисел:

[00000000h..7FFFFFFFh]=[0..+2^31-1] [FFFFFFFFh..7FFFFFFEh]=[-1..+2^31-2] [C0000000h..3FFFFFFFh]=[-2^30..+2^30-1] [80000000h..00000000h]=-2^31..0]

5.220.2.3 nmppsMin_8s7b()

```
void nmppsMin_8s7b (
    const nm8s7b * pSrcVec,
    int nSize,
    int8b * nMinValue )
```

Restrictions:

```
\~\~russian Максимальный и минимальный элементы массива должны отличаться не более чем на 127. \n
Примеры допустимых диапазонов входных чисел:\n
\~\~english The difference between the maximum and minimum elements of the array should not be more than 127. \n
Here are some examples of admissible ranges for input numbers: \n
\~\~[00h..7Fh]=[ 0..+127]
[FFh..7Eh]=[ -1..+126]
[C0h..3Fh]=[-64..+63]
[80h..00h]=[-128..0 ]
```

5.221 nmppsMaxIdx_

Поиск значения максимального элемента вектора и его положения (положений) в векторе.

Функции

- void nmppsMaxIdx_8s (nm8s7b *pSrcVec, int nSize, int *nIndex, int8b *n.MaxValue, void *pLtmpBuf, void *pGtmpBuf, int nSearchDir)
- void nmppsMaxIdx_16s (nm16s15b *pSrcVec, int nSize, int *nIndex, int16b *n.MaxValue, void *pLtmpBuf, void *pGtmpBuf, int nSearchDir)
- void nmppsMaxIdx_32s (nm32s31b *pSrcVec, int nSize, int *nIndex, int32b *n.MaxValue, void *pLtmpBuf, void *pGtmpBuf, int nSearchDir)

5.221.1 Подробное описание

Поиск значения максимального элемента вектора и его положения (положений) в векторе.

$$n.MaxValue = \max_i (pSrcVec[i])$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер вектора в элементах. Занимаемый этим вектором объем памяти должен быть кратен 64 длинным словам (nm64s[64,128,...]).
pLtmpBuf	Временный массив на локальной шине из nSize элементов.
pGtmpBuf	Временный массив на глобальной шине .
nSearchDir	Направление поиска максимума. При nSearchDir = 1, поиск ведется от начала массива. При nSearchDir = -1, поиск ведется от конца массива.
	Значение максимального элемента.

Возвращаемые значения

nIndex	Индекс первого найденного максимума среди равных.
--------	---

Возвращает

void

Restrictions:

Ограничения на параметры приводятся в описании каждой из функций.

Restrictions:

Диапазоны входных элементов могут быть "плавающими"

Например для данных nm16s15b максимальный и минимальный элементы массива должны отличаться не более чем на $2^{15}-1$. Примеры допустимых диапазонов входных чисел для типа nm16s15b :

Here are some examples of admissible ranges for input numbers:

[0000h..7FFFh]=[0..+32767] [FFFFh..7FFEh]=[-1..+32766] [C000h..3FFFh]=[-16384..+16383]
[8000h..0000h]=[-32768..0]

5.222 nmppsMinIndx_

Поиск значения минимального элемента вектора и его положения (положений) в векторе.

Функции

- void nmppsMinIndx_8s ([nm8s7b](#) *pSrcVec, int nSize, int *nIndex, [int8b](#) *nMinValue, void *p← LTmpBuf, void *pGTmpBuf, int nSearchDir)
- void nmppsMinIndx_16s ([nm16s15b](#) *pSrcVec, int nSize, int *nIndex, [int16b](#) *nMinValue, void *pLTmpBuf, void *pGTmpBuf, int nSearchDir)
- void nmppsMinIndx_32s ([nm32s31b](#) *pSrcVec, int nSize, int *nIndex, [int32b](#) *nMinValue, void *pLTmpBuf, void *pGTmpBuf, int nSearchDir)

5.222.1 Подробное описание

Поиск значения минимального элемента вектора и его положения (положений) в векторе.

$$nMinValue = \min_i (pSrcVec[i])$$

Аргументы

pSrcVec	Входной вектор.
nSize	Размер вектора в элементах. Занимаемый этим вектором объем памяти должен быть кратен 64 для длинных словам (nm64s[64,128,...]).
pLTmpBuf	Временный массив на локальнойшине .
pGTmpBuf	Временный массив на глобальнойшине из nSize элементов.
nSearchDir	Направление поиска минимума. При nSearchDir = 1, поиск ведется от начала массива. При nSearchDir = -1, поиск ведется от конца массива. Значение минимального элемента.

Возвращаемые значения

nIndex	Индекс первого найденного минимума среди равных.
--------	--

Возвращает

void

Restrictions:

Ограничения на параметры приводятся в описании каждой из функций.

Restrictions:

Диапазоны входных элементов могут быть "плавающими"

Например для данных nm16s15b максимальный и минимальный элементы массива должны отличаться не более чем на $2^{15}-1$. Примеры допустимых диапазонов входных чисел для типа nm16s15b :

Here are some examples of admissible ranges for input numbers:

[0000h..7FFFh]=[0..+32767] [FFFFh..7FFEh]=[-1..+32766] [C000h..3FFFh]=[-16384..+16383]
[8000h..0000h]=[-32768..0]

5.223 nmppsMinIndxVN

Поиск значения минимального элемента вектора длины N и его положения в векторе.

Функции

- int nmppsMinIdxV9_32s (int *pSrcVec, int nStride, int *nPos)
- int nmppsMinIdxV16_32s (int *pSrcVec, int nStride, int *nPos)
- int nmppsMinIdxV25_32s (int *pSrcVec, int nStride, int *nPos)
- int nmppsMinIdxV256_32s (int *pSrcVec, int nStride, int *nPos)
- int nmppsMinIdxV1024_32s (int *pSrcVec, int nStride, int *nPos)

5.223.1 Подробное описание

Поиск значения минимального элемента вектора длины N и его положения в векторе.

Аргументы

pSrcVec	Массив из N элементов, где N определяется числом в имени функции.
nStride	шаг между элементами в 32р. словах

Возвращаемые значения

Индекс	первого найденного минимума.
--------	------------------------------

Возвращает

Значение минимального элемента.

Restrictions:

Вычисления производятся на скалярном ядре, поэтому указатель на данные pSrcVec может ссылаться на нечетный адрес.

5.224 nmppsFirstZeroIdx

Поиск позиции первого нулевого элемента в векторе .

Функции

- int nmppsFirstZeroIdx_32s (int *pSrcVec, int nSize)

5.224.1 Подробное описание

Поиск позиции первого нулевого элемента в векторе .

Аргументы

pSrcVec	Входной массив.
nIndex	Размер массива.

Возвращает

Позиция нулевого элемента или -1 в случае если нулевой элемент не найден

Restrictions:

Вычисления производятся на скалярном ядре, поэтому указатель на данные pSrcVec может ссылаться на нечетный адрес. Поиск производится до первого нулевого элемента.

5.225 nmppsFirstNonZeroIndx

Поиск позиции первого ненулевого элемента в векторе .

Функции

- int nmppsFirstNonZeroIndx_32s (int *pSrcVec, int nSize)

5.225.1 Подробное описание

Поиск позиции первого ненулевого элемента в векторе .

Аргументы

pSrcVec	Входной массив.
nIndex	Размер массива.

Возвращает

Позиция ненулевого элемента или -1 в случае если ненулевой элемент не найден

Restrictions:

Вычисления производятся на скалярном ядре, поэтому указатель на данные pSrcVec может ссылаться на нечетный адрес. Поиск производится до первого ненулевого элемента.

5.226 nmppsLastZeroIndx

Поиск позиции последнего нулевого элемента в векторе .

Функции

- int nmppsLastZeroIdx_32s (int *pSrcVec, int nSize)

5.226.1 Подробное описание

Поиск позиции последнего нулевого элемента в векторе .

Аргументы

pSrcVec	Входной массив.
nIndex	Размер массива.

Возвращает

Позиция нулевого элемента или -1 в случае если нулевой элемент не найден

Restrictions:

Вычисления производятся на скалярном ядре, поэтому указатель на данные pSrcVec может ссылаться на нечетный адрес. Поиск производится с конца до первого нулевого элемента.

5.227 nmppsLastNonZeroIdx

Поиск позиции последнего ненулевого элемента в векторе .

Функции

- int nmppsLastNonZeroIdx_32s (int *pSrcVec, int nSize)

5.227.1 Подробное описание

Поиск позиции последнего ненулевого элемента в векторе .

Аргументы

pSrcVec	Входной массив.
nIndex	Размер массива.

Возвращает

Позиция нулевого элемента или -1 в случае если нулевой элемент не найден

Restrictions:

Вычисления производятся на скалярном ядре, поэтому указатель на данные pSrcVec может ссылаться на нечетный адрес. Поиск производится с конца до первого ненулевого элемента.

5.228 nmppsMinEvery_

Поэлементный минимум из двух векторов.

Функции

- void nmppsMinEvery_8s ([nm8s7b](#) *pSrcVec1, [nm8s7b](#) *pSrcVec2, [nm8s7b](#) *pDstMinVec, int nSize)
- void nmppsMinEvery_16s ([nm16s15b](#) *pSrcVec1, [nm16s15b](#) *pSrcVec2, [nm16s15b](#) *pDstMinVec, int nSize)
- void nmppsMinEvery_32s ([nm32s31b](#) *pSrcVec1, [nm32s31b](#) *pSrcVec2, [nm32s31b](#) *pDstMinVec, int nSize)
- void nmppsMinEvery_64s ([nm64s63b](#) *pSrcVec1, [nm64s63b](#) *pSrcVec2, [nm64s63b](#) *pDstMinVec, int nSize)

5.228.1 Подробное описание

Поэлементный минимум из двух векторов.

$$pDstMinVec[i] = \min(pSrcVec1[i], pSrcVec2[i])$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

<code>pDstMinVec</code>	Массив поэлементных минимумов для входных массивов.
-------------------------	---

Возвращает

`void`

Restrictions:

Сравниваемые пары чисел двух массивов должны отличаться не более чем на $2^{15} - 1$.

Примеры допустимых диапазонов сравниваемых пар чисел:

should not be more than $2^{15} - 1$.

Here are some examples of admissible ranges for comparable pairs of numbers:

[0000h..7FFFh]=[0..+32767]

[FFFFh..7FFEh]=[-1..+32766]

[C000h..3FFFh]=[-16384..+16383]

[8000h..0000h]=[-32768..0]

5.229 nmppsMaxEvery_

Поэлементный максимум из двух векторов.

Функции

- `void nmppsMaxEvery_8s (nm8s7b *pSrcVec1, nm8s7b *pSrcVec2, nm8s7b *pDstMaxVec, int n←Size)`
- `void nmppsMaxEvery_16s (nm16s15b *pSrcVec1, nm16s15b *pSrcVec2, nm16s15b *pDstMaxVec, int nSize)`
- `void nmppsMaxEvery_32s (nm32s31b *pSrcVec1, nm32s31b *pSrcVec2, nm32s31b *pDstMaxVec, int nSize)`
- `void nmppsMaxEvery_64s (nm64s63b *pSrcVec1, nm64s63b *pSrcVec2, nm64s63b *pDstMaxVec, int nSize)`

5.229.1 Подробное описание

Поэлементный максимум из двух векторов.

$$pDstMaxVec[i] = \max(pSrcVec1[i], pSrcVec2[i])$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstMaxVec	Массив поэлементных минимумов для входных массивов.
------------	---

Возвращает

void

Restrictions:

Сравниваемые пары чисел двух массивов должны отличаться не более чем на $2^{15} - 1$.

Примеры допустимых диапазонов сравниваемых пар чисел:

should not be more than $2^{15} - 1$.

Here are some examples of admissible ranges for comparable pairs of numbers:

[0000h..7FFFh]=[0..+32767]

[FFFFh..7FFEh]=[-1..+32766]

[C000h..3FFFh]=[-16384..+16383]

[8000h..0000h]=[-32768..0]

5.230 nmppsMinCmpLtV_

Поэлементный минимум из двух векторов.

Функции

- void nmppsMinCmpLtV_16s ([nm16s15b](#) *pSrcVec1, [nm16s15b](#) *pSrcVec2, [nm16s15b](#) *pDstMin, [nm16s15b](#) *pDstSignMask, int nSize)

5.230.1 Подробное описание

Поэлементный минимум из двух векторов.

$$pDstMin[i] = \min(pSrcVec1[i], pSrcVec2[i])$$

$$pDstSignMask[i] = \begin{cases} 11\dots1b, & pSrcVec1[i] < pSrcVec2[i] \\ 00\dots0b, & pSrcVec1[i] \leq pSrcVec2[i] \end{cases}$$

$$i = \overline{0\dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstMin	Массив поэлементных минимумов для входных массивов.
pDstSignMask	Массив знаков поэлементных разностей первого и второго векторов.

Возвращает

void

Restrictions:

Сравниваемые пары чисел двух массивов должны отличаться не более чем на $2^{15} - 1$.

Примеры допустимых диапазонов сравниваемых пар чисел:

should not be more than $2^{15} - 1$.

Here are some examples of admissible ranges for comparable pairs of numbers:

[0000h..7FFFh]=[0..+32767]
 [FFFFh..7FFEh]=[-1..+32766]
 [C000h..3FFFh]=[-16384..+16383]
 [8000h..0000h]=[-32768..0]

5.231 nmppsCmpLt0

Сравнивает элементы массива на меньше нуля.

Функции

- void nmppsCmpLt0_8s (const **nm8s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsCmpLt0_16s (const **nm16s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsCmpLt0_32s (const **nm32s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsCmpLt0_64s (const **nm64s** *pSrcVec, **nm64s** *pDstVec, int nSize)

5.231.1 Подробное описание

Сравнивает элементы массива на меньше нуля.

$$pDstVec(i) = \begin{cases} 11\dots1b, & \text{if } pSrcVec(i) < 0 \\ 00\dots0b, & \text{if } pSrcVec(i) \geq 0 \end{cases}$$

$$i = \overline{0\dots nSize - 1}$$

Если элемент входного вектора меньше 0, во все биты соответствующего элемента выходного вектора записывается 1.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.232 nmppsCmpLteC

Функция сравнения элементов массива с константой (меньше или равно)

Функции

- void nmppsCmpLteC_v2nm32f (const v2nm32f *pSrcVec, const v2nm32f *C, nm1 *evenFlags, nm1 *oddFlags, int step, int nSize)

5.232.1 Подробное описание

Функция сравнения элементов массива с константой (меньше или равно)

Аргументы

in	pSrcVec	входной массив (элемент массива представляет собой структура v2nm32f, состоящая из двух чисел float)
in	C	константа (два числа float)

Возвращаемые значения

[out]	evenFlags маска полученная после сравнения первого поля из структуры v2nm32f
[out]	oddFlags маска полученная после сравнения второго поля из структуры v2nm32f

Аргументы

in	step	шаг, с которым будут браться элементы из pSrcVec
in	nSize	размер массива (в v2nm32f)

В структуре v2nm32f два поля. Первое - v0. Второе - v1.

5.233 nmppsCmpLtC

Функция сравнения элементов массива с константой (меньше или равно)

Функции

- void nmppsCmpLtC_v2nm32f (const v2nm32f *pSrcVec, const v2nm32f *C, nm1 *evenFlags, nm1 *oddFlags, int step, int nSize)

5.233.1 Подробное описание

Функция сравнения элементов массива с константой (меньше или равно)

Аргументы

in	pSrcVec	входной массив (элемент массива представляет собой структура v2nm32f, состоящая из двух чисел float)
in	C	константа (два числа float)

Возвращаемые значения

[out]	evenFlags	маска полученная после сравнения первого поля из структуры v2nm32f
[out]	oddFlags	маска полученная после сравнения второго поля из структуры v2nm32f

Аргументы

in	step	шаг, с которым будут браться элементы из pSrcVec
in	nSize	размер массива (в v2nm32f)

В структуре v2nm32f два поля. Первое - v0. Второе - v1.

5.234 nmppsCmpEq0(Reduced)

Сравнивает элементы массива неполной разрядности на признак равенства нулю.

Функции

- void nmppsCmpEq0_8u7b (nm8u7b *pSrcVec, nm1 *pDstVec, int nSize, int nTrueFlag)
- void nmppsCmpEq0_16u15b (nm16u15b *pSrcVec, nm1 *pDstVec, int nSize, int nTrueFlag)
- void nmppsCmpEq0_32u31b (nm32u31b *pSrcVec, nm1 *pDstVec, int nSize, int nTrueFlag)

5.234.1 Подробное описание

Сравнивает элементы массива неполной разрядности на признак равенства нулю.

$$pDstVec(i) = \begin{cases} 1, & \text{if } pSrcVec(i) = 0 \\ 0, & \text{if } pSrcVec(i) \neq 0 \end{cases}, \text{if } nTrueFlag = 1$$

$$pDstVec(i) = \begin{cases} 0, & \text{if } pSrcVec(i) = 0 \\ 1, & \text{if } pSrcVec(i) \neq 0 \end{cases}, \text{if } nTrueFlag = 0$$

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах. Vector size in elements.
nTrueFlag	Младший бит nTrueFlag определяет значение выходного бита при выполнении условия.

Возвращаемые значения

pDstVec	Результирующий бинарный вектор.
---------	---------------------------------

Возвращает

void

5.234.2 Функции

5.234.2.1 nmppsCmpEq0_16u15b()

```
void nmppsCmpEq0_16u15b (
    nm16u15b * pSrcVec,
    nm1 * pDstVec,
    int nSize,
    int nTrueFlag )
```

Заметки

nSize =[1,2,3,4...]

5.234.2.2 nmppsCmpEq0_32u31b()

```
void nmppsCmpEq0_32u31b (
    nm32u31b * pSrcVec,
    nm1 * pDstVec,
    int nSize,
    int nTrueFlag )
```

Заметки

nSize =[1,2,3,4...]

5.234.2.3 nmppsCmpEq0_8u7b()

```
void nmppsCmpEq0_8u7b (
    nm8u7b * pSrcVec,
    nm1 * pDstVec,
    int nSize,
    int nTrueFlag )
```

Заметки

nSize =[1,2,3,4...]

5.235 nmppsCmpGt0

Сравнивает элементы массива на больше нуля.

Функции

- void nmppsCmpGt0_8s (const nm8s *pSrcVec, nm8s *pDstVec, int nSize)
- void nmppsCmpGt0_16s (const nm16s *pSrcVec, nm16s *pDstVec, int nSize)
- void nmppsCmpGt0_32s (const nm32s *pSrcVec, nm32s *pDstVec, int nSize)
- void nmppsCmpGt0_64s (const nm64s *pSrcVec, nm64s *pDstVec, int nSize)

5.235.1 Подробное описание

Сравнивает элементы массива на больше нуля.

$$pDstVec(i) = \begin{cases} 11\dots1b, & \text{if } pSrcVec(i) > 0 \\ 00\dots0b, & \text{if } pSrcVec(i) \leq 0 \end{cases}$$

$$i = \overline{0\dots nSize - 1}$$

Если элемент входного вектора больше 0, во все биты соответствующего элемента выходного вектора записывается 1.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.236 nmppsMinMaxEvery_

Поэлементное сравнение двух векторов.

Функции

- void nmppsMinMaxEvery_8s (**nm8s** *pSrcVec1, **nm8s** *pSrcVec2, **nm8s** *pDstMin, **nm8s** *pDstMax, int nSize)
- void nmppsMinMaxEvery_16s (**nm16s** *pSrcVec1, **nm16s** *pSrcVec2, **nm16s** *pDstMin, **nm16s** *pDstMax, int nSize)
- void nmppsMinMaxEvery_32s (**nm32s** *pSrcVec1, **nm32s** *pSrcVec2, **nm32s** *pDstMin, **nm32s** *pDstMax, int nSize)

5.236.1 Подробное описание

Поэлементное сравнение двух векторов.

$$pDstMin[i] = \min(pSrcVec1[i], pSrcVec2[i])$$

$$pDstMax[i] = \max(pSrcVec1[i], pSrcVec2[i])$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstMin	Массив поэлементных минимумов для входных массивов.
pDstMax	Массив поэлементных максимумов для входных массивов.

Возвращает

void

Restrictions:

Сравниваемые пары чисел двух массивов должны отличаться не более чем на $2^{15} - 1$.

Примеры допустимых диапазонов сравниваемых пар чисел:
should not be more than $2^{15} - 1$.

Here are some examples of admissible ranges for comparable pairs of numbers:

[0000h..7FFFh]=[0..+32767]
[FFFFh..7FFEh]=[-1..+32766]
[C000h..3FFFh]=-[-16384..+16383]
[8000h..0000h]=[-32768..0]

5.237 nmppsClipPowC_

Функция насыщения.

Функции

- void nmppsClipPowC_8s ([nm8s](#) *pSrcVec, int nClipFactor, [nm8s](#) *pDstVec, int nSize)
- void nmppsClipPowC_16s ([nm16s](#) *pSrcVec, int nClipFactor, [nm16s](#) *pDstVec, int nSize)
- void nmppsClipPowC_32s ([nm32s](#) *pSrcVec, int nClipFactor, [nm32s](#) *pDstVec, int nSize)
- void nmppsClipPowC_64s ([nm64s](#) *pSrcVec, int nClipFactor, [nm64s](#) *pDstVec, int nSize)

5.237.1 Подробное описание

Функция насыщения.

$$pDstVec[i] = \begin{cases} -2^{nClipFactor}, & \text{if } pSrcVec[i] < -2^{nClipFactor} \\ pSrcVec[i], & \text{if } -2^{nClipFactor} \leq pSrcVec[i] \leq 2^{nClipFactor} - 1 \\ 2^{nClipFactor} - 1, & \text{if } pSrcVec[i] > 2^{nClipFactor} - 1 \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nClipFactor	Показатель степени, определяющий верхний и нижний пороги насыщения. nClipFactor>0
nSize	Размер вектора в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.238 nmppsClipCC_

Функция насыщения с произвольными порогами.

Функции

- void nmppsClipCC_32s (nm32s30b *pSrcVec, int30b nNegThresh, int30b nPosThresh, nm32s30b *pDstVec, int nSize)

5.238.1 Подробное описание

Функция насыщения с произвольными порогами.

$$pDstVec[i] = \begin{cases} nPosThresh, & \text{if } pSrcVec[i] > nPosThresh \\ pSrcVec[i], & \text{if } nNegThresh \leq pSrcVec[i] \leq nPosThresh \\ nNegThresh, & \text{if } pSrcVec[i] < nNegThresh \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
nNegThresh	Нижний порог насыщения.
nPosThresh	Верхний порог насыщения.
nSize	Размер вектора в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

Restrictions:

- Диапазон изменения для $nPostThresh \subset [0 \dots 2^{30} - 1]$.
- Диапазон изменения для $nNegThresh \subset [-2^{30} \dots 0]$.

5.239 nmppsThreshold_Lt_Gt_f

Пороговая функция с нижним и верхним порогами

Функции

- void nmppsThreshold_Lt_Gt_32f (nm32f *pSrcVec, nm32f *pDstVec, float min, float max, int nSize)

5.239.1 Подробное описание

Пороговая функция с нижним и верхним порогами

$$pDstVec[i] = \begin{cases} \max, & \text{if } pSrcVec[i] > \max \\ pSrcVec[i], & \text{if } \min \leq pSrcVec[i] \leq \max \\ \min, & \text{if } pSrcVec[i] < \min \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

Аргументы

pSrcVec	Входной вектор.
---------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

min	Нижний порог насыщения.
max	Верхний порог насыщения.
nSize	Размер вектора в элементах.

Возвращает

void

5.240 nmppsClipRShiftConvert_AddC_

Сокращение разрядности данных с предварительной их обработкой.

Функции

- void nmppsClipRShiftConvertAddC_16s8s (**nm16s** *pSrcVec, int nClipFactor, int nShift, **int8b** nAddValue, **nm8s** *pDst Vec, int nSize)
- void nmppsClipRShiftConvertAddC_32s8s (**nm32s** *pSrcVec, int nClipFactor, int nShift, **int8b** nAddValue, **nm8s** *pDst Vec, int nSize)

5.240.1 Подробное описание

Сокращение разрядности данных с предварительной их обработкой.

$$pDstVec[i] = Convert(Clip(pSrcVec[i], nClipFactor) >> nShift) + nAddValue$$

$$i = \overline{0 \dots nSize - 1}$$

Сокращение разрядности данных выполняется после предварительной обработки и осуществляется путем отбрасывания старших битов.

Аргументы

pSrcVec	Входной вектор.
nClipFactor	Показатель степени, определяющий верхний и нижний пороги насыщения. =[1,2,3..15]
nShift	Параметр определяет на сколько позиций =[2,4,6...14] нужно сдвинуть биты Создано системой Dohygen
nAddValue	Добавляемая константа.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.241 nmppsClipConvert_AddC_

Сокращение разрядности данных с предварительной их обработкой.

Определения типов

- typedef [nm64u](#) NmppsWeightState

Функции

- void nmppsClipConvertAddCInitAlloc_16s8s (NmppsWeightState **ppState)
- void nmppsClipConvertAddC_16s8s ([nm16s](#) *pSrcVec, int nClipFactor, [int8b](#) nAddValue, [nm8s](#) *pDstVec, int nSize, NmppsWeightState *pState)
- void nmppsClipConvertAddCFree (NmppsWeightState *pState)

5.241.1 Подробное описание

Сокращение разрядности данных с предварительной их обработкой.

$$pDstVec[i] = Convert(Clip(pSrcVec[i], nClipFactor)) + nAddValue$$

$$i = \overline{0 \dots nSize - 1}$$

Сокращение разрядности данных выполняется после предварительной обработки и осуществляется путем отбрасывания старших битов.

Аргументы

pSrcVec	Входной вектор.
nClipFactor	Показатель степени, определяющий верхний и нижний пороги насыщения. =[1,2,3...15]
nAddValue	Добавляемая константа.
nSize	Размер векторов в элементах. указатель на матрицу коэффициентов для векторного умножителя. Для уекорения.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.242 nmppsCmpEqC

Сравнивает элементы массива на признак равенства константе.

Функции

- void nmppsCmpEqC_16u15b (**nm16u15b** *pSrcVec, **uint15b** nCmpVal, **nm16s** *pDstVec, int nSize, **int16b** nTrueFlag)
- void nmppsCmpEqC_8u7b (**nm8u7b** *pSrcVec, **uint7b** nCmpVal, **nm8s** *pDstVec, int nSize, **int8b** nTrueFlag)
- void nmppsCmpEqC_4u3b (**nm4u3b** *pSrcVec, **uint3b** nCmpVal, **nm4s** *pDstVec, int nSize, **int4b** nTrueFlag)

5.242.1 Подробное описание

Сравнивает элементы массива на признак равенства константе.

$$pDstVec(i) = \begin{cases} nTrueFlag, & \text{if } pSrcVec(i) = nCmpVal \\ 0, & \text{if } pSrcVec(i) \neq nCmpVal \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\^

Аргументы

pSrcVec	Входной вектор.
nCmpVal	Значение константы для сравнения
nSize	Размер векторов в элементах.
nTrueFlag	Значение флага, устанавливаемого при выполнении условия

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.243 nmppsCmpNe0

Сравнивает элементы массива на признак неравенства нулю.

Функции

- void nmppsCmpNe0_8s (const **nm8s** *pSrcVec, **nm8s** *pDstVec, int nSize)
- void nmppsCmpNe0_16s (const **nm16s** *pSrcVec, **nm16s** *pDstVec, int nSize)
- void nmppsCmpNe0_32s (const **nm32s** *pSrcVec, **nm32s** *pDstVec, int nSize)
- void nmppsCmpNe0_64s (const **nm64s** *pSrcVec, **nm64s** *pDstVec, int nSize)

5.243.1 Подробное описание

Сравнивает элементы массива на признак неравенства нулю.

$$pDstVec(i) = \begin{cases} -1, & \text{if } pSrcVec(i) \neq 0 \\ 0, & \text{if } pSrcVec(i) = 0 \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.244 nmppsCmpEq0

Сравнивает элементы массива на признак равенства нулю.

Функции

- void nmppsCmpEq0_32s (const **nm32s** *pSrcVec, **nm32s** *pDstVec, int nSize)

5.244.1 Подробное описание

Сравнивает элементы массива на признак равенства нулю.

$$pDstVec(i) = \begin{cases} -1, & \text{if } pSrcVec(i) = 0 \\ 0, & \text{if } pSrcVec(i) \neq 0 \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\^

Аргументы

pSrcVec	Входной вектор.
---------	-----------------

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

nSize	Размер векторов в элементах.
-------	------------------------------

Возвращает

void

5.245 nmppsCmpNeC

Сравнивает элементы массива на признак неравенства константе.

Функции

- void nmppsCmpNeC_8s (const **nm8s** *pSrcVec, **int8b** nCmpVal, **nm8s** *pDstVec, int nSize)
- void nmppsCmpNeC_16s (const **nm16s** *pSrcVec, **int16b** nCmpVal, **nm16s** *pDstVec, int nSize)
- void nmppsCmpNeC_32s (const **nm32s** *pSrcVec, **int32b** nCmpVal, **nm32s** *pDstVec, int nSize)
- void nmppsCmpNeC_64s (const **nm64s** *pSrcVec, **int64b** nCmpVal, **nm64s** *pDstVec, int nSize)

5.245.1 Подробное описание

Сравнивает элементы массива на признак неравенства константе.

$$i = \overline{0 \dots nSize - 1}$$

\^

Аргументы

pSrcVec	Входной вектор.
nCmpVal	Значение константы для сравнения
nSize	Размер векторов в элементах.
nTrueFlag	Значение флага, устанавливаемого при выполнении условия

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.246 nmppsCmpNeC_flag

Сравнивает элементы массива на признак неравенства константе.

Функции

- void nmppsCmpNeC_8u7b (**nm8u7b** *pSrcVec, **uint7b** nCmpVal, **nm8s** *pDstVec, int nSize, **int8b** nTrueFlag)
- void nmppsCmpNeC_16u15b (**nm16u15b** *pSrcVec, **uint15b** nCmpVal, **nm16s** *pDstVec, int nSize, **int16b** nTrueFlag)

5.246.1 Подробное описание

Сравнивает элементы массива на признак неравенства константе.

$$pDstVec(i) = \begin{cases} nTrueFlag, & \text{if } pSrcVec(i) \neq nCmpVal \\ 0, & \text{if } pSrcVec(i) = nCmpVal \end{cases}$$

$$pDstVec(i) = \begin{cases} nTrueFlag, & \text{if } pSrcVec(i) \neq nCmpVal \\ 0, & \text{if } pSrcVec(i) = nCmpVal \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\^

Аргументы

pSrcVec	Входной вектор.
nCmpVal	Значение константы для сравнения
nSize	Размер векторов в элементах.
nTrueFlag	Значение флага, устанавливаемого при выполнении условия

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.247 nmppsCmpLtC

Функция сравнения элементов массива с константой (меньше)

Функции

- void nmppsCmpLtC_8s7b (const **nm8s7b** *pSrcVec, **int8b** nCmpVal, **nm8s** *pDstVec, int nSize)
- void nmppsCmpLtC_16s15b (const **nm16s15b** *pSrcVec, **int16b** nCmpVal, **nm16s** *pDstVec, int nSize)
- void nmppsCmpLtC_32s31b (const **nm32s31b** *pSrcVec, **int32b** nCmpVal, **nm32s** *pDstVec, int nSize)
- void nmppsCmpLtC_64s63b (const **nm64s63b** *pSrcVec, **int64b** nCmpVal, **nm64s** *pDstVec, int nSize)

5.247.1 Подробное описание

Функция сравнения элементов массива с константой (меньше)

Аргументы

pSrcVec	Входной вектор.
nCmpVal	Значение константы для сравнения

Возвращаемые значения

pDstVec	Результирующий вектор.
----------------	------------------------

Аргументы

nSize	Размер векторов в элементах.
--------------	------------------------------

Возвращает

void

5.248 nmppsCmpGtC

Функция сравнения элементов массива с константой (больше)

Функции

- void nmppsCmpGtC_8s7b (const **nm8s7b** *pSrcVec, **int8b** nCmpVal, **nm8s** *pDstVec, int nSize)
- void nmppsCmpGtC_16s15b (const **nm16s15b** *pSrcVec, **int16b** nCmpVal, **nm16s** *pDstVec, int nSize)
- void nmppsCmpGtC_32s31b (const **nm32s31b** *pSrcVec, **int32b** nCmpVal, **nm32s** *pDstVec, int nSize)
- void nmppsCmpGtC_64s63b (const **nm64s63b** *pSrcVec, **int64b** nCmpVal, **nm64s** *pDstVec, int nSize)
- void nmppsCmpGtC_v2nm32f (const **v2nm32f** *pSrcVec, const **v2nm32f** *C, **nm1** *evenFlags, **nm1** *oddFlags, int step, int nSize)

5.248.1 Подробное описание

Функция сравнения элементов массива с константой (больше)

Функция сравнения элементов массива с константой (больше или равно)

Аргументы

pSrcVec	Входной вектор.
nCmpVal	Значение константы для сравнения

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Аргументы

nSize	Размер векторов в элементах.
-------	------------------------------

Возвращает

void

Аргументы

in	pSrcVec	входной массив (элемент массива представляет собой структура v2nm32f, состоящая из двух чисел float)
in	C	константа (два числа float)

Возвращаемые значения

[out]	evenFlags	маска полученная после сравнения первого поля из структуры v2nm32f
[out]	oddFlags	маска полученная после сравнения второго поля из структуры v2nm32f

Аргументы

in	step	шаг, с которым будут браться элементы из pSrcVec
in	nSize	размер массива (в v2nm32f)

В структуре v2nm32f два поля. Первое - v0. Второе - v1.

5.249 nmppsCmpGteC

Функция сравнения элементов массива с константой (больше или равно)

Функции

- void nmppsCmpGteC_v2nm32f (const **v2nm32f** *pSrcVec, const **v2nm32f** *C, **nm1** *evenFlags, **nm1** *oddFlags, int step, int nSize)

5.249.1 Подробное описание

Функция сравнения элементов массива с константой (больше или равно)

Аргументы

in	pSrcVec	входной массив (элемент массива представляет собой структура v2nm32f, состоящая из двух чисел float)
in	C	константа (два числа float)

Возвращаемые значения

[out]	evenFlags	маска полученная после сравнения первого поля из структуры v2nm32f
[out]	oddFlags	маска полученная после сравнения второго поля из структуры v2nm32f

Аргументы

in	step	шаг, с которым будут браться элементы из pSrcVec
in	nSize	размер массива (в v2nm32f)

В структуре v2nm32f два поля. Первое - v0. Второе - v1.

5.250 nmppsCmpEqV_

Поэлементное сравнение элементов (неполной разрядности) двух векторов на признак равенства.

Функции

- void nmppsCmpEqV_16u15b (**nm16u15b** *pSrcVec1, **nm16u15b** *pSrcVec2, **nm16s** *pDstVec, int nSize, **int16b** nTrueFlag)
- void nmppsCmpEqV_8u7b (**nm8u7b** *pSrcVec1, **nm8u7b** *pSrcVec2, **nm8s** *pDstVec, int nSize, **int8b** nTrueFlag)

5.250.1 Подробное описание

Поэлементное сравнение элементов (неполной разрядности) двух векторов на признак равенства.

$$pDstVec(i) = \begin{cases} nTrueFlag, & \text{if } pSrcVec(i) = nCmpVal \\ 0, & \text{if } pSrcVec(i) \neq nCmpVal \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер векторов в элементах.
nTrueFlag	Значение флага, устанавливаемого при выполнении условия

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.251 nmppsCmpNe

Сравнивает элементы массива на признак неравенства константе.

Функции

- void nmppsCmpNe_8s (const **nm8s** *pSrcVec1, const **nm8s** *pSrcVec2, **nm8s** *pDstVec, int nSize)
- void nmppsCmpNe_16s (const **nm16s** *pSrcVec1, const **nm16s** *pSrcVec2, **nm16s** *pDstVec, int nSize)
- void nmppsCmpNe_32s (const **nm32s** *pSrcVec1, const **nm32s** *pSrcVec2, **nm32s** *pDstVec, int nSize)
- void nmppsCmpNe_64s (const **nm64s** *pSrcVec1, const **nm64s** *pSrcVec2, **nm64s** *pDstVec, int nSize)

5.251.1 Подробное описание

Сравнивает элементы массива на признак неравенства константе.

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2
nSize	Размер векторов в элементах

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Возвращает

void

5.252 nmppsCmpLt

Функция сравнения элементов массива с константой (меньше)

Функции

- void nmppsCmpLt_8s7b (const **nm8s** *pSrcVec1, const **nm8s** *pSrcVec2, **nm8s** *pDstVec, int nSize)
- void nmppsCmpLt_16s15b (const **nm16s** *pSrcVec1, const **nm16s** *pSrcVec2, **nm16s** *pDstVec, int nSize)
- void nmppsCmpLt_32s31b (const **nm32s** *pSrcVec1, const **nm32s** *pSrcVec2, **nm32s** *pDstVec, int nSize)
- void nmppsCmpLt_64s63b (const **nm64s** *pSrcVec1, const **nm64s** *pSrcVec2, **nm64s** *pDstVec, int nSize)

5.252.1 Подробное описание

Функция сравнения элементов массива с константой (меньше)

Аргументы

pSrcVec1	Входной вектор1
pSrcVec2	Входной вектор2

Возвращаемые значения

pDstVec	Результирующий вектор
---------	-----------------------

Аргументы

nSize	Размер векторов в элементах
-------	-----------------------------

Возвращает

void

5.253 nmppsCmpNeV_

Поэлементное сравнение элементов двух векторов на признак неравенства.

Функции

- void nmppsCmpNeV_16u ([nm16u15b](#) *pSrcVec1, [nm16u15b](#) *pSrcVec2, [nm16s](#) *pDstVec, int n←Size, [int16b](#) nTrueFlag)
- void nmppsCmpNeV_8u ([nm8u7b](#) *pSrcVec1, [nm8u7b](#) *pSrcVec2, [nm8s](#) *pDstVec, int nSize, [int8b](#) nTrueFlag)

5.253.1 Подробное описание

Поэлементное сравнение элементов двух векторов на признак неравенства.

$$pDstVec(i) = \begin{cases} nTrueFlag, & \text{if } pSrcVec(i) \neq nCmpVal \\ 0, & \text{if } pSrcVec(i) = nCmpVal \end{cases}$$

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec1	Первый входной вектор.
pSrcVec2	Второй входной вектор.
nSize	Размер векторов в элементах.
nTrueFlag	Значение флага, устанавливаемого при выполнении условия

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.254 Vec_ClipRShiftConvert_AddC

Изменение разрядности элементов вектора с клиппированием

Преобразование 32р знаковых данных к восьмиразрядным осуществляется вырезанием битов 16..31.. ,затем клиппированием и вырезанием восьми младших бит реультата. При клиппировании используется глобальная константа типа nm64s с именем _F1CR_16_to_x. По умолчанию, там лежит ff00_ff00_ff00_ff00 соответствующая диапазону результата [-255..256]. К каждому полученному 64р слову результата прибавляется значение глобальной переменной _VR_16_to_x, которая по умолчанию равна нулю.

Функции

- void Vec_ClipRShiftConvert_AddC ([nm32s](#) *pSrcVec, [nm8u](#) *pDstVec, int nSize)

5.254.1 Подробное описание

Изменение разрядности элементов вектора с клиппированием

Преобразование 32р знаковых данных к восьмиразрядным осуществляется вырезанием битов 16..31.. ,затем клиппированием и вырезанием восьми младших бит реультата. При клиппировании используется глобальная константа типа nm64s с именем _F1CR_16_to_x. По умолчанию, там лежит ff00_ff00_ff00_ff00 соответствующая диапазону результата [-255..256]. К каждому полученному 64р слову результата прибавляется значение глобальной переменной _VR_16_to_x, которая по умолчанию равна нулю.

Аргументы

pSrcVec	Входной вектор.
nSize	Размер векторов в элементах.

Возвращаемые значения

pDstVec	Результирующий вектор.
---------	------------------------

Возвращает

void

5.255 nmppsAddr_

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

Функции

- `__INLINE__ nm1 * nmppsAddr_1 (const nm1 *pVec, int nIndex)`
- `nm2s * nmppsAddr_2s (const nm2s *pVec, int nIndex)`
- `nm4s * nmppsAddr_4s (const nm4s *pVec, int nIndex)`
- `__INLINE__ nm8s * nmppsAddr_8s (const nm8s *pVec, int nIndex)`
- `__INLINE__ nm16s * nmppsAddr_16s (const nm16s *pVec, int nIndex)`
- `__INLINE__ nm32s * nmppsAddr_32s (const nm32s *pVec, int nIndex)`
- `__INLINE__ nm32f * nmppsAddr_32f (const nm32f *pVec, int nIndex)`
- `__INLINE__ nm64s * nmppsAddr_64s (const nm64s *pVec, int nIndex)`
- `__INLINE__ nm64f * nmppsAddr_64f (const nm64f *pVec, int nIndex)`
- `nm2u * nmppsAddr_2u (const nm2u *pVec, int nIndex)`
- `nm4u * nmppsAddr_4u (const nm4u *pVec, int nIndex)`
- `__INLINE__ nm8u * nmppsAddr_8u (const nm8u *pVec, int nIndex)`
- `__INLINE__ nm16u * nmppsAddr_16u (const nm16u *pVec, int nIndex)`
- `__INLINE__ nm32u * nmppsAddr_32u (const nm32u *pVec, int nIndex)`
- `__INLINE__ nm64u * nmppsAddr_64u (const nm64u *pVec, int nIndex)`

5.255.1 Подробное описание

Возвращает адрес ячейки памяти, содержащей указанный элемент.

Реализация для процессора NeuroMatrix возвращает адрес, выровненный в памяти на 32 бита.

Аргументы

pVec	Входной вектор.
nIndex	Индекс элемента.

Возвращает

Адрес ячейки памяти.

Заметки

Для ускорения работы на РС возможно использование макроса ADDR(ptr, index), который раскрывается на РС как (ptr+index), а на NM как вызов функции nmppsAddr_.

Аргументы

pVec	Входной вектор.
nIndex	Индекс элемента.

Возвращает

Адрес ячейки памяти.

Заметки

Для ускорения работы на РС возможно использование макроса ADDR(ptr, index), который раскрывается на РС как (ptr+index), а на NM как вызов функции nmppsAddr_.

5.256 nmppsSetVal_

Модификация элемента вектора.

Функции

- void nmppsPut_1 (**nm1** *pVec, int nIndex, **int1b** nVal)
- void nmppsPut_2s (**nm2s** *pVec, int nIndex, **int2b** nVal)
- void nmppsPut_4s (**nm4s** *pVec, int nIndex, **int4b** nVal)
- void nmppsPut_8s (**nm8s** *pVec, int nIndex, **int8b** nVal)
- void nmppsPut_16s (**nm16s** *pVec, int nIndex, **int16b** nVal)
- **_ _ INLINE_ _** void nmppsPut_32s (**nm32s** *pVec, int nIndex, **int32b** nVal)
- **_ _ INLINE_ _** void nmppsPut_64s (**nm64s** *pVec, int nIndex, **int64b** nVal)
- **_ _ INLINE_ _** void nmppsPut_2u (**nm2u** *pVec, int nIndex, **uint2b** nVal)
- **_ _ INLINE_ _** void nmppsPut_4u (**nm4u** *pVec, int nIndex, **uint4b** nVal)
- **_ _ INLINE_ _** void nmppsPut_8u (**nm8u** *pVec, int nIndex, **uint8b** nVal)
- **_ _ INLINE_ _** void nmppsPut_16u (**nm16u** *pVec, int nIndex, **uint16b** nVal)
- **_ _ INLINE_ _** void nmppsPut_32u (**nm32u** *pVec, int nIndex, **uint32b** nVal)
- **_ _ INLINE_ _** void nmppsPut_64u (**nm64u** *pVec, int nIndex, **uint64b** nVal)

5.256.1 Подробное описание

Модификация элемента вектора.

$$pVec(nIndex) = Val$$

Аргументы

pVec	Вектор.
nIndex	Позиция элемента
nVal	Значение элемента

Возвращает

void

$$pVec(nIndex) = Val$$

Аргументы

pVec	Вектор.
nIndex	Позиция элемента
nVal	Значение элемента

Возвращает

void

5.257 nmppsGetVal_

Извлекает значение элемента вектора.

Функции

- void nmppsGetVal_1 (const **nm1** *pVec, int nIndex, **int1b** *nVal)
- void nmppsGetVal_2s (const **nm2s** *pVec, int nIndex, **int2b** *nVal)
- void nmppsGetVal_4s (const **nm4s** *pVec, int nIndex, **int4b** *nVal)

- void nmppsGetVal_8s (const **nm8s** *pVec, int nIndex, **int8b** *nVal)
- void nmppsGetVal_16s (const **nm16s** *pVec, int nIndex, **int16b** *nVal)
- **_ _INLINE_ _** void nmppsGetVal_32s (const **nm32s** *pVec, int nIndex, **int32b** *nVal)
- **_ _INLINE_ _** void nmppsGetVal_64s (const **nm64s** *pVec, int nIndex, **int64b** *nVal)
- void nmppsGetVal_2u (const **nm2u** *pVec, int nIndex, **uint2b** *nVal)
- void nmppsGetVal_4u (const **nm4u** *pVec, int nIndex, **uint4b** *nVal)
- void nmppsGetVal_8u (const **nm8u** *pVec, int nIndex, **uint8b** *nVal)
- void nmppsGetVal_16u (const **nm16u** *pVec, int nIndex, **uint16b** *nVal)
- **_ _INLINE_ _** void nmppsGetVal_32u (const **nm32u** *pVec, int nIndex, **uint32b** *nVal)
- **_ _INLINE_ _** void nmppsGetVal_64u (const **nm64u** *pVec, int nIndex, **uint64b** *nVal)

5.257.1 Подробное описание

Извлекает значение элемента вектора.

Аргументы

pVec	Вектор.
nIndex	Позиция элемента.

Возвращаемые значения

nVal	Значение элемента.
-------------	--------------------

Возвращает

void

Аргументы

pVec	Вектор.
nIndex	Позиция элемента.

Возвращаемые значения

nVal	Значение элемента.
-------------	--------------------

Возвращает

void

5.258 nmppsGetVal_(return)

Извлекает значение элемента вектора.

Функции

- `int2b nmppsGet_2s (const nm2s *pVec, int nIndex)`
- `int4b nmppsGet_4s (const nm4s *pVec, int nIndex)`
- `int8b nmppsGet_8s (const nm8s *pVec, int nIndex)`
- `int16b nmppsGet_16s (const nm16s *pVec, int nIndex)`
- `_ _INLINE_ _ int32b nmppsGet_32s (const nm32s *pVec, int nIndex)`
- `uint1b nmppsGet_1 (const nm1 *pVec, int nIndex)`
- `uint2b nmppsGet_2u (const nm2u *pVec, int nIndex)`
- `uint4b nmppsGet_4u (const nm4u *pVec, int nIndex)`
- `uint8b nmppsGet_8u (const nm8u *pVec, int nIndex)`
- `uint16b nmppsGet_16u (const nm16u *pVec, int nIndex)`
- `_ _INLINE_ _ uint32b nmppsGet_32u (const nm32u *pVec, int nIndex)`

5.258.1 Подробное описание

Извлекает значение элемента вектора.

Аргументы

pVec	Вектор.
nIndex	Позиция элемента.

Возвращает

Значение элемента.

Аргументы

pVec	Вектор.
nIndex	Позиция элемента.

Возвращает

Значение элемента.

5.259 VEC_QSort

Сортировка массива по убыванию.

Функции

- void nmppsQSort_32s ([nm32s](#) *pSrcDstVec, int nSize)

5.259.1 Подробное описание

Сортировка массива по убыванию.

Аргументы

pSrcDstVec	Входной и результирующий вектор.
nSize	Размер вектора в элементах.

Возвращает

void

Restrictions:

Функция работает рекурсивно, передавая параметры через стек, поэтому размер стека должен быть больше $4 * \log_2(nSize)$ 32-битных слов в лучшем случае (элементы массива расположены беспорядочно) и больше $6 * nSize$ 32-битных слов в худшем случае (элементы массива уже упорядочены)

5.260 nmppsRemap_

Переупорядочивание элементов вектора по таблице.

Функции

- void nmppsRemap_32u ([nm32u](#) *pSrcVec, [nm32u](#) *pDstVec, [nm32s](#) *pRemapTable, int nDstVecSize)
- void nmppsRemap_8u ([nm8u](#) *pSrcVec, [nm8u](#) *pDstVec, [nm32s](#) *pRemapTable, int nSrcVecSize, int nDstVecSize, void *pTmpBuf1, void *pTmpBuf2)

5.260.1 Подробное описание

Переупорядочивание элементов вектора по таблице.

$$pDstVec[i] = pSrcVec[pRemapTable[i]],$$

$$i = \overline{0 \dots nSize - 1}$$

\~

Аргументы

pSrcVec	Входной вектор.
pRemapTable	Таблица новых индексов для переупорядочивания.
nDstVecSize	Размер результирующего вектора в элементах.
pTmpBuf1	Временный массив nm32u pTmpBuf1[nSrcVecSize].
pTmpBuf2	Временный массив nm32u pTmpBuf2[nDstVecSize]. Результирующий вектор nm8u pDstVec[nDstVecSize].

Возвращает

void

```
// Функция
// void nmppsRemap_8u(nm8u* pSrcVec, nm8u* pDstVec, nm32s* pRemapTable, int nSrcVecSize, int nDstVecSize, void*
// pTmpBuf1, void* pTmpBuf2);
// выполняет следующие действия:
nmppsConvert_8u((nm8u*) pSrcVec, (nm32u*) pTmpBuf1, nSrcVecSize);
nmppsRemap_32u((nm32u*) pTmpBuf1, (nm32u*) pTmpBuf2, RemapTable, DstVecSize);
nmppsConvert_32s((nm32s*) pTmpBuf2, (nm8s*) pDstVec, DstVecSize);
```

Заметки

Возможность использования *inplace* параметров определяется исходя из последовательности процессов чтения/записи:
the folowing sequence of actions :

- pSrcVec => pTmpBuf1 - *inplace* запрещен;
- pTmpBuf1=>pTmpBuf2 - *inplace* запрещен;
- pTmpBuf2=>pDstVec - *inplace* разрешен;

5.261 nmppSplitTmp

Расщепляет массив на два, группируя по четным и нечетным элементам

Функции

- void nmppsSplitTmp_8s (const **nm8s** *src, **nm8s** *dst1, **nm8s** *dst2, int size, **nm8s** *tmpSizeOfDst)

5.261.1 Подробное описание

Расщепляет массив на два, группируя по четным и нечетным элементам

Аргументы

in	src	Входной массив
out	dst1	Выходной массив размера size/2
out	dst2	Выходной массив размера size/2
in	size	Размер исходного массива в элементах. Кратность параметра size должна соответствовать двум длинным 64-р. словам.
in	tmpSizeOfDst	Временный массив размера size/2

Возвращает

Details Максимальная производительность достигается при размещении входных, выходных и временных массивов в разных банках памяти. Массивы dst1 и dst2 могут находиться в одном банке. Макс производительность на 64-р. слово результата = 2.1 такта (при size=10240 байт) и 2.5 такта (при size=4096 байт)

5.262 nmppSplit

Расщепляет массив на два массива, группируя по четным и нечетным элементам

Функции

- void nmppsSplit_8s (const **nm8s** *src, **nm8s** *dst1, **nm8s** *dst2, int size)
- void nmppsSplit_16s (const **nm16s** *src, **nm16s** *dst1, **nm16s** *dst2, int size)
- void nmppsSplit_32s (const **nm32s** *src, **nm32s** *dst1, **nm32s** *dst2, int size)

5.262.1 Подробное описание

Расщепляет массив на два массива, группируя по четным и нечетным элементам

Аргументы

in	src	Входной массив
out	dst1	Выходной массив размера size/2
out	dst2	Выходной массив размера size/2
in	size	Размер исходного массива в элементах. Кратность параметра size должна соответствовать двум длинным 64-р. словам.

Возвращает

Details Максимальная производительность достигается при размещении входных, выходных массивов в разных банках памяти. Массивы dst1 и dst2 могут находиться в одном банке. Макс производительность на 64-р. слово результата = 2.14 такта (при size=10240 байт) и 2.6 такта (при size=4096 байт)

5.263 nmppMerge

Собирает массив из двух, чередуя элементы из каждого. Функция обратная nmppsSplit.

Функции

- void nmppsMerge_8s (const **nm8s** *src0, const **nm8s** *src1, **nm8s** *dst, int sizeSrc)
- void nmppsMerge_16s (const **nm16s** *src0, const **nm16s** *src1, **nm16s** *dst, int sizeSrc)
- void nmppsMerge_32s (const **nm32s** *src0, const **nm32s** *src1, **nm32s** *dst, int sizeSrc)

5.263.1 Подробное описание

Собирает массив из двух, чередуя элементы из каждого. Функция обратная nmppsSplit.

Аргументы

in	src0	Входной массив размера sizeSrc
in	src1	Входной массив размера sizeSrc
out	dst	Выходной массив размера 2*sizeSrc
in	sizeSrc	Размер выходного массива в элементах. Кратность параметра sizeSrc должна соответствовать 64-р. слову.

Возвращает

Details

5.264 nmppSplit_32fcr

Расщепляет массив на два, группируя по четным и нечетным элементам

Функции

- void nmppsSplit_32fcr (const **nm32fcr** *pSrcVec, **nm32fcr** *pDstVec1, **nm32fcr** *pDstVec, int sizeSrc)

5.264.1 Подробное описание

Расщепляет массив на два, группируя по четным и нечетным элементам

Аргументы

in	pSrcVec	Входной массив
out	pDstVec1	Выходной массив размера size/2
out	pDstVec2	Выходной массив размера size/2
in	sizeSrc	Размер исходного массива в элементах (должен быть четным)

Возвращает

Details Максимальная производительность достигается при размещении входных, выходных массивов в разных банках памяти. Массивы dst1 и dst2 могут находиться в одном банке. Макс производительность на 64-р. слово результата = 1 такт

5.265 nmppsDecimate

Делает выборку элементов из массива с некоторым шагом

Функции

- void nmppsDecimate_16s ([nm16s](#) *pSrc, int startPos, int step, [nm16s](#) *pDst, int nSizeDst)
- void nmppsDecimate_32s ([nm32s](#) *pSrc, int startPos, int step, [nm32s](#) *pDst, int nSizeDst)

5.265.1 Подробное описание

Делает выборку элементов из массива с некоторым шагом

Аргументы

in	pSrc	Входной массив
in	startPos	Положение элемента в 64-р. слове
out	step	Шаг выборки. Кратность параметра step должна соответствовать длинному 64-р. слову.
out	pDst	Выходной массив
in	nSizeDst	Размер результирующего массива в элементах. Кратность параметра size должна соответствовать длинному 64-р. слову.

Возвращает

Details Максимальная производительность достигается при размещении входных, выходных массивов в разных банках памяти.

5.266 Целочисленные функции

Группы

- Векторные функции
- Матричные функции
- Функции обработки сигналов
- Функции обработки изображений
- Скалярные функции
- Базовые регистровые функции библиотеки
- `implement`

5.266.1 Подробное описание

5.267 Функции с плавающей точкой

Группы

- Векторные функции
- Матричные функции
- Функции обработки сигналов
- Базовые регистровые функции библиотеки
- `nblas`

5.267.1 Подробное описание

5.268 Типы данных

Группы

- Типы векторных данных
- Типы скалярных данных

5.268.1 Подробное описание

5.269 Векторные функции

Группы

- Функции поддержки
- Инициализация и копирование
- Арифметические операции
- Логические и бинарные операции
- Операции сравнения
- Статистические функции
- Переупорядочивание и сортировка

5.269.1 Подробное описание

5.270 Векторные функции

Группы

- Инициализация и копирование
- Арифметические операции
- Операции сравнения
- Элементарные математические функции

5.270.1 Подробное описание

5.271 Матричные функции

Группы

- Инициализация и копирование
- Функции поддержки
- Векторно-матричные операции

5.271.1 Подробное описание

5.272 Матричные функции

Группы

- Обращение матрицы

5.272.1 Подробное описание

5.273 Функции обработки сигналов

Группы

- Свертка
- Масочная фильтрация
- Изменение размеров
- Быстрое преобразование Фурье

5.273.1 Подробное описание

5.274 Функции обработки сигналов

Группы

- Быстрое преобразование Фурье

5.274.1 Подробное описание

5.275 Функции обработки изображений

Группы

- [Floodfill](#)

Исполняет разделение бинарной картинки на односвязные области. Пример вызова: no=VL_<
FloodFill32b(pSrcImage, Tetr,Image, pTmpBuff, nSrcWidth, nSrcHeight);.

- Переупорядочивание изображений
- Арифметические действия
- Масочная фильтрация
- Инициализация и копирование
- Изменение размеров
- Операции выборки
- Функции поддержки

5.275.1 Подробное описание

5.276 Скалярные функции

Группы

- Инициализация
- Integer operations
- Fix-point 64
- Fix-point 32
- Арифметические операции
- Функции деинтерлейсинга

5.276.1 Подробное описание

5.276.1.1 Введение

Назначением данной библиотеки является предоставление базовых операций по работе со скалярными данными для процессора NM6403, NM6404, NM6405.

В состав библиотеки входят арифметические, тригонометрические функции, функции для работы с данными в формате с фиксированной точкой.

Библиотека предназначена для быстрой разработки эффективных пользовательских программ на языке высокого уровня(C++).

Назначением данной библиотеки является предоставление базовых операций обработки изображений для процессора NM6403, NM6404, NM6405. В состав библиотеки входят функции двумерной фильтрации, арифметические действия и цветовые преобразования. Библиотека предназначена для быстрой разработки эффективных пользовательских программ на языке высокого уровня с использованием преимуществ архитектуры данного процессора.

Функции библиотеки имеют C++ интерфейс. Большинство функций библиотеки реализованы на языке ассемблера с использованием векторных инструкций и оптимизированы под архитектуру процессора NM6403.

Для удобства разработки прикладных программ библиотека содержит аналогичные реализации функций для процессоров серии x86, выполненных на языке C++. Данные реализации позволяют выполнять написанные с использованием данной библиотеки прикладные программы на персональном компьютере.

Назначением данной библиотеки является предоставление базовых операций по обработке матриц для процессорах NM6403, NM6404, NM6405. В состав библиотеки входят арифметические операции над матрицами. Библиотека предназначена для быстрой разработки эффективных пользовательских программ как на языке высокого уровня(C++).

Функции библиотеки имеют C++ интерфейс. Большинство функций библиотеки реализованы на языке ассемблера с использованием векторных инструкций и оптимизированы под архитектуру процессора NM6403.

Для удобства разработки прикладных программ библиотека содержит аналогичные реализации функций для процессоров серии x86, выполненных на языке C++. Данные реализации позволяют выполнять написанные с использованием данной библиотеки прикладные программы на персональном компьютере.

Назначением данной библиотеки является предоставление базовых функций по обработке сигналов для процессоров NM6403,NM6404,NM6405. В состав библиотеки входят функции одномерной КИХ фильтрации, нелинейной фильтрации, передискретизации. Библиотека предназначена для быстрой разработки эффективных пользовательских программ как на языке высокого уровня(C++).

Функции библиотеки имеют C++ интерфейс. Большинство функций библиотеки реализованы на языке ассемблера с использованием векторных инструкций и оптимизированы под архитектуру процессоров NM6403.

Для удобства разработки прикладных программ библиотека содержит аналогичные реализации функций для процессоров серии x86, выполненных на языке C++. Данные реализации позволяют выполнять написанные с использованием данной библиотеки прикладные программы на персональном компьютере.

Назначением данной библиотеки является предоставление базовых операций по обработке одномерных массивов (векторов) для процессоров NM6405,NM6406, систем на кристале с ядром NMC.

В состав библиотеки входят логические и арифметические функции, операции сравнения, инициализации, копирования, преобразования разрядностей и т.п. Библиотека предназначена для быстрой разработки эффективных пользовательских программ как на языке высокого уровня(C++), так и на языке ассемблера с помощью прилагаемой библиотеки ядра низкоуровневых функций. Функции библиотеки имеют C++ интерфейс.

Большинство функций библиотеки реализованы на языке ассемблера с использованием векторных инструкций и оптимизированы под архитектуру процессоров NMC. Для удобства разработки прикладных программ библиотека содержит аналогичные реализации функций для процессоров серии x86, выполненных на языке C++. Данные реализации позволяют выполнять написанные с использованием данной библиотеки прикладные программы на персональном компьютере.

Функции векторного ядра библиотеки

Функции различных библиотек: nmply,nmpls, nmpli, nmplm и др. , имеющие C++ интерфейсы, в своей реализации используют вызовы функций ядра. Функции ядра не имеют C++ интерфейса. Их вызов возможен только из ассемблера процессора NeuroMatrix. Передача параметров и настройка функций производится через регистры.

Одна и та же функция ядра может использоваться при реализации одной или нескольких функций библиотеки. Функции ядра также могут быть использованы для реализации пользовательских функций. Использование функций ядра позволяет минимизировать время разработки, уменьшить размер кода и получить максимальную производительность.

5.277 Базовые регистровые функции библиотеки

Группы

- Элементарные функции
- функции взвешенного суммирования
- Целевые функции

5.277.1 Подробное описание

5.278 implement

Группы

- контроль переполнения

5.278.1 Подробное описание

5.279 Базовые регистровые функции библиотеки

Группы

- Элементарные функции

5.279.1 Подробное описание

5.280 nmblas

Группы

- BLASS-LEVEL1
- BLASS-LEVEL2
- BLASS-LEVEL3

5.280.1 Подробное описание

5.281 контроль переполнения

Структуры данных

- class `tcube< T >`
- class `nmmtr< T >`
- class `nmmtrpack< T >`
- class `nmvecpack< T >`
- class `vec< T >`

Макросы

- `#define GetVec getvec`

Функции

- `_ _INLINE_ _ ostream & operator<< (ostream &s, mtr< unsigned char > &mtr)`

5.281.1 Подробное описание

определяет классы, предназначенные для контроля переполнения при реализации библиотеки на РС.

—реализации библиотеки на РС производит контроль переполнения с выдачей сообщения об ошибке пользователю библиотеки. Для этой цели определены шаблонные классы для вектора, матрицы и скалярного числа, позволяющие производить базовые операции над их элементами.

5.281.2 Макросы

5.281.2.1 GetVec

```
#define GetVec getvec
```

Класс матриц.

Примеры:

```
int Test[10]={1,125,3,4,5,6,7,8,9,10};  
int Res[10];  
mtr<int> AA0(3,3);  
vec<int> A0(3);  
scalar<int> a0(2);  
mtr<int> BB0(3,3);  
mtr<int> CC0(3,3);  
vec<int> C0(3);  
BB0.SetData(Test);  
BB0=AA0;  
A0 =AA0[2];  
a0 =AA0[2][2];  
CC0=AA0+BB0;  
CC0=AA0*a0;  
C0 =AA0*A0;  
CC0=AA0*BB0;
```

5.281.3 Функции

5.281.3.1 operator<<()

```
--INLINE__ ostream & operator<< (
    ostream & s,
    mtr< unsigned char > & mtr )
```

Класс матриц.

Примеры:

```
int Test[10]={1,125,3,4,5,6,7,8,9,10};
int Res [10];
mtr<int> AA0(3,3);
vec<int> A0(3);
scalar<int> a0(2);
mtr<int> BB0(3,3);
mtr<int> CC0(3,3);
vec<int> C0(3);
BB0.SetData(Test);
BB0=AA0;
A0 =AA0[2];
a0 =AA0[2][2];
CC0=AA0+BB0;
CC0=AA0*a0;
C0 =AA0*A0;
CC0=AA0*BB0;
```

5.282 Элементарные функции

Группы

- [Vec_0_sub_data](#)
- [Vec_activate_data](#)
- [Vec_activate_data_add_0](#)
- [Vec_activate_data_xor_data](#)
- [Vec_activate_data_add_ram](#)
- [Vec_afifo](#)
- [Vec_data](#)
- [Vec_data_add_ram](#)
- [Vec_data_and_ram](#)
- [Vec_data_or_ram](#)
- [Vec_data_sub_ram](#)
- [Vec_data_xor_ram](#)
- [Vec_And](#)
- [Vec_Or](#)
- [Vec_Xor](#)
- [Vec_Add](#)
- [Vec_Sub](#)
- [Vec_not_data](#)
- [Vec_ram](#)
- [Vec_ram_sub_data](#)
- [Vec_vsum_activate_data_0](#)

- void [vec_Mask \(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar2, nmreg gr2, nmreg gr5, nmreg ar6, nmreg gr6\)](#)

5.282.1 Подробное описание

5.282.2 Функции

5.282.2.1 vec_Mask()

```
void vec_Mask (
    nmreg ar0,
    nmreg gr0,
    nmreg ar1,
    nmreg gr1,
    nmreg ar2,
    nmreg gr2,
    nmreg gr5,
    nmreg ar6,
    nmreg gr6 )
```

\`~
\defgroup vec_Mask

\`~russian Ядро функции nmppsMaskV_().
\`~english nmppsMaskV_() function core.

\`~\`~russian Действие функции эквивалентно следующим псевдоинструкциям:
\`~\`~english The function operation is equivalent to the following pseudoinstructions:
\`~\`~

```
rep N ram =[ar0++gr0];
rep N data=[ar1++gr1]  with data;
rep N data=[ar2++gr2]  with mask data,ram,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar2	указатель на столбец SrcMtr2 (маска)
gr2	SrcMtr3 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для arg6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar6.

5.283 Элементарные функции

Группы

- [Vec_copyEvenToEven_32f](#)
- [Vec_CopyOddToEven_32f](#)

5.283.1 Подробное описание

5.284 функции взвешенного суммирования

Группы

- [Vec_ClipMul2D2W8_AddVr](#)
- [Vec_ClipMulNDNW2_AddVr](#)
- [Vec_ClipMulNDNW4_AddVr](#)
- [Vec_ClipMulNDNW8_AddVr](#)
- [Vec_Mul2D2W1_AddVr](#)
- [Vec_Mul2D2W2_AddVr](#)
- [Vec_Mul2D2W4_AddVr](#)
- [Vec_Mul2D2W8_AddVr](#)
- [Vec_Mul3D3W2_AddVr](#)
- [Vec_Mul3D3W8_AddVr](#)
- [Vec_Mul4D4W2_AddVr](#)
- [Vec_MulVN_AddVN](#)
- [Vec_vsum_data_0](#)
- [Vec_vsum_data_vr](#)
- [Vec_vsum_shift_data_0](#)
- [Vec_vsum_shift_data_vr](#)
- [Vec_vsum_shift_data_afifo](#)

5.284.1 Подробное описание

5.285 Целевые функции

Группы

- [`
`](#)
- [Vec_data_add_afifo](#)
- [Vec_FilterCoreRow2](#)
- [Vec_FilterCoreRow4](#)
- [Vec_FilterCoreRow8](#)
- [Vec_Abs](#)
- [Vec_ClipExt](#)

- [Vec_IncNeg](#)
- [Vec_SubAbs](#)
- [Vec_SubVN_Abs](#)
- [Vec_Swap](#)
- [Vec_MUL_2V4toW8_shift](#)
- [Vec_MUL_2V8toW16_shift](#)
- [Vec_vsum_data_afifo](#)
- [Vec_CompareMinV](#)

Поэлементный поиск минимального

Действие функции эквивалентно следующим псевдоинструкциям:

- [Vec_CompareMaxV](#)

Поэлементный поиск максимального

Действие функции эквивалентно следующим псевдоинструкциям:

- [Vec_DupValueInVector8](#)

Размножение 8-ми битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

- [Vec_DupValueInVector16](#)

Размножение 16-ти битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

- [Vec_BuildDiagWeights8](#)

Построение диагональной матрицы весовых коэффициентов (8x8).

- [Vec_BuildDiagWeights16](#)

Построение диагональной матрицы весовых коэффициентов (16x16).

- [Vec_MaxVal_v8nm8s](#)

Поиск максимума в 8 байтах

- [Vec_MaxVal_v4nm16s](#)

Поиск максимума в 4-х 16р. элементах

- [Vec_MaxVal](#)

Поиск максимумов в колонках матрицы SrcMtr1.

- [Vec_MinVal_v8nm8s](#)

Поиск минимума в 8 байтах

- [Vec_MinVal_v4nm16s](#)

Поиск минимума в 4-х 16р. элементах

- [Vec_MinVal](#)

Поиск минимумов в колонках матрицы SrcMtr1.

- [Vec_AccMul1D1W32_AddVr](#)

Умножение с накоплением

Действие функции эквивалентно следующим псевдоинструкциям:

5.285.1 Подробное описание

5.286 Vec_0_sub_data

Функции

- void vec_0_sub_data ([nmreg nb1](#), [nmreg ar0](#), [nmreg gr0](#), [nmreg gr5](#), [nmreg ar6](#), [nmreg gr6](#))

5.286.1 Подробное описание

\~

\~russian Ядро функции nmppsNeg().
\~english nmppsNeg() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with 0-data;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.287 Vec_activate_data

Функции

- void vec_activate_data (**nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.287.1 Подробное описание

\~

Ядро функции nmppsCmpLt0_().

Действие функции эквивалентно следующим псевдоинструкциям:
rep N data=[ar0++gr0] with activate data;
rep N [ar6++gr6]=afifo;

Аргументы

f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.288 Vec_activate_data_add_0

Функции

- void vec_activate_data_add_0 ([nmreg](#) f1cr, [nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.288.1 Подробное описание

\~

Функция производит арифметическую активацию.

Ядро функции nmppsClipPowC_().

Действие функции эквивалентно следующим псевдоинструкциям:
rep N data=[ar0++|gr0] with activate data + 0;
rep N [ar6++|gr6]=afifo;

Аргументы

f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.289 Vec_activate_data_xor_data

Функции

- void vec_activate_data_xor_data ([nmreg f1cr](#), [nmreg ar0](#), [nmreg gr0](#), [nmreg gr5](#), [nmreg ar6](#), [nmreg gr6](#))

5.289.1 Подробное описание

\~

\~russian Функция позволяет вычислить приближенное значение модуля.
\~english The function allow calculate approximate absolute value.

\~
\~russian Ядро функции nmppsAbs1().
\~english nmppsAbs1() function core.
\~

Действие функции эквивалентно следующим псевдоинструкциям:
rep N data=[ar0++|gr0] with activate data xor data;
rep N [ar6++|gr6]=afifo;

Аргументы

f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.290 Vec_activate_data_add_ram

Функции

- void vec_activate_data_add_ram (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.290.1 Подробное описание

\~

\~
russian Функция выполняет арифметическую активацию с прибавлением константы.
\~english The function executes arithmetic activation with adding a constant.

\~
russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N ram =[ar1]
rep N data=[ar0++gr0] with activate data+ram;
rep N [ar6++gr6] = afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на 64р. слово
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, ar6.

5.291

Функции

- void vec_Add_VV_shift (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.291.1 Подробное описание

\~

\~russian Функция служит для суммирования двух массивов со сдвигом результата на 1 бит вправо.
\~english The function serves for two arrays summation with shifting the result for 1 bit to the right.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:

\~english The function operation is equivalent to the following pseudoinstructions:

\~

```
rep N ram = [ar4];
rep N data = [ar0++gr0] with data + 0;
rep N data = [ar1++gr1] with data + afifo;
rep N with mask ram,shift afifo,0;
rep N [ar6++gr6] = afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на строки
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на 64р. слово (маска)
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar6,gr2,gr5.

5.292 Vec_afifo

Функции

- void vec_afifo (**nmreg** ar0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.292.1 Подробное описание

\~

\~russian Функция служит для заполнения массива константой.
\~english The function serves for filling an array with a constant.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep 1 data=[ar0] with data;
rep 1*N [ar6++gr6]=afifo with afifo;
```

Аргументы

ар0	указатель на 64р. слово
гт5	Высота матриц N = [0,1,2...31,32,33,...]
ар6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.293 Vec_data

Функции

- void vec_data ([nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.293.1 Подробное описание

\~

\~russian Ядро функции nmppsCopy_().
\~english Core of nmppsCopy_() function.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with data;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.294 Vec_copyEvenToEven_32f

Функции

- void vec_CopyEvenToEven_32f (**nmreg** ar0, **nmreg** ar6, **nmreg** gr5)

5.294.1 Подробное описание

\~russian Ядро функций nmppsCopyEvenToEven_32f и nmppsCopyOddToOdd_32f.
\~english Core of nmppsCopyEvenToEven_32f and nmppsCopyOddToOdd_32f functions.

\~russian Функция копирует числа с плавающей точкой одинарной точности, лежащие с четного адреса, на четный:
\~english The function copies single-precision floating-point numbers from an even address to an even address
\~

Аргументы

ar0	указатель (может быть только четным) на входной массив чисел с плавающей точкой одинарной точности
ar6	указатель (может быть только четным) на выходной массив чисел с плавающей точкой одинарной точности
gr5	размер (может быть любой) входного массива size = [0, 1, 2 ... 31, 32, 33 ...]

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, ar6.

5.295 Vec_CopyOddToEven_32f

Функции

- void vec_CopyOddToEven_32f (**nmreg** ar0, **nmreg** ar6, **nmreg** gr5)

5.295.1 Подробное описание

\~russian Ядро функций nmppsCopyOddToEven_32f и nmppsCopyEvenToOdd_32f.
\~english Core of nmppsCopyOddToEven_32f and nmppsCopyEvenToOdd_32f functions.

\~russian Функция копирует числа с плавающей точкой одинарной точности, лежащие с нечетного адреса, на четный:
\~english The function copies single-precision floating-point numbers from an odd address to an even address
\~
\~

Аргументы

ar0	указатель (может быть только нечетным) на входной массив чисел с плавающей точкой одинарной точности
ar6	указатель (может быть только четным) на выходной массив чисел с плавающей точкой одинарной точности
gr5	размер (может быть любой кроме 0) входного массива size = [1, 2 ... 31, 32, 33 ...]

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, ar6.

5.296 Vec_data_add_afifo

Функции

- void vec_data_add_afifo (**nmreg** nb1, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6)

5.296.1 Подробное описание

\~

\~russian Ядро функции nmppsSum().
\~english Core of nmppsSum() function.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep 1 data=vfalse;
.repeat N;
rep 1 data=[ar0+++gr0] with data + afifo; (rep1 N times)
.endrepeat;
rep 1 [ar6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на 64р. слово

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0.

5.297 Vec_data_add_ram

Функции

- void vec_data_add_ram ([nmreg nb1](#), [nmreg ar0](#), [nmreg gr0](#), [nmreg ar1](#), [nmreg gr5](#), [nmreg ar6](#), [nmreg gr6](#))

5.297.1 Подробное описание

\~

\~russian Ядро функции nmppsAddC().
\~english Core of nmppsAddC() function.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~

```
rep N ram=[ar1];
rep N data=[ar0++gr0] with data+ram;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr stride
ar1	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.298 Vec_data_and_ram

Функции

- void vec_data_and_ram (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.298.1 Подробное описание

\~

\~russian Ядро функции nmppsAndC_().
\~english Core of nmppsAndC_() function.

\~\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~english The function operation is equivalent to the following pseudoinstructions:
\~\~

```
rep N ram =[ar1];
rep N data=[ar0++gr0] with data and ram;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.299 Vec_data_or_ram

Функции

- void vec_data_or_ram (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.299.1 Подробное описание

\~

\~russian Ядро функции nmppsOrC_().
\~english Core of nmppsOrC_() function.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:

\~

```
rep N ram =[ar1];
rep N data=[ar0++gr0] with data or ram;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.300 Vec_data_sub_ram

Функции

- void vec_data_sub_ram ([nmreg](#) nb1, [nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) ar1, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.300.1 Подробное описание

\~

\~russian Ядро функции nmppsSubC().
\~english Core of nmppsSubC() function.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:

```
rep N ram=[ar1];
rep N data=[ar0++gr0] with data-ram;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
ar1	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.301 Vec_data_xor_ram

Функции

- void vec_data_xor_ram (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.301.1 Подробное описание

\~

\~russian Ядро функции nmppsXorC_().
\~english Core of nmppsXorC_() function.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~
\code
 rep N ram = [ar1];
 rep N data = [ar0++gr0] with data xor ram;
 rep N [ar6++gr6] = afifo;
\endcode

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.302 Vec_FilterCoreRow2

Функции

- void vec_FilterCoreRow2 ([nmreg](#) ar0, [nmreg](#) ar4, [nmreg](#) ar6, [nmreg](#) gr1, [nmreg](#) gr4, [nmreg](#) gr6)

5.302.1 Подробное описание

\~

5.303 Vec_FilterCoreRow4

Функции

- void vec_FilterCoreRow4 ([nmreg](#) ar0, [nmreg](#) ar4, [nmreg](#) ar6, [nmreg](#) gr1, [nmreg](#) gr4, [nmreg](#) gr6)

5.303.1 Подробное описание

5.304 Vec_FilterCoreRow8

Функции

- void vec_FilterCoreRow8 ([nmreg](#) ar0, [nmreg](#) ar4, [nmreg](#) ar6, [nmreg](#) gr1, [nmreg](#) gr4, [nmreg](#) gr6)

5.304.1 Подробное описание

5.305 Vec_And

Функции

- void vec_And (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.305.1 Подробное описание

\~russian Ядро функции nmppsAnd_().
\~english nmppsAnd_() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with data;
rep N data=[ar1++gr1] with data and afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для ar6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.306 Vec_Or

Функции

- void vec_Or (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.306.1 Подробное описание

\~

\~
\~russian Ядро функции nmppsOr_().
\~english nmppsOr_() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with data;
rep N data=[ar1++gr1] with data or afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для ar6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.307 Vec_Xor

Функции

- void vec_Xor ([nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) ar1, [nmreg](#) gr1, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.307.1 Подробное описание

\~

\~russian Ядро функции nmppsXor_().
\~english nmppsXor_() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with data;
rep N data=[ar1++gr1] with data xor afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для ar6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.308 Vec_Abs

Функции

- void vec_Abs (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.308.1 Подробное описание

\~

\~ russian Ядро функции nmppsAbs().
\~ english nmppsAbs() function core.

\~
\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~ english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N ram =[ar0++gr0]  with activate data;
rep N           with vsum afifo,ram,ram;
rep N [ar6++gr6]= afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	установлено системой Doxygen Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.309 Vec_Add

Функции

- void vec_Add (**nmreg** nb1, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.309.1 Подробное описание

\~

\~ russian Ядро функции nmppsAdd().
\~ english nmppsAdd() function core.

\~\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~ english The function operation is equivalent to the following pseudoinstructions:
\~\~

```
rep N data=[ar0+++gr0] with data;
rep N data=[ar1+++gr1] with data + afifo;
rep N [ar6+++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для аг6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.310 Vec_ClipExt

Функции

- void vec_ClipExt (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** ar2, **nmreg** ar3, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.310.1 Подробное описание

\~

\~russian Ядро функции nmpssClipCC_().
\~english nmpssClipCC_() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N ram = [ar1];
rep N data = [ar0++gr0] with data-ram;
rep N data = [ar2]      with activate afifo+data;
rep N data = [ar3]      with activate afifo-data;
rep N [ar6++gr6] = afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
ar1	указатель на 64р. слово-константу
ar2	указатель на 64р. слово-константу
ar3	указатель на 64р. слово-константу
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для аг6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

nmpssClipCC_

5.311 Vec_ClipMul2D2W8_AddVr

Функции

- void vec_ClipMul2D2W8_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.311.1 Подробное описание

\~russian Взвешенное умножение двух массивов с накоплением и активацией
\~english Weighted multiplication of two arrays with accumulation and activation

\~
\~russian Ядро функции nmpssClipPowC_RShift_Convert_AddC_().
\~english nmpssClipPowC_RShift_Convert_AddC_() function core.

Действие функции эквивалентно следующим псевдоинструкциям:

```
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,vr;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,activate data,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
vr	константа для суммирования
ar0	SrcMtr1
gr0	SrcMtr1 stride
ar1	SrcMtr2
gr1	SrcMtr2 stride
ar4	2 матрицы весовых коэффициентов по 8 64р. слов
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для ar6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6,gr7.

5.312 Vec_ClipMulNDNW2_AddVr

Функции

- void vec_ClipMulNDNW2_AddVr (**nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6**)

5.312.1 Подробное описание

\~

\~
russian Взвешенное умножение нескольких массивов с накоплением и активацией
\~english Weighted multiplication of several arrays with accumulation and activation

\~
russian Ядро функции SIG_Filter().
\~english nmppsFilter_() function core.
\~

Действие функции эквивалентно следующим псевдоинструкциям:

```
ar2=ar0;
gr2=[ar1++];
ar0=ar2+gr2;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,vr;
.repeat K-1;
gr2=[ar1++];
ar0=ar2+gr2;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,afifo;
.endrepeat;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 2 строки (sb=2)
f1cr	задает функцию активации
vr	константа для суммирования
ar0	задает базовый адрес для входных массивов (как правило адрес первого массива)
gr0	шаг чтения входного массива stride for input arrays
ar1	массив адресных смещений входных массивов относительно ar0
gr1	количество массив - K
ar4	массив из K матриц весовых коэффициентов по 2 64р. слов
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для аг6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,gr1,ar2,gr2,ar3,gr3,ar4,ar6,gr7.

5.313 Vec_ClipMulNDNW4_AddVr

Функции

- void vec_ClipMulNDNW4_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.313.1 Подробное описание

\^

\^ russian Взвешенное умножение нескольких массивов с накоплением и активацией
\^ english Weighted multiplication of several arrays with accumulation and activation

\^
\^ russian Ядро функции SIG_Filter().
\^ english nmppsFilter_() function core.
\^

Действие функции эквивалентно следующим псевдоинструкциям:

```
ar2=ar0;
gr2=[ar1++];
ar0=ar2+gr2;
rep 4 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,vr;
.repeat K-1;
gr2=[ar1++];
ar0=ar2+gr2;
rep 4 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,afifo;
.endrepeat;
rep N [ar6++ gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 4 строки (sb=20002h)
f1cr	задает функцию активации
vr	константа для суммирования
ar0	задает базовый адрес для входных массивов (как правило адрес первого массива)
gr0	stride for input arrays
ar1	массив адресных смещений входных массивов относительно ar0
gr1	количество массив - K
ar4	массив из K матриц весовых коэффициентов по 4 64р. слов

Аргументы

gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для ar6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,gr1,ar2,gr2,ar3,gr3,ar4,ar6,gr7.

5.314 Vec_ClipMulNDNW8_AddVr

Функции

- void vec_ClipMulNDNW8_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.314.1 Подробное описание

\~

\~ russian Взвешенное умножение нескольких массивов с накоплением и активацией
\~ english Weighted multiplication of several arrays with accumulation and activation

\~\~ russian Ядро функции SIG_Filter().
\~\~ english nmppsFilter_() function core.
\~

Действие функции эквивалентно следующим псевдоинструкциям:

```

ar2=ar0;
gr2=[ar1++];
ar0=ar2+gr2;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,vr;
.repeat K-1;
gr2=[ar1++];
ar0=ar2+gr2;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,activate data,afifo;
.endrepeat;
rep N [ar6++gr6]=afifo;

```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 8 строки (sb=2020202h)

Аргументы

f1cr	задает функцию активации
vr	константа для суммирования
ar0	\ задает базовый адрес для входных массивов (как правило адрес первого массива) \ set base address for input arrays
gr0	stride for input arrays
ar1	массив адресных смещений входных массивов относительно ar0
gr1	\ количество массив - K \ number of arrays - K
ar4	массив из K матриц весовых коэффициентов по 8 64р. слов
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на результирующий столбец
gr6	межстрочный шаг для аг6

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,gr1,ar2,gr2,ar3,gr3,ar4,ar6,gr7.

5.315 Vec_IncNeg

Функции

- void vec_IncNeg (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.315.1 Подробное описание

\~

\~ russian Увеличивает отрицательные числа на 1.
\~ english It increases negative numbers by 1.

\~\~ russian Применяется в nmppsDivC().
\~\~ english It is used in nmppsDivC().

Действие функции эквивалентно следующим псевдоинструкциям:
rep N ram = [ar0++gr0] with activate data;
rep N with ram - afifo;
rep N [ar6++gr6] = afifo;

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.316 Vec_Mul2D2W1_AddVr

Функции

- void vec_Mul2D2W1_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.316.1 Подробное описание

\~

\~russian Ядро функции nmppsConvert_64s(nm64s*,nm32s*,int).
\~english nmppsConvert_64s(nm64s*,nm32s*,int) function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep 1 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 1 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение в 1 строку

Аргументы

vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на 2 матрицы весовых коэффициентов по 2 строки в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6,gr7.

5.317 Vec_Mul2D2W2_AddVr

Функции

- void vec_Mul2D2W2_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.317.1 Подробное описание

\~

\~russian Ядро функции nmppsConvert_32s(nm32s*,nm16s*,int).
\~english nmppsConvert_32s(nm32s*,nm16s*,int) function core.

\~\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~english The function operation is equivalent to the following pseudoinstructions:
\~\~

```
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 2 строки
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на 2 матрицы весовых коэффициентов по 2 строки в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6,gr7.

5.318 Vec_Mul2D2W4_AddVr

Функции

- void vec_Mul2D2W4_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.318.1 Подробное описание

\~

\~ russian Ядро функции nmppsConvert_16s(nm16s* pSrcVec, nm8s* pDstVec, int nSize).
\~ english nmppsConvert_16s(nm16s* pSrcVec, nm8s* pDstVec, int nSize) function core.

\~\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~ english The function operation is equivalent to the following pseudoinstructions:
\~\~

```
rep 4 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 4 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 4 строки
f1cr	задает функцию активации
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на 2 матрицы весовых коэффициентов по 4 строки в каждой
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6,gr7.

5.319 Vec_Mul2D2W8_AddVr

Функции

- void vec_Mul2D2W8_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.319.1 Подробное описание

\~

\~russian Применяется в MTR_Copyua().
\~english It is used in MTR_Copyua().

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на 2 матрицы весовых коэффициентов по 8 строк в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6,gr7.

5.320 Vec_Mul3D3W2_AddVr

Функции

- void vec_Mul3D3W2_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar2, **nmreg** gr2, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.320.1 Подробное описание

\~

\~russian
\~english\~\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~english The function operation is equivalent to the following pseudoinstructions:
\~\~

```

rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar2++gr2] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;

```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 2 строки
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar2	указатель на столбец SrcMtr3
gr2	SrcMtr3 stride
ar4	указатель на 3 матрицы весовых коэффициентов по 2 строки в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar6,gr7.

5.321 Vec_Mul3D3W8_AddVr

Функции

- void vec_Mul3D3W8_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar2, **nmreg** gr2, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.321.1 Подробное описание

\~

\~
\~russian
\~english

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep 8 wfifo=[ar4++],ftw,wtw;
rep N data =[ar2++gr2] with vsum ,data,afifo;
rep N [ar6++ gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar2	указатель на столбец SrcMtr3
gr2	SrcMtr3 stride
ar4	указатель на 3 матрицы весовых коэффициентов по 8 строк в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar6,gr7.

5.322 Vec_Mul4D4W2_AddVr

Функции

- void vec_Mul4D4W2_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar2, **nmreg** gr2, **nmreg** ar3, **nmreg** gr3, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.322.1 Подробное описание

\~

\~ russian Ядро функции nmppsConvert_32s(nm32s* pSrcVec, nm8s* pDstVec, int nSize).
\~ english nmppsConvert_32s(nm32s* pSrcVec, nm8s* pDstVec, int nSize) function core.

\~\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~ english The function operation is equivalent to the following pseudoinstructions:
\~\~

```

rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar0++gr0] with vsum ,data,vr;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar1++gr1] with vsum ,data,afifo;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar2++gr2] with vsum ,data,afifo;
rep 2 wfifo=[ar4++],ftw,wtw;
rep N data =[ar3++gr3] with vsum ,data,afifo;
rep N [ar6++gr6]=afifo;

```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 2 строки
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar2	указатель на столбец SrcMtr3
gr2	SrcMtr3 stride
ar3	указатель на столбец SrcMtr4
gr3	SrcMtr4 stride
ar4	указатель на 4 матрицы весовых коэффициентов по 2 строки в каждой
gr4	дублирует nb1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar3,ar6,gr7.

5.323 Vec_MulVN_AddVN

Функции

- void vec_MulVN_AddVN (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.323.1 Подробное описание

\~

\~russian Ядро функции MTR_MulC_AddVsVc().
\~english MTR_MulC_AddVsVc() function core.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:

\~

\~

```
rep N data=[ar0+++gr0] with vsum ,data, vr;
rep N data=[ar1+++gr1] with afifo+data;
rep N [ar6+++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.324 Vec_Sub

Функции

- void vec_Sub (**nmreg** nb1, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.324.1 Подробное описание

\~

\~russian Ядро функции nmppsSub().
\~english nmppsSub() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with data;
rep N data=[ar1++gr1] with data - afifo;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.325 Vec_SubAbs

Функции

- void vec_SubAbs (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.325.1 Подробное описание

\~

\~russian Ядро функции nmppsAbsDiff().
\~english nmppsAbsDiff() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0]  with data
rep N data=[ar1++gr1]  with afifo-data;
rep N [ar4],ram =afifo with activate afifo;
rep N           with vsum afifo,ram,ram;
rep N [ar6++gr6]= afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на временный буфер (1 64р. слово)
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.326 Vec_SubVN_Abs

Функции

- void vec_SubVN_Abs (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.326.1 Подробное описание

\~

\~russian Ядро функции mtr_SubMV_Abs().
\~english mtr_SubMV_Abs() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N ram=[ar1]
rep N data=[ar0++gr0]    with data-ram;
rep N [ar2++]={afifo}    with activate afifo;
// ar2,ar5- internal pointers to temporary buffer size of long[32]
rep N data=[ar5++]      with vsum afifo,data,data;
rep N [ar6++gr6]={afifo};
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
ar1	указатель на маску (1 64р. слово)
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.327 Vec_Swap

Функции

- void vec_Swap (**nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.327.1 Подробное описание

```
\`russian Ядро функции mtr_SubVN_Abs().
\`english mtr_SubVN_Abs() function core.
\`
```

Функция осуществляет два одновременных копирования:

[ar0++gr0] => [ar4++gr4]
[ar1++gr1] => [ar6++gr6]

если ar6=ar0, gr6=gr0, ar4=ar1, gr4=gr1

то выполняется перестановка двух векторов

Действие функции эквивалентно следующим псевдоинструкциям:

```
rep N ram =[ar0++gr0];
rep N data=[ar1++gr1]  with data;
rep N [ar6++gr6]=afifo with ram;
rep N [ar4++gr4]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar4	указатель на столбец DstMtr1
gr4	DstVec1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr2
gr6	DstVec2 stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, ar1, ar4, ar6.

5.328 Vec_MUL_2V4toW8_shift

Функции

- void vec_MUL_2V4toW8_shift ([nmreg](#) nb1, [nmreg](#) sb, [nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) ar1, [nmreg](#) gr1, [nmreg](#) ar3, [nmreg](#) gr4, [nmreg](#) ar5, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.328.1 Подробное описание

\~

\~russian Ядро функции SIG_ResizeDown2(nm8u7b* pSrcVec, nm8u7b* pDstVec, int nSize).
\~english SIG_ResizeDown2(nm8u7b* pSrcVec, nm8u7b* pDstVec, int nSize) function core.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:

\~english The function operation is equivalent to the following pseudoinstructions:

\~

```
rep N ram=[ar3];
rep 8 wfifo=[ar5++],ftw,wtw;
rep N data =[ar0++gr0],ftw,wtw with vsum ,data,0;
rep N data =[ar1++gr1]      with vsum ,data,afifo;
rep N           with mask ram,shift afifo,0;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 4 строки
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar3	указатель на 64р. маску
gr4	дублирует nb1
ar5	указатель на матрицу весовых коэффициентов (16 64р. слов)
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar4,ar6,gr2,gr5.

5.329 Vec_MUL_2V8toW16_shift

Функции

- void vec_MUL_2V8toW16_shift (**nmreg** nb1, **nmreg** sb, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar3, **nmreg** gr4, **nmreg** ar5, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.329.1 Подробное описание

\~

\~russian Ядро функции SIG_ResizeDown2(nm16u15b* pSrcVec, nm16u15b* pDstVec, int nSize).
\~english SIG_ResizeDown2(nm16u15b* pSrcVec, nm16u15b* pDstVec, int nSize) function core.

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:

\~english The function operation is equivalent to the following pseudoinstructions:

\~

```
rep N ram=[ar3];
rep 16 wfifo=[ar5++],ftw,wtw;
rep N data =[ar0++gr0],ftw,wtw with vsum ,data,0;
rep N data =[ar1++gr1]      with vsum ,data,afifo;
rep N           with mask ram,shift afifo,0;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки
sb	задает разбиение на 2 строки
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
ar3	указатель на 64р. маску
gr4	дублирует nb1
ar5	указатель на матрицу весовых коэффициентов (16 64р. слов)
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar2,ar4,ar6,gr2,gr5.

5.330 Vec_not_data

Функции

- void vec_not_data (**nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.330.1 Подробное описание

\~

\~russian Ядро функции nmppsNot_().
\~english nmppsNot_() function core.

\~
\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with not data;
rep N [ar6++gr6]=afifo;
```

Аргументы

ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar6.

5.331 Vec_ram

Функции

- void vec_ram ([nmreg](#) ar0, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.331.1 Подробное описание

\~

\~russian Ядро функции nmppsSet_().
\~english nmppsSet_() function core.

\~
\~russian Функция служит для заполнения массива константой.
\~english the function serves for filling an array with a constant.
\~

Действие функции эквивалентно следующим псевдоинструкциям:
rep N ram=[ar0] with data;
rep N [ar6++gr6]=afifo;

Аргументы

ar0	указатель на столбец SrcMtr1
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.332 Vec_ram_sub_data

Функции

- void vec_ram_sub_data ([nmreg nb1](#), [nmreg ar0](#), [nmreg gr0](#), [nmreg gr5](#), [nmreg ar6](#), [nmreg gr6](#))

5.332.1 Подробное описание

\~

\~ russian Ядро функции nmppsSubCRev().
\~ english nmppsSubCRev() function core.

\~\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~\~ english The function operation is equivalent to the following pseudoinstructions:
\~\~

```
rep N ram=[ar1];
rep N data=[ar0++gr0] with ram-data;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.333 Vec_vsum_activate_data_0

Функции

- void vec_vsum_activate_data_0 (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.333.1 Подробное описание

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with vsum ,activate,0;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1	указатель на столбец SrcMtr2
gr1	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.334 Vec_vsum_data_0

Функции

- void vec_vsum_data_0 (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.334.1 Подробное описание

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0+++gr0] with vsum ,data,0;
rep N [ar6+++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.335 Vec_vsum_data_afifo

Функции

- void vec_vsum_data_afifo (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6)

5.335.1 Подробное описание

\~

\~russian Используется в nmppsSum(nm1* pSrcVec, void* pTmpBuf, int nSize)
\~english It is used in nmppsSum(nm1* pSrcVec, void* pTmpBuf, int nSize)

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~

```
rep 1 data=vfalse;
.repeat N;
rep 1 data=[ar0++gr0] with vsum ,data,afifo;
.endrepeat;
rep 1 [ar6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на строки
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0.

5.336 Vec_vsum_data_vr

Функции

- void vec_vsum_data_vr (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.336.1 Подробное описание

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~

```
rep N data=[ar0++gr0] with vsum ,data,vr;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
woper	в рабочей матрице должны быть загружены весовые коэффициенты
vr	константа для суммирования
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.337 Vec_vsum_shift_data_0

Функции

- void vec_vsum_shift_data_0 (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.337.1 Подробное описание

\~

\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~ english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0++gr0] with vsum ,shift data,0;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк

Аргументы

woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.338 Vec_vsum_shift_data_vr

Функции

- void vec_vsum_shift_data_vr (**nmreg** nb1, **nmreg** sb, **nmreg** woper, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.338.1 Подробное описание

\~

\~russian Действие функции эквивалентно следующим псевдоинструкциям:
\~english The function operation is equivalent to the following pseudoinstructions:
\~
\~

```
rep N data=[ar0+++gr0] with vsum ,shift data, vr;
rep N [ar6+++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
woper	в рабочей матрице должны быть загружены весовые коэффициенты
vr	константа для суммирования
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ар6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar6.

5.339 Vec_vsum_shift_data_afifo

Функции

- void vec_vsum_shift_data_afifo (**nmreg** nb1, **nmreg** sb, **nmreg** f1cr, **nmreg** woper, **nmreg** ar0, **nmreg** gr0, **nmreg** gr5, **nmreg** ar6)

5.339.1 Подробное описание

\~

\~ russian Действие функции эквивалентно следующим псевдоинструкциям:
\~ english The function operation is equivalent to the following pseudoinstructions:

```
rep 1  data=vfalse;
rep 1*N data=[ar0++gr0] with vsum , shift data,afifo; (rep1 N times)
rep 1  [ar6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 8 строк
f1cr	задает функцию активации
woper	в рабочей матрице должны быть загружены весовые коэффициенты
ar0	указатель на столбец SrcMtr
gr0	SrcMtr stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0.

5.340 Vec_CompareMinV

Поэлементный поиск минимального

Действие функции эквивалентно следующим псевдоинструкциям:

Функции

- void vec_CompareMinV (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar1, **nmreg** gr1, **nmreg** ar3, **nmreg** gr3, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.340.1 Подробное описание

Поэлементный поиск минимального

Действие функции эквивалентно следующим псевдоинструкциям:

\~

```
rep N ram=[ar0++gr0];
rep N data=[ar1++gr1] with ram-data;
rep N           with activate afifo;
rep N data=[ar3++gr3] with mask afifo, ram, data;
rep N [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr1 stride
ar1,ar3	указатель на столбец SrcMtr2
gr1,gr3	SrcMtr2 stride
gr5	Высота матриц N = [0,1,2...31,32,33,...]
ar6	указатель на столбец DstMtr
gr6	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar3,ar6.

5.341 Vec_CompareMaxV

Поэлементный поиск максимального

Действие функции эквивалентно следующим псевдоинструкциям:

Функции

- void vec_CompareMaxV ([nmreg](#) nb1, [nmreg](#) f1cr, [nmreg](#) ar0, [nmreg](#) gr0, [nmreg](#) ar1, [nmreg](#) gr1, [nmreg](#) ar3, [nmreg](#) gr3, [nmreg](#) gr5, [nmreg](#) ar6, [nmreg](#) gr6)

5.341.1 Подробное описание

Поэлементный поиск максимального

Действие функции эквивалентно следующим псевдоинструкциям:

\~

```
rep N ram=[ar0++gr0];
rep N data=[ar1++gr1] with ram-data;
rep N           with activate afifo;
rep N data=[ar3++gr3] with mask afifo, data, ram;
rep N [ar6++gr6]=afifo;
```

Аргументы

<code>nb1</code>	задает разбиение на колонки (необходимо wtw)
<code>f1cr</code>	задает функцию активации
<code>ar0</code>	указатель на столбец SrcMtr1
<code>gr0</code>	SrcMtr1 stride
<code>ar1,ar3</code>	указатель на столбец SrcMtr2
<code>gr1,gr3</code>	SrcMtr2 stride
<code>gr5</code>	Высота матриц N = [0,1,2...31,32,33,...]
<code>ar6</code>	указатель на столбец DstMtr
<code>gr6</code>	DstMtr stride

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0,ar1,ar3,ar6.

5.342 Vec_DupValueInVector8

Размножение 8-ми битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

Функции

- void vec_DupValueInVector8 ([nmreg](#) ar1, [nmreg](#) gr1)

5.342.1 Подробное описание

Размножение 8-ми битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

\~

```
gr1 = gr1 and 0FFh;
[ar1] = gr1 + (gr1 << 8) + (gr1 << 16) + (gr1 << 24) +
        (gr1 << 32) + (gr1 << 40) + (gr1 << 48) + (gr1 << 56).
ar1 += 2;
```

Аргументы

ар1	Адрес 64-х битового вектора.
gr1	Значение (8 бит).

Restrictions:

При выходе из функции изменяется содержимое регистров: ar1, gr1.

5.343 Vec_DupValueInVector16

Размножение 16-ти битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

Функции

- void vec_DupValueInVector16 ([nmreg](#) ar1, [nmreg](#) gr1)

5.343.1 Подробное описание

Размножение 16-ти битового значения по всему вектору.

Действие функции эквивалентно следующим псевдоинструкциям:

\~

```
gr1 = gr1 and 0FFFFh;
[ar1] = gr1 + (gr1 << 16) + (gr1 << 32) + (gr1 << 48).
ar1 += 2;
```

Аргументы

ар1	Адрес 64-х битового вектора.
gr1	Значение (8 бит).

Restrictions:

При выходе из функции изменяется содержимое регистров: ar1, gr1.

5.344 Vec_BuildDiagWeights8

Построение диагональной матрицы весовых коэффициентов (8x8).

Функции

- void vec_BuildDiagWeights8 (**nmreg** ar1, **nmreg** gr1)

5.344.1 Подробное описание

Построение диагональной матрицы весовых коэффициентов (8x8).

\~

Аргументы

ар1	Адрес 64-х буфера весовых коэффициентов (8x64 бит).
gr1	Значение (8 бит).

Restrictions:

При выходе из функции изменяется содержимое регистров: ar1, gr1.

5.345 Vec_BuildDiagWeights16

Построение диагональной матрицы весовых коэффициентов (16x16).

Функции

- void vec_BuildDiagWeights16 (**nmreg** ar1, **nmreg** gr1)

5.345.1 Подробное описание

Построение диагональной матрицы весовых коэффициентов (16x16).

\~

Аргументы

ar1	Адрес 64-х буфера весовых коэффициентов (4x64 бит).
gr1	Значение (16 бит).

Restrictions:

При выходе из функции изменяется содержимое регистров: ar1, gr1.

5.346 Vec_MaxVal_v8nm8s

Поиск максимума в 8 байтах

Функции

- void vec_MaxVal_v8nm8s ([nmreg](#) ar0, [nmreg](#) gr7)

5.346.1 Подробное описание

Поиск максимума в 8 байтах

\~

Аргументы

ar0	Адрес 64р. слова
-----	------------------

Возвращаемые значения

gr7	Максимум из 8 байт
-----	--------------------

Restrictions:

При выходе из функции изменяется содержимое регистров: gr0, gr1, gr2, gr3, ar5, gr5, gr7.

5.347 Vec_MaxVal_v4nm16s

Поиск максимума в 4-х 16р. элементах

Функции

- void vec_MaxVal_v4nm16s (**nmreg** ar0, **nmreg** gr7)

5.347.1 Подробное описание

Поиск максимума в 4-х 16р. элементах

\^

Аргументы

ар0	Адрес 64р. слова
-----	------------------

Возвращаемые значения

gr7	Максимум из 8 байт
-----	--------------------

Restrictions:

При выходе из функции изменяется содержимое регистров: gr0, gr1, gr2, gr3, ar5, gr5, gr7.

5.348 Vec_MaxVal

Поиск максимумов в колонках матрицы SrcMtr1.

Функции

- void vec_MaxVal (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar4, **nmreg** gr5, **nmreg** ar6)

5.348.1 Подробное описание

Поиск максимумов в колонках матрицы SrcMtr1.

\^

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr stride
ar4	указатель на временный массив размером nm64s[64]
gr5	Высота матрицы SrcMtr1 N = [32,64,...]

Возвращаемые значения

ар6	указатель на 64р. слово результатов (максимумов)
-----	--

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, gr0, ar4, ar3, ar6, gr7.

5.349 Vec_MinVal_v8nm8s

Поиск минимума в 8 байтах

Функции

- void vec_MinVal_v8nm8s (**nmreg** ar0, **nmreg** gr7)

5.349.1 Подробное описание

Поиск минимума в 8 байтах

\~

Аргументы

ар0	Адрес 64р. слова
-----	------------------

Возвращаемые значения

gr7	Максимум из 8 байт
-----	--------------------

Restrictions:

При выходе из функции изменяется содержимое регистров: gr0, gr1, gr2, gr3, ar5, gr5, gr7.

5.350 Vec_MinVal_v4nm16s

Поиск минимума в 4-х 16р. элементах

Функции

- void vec_MinVal_v4nm16s (**nmreg** ar0, **nmreg** gr7)

5.350.1 Подробное описание

Поиск минимума в 4-х 16р. элементах

\~

Аргументы

arg0	Адрес 64р. слова
------	------------------

Возвращаемые значения

gr7	Максимум из 8 байт
-----	--------------------

Restrictions:

При выходе из функции изменяется содержимое регистров: gr0, gr1, gr2, gr3, ar5, gr5, gr7.

5.351 Vec_MinVal

Поиск минимумов в колонках матрицы SrcMtr1.

Функции

- void vec_MinVal (**nmreg** nb1, **nmreg** f1cr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar4, **nmreg** gr5, **nmreg** ar6)

5.351.1 Подробное описание

Поиск минимумов в колонках матрицы SrcMtr1.

\~

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
f1cr	задает функцию активации
ar0	указатель на столбец SrcMtr1
gr0	SrcMtr stride
ar4	указатель на временный массив размером nm64s[64]
gr5	Высота матрицы SrcMtr1 N = [32,64,...]

Возвращаемые значения

arg6	указатель на 64р. слово результатов (максимумов)
------	--

Restrictions:

При выходе из функции изменяется содержимое регистров: ar0, gr0, ar4, ar3, ar6, gr7.

5.352 Vec_AccMul1D1W32_AddVr

Умножение с накоплением

Действие функции эквивалентно следующим псевдоинструкциям:

Функции

- void vec_AccMul1D1W32_AddVr (**nmreg** nb1, **nmreg** sb, **nmreg** vr, **nmreg** ar0, **nmreg** gr0, **nmreg** ar4, **nmreg** gr4, **nmreg** gr5, **nmreg** ar6, **nmreg** gr6)

5.352.1 Подробное описание

Умножение с накоплением

Действие функции эквивалентно следующим псевдоинструкциям:

```
\^

rep 32 wfifo=[ar4++],ftw,wtw;
rep 32 wfifo data = [ar0++gr0] with vsum ,data,vr;
with gr5--;
with gr5--;
<Loop>
  rep 32 wfifo=[ar4++],ftw,wtw;
  rep 32 data=[ar0++gr0] with vsum ,data,afifo;
if <>0 goto Loop with gr5--;
rep 32 [ar6++gr6]=afifo;
```

Аргументы

nb1	задает разбиение на колонки (необходимо wtw)
sb	задает разбиение на 32 строки
vt	константа для суммирования
arg0	указатель на столбец SrcMtr1
gr0	SrcMtr stride
ar4	матрицы весовых коэффициентов
gr4	дублирует nb1
gr5	кол-во итераций умножений с накоплением

Возвращаемые значения

arg6	указатель на столбец DstMtr, состоящий из 32 длинных слов
------	---

Аргументы

gr6	DstMtr stride
-----	---------------

Глава 6

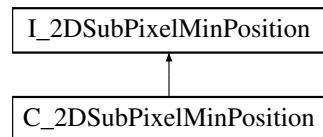
Структуры данных

6.1 Класс C_2DSubPixelMinPosition

Открытые члены

- virtual void [Find](#) (float *S9, float &dx, float &dy)
- virtual void [Release](#) ()

Граф наследования: C_2DSubPixelMinPosition:



6.1.1 Методы

6.1.1.1 Find()

```
virtual void C_2DSubPixelMinPosition::Find (
    float * S9,
    float & dx,
    float & dy )  [virtual]
```

Замещает [I_2DSubPixelMinPosition](#).

6.1.1.2 Release()

```
virtual void C_2DSubPixelMinPosition::Release ( ) [virtual]
```

Замещает [I_2DSubPixelMinPosition](#).

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/isubpixel2dimpl.h

6.2 Класс C_2DTrigSubPixelMinPosition

Открытые члены

- virtual void [Find](#) (float *S9, float &dx, float &dy)
- virtual void [Release](#) ()

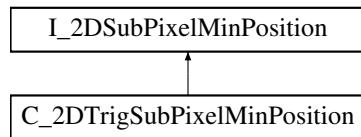
Защищенные члены

- float S9Interpolation (float x, float y, float *S9)

Защищенные данные

- float Teta [8]

Граф наследования: C_2DTrigSubPixelMinPosition:



6.2.1 Методы

6.2.1.1 Find()

```
virtual void C_2DTrigSubPixelMinPosition::Find (
    float * S9,
    float & dx,
    float & dy ) [virtual]
```

Замещает [I_2DSubPixelMinPosition](#).

6.2.1.2 Release()

```
virtual void C_2DTrigSubPixelMinPosition::Release ( ) [virtual]
```

Замещает [I_2DSubPixelMinPosition](#).

Объявления и описания членов класса находятся в файле:

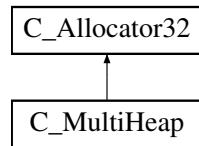
- g:/nmpp/include/nmpli/isubpixel2dimpl.h

6.3 Класс C_Allocator32

Открытые члены

- void * Allocate ()
- int Release ()

Граф наследования: C_Allocator32:



Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/multiheap.h

6.4 Шаблон класса C_BoxImg< T >

Открытые члены

- C_BoxImg ([C_MultiHeap](#) &MultiHeap, int Width, int Height, int BorderHeight=0, int FillMode=BOX_nmppiFILL_FF)
- void Lock ()
- void Unlock ()
- void Fill (int FillMode)
- C_BoxImg (T *Data, int Width, int Height, int BorderHeight=0)
- void Init (T *Data, int Width, int Height, int BorderHeight=0)
- T * Addr (int y, int x)
- T * Allocate ([C_MultiHeap](#) &MultiHeap, int Width, int Height, int BorderHeight=0)
- T * Allocate ([C_MultiHeap](#) &MultiHeap)
- int Release ()
- __INLINE__ T * operator[] (int idx)

Поля данных

- int nWidth
- int nHeight
- int sizeBox
- int sizeData
- int nBorder
- T * pBox
- T * pData
- [C_MultiHeap](#) * pHeap

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/multiheap.h

6.5 Шаблон класса C_BoxVec< T >

Открытые члены

- C_BoxVec (T *Data, int SizeData, int SizeBorder=0)
- C_BoxVec ([C_MultiHeap](#) &MultiHeap, int SizeData, int Border=0)
- T * Addr (int idx)
- int Assign (T *Data, int SizeData, int SizeBorder=0)
- T * Allocate ([C_MultiHeap](#) &MultiHeap, int SizeData, int Border=0)
- T * Allocate ([C_MultiHeap](#) &MultiHeap)
- int Release ()

Поля данных

- int sizeData
- int sizeBox
- int nBorder
- T * pBox
- T * pData
- [C_MultiHeap](#) * pHeap

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/multiheap.h

6.6 Класс C_Heap

класс - куча

```
#include <multiheap.h>
```

Открытые члены

- `C_Heap (void *addrHeap, size_t32 size32Heap)`
конструктор - создает кучу в указанной памяти
- `void Create (void *addrHeap, size_t32 size32Heap)`
создает кучу в указанной памяти
- `int IsMine (void *addr)`
устанавливает принадлежность к куче
- `size_t32 AllocateMaxAvail ()`
Возвращает объем свободной памяти в пуле в 32р. словах
- `int * Allocate (size_t32 size32Buffer)`
Выделяет буфер в куче
- `int ReleaseBuffer (S_BufferInfo *pDelBuffer)`
удаляет структуру буфера из списка
- `int Release (void *p)`
особождает память по адресу
- `void Lock (void *p)`
блокирует указатель от удаления через Release
- `void LockAll ()`
блокирует все указатели от удаления через Release
- `void UnlockAll ()`
разблокирует все указатели для удаления через Release
- `void Unlock (void *p)`
разблокирует все указатели для удаления через Release
- `void ReleaseAll ()`
удаляет все указатели из кучи
- `void LockHeap ()`
Запрещает операции с кучей
- `void UnlockHeap ()`
Разрешает операции с кучей
- `int Check ()`

Поля данных

- `S_BufferInfo * pZeroBuffer`
< указатель на нулевой буфер в списке (с нулевым размером)
- `int * pHeapEnd`
< указатель на слово следующее за концом кучи
- `int size32HeapAvail`
< размер общей свободной памяти в куче
- `bool isHeapLocked`
< запрещает операции с кучей
- `int status`

6.6.1 Подробное описание

класс - куча

6.6.2 Методы

6.6.2.1 AllocateMaxAvail()

```
size_t32 C_Heap::AllocateMaxAvail() [inline]
```

Возвращает объем свободной памяти в пуле в 32р. словах

Возвращает максимальный размер буфера в 32р. словах, который можно выделить в куче

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/multiheap.h

6.7 Шаблон класса C_Img< T >

```
#include <iSupport.h>
```

Открытые члены

- C_Img (int nWidth, int nHeight, int nStride, int nBorder, void *(*allocator32)(int))
- C_Img (T *pData, int nWidth, int nHeight, int nStride, int nBorder)
- void Fill (T color)

Поля данных

- int m_nBorder
- T * m_pContainer
- T * m_pData
- int m_nWidth
- int m_nHeight
- int m_nStride
- int m_nSize

6.7.1 Подробное описание

```
template<class T>
class C_Img< T >
```

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iSupport.h

6.8 Класс C_MultiHeap

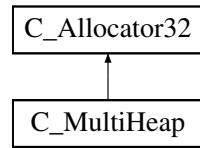
Открытые члены

- `C_MultiHeap (int Mode=ALLOCATE_FORWARD)`
- `int Error ()`
- `void Mode (int mode, void **legend=0)`
- `unsigned Rand ()`
Генератор случайных чисел
- `unsigned Rand (unsigned min, unsigned max)`
- `C_Heap & operator[] (int idxHeap)`
- `int CreateHeap (void *addrHeap, size_t32 size32Heap)`
создает кучу по адресу указанного размера (полный размер со служебными данными)
- `void * Allocate (size_t32 size32Buffer)`
обходит кучи в заданном в AllocateMode порядке и выделяет память заданного размера
- `void * Allocate (size_t32 size32Buffer, int nPriorHeap0, int nPriorHeap1=-1, int nPriorHeap2=-1, int nPriorHeap3=-1, int nPriorHeap4=-1, int nPriorHeap5=-1)`
обходит кучи в заданном порядке и выделяет память заданного размера
- `void * AllocateWith (size_t32 size32Buffer, void *addrInTheSameHeap)`
выделяет массив в той же куче где и указатель
- `int Which (void *addr)`
Возвращает номер кучи к которой принадлежит адрес
- `void Lock (void *addr)`
- `int Unlock (void *addr)`
- `int LockAll ()`
- `int UnlockAll ()`
- `int Release (void *addr)`
- `void ReleaseAll ()`
Удаляет все незаблокированные указатели из куч
- `void LockHeap (int idxHeap)`
Запрещает операции Allocate и Release с кучей
- `void UnlockHeap (int idxHeap)`
Разрешает операции Allocate и Release с кучей
- `int Check ()`

Поля данных

- `C_Heap pHeap [MAX_NUM_BANKS]`
массив куч
- `unsigned numHeaps`
число проинициализированных куч
- `unsigned numAllocateFails`
число ошибок выделения куч
- `unsigned AllocateMode`
порядок обхода куч при поиске свободного места
- `void ** pAllocateLegend`
история номеров куч использованных в последних 8 Allocate
- `unsigned idxAllocateLegend`
- `long long allocateHistory`

Граф наследования: C_MultiHeap:



Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/multiheap.h

6.9 Класс C_PlessyCornerDetector

Открытые члены

- virtual void **Allocate** (int w, int h, int ww)
- virtual void **DeAllocate** ()
- virtual void **FindCorners** (unsigned char *Picture, int w, int h, int ww, float *px, float *py, int &nc)
- virtual void **Release** ()
- virtual void **SetThreshold** (float _threshold)=0

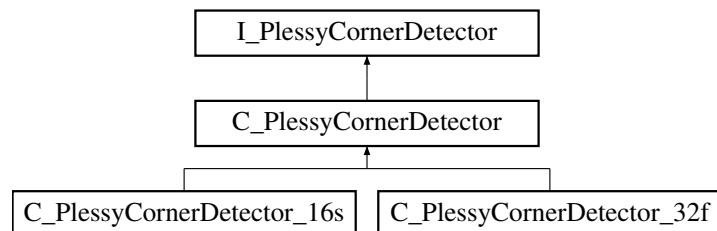
Защищенные члены

- virtual void CountDer (unsigned char *Picture, int w, int h, int ww)
- virtual void CountPlessy (int w, int h, int ww)

Защищенные данные

- **I_2DSubPixelMinPosition** * SubPixelMinPosition
- short * fxi
- short * fyi
- short * Picture16
- float * SumSxxSyy
- float * Subxy
- float S9 [9]
- float threshold
- unsigned char * cres

Граф наследования: C_PlessyCornerDetector:



6.9.1 Методы

6.9.1.1 Allocate()

```
virtual void C_PlessyCornerDetector::Allocate (
    int w,
    int h,
    int ww ) [virtual]
```

Замещает [I_PlessyCornerDetector](#).

6.9.1.2 DeAllocate()

```
virtual void C_PlessyCornerDetector::DeAllocate ( ) [virtual]
```

Замещает [I_PlessyCornerDetector](#).

6.9.1.3 FindCorners()

```
virtual void C_PlessyCornerDetector::FindCorners (
    unsigned char * Picture,
    int w,
    int h,
    int ww,
    float * px,
    float * py,
    int & nc ) [virtual]
```

Замещает [I_PlessyCornerDetector](#).

6.9.1.4 Release()

```
virtual void C_PlessyCornerDetector::Release ( ) [virtual]
```

Замещает [I_PlessyCornerDetector](#).

6.9.1.5 SetThreshold()

```
virtual void C_PlessyCornerDetector::SetThreshold (
    float _threshold ) [pure virtual]
```

Замещает [I_PlessyCornerDetector](#).

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iPlessyDetector.h

6.10 Класс C_PlessyCornerDetector_16s

Открытые члены

- virtual void [Allocate](#) (int w, int h, int ww)
- virtual void [DeAllocate](#) ()
- virtual void [SetThreshold](#) (float _threshold)

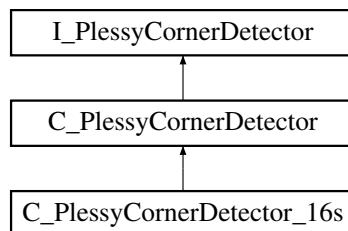
Защищенные члены

- virtual void [CountPlessy](#) (int w, int h, int ww)

Защищенные данные

- short * sxxi
- short * sxyi
- short * syyi
- short * Sxxi
- short * Sxyi
- short * Syyi
- short * Sxxit
- short * Syyit
- short * SumSxxSyyi
- short * MulSxxSyyi
- short * MulSxySxyi
- short * Subxxyi
- short * cSubxxyi
- short threshold

Граф наследования: C_PlessyCornerDetector_16s:



6.10.1 Методы

6.10.1.1 Allocate()

```
virtual void C_PlessyCornerDetector_16s::Allocate (
    int w,
    int h,
    int ww ) [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.10.1.2 CountPlessy()

```
virtual void C_PlessyCornerDetector_16s::CountPlessy (
    int w,
    int h,
    int ww ) [protected], [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.10.1.3 DeAllocate()

```
virtual void C_PlessyCornerDetector_16s::DeAllocate () [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.10.1.4 SetThreshold()

```
virtual void C_PlessyCornerDetector_16s::SetThreshold (
    float _threshold ) [virtual]
```

Замещает [C_PlessyCornerDetector](#).

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iPlessyDetector.h

6.11 Класс C_PlessyCornerDetector_32f

Открытые члены

- virtual void [Allocate](#) (int w, int h, int ww)
- virtual void [DeAllocate](#) ()
- virtual void [SetThreshold](#) (float _threshold)

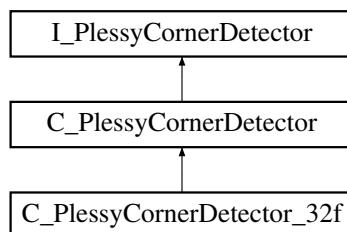
Защищенные члены

- virtual void [CountPlessy](#) (int w, int h, int ww)
- virtual void [CountDer](#) (unsigned char *Picture, int w, int h, int ww)

Защищенные данные

- float * fx
- float * fy
- float * sxx
- float * sxy
- float * syy
- float * Sxx
- float * Sxy
- float * Syy
- float * MulSxxSyy
- float * MulSxySxy
- float * cSubxy
- float threshold

Граф наследования: C_PlessyCornerDetector_32f:



6.11.1 Методы

6.11.1.1 Allocate()

```
virtual void C_PlessyCornerDetector_32f::Allocate (
    int w,
    int h,
    int ww ) [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.11.1.2 CountDer()

```
virtual void C_PlessyCornerDetector_32f::CountDer (
    unsigned char * Picture,
    int w,
    int h,
    int ww ) [protected], [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.11.1.3 CountPlessy()

```
virtual void C_PlessyCornerDetector_32f::CountPlessy (
    int w,
    int h,
    int ww ) [protected], [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.11.1.4 DeAllocate()

```
virtual void C_PlessyCornerDetector_32f::DeAllocate ( ) [virtual]
```

Переопределяет метод предка [C_PlessyCornerDetector](#).

6.11.1.5 SetThreshold()

```
virtual void C_PlessyCornerDetector_32f::SetThreshold (
    float _threshold ) [virtual]
```

Замещает [C_PlessyCornerDetector](#).

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iPlessyDetector.h

6.12 Шаблон класса C_RingBuffer< T >

Открытые члены

- void Sleep (clock_t msec)
- C_RingBuffer (T *buffer, size_t count, t_bytcpy pushmemcpy, t_bytcpy popmemcpy)
- bool Init (T *buffer, size_t count, t_bytcpy pushmemcpy, t_bytcpy popmemcpy)
- INLINE bool IsFull ()
- INLINE bool IsEmpty ()
- INLINE size_t PushAvail ()
- INLINE size_t PopAvail ()
- T * Head ()
- T * Tail ()
- bool MoveHead (int numElements)
- bool MoveTail (int numElements)
- size_t Push (T *pSrcElements, size_t numElements, int ExitMode=EXIT_ON_COMPLETED)
- size_t Push (T pSrcElement)
- void PushRequest (T *pSrcElements, size_t numElements)
- bool isPushCompleted ()
- size_t Pop (T *pDstElements, size_t numElements, int ExitMode=EXIT_ON_COMPLETED)

Поля данных

- T * data
 - физический адрес кольцевого буфера входных данных
- size_t size
 - размер кольцевого буфера входных данных (в элементах; гарантируется что это степень двойки)
- size_t head
 - сколько элементов ОТ НАЧАЛА ПОТОКА код MASTER уже записал в буфер входных данных [заполняется MASTER]
- size_t tail
 - сколько элементов ОТ НАЧАЛА ПОТОКА код SLAVE уже прочитал (обработал) [заполняется SLAVE]
- size_t * head_addr
- size_t * tail_addr
- size_t id
- t_bytcpy push_memcpy
- t_bytcpy pop_memcpy
- memcpy_ptr dma_init
- size_t(* dma_check)()
- T * dma_ptr
- size_t dma_left
- size_t dma_size
- unsigned timeout
- unsigned time2sleep
- int pad [16-3-5 *sizeof(memcpy_ptr)/sizeof(int)]
- резервные поля

6.12.1 Методы

6.12.1.1 PushRequest()

```
template<class T>
void C_RingBuffer< T >::PushRequest (
    T * pSrcElements,
    size_t numElements ) [inline]
```

!!! Head or head ??

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/ringbuffer_.h

6.13 Шаблон класса C_RingBufferRemote< T >

Открытые члены

- C_RingBufferRemote (size_t ringbuffer_addr, t_bytcpy push_memcopy, t_bytcpy pop_↔ memcopy)
- size_t GetHead ()
- size_t GetTail ()
- void SetHead ()
- void SetTail ()
- bool Init (size_t ringbuffer_addr, t_bytcpy push_memcopy, t_bytcpy pop_memcopy)
- __INLINE__ bool IsFull ()
- __INLINE__ bool IsEmpty ()
- __INLINE__ size_t GetWriteAvail ()
- __INLINE__ size_t GetReadAvail ()
- bool Push (int numElements)
- bool Pop (int numElements)
- size_t Push (T *pSrcElements, size_t numElements, int ExitMode=EXIT_ON_COMPLETED)
- size_t Pop (T *pDstElements, size_t numElements, int ExitMode=EXIT_ON_COMPLETED)
- size_t View (T *pDstElements, size_t numElements, int ExitMode=EXIT_ON_COMPLETED)

Поля данных

- size_t data_addr
физический адрес кольцевого буфера входных данных
- size_t head_addr
сколько элементов ОТ НАЧАЛА ПОТОКА код MASTER уже записал в буфер входных данных
[заполняется MASTER]
- size_t tail_addr
сколько элементов ОТ НАЧАЛА ПОТОКА код SLAVE уже прочитал (обработал) [заполняется SLAVE]
- size_t size
размер кольцевого буфера входных данных (в элементах; гарантируется что это степень двойки)
- size_t head
сколько элементов ОТ НАЧАЛА ПОТОКА код MASTER уже записал в буфер входных данных
[заполняется MASTER]
- size_t tail

- сколько элементов ОТ НАЧАЛА ПОТОКА код SLAVE уже прочитал (обработал) [заполняется SLAVE]
- size_t id
 - сколько элементов ОТ НАЧАЛА ПОТОКА код SLAVE уже прочитал (обработал) [заполняется SLAVE]
 - bool isConnected
 - t_bytcpy push_memcpy
 - t_bytcpy pop_memcpy
 - t_memcpy dma_init
 - size_t(* dma_check)()
 - T *dma_ptr
 - size_t dma_left
 - size_t dma_size
 - unsigned timeout
 - unsigned time2sleep
 - int pad [16-3-5 *sizeof(t_bytcpy)/sizeof(int)]
- резервные поля

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/ringremote.h

6.14 Класс Cplx_float

Поля данных

- double Re
- double Im

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpls/fftext.h

6.15 Структура ds_struct

Поля данных

- int nnSpot
 - общее число отбракованных пятен (по данному признаку)
- int nnPxl
 - суммарное число пикселов в отбракованных пятнах
- int dttSpot
 - общее время обработки отбракованных пятен (в тактах процессора)

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iFloodFill.h

6.16 Класс EnterHardMode

Объявления и описания членов класса находятся в файле:

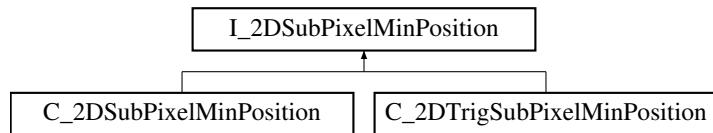
- g:/nmpp/include/macros_fpu.h

6.17 Класс I_2DSubPixelMinPosition

Открытые члены

- virtual void Find (float *S9, float &dx, float &dy)=0
- virtual void Release ()=0

Граф наследования:I_2DSubPixelMinPosition:



Объявления и описания членов класса находятся в файле:

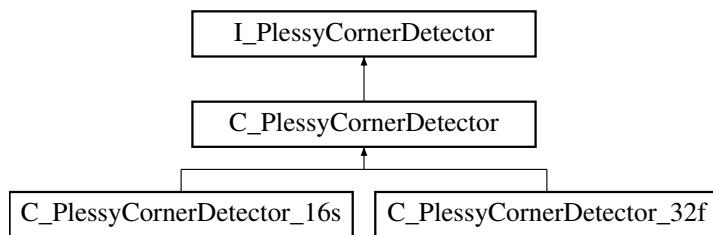
- g:/nmpp/include/nmpli/isubpixel2d.h

6.18 Класс I_PlessyCornerDetector

Открытые члены

- virtual void Allocate (int w, int h, int ww)=0
- virtual void DeAllocate ()=0
- virtual void FindCorners (unsigned char *Picture, int w, int h, int ww, float *px, float *py, int &nc)=0
- virtual void Release ()=0
- virtual void SetThreshold (float _threshold)=0

Граф наследования:I_PlessyCornerDetector:



Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iPlessy.h

6.19 Структура int15in16x4

Поля данных

- signed short item [4]

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.20 Структура int30in32x2

Поля данных

- int item [2]

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.21 Структура int31in32x2

Поля данных

- int item [2]

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.22 Шаблон класса mtr< T >

Открытые члены

- void origin (int y, int x)
- void setval (int y, int x, const T &val)
- void setvalx (int y, int x, const T &val)
- T getval (int y, int x) const
- T getvalx (int y, int x) const
- mtr (int nHeight, int nWidth, int nBorder=0)
- void resize (int nHeight, int nWidth, int nBorder=0)
- mtr (const mtr< T > &matr)
- void assign (T *Data, int nHeight, int nWidth, int nStride=0)
- mtr (T *Data, int nHeight, int nWidth, int nStride=0)
- mtr< T > & operator= (mtr< T > &matr)

- `mtr< T > & operator= (T &val)`
- `T * operator[] (int row) const`
- `T & index (int idx)`
- `mtr< T > & operator*= (const T val)`
- `mtr< T > operator* (const T &val) const`
- `vec< T > operator* (const vec< T > &vect)`
- `mtr< T > operator* (const mtr< T > &matr)`
- `mtr< T > & operator+= (const T &val)`
- `mtr< T > & operator+= (const mtr< T > &matr)`
- `mtr< T > operator+ (const mtr< T > &matr) const`
- `mtr< T > & operator-= (const mtr< T > &matr)`
- `mtr< T > & operator-= (const T &val)`
- `mtr< T > operator- (const mtr< T > &matr) const`
- `mtr< T > operator- () const`
- `mtr< T > & operator/= (const T val)`
- `mtr< T > operator/ (const T val) const`
- `mtr< T > & operator>>= (const int shr)`
- `mtr< T > operator>> (const int shr) const`
- `mtr< T > & operator<<= (const int shl)`
- `mtr< T > operator<< (const int shl) const`
- `mtr< T > & operator&= (const T &val)`
- `mtr< T > & operator&= (const mtr< T > &matr)`
- `mtr< T > operator& (const mtr< T > &matr) const`
- `mtr< T > & operator|= (const mtr< T > &matr)`
- `mtr< T > operator| (const mtr< T > &matr) const`
- `mtr< T > & operator^= (const mtr< T > &matr)`
- `mtr< T > & operator^= (const T &val)`
- `mtr< T > operator^ (const mtr< T > &matr) const`
- `mtr< T > operator^ (const T &val) const`
- `mtr< T > operator~ ()`
- `void set (const T val)`
- `mtr< T > transpose ()`
- `mtr< T > & diag (T val)`
- `T * addr (int y, int x)`
- `void reset ()`
- `int sum ()`
- `vec< T > getvec (int y) const`
- `vec< T > getcol (int x) const`
- `T minpos (int &ypos, int &xpos)`
- `T maxpos (int &ypos, int &xpos)`
- `void CopyTo (T *pData)`
- `void CopyFrom (T *pData)`
- `template<class T2 >`
`void ConvertTo (T2 *pData)`
- `template<class T2 >`
`void ConvertFrom (T2 *pData)`
- `mtr< double > & operator<<= (const int Shl)`

Поля данных

- `int m_border`
- `int m_stride`
- `int m_height`
- `int m_width`
- `int m_size`
- `int m_x0`
- `int m_y0`
- `T * m_data`

Защищенные данные

- `T * m_container`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tmatrix.h`

6.23 Структура nm16sc

Поля данных

- `signed short r`
- `signed short c`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmtype.h`

6.24 Класс nmchar

Открытые члены

- `nmchar (unsigned int *p, int offset)`
- `nmchar (nmchar &ch)`
- `nmchar & operator= (nmchar ch)`
- `unsigned int operator+ (nmchar &ch)`
- `nmchar & operator&= (unsigned int val)`
- `nmchar & operator|= (unsigned int val)`
- `nmchar & operator= (unsigned int val)`
- `operator unsigned char ()`
- `uint8ptr operator& ()`

Поля данных

- `unsigned int * adr`
- `int idx`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmchar.h`

6.25 Шаблон класса nmchar1D< N >

Открытые члены

- `nmchar` & operator[] (int idx)
- operator `uint8ptr` ()
- `unsigned int * ptr` ()

Поля данных

- `nmchar` deref
- `unsigned int data [(N+3)/4]`

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmchar.h

6.26 Шаблон класса nmchar2D< Y, X >

Открытые члены

- `uint8ptr` & operator[] (int idx)
- `unsigned int * ptr` ()

Поля данных

- `uint8ptr arr`
- `unsigned int data [Y *X/4]`

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmchar.h

6.27 Шаблон класса nmintpack< T >

```
#include <tnmvecpack.h>
```

Открытые члены

- `nmintpack` (T *base, int idx)
- `nmintpack< T > & operator= (const nmintpack< T > &val)`
- `nmintpack< T > & operator= (const int &val)`
- operator `int (void) const`
- `__INLINE__ int intdisp (int indx)`
- `__INLINE__ int bitdisp (int indx)`

Поля данных

- `T * m_container`
- `int m_disp`

6.27.1 Подробное описание

```
template<class T>
class nmintpack< T >
```

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tmvecpack.h`

6.28 Шаблон класса nmmtr< T >

```
#include <tmmtr.h>
```

Открытые члены

- `nmmtr (int Height, int Width, int Border=0)`
- `nmvec< T > operator[] (int y) const`
- `nmmtr (nmmtr< T > &mtr)`
- `nmmtr (T *Data, int Height, int Width, int Stride=0)`
- `nmmtr (const T *Data, int Height, int Width, int Stride=0)`
- `nmmtr< T > & operator= (const nmmtr< T > &mtr)`
- `nmmtr< T > & operator*= (const nmint< T > &val)`
- `template<class T2 >`
- `nmmtr< T2 > operator* (const nmint< T2 > &val)`
- `template<class T2 >`
- `nmvec< T2 > operator* (const nmvec< T2 > &vec)`
- `template<class T2 >`
- `nmmtr< T2 > operator* (const nmmtr< T2 > &mtr)`
- `nmvec< T > & operator+= (const nmint< T > &val)`
- `nmmtr< T > & operator+= (const nmmtr< T > &mtr)`
- `nmmtr< T > operator+ (const nmmtr< T > &mtr) const`
- `nmmtr< T > & operator-= (const nmmtr< T > &mtr)`
- `nmmtr< T > operator- (const nmmtr< T > &mtr) const`
- `nmmtr< T > operator- () const`
- `nmmtr< T > & operator/= (const T val)`
- `nmmtr< T > operator/ (const nmint< T > val) const`
- `nmmtr< T > & operator>>= (const int shr)`
- `nmmtr< T > operator>> (const int shr) const`
- `nmmtr< T > & operator<<= (const int shl)`
- `nmmtr< T > operator<< (const int shl) const`
- `nmmtr< T > & operator|= (const nmmtr< T > &mtr)`
- `nmmtr< T > operator| (const nmmtr< T > &mtr) const`
- `nmmtr< T > & operator&= (const nmmtr< T > &mtr)`
- `nmmtr< T > operator& (const nmmtr< T > &mtr) const`
- `nmmtr< T > & operator^= (const nmint< T > &val)`

- `nmmtr< T > operator^ (const nmint< T > &val) const`
- `nmmtr< T > operator~ () const`
- `T * addr (int y, int x)`
- `nmvec< T > vec (int y)`
- `void fill (nmint< T > &nVal)`
- `nmmtr< T > transpose ()`
- template<class T2 >
 `void set (nmmtr< T2 > &mSrcMtr) const`
- template<class T2 >
 `void set (mtr< T2 > &Mtr)`
- `void set (const T val)`
- `void reset ()`

Поля данных

- `int m_height`
- `int m_width`
- `int m_size`
- `int m_stride`
- `int m_border`
- `T * m_data`

Защищенные данные

- `T * m_container`

6.28.1 Подробное описание

```
template<class T>
class nmmtr< T >
```

класс матриц.

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/ttnmmtr.h`

6.29 Шаблон класса nmmtrpack< T >

```
#include <ttnmmtrpack.h>
```

Открытые члены

- `nmmtrpack (int Height, int Width, int Border=0)`
- `__INLINE__ nmvecpack< T > operator[] (int y) const`
- `nmmtrpack (const nmmtrpack< T > &mtr)`
- `nmmtrpack (T *Data, int Height, int Width, int Stride=0)`
- `nmmtrpack (const T *Data, int Height, int Width, int Stride=0)`
- `nmmtrpack< T > & operator= (const nmmtrpack< T > &mtr)`
- `nmmtrpack< T > & operator*= (const nmint< T > &val)`
- `template<class T2 >`
`nmmtrpack< T2 > operator* (const nmint< T2 > &val)`
- `template<class T2 >`
`nmvec< T2 > operator* (const nmvec< T2 > &vec)`
- `template<class T2 >`
`nmmtrpack< T2 > operator* (const nmmtrpack< T2 > &mtr)`
- `nmvec< T > & operator+= (const nmint< T > &val)`
- `nmmtrpack< T > & operator+= (const nmmtrpack< T > &mtr)`
- `nmmtrpack< T > operator+ (const nmmtrpack< T > &mtr) const`
- `nmmtrpack< T > & operator-= (const nmmtrpack< T > &mtr)`
- `nmmtrpack< T > operator- (const nmmtrpack< T > &mtr) const`
- `nmmtrpack< T > operator- () const`
- `nmmtrpack< T > & operator/= (const T val)`
- `nmmtrpack< T > operator/ (const nmint< T > val) const`
- `nmmtrpack< T > & operator>>= (const int shr)`
- `nmmtrpack< T > operator>> (const int shr) const`
- `nmmtrpack< T > & operator<<= (const int shl)`
- `nmmtrpack< T > operator<< (const int shl) const`
- `nmmtrpack< T > & operator|= (const nmmtrpack< T > &mtr)`
- `nmmtrpack< T > operator| (const nmmtrpack< T > &mtr) const`
- `nmmtrpack< T > & operator&= (const nmmtrpack< T > &mtr)`
- `nmmtrpack< T > operator& (const nmmtrpack< T > &mtr) const`
- `nmmtrpack< T > & operator^= (const nmint< T > &val)`
- `nmmtrpack< T > operator^ (const nmint< T > &val) const`
- `nmmtrpack< T > operator~ () const`
- `__INLINE__ nmvec< T > GetVec (int y)`
- `__INLINE__ nmvec< T > GetVec (int y, int x, int len)`
- `__INLINE__ nmmtrpack< T > & SetMtr (int y, int x, nmmtrpack< T > &mSrc)`
- `__INLINE__ nmmtrpack< T > & GetMtr (int y, int x, nmmtrpack< T > &mRes)`
- `__INLINE__ nmmtrpack< T > GetMtr (int y, int x, int height, int width)`
- `__INLINE__ T * Addr (int y, int x)`
- `template<class T2 >`
`void Set (nmmtrpack< T2 > &mSrcMtr) const`
- `void InitConst (nmint< T > &nVal)`
- `nmmtrpack< T > transpose ()`
- `void reset ()`

Поля данных

- `int m_height`
- `int m_width`
- `int m_size`
- `int m_stride`
- `int m_border`
- `T * m_data`

Защищенные данные

- `T * m_container`
- `bool m_flag_new`

6.29.1 Подробное описание

```
template<class T>
class nmmtrpack< T >
```

класс матриц.

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tmmtrpack.h`

6.30 Структура NmppsFFTSpec

Поля данных

- `nm32sc * buffer [FFT_SPEC_NUM_BUFFERS]`
- `void * fftTable [FFT_SPEC_NUM_TABLES]`
- `RoundShift norm [4]`
- `int shift [FFT_SPEC_NUM_SHIFTS]`
- `int amp [FFT_SPEC_NUM_AMPLITUDES]`
- `int round [8]`
- `Free32Func * free`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmpls/fft.h`

6.31 Структура NmppsFFTSpec_32fcr

Поля данных

- `nm32fcr * Buffers [NUMBUFF1]`
- `nm32fcr * Buffs [NUMBUFF2]`
- `int order`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/fft_32fcr.h`

6.32 Структура NmppsFrame_16s

Поля данных

- `void * pull`
- `nm16s * data`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/malloc32.h`

6.33 Структура NmppsFrame_16u

Поля данных

- `void * pull`
- `nm16u * data`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/malloc32.h`

6.34 Структура NmppsFrame_32s

Поля данных

- `void * pull`
- `nm32s * data`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/malloc32.h`

6.35 Структура NmppsFrame_32u

Поля данных

- `void * pull`
- `nm32u * data`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/malloc32.h`

6.36 Структура NmppsFrame_64s

Поля данных

- void * pull
- **nm64s** * data

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.37 Структура NmppsFrame_64u

Поля данных

- void * pull
- **nm64u** * data

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.38 Структура NmppsFrame_8s

Поля данных

- void * pull
- **nm8s** * data

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.39 Структура NmppsFrame_8u

Поля данных

- void * pull
- **nm8u** * data

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.40 Структура NmppsMallocSpec

Поля данных

- Malloc32Func * allocator [4]
- enum MALLOC32_MODE mode
- uint32 random
- fseq32 priority
- uint32 status
- uint32 time
- uint32 timeBest
- uint32 routePos
- fseq64 route [NMPPS_MALLOC_LIMIT/16]
- fseq64 bestRoute [NMPPS_MALLOC_LIMIT/16]
- void * allocHistory [NMPPS_MALLOC_LIMIT]
- void * freeHistory [NMPPS_MALLOC_LIMIT]
- uint32 allocHistoryPos
- uint32 freeHistoryPos
- uint32 firstPass

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.41 Структура NmppsTmpSpec

Поля данных

- void * buffer0
- void * buffer1

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.42 Структура nmreg

```
#include <nmtpe.h>
```

Поля данных

- int nVal

6.42.1 Подробное описание

\~

NM регистр.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.43 Класс nmshort

Открытые члены

- `__INLINE__ nmshort (nmshort &ch)`
- `__INLINE__ nmshort & operator= (nmshort ch)`
- `__INLINE__ unsigned int operator+ (nmshort &ch)`
- `__INLINE__ nmshort & operator= (unsigned int val)`
- `__INLINE__ operator unsigned char ()`
- `__INLINE__ uint16ptr operator& ()`

Поля данных

- `unsigned int * adr`
- `int idx`

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmshort.h

6.44 Шаблон класса nmshort2D< Y, X >

Открытые члены

- `uint16ptr & operator[] (int idx)`
- `unsigned int * ptr ()`

Поля данных

- `uint16ptr arr`
- `unsigned int data [Y *X/2]`

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmshort.h

6.45 Шаблон класса nmvecpack< T >

```
#include <tnmvecpack.h>
```

Открытые члены

- nmvecpack (void *Data, int Size, int Border=0)
- nmvecpack (int Size, int Border=0)
- nmvecpack (const nmvecpack< T > &vec)
- nmvecpack< T > & operator= (const nmvecpack< T > &vec)
- __INLINE__ nmintpack< T > operator[] (int idx)
- template<class T2 >
nmvecpack< T2 > & operator*= (const nmint< T2 > val)
- template<class T2 >
nmvecpack< T2 > operator* (const nmint< T2 > &val) const
- template<class T2 >
nmint< T2 > operator* (const nmvecpack< T2 > &vec) const
- nmvecpack< T > & operator+= (const nmint< T > &val)
- nmvecpack< T > & operator+= (const nmvecpack< T > &vec)
- nmvecpack< T > operator+ (const nmint< T > &val) const
- nmvecpack< T > operator+ (const nmvecpack< T > &vec) const
- nmvecpack< T > & operator-= (const nmint< T > &val)
- nmvecpack< T > & operator-= (const nmvecpack< T > &vec)
- nmvecpack< T > operator- () const
- nmvecpack< T > operator- (const nmint< T > &val) const
- nmvecpack< T > operator- (const nmvecpack< T > &vec) const
- nmvecpack< T > & operator/= (const nmint< T > val)
- nmvecpack< T > operator/ (const T val) const
- nmvecpack< T > & operator>>= (const int shr)
- nmvecpack< T > operator>> (const int shr) const
- nmvecpack< T > & operator<<= (const int shl)
- nmvecpack< T > operator<< (const int shl) const
- nmvecpack< T > & operator|= (const nmint< T > &val)
- nmvecpack< T > & operator|= (const nmvecpack< T > &vec)
- nmvecpack< T > operator| (const nmint< T > &val) const
- nmvecpack< T > operator| (const nmvecpack< T > &vec) const
- nmvecpack< T > & operator&= (const nmint< T > &val)
- nmvecpack< T > & operator&= (const nmvecpack< T > &vec)
- nmvecpack< T > operator& (const nmint< T > &val) const
- nmvecpack< T > operator& (const nmvecpack< T > &vec) const
- nmvecpack< T > & operator^= (const nmint< T > &val)
- nmvecpack< T > & operator^= (const nmvecpack< T > &vec)
- nmvecpack< T > operator^ (const nmint< T > &val) const
- nmvecpack< T > operator^ (const nmvecpack< T > &vec) const
- nmvecpack< T > & operator~ () const
- bool operator== (const nmvecpack< T > &vec) const
- bool operator!= (const nmvecpack< T > &vec) const
- template<class T2 >
void SetData (T2 *Data)
- template<class T2 >
void GetData (T2 *Data)
- void reset ()

Поля данных

- int m_size
- T * m_data

Защищенные данные

- T * m_container
- int m_border

6.45.1 Подробное описание

```
template<class T>
class nmvecpack< T >
```

Класс векторов.

Примеры:

```
int Test[10]={1,125,3,4,5,6,7,8,9,10};
int Res [10];
nmvecpack<int> A0(3);
nmvecpack<int> B0(3);
nmvecpack<int> C0(3);
```

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmtl/tnmvecpack.h

6.46 Структура RGB24_nm8u

```
#include <iDef.h>
```

Поля данных

- unsigned char nB:8
- unsigned char nG:8
- unsigned char nR:8

6.46.1 Подробное описание

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iDef.h

6.47 Структура RGB32_nm10s

Поля данных

- int nB:10
- int nG:10
- int nR:10
- int nA:2

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iDef.h

6.48 Структура RGB32_nm10u

Поля данных

- unsigned int nB:10
- unsigned int nG:10
- unsigned int nR:10
- unsigned int nA:2

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iDef.h

6.49 Структура RGB32_nm8s

Поля данных

- int nB:8
- int nG:8
- int nR:8
- int nA:8

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iDef.h

6.50 Структура RGB32_nm8u

Поля данных

- unsigned int nB:8
- unsigned int nG:8
- unsigned int nR:8
- unsigned int nA:8

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iDef.h

6.51 Структура RoundShift

Поля данных

- int round
- int shift

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpls/fft.h

6.52 Класс RPoint

```
#include <iCellTexture.h>
```

Открытые члены

- RPoint (const **RPoint** &p)
- RPoint (double _x, double _y)
- **RPoint** & operator= (const **RPoint** &p)

Поля данных

- double x
- double y

6.52.1 Подробное описание

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpli/iCellTexture.h

6.53 Структура S_BufferInfo

класс буфер - заголовок в начале выделяемой динамической памяти

```
#include <multiheap.h>
```

Открытые члены

- int * DataBegin ()
Возвращает указатель на данные в буфере
- int * DataEnd ()
Возвращает указатель на конец данных в буфере (следу)
- int * EndGuardBits ()
Возвращает указатель на конечные защитные поля (2 слова)
- unsigned CheckGuardBits ()

Поля данных

- int guardInfoBits0
- size_t32 size32Buffer
 - < размер массива данных в буфере в 32-р словах
- S_BufferInfo * pPrevBuffer
 - < указатель на предыдущий буфер в списке
- S_BufferInfo * pNextBuffer
 - < указатель на следующий буфер в списке
- bool isLocked
 - < флаг блокировки буфера, запрещающий его удаление с помощью Release()
- int guardInfoBits1

6.53.1 Подробное описание

класс буфер - заголовок в начале выделяемой динамической памяти

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/multiheap.h

6.54 Структура s_int32x2

Поля данных

- int hi
- int lo

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.55 Структура s_nm32fc

Поля данных

- float re
- float im

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.56 Структура s_nm32fcr

Поля данных

- float im
- float re

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.57 Структура s_nm32sc

Поля данных

- nm32s re
- nm32s im

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.58 Структура s_nm64sc

Поля данных

- long long re
- long long im

6.58.1 Поля

6.58.1.1 im

long long s_nm64sc::im

Мнимая часть комплексного числа.

6.58.1.2 re

long long s_nm64sc::re

Вещественная часть комплексоного числа.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.59 Структура S_nmppiFilterKernel

Поля данных

- `nm32s * pDispArray`
- `nm32s * pWeightMatrix`
- `int nKerWidth`
- `int nKerHeight`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmpli/iFilter.h`

6.60 Структура S_nmppiFilterKernel_32s32s

Поля данных

- `nm32s * pDispArray`
- `nm32s * pWeightMatrix`
- `int nKerWidth`
- `int nKerHeight`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmpli/iFilter.h`

6.61 Структура s_v16nm16s

```
#include <nmtype.h>
```

Поля данных

- `unsigned long long vec [4]`

6.61.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 16р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmtype.h`

6.62 Структура s_v16nm16u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [4]

6.62.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 16р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.63 Структура s_v16nm32s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [8]

6.63.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 32р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.64 Структура s_v16nm32u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [8]

6.64.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 32р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.65 Структура s_v16nm4u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.65.1 Подробное описание

Тип векторной структуры, состоящей из 16-ти 4-р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.66 Структура s_v16nm8s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.66.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 8р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.67 Структура s_v16nm8u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.67.1 Подробное описание

\~

Тип векторной структуры, состоящей из 16-ти 8р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.68 Структура s_v2nm32f

Поля данных

- float v0
- float v1

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.69 Структура s_v2nm32s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.69.1 Подробное описание

\~

Тип векторной структуры, состоящей из 2-х 32р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.70 Структура s_v2nm32u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.70.1 Подробное описание

\~

Тип векторной структуры, состоящей из 2-х 32р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.71 Структура s_v4nm16s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.71.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 16р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.72 Структура s_v4nm16u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.72.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 16р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.73 Структура s_v4nm32f

Поля данных

- float vec [4]

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.74 Структура s_v4nm32s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.74.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 32р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.75 Структура s_v4nm32u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.75.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 32р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.76 Структура s_v4nm64f

Поля данных

- double vec [4]

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.77 Структура s_v4nm8u

#include <nmtype.h>

Поля данных

- unsigned long long vec [1]

6.77.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 8р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.78 Структура s_v8nm16s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.78.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 16р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.79 Структура s_v8nm16u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [2]

6.79.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 16р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.80 Структура s_v8nm32s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [4]

6.80.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 32р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.81 Структура s_v8nm32u

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [4]

6.81.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 32р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.82 Структура s_v8nm8s

```
#include <nmtype.h>
```

Поля данных

- unsigned long long vec [1]

6.82.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 8р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.83 Структура s_v8nm8u

```
#include <nmtpe.h>
```

Поля данных

- unsigned long long vec [1]

6.83.1 Подробное описание

\~

Тип векторной структуры, состоящей из 8-ми 8р. чисел без знака.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtpe.h

6.84 Структура SpecTmp1

Поля данных

- void * buffer
- int status
- int mode
- fseq64 route

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/malloc32.h

6.85 Структура spot_struct

Поля данных

- int Xmin
координаты минимального прямоугольника, содержащего пятно
- int Ymin
координаты минимального прямоугольника, содержащего пятно
- int Xmax
координаты минимального прямоугольника, содержащего пятно
- int Ymax
координаты минимального прямоугольника, содержащего пятно
- int noPxl
номер начального пикселя следующего пятна в массиве pixels
- int dtSpot
время обработки пятна (в тактах процессора)

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iFloodFill.h

6.86 Структура tagSegmentInfo

Поля данных

- int xMin
- int yMin
- int xMax
- int yMax
- int N

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmpli/iFloodFill.h

6.87 Шаблон класса tcube< T >

```
#include <tnmcube.h>
```

Открытые члены

- tcube ([tcube< T > &cube](#))
- tcube (int Height, int Width, int Depth, int WStride=0, int CStride=0)
- tcube (T *data, int Height, int Width, int Depth, int WStride=0, int CStride=0)
- [tcube< T > & operator= \(tcube< T > &cube\)](#)
- [tmtr< T > operator\[\] \(int y\) const](#)
- T * data ()
- void printChannel (int ch)
- void fill (T value)

Поля данных

- int m_height
- int m_width
- int m_depth
- int m_ystride
- int m_wstride
- int m_cstride
- T * m_data

Зашщищенные данные

- T * m_container

6.87.1 Подробное описание

```
template<class T>
class tcube< T >
```

класс матриц.

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmtl/tnmcube.h

6.88 Шаблон класса tfixpoint< T, point >

Открытые члены

- INLINE tfixpoint (const int val)
- INLINE tfixpoint (const float val)
- INLINE tfixpoint (const double val)
- INLINE **tfixpoint**< T, point > & operator= (const int val)
- INLINE **tfixpoint**< T, point > & operator= (const float val)
- **tfixpoint**< T, point > & operator= (const double val)
- INLINE **tfixpoint**< T, point > & operator= (const **tfixpoint**< T, point > &val)
- INLINE **tfixpoint**< T, point > & operator-= (const **tfixpoint**< T, point > &val)
- INLINE **tfixpoint**< T, point > & operator+= (const **tfixpoint**< T, point > &val)
- INLINE **tfixpoint**< T, point > operator- (const **tfixpoint**< T, point > &val) const
- INLINE **tfixpoint**< T, point > operator+ (const **tfixpoint**< T, point > &val) const
- INLINE **tfixpoint**< T, point > & operator++ (int)
- INLINE **tfixpoint**< T, point > & operator-- (int)
- template<class T2 , int point2>
 INLINE **tfixpoint**< T, point > & operator*= (const **tfixpoint**< T2, point2 > val)
- INLINE **tfixpoint**< T, point > & operator*=(const int val)
- INLINE **tfixpoint**< T, point > & operator*=(const float val)
- INLINE **tfixpoint**< T, point > & operator*=(const double val)
- template<int point2>
 INLINE **tfixpoint**< T, point > operator* (const **tfixpoint**< T, point2 > val) const
- INLINE **tfixpoint**< T, point > operator* (const int val) const
- INLINE **tfixpoint**< T, point > & operator/= (const **tfixpoint**< T, point > val)
- INLINE **tfixpoint**< T, point > operator/ (const **tfixpoint**< T, point > val) const
- INLINE **tfixpoint**< T, point > & operator>>= (const int y)
- INLINE **tfixpoint**< T, point > operator>> (const int y) const
- INLINE **tfixpoint**< T, point > & operator<<= (const int y)
- INLINE **tfixpoint**< T, point > operator<< (const int n) const
- INLINE **tfixpoint**< T, point > & operator^= (**tfixpoint**< T, point > &val)
- INLINE **tfixpoint**< T, point > operator^ (**tfixpoint**< T, point > &val) const
- INLINE **tfixpoint**< T, point > operator- () const
- bool operator> (const **tfixpoint**< T, point > &y) const
- bool operator>= (const **tfixpoint**< T, point > &y) const
- bool operator< (const **tfixpoint**< T, point > &y) const
- bool operator<= (const **tfixpoint**< T, point > &y) const
- bool operator== (const **tfixpoint**< T, point > &y) const
- bool operator!= (const **tfixpoint**< T, point > &y) const
- float ft ()

Поля данных

- `T m_value`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tfixpoint.h`

6.89 Структура Tmp2BuffSpec

Поля данных

- `void * buffer0`
- `void * buffer1`
- `fseq64 route`
- `int mode`
- `int status`

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmtype.h`

6.90 Шаблон класса tmtr< T >

Открытые члены

- `tmtr (tmtr< T > &mtr)`
- `tmtr (int nHeight, int nWidth, int nStride=0)`
- `tmtr (T *Data, int nHeight, int nWidth, int nStride=0)`
- `T * ref (int y, int x)`
- `T * operator[] (int row) const`
- `void fill (T &value)`

Поля данных

- `int m_stride`
- `int m_height`
- `int m_width`
- `T * m_data`

Защищенные данные

- `T * m_container`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tnmcube.h`

6.91 Класс uint16ptr

Открытые члены

- `__INLINE__ unsigned char * x86addr ()`
- `__INLINE__ uint16ptr (void *p)`
- `__INLINE__ uint16ptr (unsigned short *p)`
- `__INLINE__ uint16ptr (short *p)`
- `__INLINE__ uint16ptr (const uint16ptr &p)`
- `__INLINE__ uint16ptr (unsigned int *p)`
- `__INLINE__ uint16ptr (unsigned int *p, int offset)`
- `__INLINE__ nmshort & operator[] (int idx)`
- `__INLINE__ uint16ptr & operator= (unsigned int *ptr)`
- `__INLINE__ bool operator< (uint16ptr ptr)`
- `__INLINE__ bool operator>= (uint16ptr ptr)`
- `__INLINE__ int operator- (uint16ptr ptr)`
- `__INLINE__ uint16ptr & operator= (const uint16ptr &p)`
- `__INLINE__ unsigned int * ptr ()`
- `__INLINE__ nmshort & operator* ()`
- `__INLINE__ uint16ptr operator+ (int idx)`
- `__INLINE__ uint16ptr operator- (int idx)`
- `__INLINE__ uint16ptr & operator+= (int idx)`
- `__INLINE__ uint16ptr & operator-= (int idx)`
- `__INLINE__ uint16ptr operator++ (int)`
- `__INLINE__ uint16ptr & operator++ ()`
- `__INLINE__ unsigned int operator== (unsigned int N)`
- `__INLINE__ bool operator== (uint16ptr ptr)`

Поля данных

- `unsigned int * addr`
- `int idx`
- `nmshort arref`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmshort.h`

6.92 Класс uint8ptr

Открытые члены

- `unsigned char * x86addr ()`
- `uint8ptr (const void *p)`
- `uint8ptr (const unsigned char *p)`
- `uint8ptr (const char *p)`
- `uint8ptr (const uint8ptr &p)`
- `uint8ptr (unsigned int *p)`
- `uint8ptr (unsigned int *p, int offset)`
- `nmchar & operator[] (int idx)`

- `uint8ptr & operator= (unsigned int *ptr)`
- `bool operator< (uint8ptr ptr)`
- `bool operator> (uint8ptr ptr)`
- `bool operator>= (uint8ptr ptr)`
- `int operator- (uint8ptr ptr)`
- `uint8ptr & operator= (const uint8ptr &p)`
- `unsigned int * ptr ()`
- `nmchar & operator* ()`
- `uint8ptr operator+ (int idx)`
- `uint8ptr operator- (int idx)`
- `uint8ptr & operator+= (int idx)`
- `uint8ptr & operator-= (int idx)`
- `uint8ptr operator++ (int)`
- `uint8ptr & operator++ ()`
- `bool operator== (uint8ptr ptr)`

Поля данных

- `unsigned int * addr`
- `int idx`
- `nmchar arref`

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmchar.h`

6.93 Структура v16nm4s

```
#include <nmtpe.h>
```

Поля данных

- `unsigned long long vec [1]`

6.93.1 Подробное описание

Тип векторной структуры, состоящей из 16-ти 4-р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- `g:/nmpp/include/nmtpe.h`

6.94 Структура v4nm8s

```
#include <nmtpe.h>
```

Поля данных

- unsigned long long vec [1]

6.94.1 Подробное описание

\~

Тип векторной структуры, состоящей из 4-х 8р. чисел со знаком.

Объявления и описания членов структуры находятся в файле:

- g:/nmpp/include/nmtype.h

6.95 Шаблон класса vec< T >

```
#include <tvector.h>
```

Открытые члены

- void resize (int Size, int Border=0)
- vec (T *Data, int Size, int Border=0)
- vec (int Size, int Border=0)
- vec (const vec< T > &vect)
- void reset ()
- vec< T > & InitRamp (T StartValue, T Increment)
- int MaxPos (T &maxval)
- int MinPos (T &minval)
- double Mean ()
- T & CustomMax ()
- vec< T > & operator= (const T &val)
- vec< T > & operator= (const vec< T > &vect)
- T * addr (int idx)
- T & operator[] (size_t idx)
- vec< T > & operator*= (const T &val)
- vec< T > operator* (const T &val) const
- T operator* (const vec< T > &vect) const
- vec< T > operator* (const mtr< T > matr) const
- vec< T > & operator+= (const T &val)
- vec< T > & operator+= (const vec< T > &vect)
- vec< T > operator+ (const T &val) const
- vec< T > operator+ (const vec< T > &vect) const
- vec< T > & operator-= (const T &val)
- vec< T > & operator-= (const vec< T > &vect)
- vec< T > operator- (const vec< T > &vect) const
- vec< T > operator- () const
- vec< T > & operator/= (const T val)
- vec< T > operator/ (const T val) const
- vec< T > & operator>>= (const int shr)
- vec< T > operator>> (const int shr) const

- `vec< T > & operator<<= (const int shl)`
- `vec< T > operator<< (const int shl) const`
- `vec< T > & operator&= (const T &val)`
- `vec< T > & operator&= (const vec< T > &vect)`
- `vec< T > operator& (const T &val) const`
- `vec< T > operator& (const vec< T > &vect) const`
- `vec< T > & operator|= (const T &val)`
- `vec< T > & operator|= (const vec< T > &vect)`
- `vec< T > operator| (const T &val) const`
- `vec< T > operator| (const vec< T > &vect) const`
- `vec< T > & operator^= (const T &val)`
- `vec< T > & operator^= (const vec< T > &vect)`
- `vec< T > operator^ (const T &val) const`
- `vec< T > operator^ (const vec< T > &vect) const`
- `int sum ()`
- `bool operator== (const vec< T > &vect)`
- `bool operator!= (const vec< T > &vect)`

Поля данных

- `int m_border`
- `int size`
- `T * m_data`

Зашщищенные данные

- `T * m_container`

6.95.1 Подробное описание

```
template<class T>
class vec< T >
```

Класс векторов.

Примеры:

```
int Test[10]={1,125,3,4,5,6,7,8,9,10};
int Res [10];
vec<int> A0(3);
vec<int> B0(3);
vec<int> C0(3);
scalar<int> a0(2);
A0[0]=1;
A0[1]=2;
A0[2]=3;
B0.SetData(Test);
B0=A0;
a0=A0[1];
C0=A0+B0;
C0=A0*a0;
a0=A0*A0;
```

Объявления и описания членов класса находятся в файле:

- `g:/nmpp/include/nmtl/tvector.h`

6.96 Класс Wi_4096_fixed

Поля данных

- char Re
- char Im

Объявления и описания членов класса находятся в файле:

- g:/nmpp/include/nmpls/fftext.h

Глава 7

Файлы

7.1 crtdbg2.h

```
1 #ifdef ENABLE__ASSERTE
2 #ifndef __NM_
3 #include <crtdbg.h>
4 #else
5 #define __ASSERTE
6 #endif
7
8 #define ASSERTE __ASSERTE
9 #else
10 #define ASSERTE(expression)
11#endif
```

7.2 fft.h

```
1 #include "./nmpls/fft.h"
```

7.3 fft.h

```
1 //*****
2 /*          RC Module Inc., Moscow, Russia          */
3 /*          NeuroMatrix(r) NM6403 Software          */
4 /*
5 /*  Fast Fourie Transform Library                  */
6 /*  (C-callable functions)                         */
7 /*
8 /* $Workfile:: sFFT.h                           */
9 /* Contents:      Header file of FFT routines   */
10 /*
11 /*
12 /* Author:      S.Mushkaev                      */
13 /*
14 /* Version      1.0                            */
15 /* Start Date: 03.07.2001                      */
16 /* Release $Date: 2005/07/13 14:19:56 $        */
17 /*
18 /*
19 //*****
20
21 // LIBRARY nmfft.lib
22
23
24
25 #ifndef _SFFT2_H_INCLUDED_
26 #define _SFFT2_H_INCLUDED_
27
28
29 #include "nmtype.h"
30
31 //*****
```

```

33
34 /*
35 #ifndef _NMCMPLX_H_INCLUDED_
36 struct nm32sc
37 {
38 int re;//Real;
39 int im;//Imag;
40 nm32sc():
41 re(0),im(0){} //Real(0),Imag(0){}
42 nm32sc(int _Real,int _Imag):
43 re(_Real),im(_Imag){};//Real(_Real),Imag(_Imag){}
44 };
45 #endif // _NMCMPLX_H_INCLUDED_
46 */
47 // #include "nmpp.h"
48
50 // The functions listed below are forward and inversed FFT routines for
51 // 256,512,1024 or 2048-point compex data, represented as arrays of nm32sc type.
52 // Each complex number is stored in 64-bit word.
53 // The lower 32-bits is real part of complex number.
54 // The higher 32-bits is imaginary part of complex number;
55 // The admissible input range of data depends on dimension of array,
56 // mode of calculation accuracy and on/off mode of intermediate and final scaling down (shift normalization) of results.
57 // This range guarantee against overflow during calculation process.
58 // The table of ranges you may find in "FFT Library Programmer's manual"
59 //
60 // The mode of calculation accuracy tells how sine-cosine coeffecients are represented if fixed-point format.
61 // When the 7-bit accuracy mode is used, output shape accuracy approaches of the maximum,
62 // but output is reduced by around 2%.
63 // If the 6-bit accuracy mode is used, then output range corresponds to result of Fourier transform
64 // based on floating-point arithmetic, but output is less precise.
65 // The mode of calculation accuracy may be set or switched by appropriate ***Set6bit() or ***Set7bit() function
66 // NOTE: At least on time the accuracy setting function must be called before FFT routine executing.
67 #include "malloc32.h"
68
69 #ifdef __cplusplus
70     extern "C" {
71 #endif
72 //===== Forward FFT 256
73 =====
74
75 #include "time.h"
76 #include "malloc32.h"
77
78
79
80 #define FFT_SPEC_NUM_BUFFERS 2
81 #define FFT_SPEC_NUM_TABLES 2
82 #define FFT_SPEC_NUM_SHIFTS 8
83 #define FFT_SPEC_NUM_AMPLITUDES 8
84     typedef struct {
85         int round;
86         int shift;
87     } RoundShift;
88     typedef struct {
89
90
91
92         nm32sc* buffer[FFT_SPEC_NUM_BUFFERS];
93         void* ftTable[FFT_SPEC_NUM_TABLES];
94         RoundShift norm[4];
95         int shift[FFT_SPEC_NUM_SHIFTS];
96         int amp[FFT_SPEC_NUM_AMPLITUDES];
97         int round[8];
98         //int amp;
99         //void* coreOut;
100
101     Free32Func* free;
102     //NmppsAllocation allocOrder;
103 } NmppsFFTSpec;
104
105
106
107
108 // int nmppsFFT256FwdInitAlloc(Malloc32Func* allocate, Free32Func* free, NmppsFFTSpec* spec);
109 // void nmppsFFT256Fwd(nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec);
110 // void nmppsFFTFree(NmppsFFTSpec* spec );
111 //
112 // int nmppsFFT256FwdInitCustomAlloc(Malloc32Func* allocate, Free32Func* free, NmppsFFTSpec* spec);
113 // void nmppsFFT256FwdOptimize(void* src, void* dst, uint64* allocOrder);
114
115
116
117 // #ifdef __NM_
118 // #define sizeof32(t) sizeof(t)
119 // #else

```

```

120 // #define sizeof32(t) (sizeof(t)*4)
121 // #endif
122
123 #define NMPP_OPTIMIZE_DISABLE 1
124 #define NMPP_NORMALIZE_DISABLE 2
125 #define NMPP_FFT_6BIT 4
126 #define NMPP_FFT_7BIT 0
127 #define NMPP_OK 0
128 #define NMPP_ERROR -1
129
130 void nmppsFFTFree(NmppsFFTSpec* spec );
131
132 #define nmppsFFT16HiFwd nmppsFFT16Fwd242
133 #define nmppsFFT16HiFwdRaw nmppsFFT16Fwd242Raw
134
135 int nmppsFFT16HiFwdInit(NmppsFFTSpec* spec, void* buffer0, void* buffer1, void* fftTable0, void* fftTable1);
136 int nmppsFFT16HiFwdInitAlloc(NmppsFFTSpec* spec, const void* src, const void* dst, int settings);
137 void nmppsFFT16HiFwd(const nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec);
138 void nmppsFFT16HiFwdRaw(const nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec);
139
140 void nmppsFFT32FwdRaw(const nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec);
141 void nmppsFFT32Fwd ((const nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec));
142 int nmppsFFT32FwdInitAlloc( NmppsFFTSpec* spec, int settings);
143 void nmppsFFT32Free( NmppsFFTSpec* spec);
144 //void nmppsFFT32FwdRawRef2x16( nm32sc* src, nm32sc* dst);
145
146 // #define nmppsFFT64FwdRaw8x8 nmppsFFT64Fwd
147 void nmppsFFT64Fwd8x8Raw(const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
148 void nmppsFFT64Fwd ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec));
149 void nmppsFFT64Fwd2x4x8 (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
150 int nmppsFFT64FwdInitAlloc( NmppsFFTSpec* spec, int settings);
151 void nmppsFFT64Free( NmppsFFTSpec* spec);
152 //void nmppsFFT64FwdRawRef2x4x8( nm32sc* src, nm32sc* dst);
153
154 void nmppsFFT256Fwd (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
155 int nmppsFFT256FwdOptimize(void* src, void* dst, fseq64* allocOrder) ;
156 int nmppsFFT256FwdInitAlloc( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
157 int nmppsFFT256FwdInitAllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
158
159 void nmppsFFT256Inv(const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
160 int nmppsFFT256InvOptimize (const void* src, const void* dst, fseq64* allocOrder) ;
161 int nmppsFFT256InvInit Alloc (NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
162 int nmppsFFT256InvInit AllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
163
164 void nmppsFFT512FwdRaw ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec));
165 void nmppsFFT512Fwd ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec));
166 int nmppsFFT512FwdOptimize ((const void* src, const void* dst, fseq64* allocOrder) ;
167 int nmppsFFT512FwdInitAlloc ( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
168 int nmppsFFT512FwdInitAllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
169
170 void nmppsFFT512Inv ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
171 void nmppsFFT512InvRaw ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec));
172 int nmppsFFT512InvOptimize ((const void* src, const void* dst, fseq64* allocOrder) ;
173 int nmppsFFT512InvInit Alloc (NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
174 int nmppsFFT512InvInit AllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
175
176 void nmppsFFT1024Fwd ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
177 int nmppsFFT1024FwdOptimize ((const void* src, const void* dst, fseq64* allocOrder) ;
178 int nmppsFFT1024FwdInitAlloc ( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
179 int nmppsFFT1024FwdInitAllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
180
181 void nmppsFFT1024Inv ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
182 int nmppsFFT1024InvOptimize ((const void* src, const void* dst, fseq64* allocOrder) ;
183 int nmppsFFT1024InvInit Alloc (NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
184 int nmppsFFT1024InvInit AllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
185
186 void nmppsFFT2048Fwd ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
187 void nmppsFFT2048FwdInit ((NmppsFFTSpec* specFFT, void* datbuf0, void* datbuf1, void* tblbuf0, void* tblbuf1,
int settings );
188
189 void nmppsFFT2048Fwd4888 ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
190 void nmppsFFT2048Fwd4888Raw ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
191 void nmppsFFT2048FwdRaw ((const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
192 // int nmppsFFT2048Fwd4888_ (nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
193 int nmppsFFT2048FwdOptimize ((const void* src, const void* dst, fseq64* allocOrder) ;
194 int nmppsFFT2048FwdInitAlloc (NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
195 int nmppsFFT2048FwdInitAlloc4888( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
196 int nmppsFFT2048FwdInitAllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int
settings);
197
198

```

```

199 void nmppsFFT2048Inv      (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
200 void nmppsFFT2048InvRaw   (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
201 int nmppsFFT2048InvInit   (NmppsFFTSpec* spec, int settings, nm64s *buf0, nm64s* buf1, nm64s* tbl0, nm64s*
202     tbl1);
203 int nmppsFFT2048InvInitAlloc (NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
204 int nmppsFFT2048InvInitSinCos (NmppsFFTSpec* spec, int settings);
205 //int nmppsFFT2048Inv4888RawRef (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
206 //int nmppsDFT2048InvRef_f (const nm32sc* src, nm32sc* dst);
207 //int nmppsFFT2048Inv28888Ref_f (const nm32sc* src, nm32sc* dst);
208 //int nmppsDFT2048Inv_i (const nm32sc* src, nm32sc* dst, int bits); //int
209 nmppsFFT2048InvInitAllocCustom( NmppsFFTSpec** specFFT, Malloc32Func* allocate, Free32Func* free, int settings);
210
211 #define nmppsFFT8192Fwd      nmppsFFT8192Fwd28888
212 #define nmppsFFT8192FwdRaw   nmppsFFT8192Fwd28888Raw
213 #define nmppsFFT8192FwdInit   nmppsFFT8192Fwd28888Init
214 #define nmppsFFT8192FwdInitAlloc nmppsFFT8192Fwd28888InitAlloc
215
216 #define nmppsFFT8192Inv      nmppsFFT8192Inv28888
217 #define nmppsFFT8192InvRaw   nmppsFFT8192Inv28888Raw
218 #define nmppsFFT8192InvInit   nmppsFFT8192Inv28888Init
219 #define nmppsFFT8192InvInitAlloc nmppsFFT8192Inv28888InitAlloc
220
221 #define FFT2048_TBL0_SIZE64 2048
222 #define FFT2048_TBL1_SIZE64 2048
223 #define FFT2048_TBL2_SIZE64 (4*4*2+4*8*8+32*8*8+256*8*8)/8
224 #define FFT2048_TBL3_SIZE64 (4*4*2+4*8*8+32*8*8+256*8*8)/8
225
226 #define FFT8192_TBL0_SIZE64 9344/8
227 #define FFT8192_TBL1_SIZE64 65536/8
228 #define FFT8192_TBL2_SIZE64 9344/8
229 #define FFT8192_TBL3_SIZE64 65536/8
230
231 //void nmppsFFT8192Fwd      (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
232 int nmppsFFT8192Fwd28888   (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
233 int nmppsFFT8192Fwd28888Raw (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
234 int nmppsFFT8192Fwd28888RawRef (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
235 int nmppsFFT8192Fwd28888Init (NmppsFFTSpec* spec, int settings, nm64s *buf0, nm64s* buf1, nm64s* tbl0, nm64s*
236     tbl1, nm64s* tbl2, nm64s* tbl3);
237 int nmppsFFT8192Fwd28888InitAlloc( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
238 int nmppsDFT8192FwdRef_f (const nm32sc* src, nm32sc* dst);
239 int nmppsFFT8192Fwd28888Ref_f (const nm32sc* src, nm32sc* dst);
240 int nmppsDFT8192Fwd_i (const nm32sc* src, nm32sc* dst, int bits);
241
242 int nmppsFFT8192Inv28888   (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
243 int nmppsFFT8192Inv28888Raw (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
244 int nmppsFFT8192Inv28888RawRef (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
245 int nmppsFFT8192Inv28888Init (NmppsFFTSpec* spec, int settings, nm64s *buf0, nm64s* buf1, nm64s* tbl0, nm64s*
246     tbl1, nm64s* tbl2, nm64s* tbl3);
247 int nmppsFFT8192Inv28888InitAlloc( NmppsFFTSpec** spec, const void* src, const void* dst, int settings);
248 int nmppsFFT8192Inv28888InitSinCos( NmppsFFTSpec* spec, int settings);
249 int nmppsDFT8192InvRef_f (const nm32sc* src, nm32sc* dst);
250 int nmppsFFT8192Inv28888Ref_f (const nm32sc* src, nm32sc* dst);
251 int nmppsDFT8192Inv_i (const nm32sc* src, nm32sc* dst, int bits);
252
253 void nmppsFFTResetSpec(NmppsFFTSpec* spec);
254
255 //#include "nmtl/tcmplx.h"
256 void nmppsFFT32FwdRef2x16 (nm32sc* src, nm32sc* dst); // C++
257 void nmppsFFT32FwdRef2x16_f(nm32sc* src, nm32sc* dst); // C++
258
259
260
261 void nmppsFFT64FwdRef_f (nm32sc* src, nm32sc* dst);
262 void nmppsFFT64FwdRef2x32_f (nm32sc* src, nm32sc* dst);
263 void nmppsFFT64FwdRef2x4x8_f(nm32sc* src, nm32sc* dst);
264 void nmppsFFT64FwdRef2x4x8 (nm32sc* src, nm32sc* dst);
265 void nmppsFFT64FwdRef4x4x4_f(nm32sc* src, nm32sc* dst);
266 void nmppsFFT64FwdRef8x8_f (nm32sc* src, nm32sc* dst);
267 void nmppsFFT64FwdRef8x8 (nm32sc* src, nm32sc* dst);
268
269 void nmppsDFT512Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
270 void nmppsFFT512Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
271 void nmppsDFT512Inv_RefFloat(const nm32sc* src, nm32sc* dst);
272
273 //id nmppsFFT2048Fwd_f (const nm32sc* src, nm32sc* dst, const NmppsFFTSpec* spec);
274 void nmppsDFT2048Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
275 void nmppsDFT2048Inv_RefFloat(const nm32sc* src, nm32sc* dst);
276 void nmppsFFT2048Fwd4888_RefFloat(const nm32sc* src, nm32sc* dst);
277 void nmppsFFT2048Inv4888_RefFloat(const nm32sc* src, nm32sc* dst);
278 void nmppsFFT2048Fwd4888_RefInt(const nm32sc* src, nm32sc* dst);
279 void nmppsFFT2048Inv4888_RefInt(const nm32sc* src, nm32sc* dst);
280 //void nmppsFFT2048Fwd4888(const nm32sc* src, nm32sc* dst);
281

```

```

282 //void nmppsFFT64FwdRef2x2x16_f( nm32sc* src, nm32sc* dst);
283
284 #ifdef __cplusplus
285     };
286 #endif
287
288 #define SKIP_SINCOS 4
289
290 #endif

```

7.4 fft_32fcr.h

```

1 #ifndef FFT_32FCR_H_INCLUDED_
2 #define _FFT_32FCR_H_INCLUDED_
3
4 #ifdef __cplusplus
5     extern "C" {
6 #endif
7
8 #include "nmtype.h"
9
10 #define NUMBUFF1      21
11 #define NUMBUFF2      4
12
13 #define pSrcVecFFTfwdInitAlloc_32fcr(SPEC, SIZE)          nmppsFFT##SIZE##FwdInitAlloc_32fcr(SPEC)
14 #define FFTfwdInitAlloc_32fcr(SPEC, SIZE)                  pSrcVecFFTfwdInitAlloc_32fcr(SPEC, SIZE)
15
16 #define pSrcVecFFTInvInitAlloc_32fcr(SPEC, SIZE)          nmppsFFT##SIZE##InvInitAlloc_32fcr(SPEC)
17 #define FFTInvInitAlloc_32fcr(SPEC, SIZE)                  pSrcVecFFTInvInitAlloc_32fcr(SPEC, SIZE)
18
19 #define pSrcVecFFTfwd_32fcr(SRC, DST, SPEC, SIZE)          nmppsFFT##SIZE##Fwd_32fcr(SRC, DST,
20 #define SPEC)                                              pSrcVecFFTfwd_32fcr(SRC, DST, SPEC, SIZE)
21
22 #define pSrcVecFFTInv_32fcr(SRC, DST, SPEC, SIZE)          nmppsFFT##SIZE##Inv_32fcr(SRC, DST, SPEC)
23 #define FFTInv_32fcr(SRC, DST, SPEC, SIZE)                  pSrcVecFFTInv_32fcr(SRC, DST, SPEC, SIZE)
24
25 typedef struct
26 {
27     nm32fcr* Buffers[NUMBUFF1];
28     nm32fcr* Buffs[NUMBUFF2];
29     int order;
30 } NmppsFFTSpec_32fcr;
31
32 // sFFT_32fcr
33
45 int nmppsFFT16FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
46 int nmppsFFT32FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
47 int nmppsFFT64FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
48 int nmppsFFT128FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
49 int nmppsFFT256FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
50 int nmppsFFT512FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
51 int nmppsFFT1024FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
52 int nmppsFFT2048FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
53 int nmppsFFT4096FwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
55
67 int nmppsFFT16InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
68 int nmppsFFT32InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
69 int nmppsFFT64InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
70 int nmppsFFT128InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
71 int nmppsFFT256InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
72 int nmppsFFT512InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
73 int nmppsFFT1024InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
74 int nmppsFFT2048InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
75 int nmppsFFT4096InvInitAlloc_32fcr(NmppsFFTSpec_32fcr** addr);
77
92 void nmppsDFT8Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
94
106 void nmppsFFT16Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
108
134 void nmppsFFT32Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
136
162 void nmppsFFT64Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
164
186 void nmppsFFT128Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
188
211 void nmppsFFT256Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
213
239 void nmppsFFT512Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
241
266 void nmppsFFT1024Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
268
293 void nmppsFFT2048Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);

```

```

295
319 void nmppsFFT4096Fwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
321
322
335 void nmppsDFT8Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
337
338
349 void nmppsFFT16Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
351
352
377 void nmppsFFT32Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
379
380
405 void nmppsFFT64Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
407
408
433 void nmppsFFT128Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
435
460 void nmppsFFT256Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
462
463
488 void nmppsFFT512Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
490
519 void nmppsFFT1024Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
521
546 void nmppsFFT2048Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
548
573 void nmppsFFT4096Inv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
575
589 //int nmppsFFTfwd_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr *Spc);
590
591
601 //int nmppsFFTfwdInitAlloc_32fcr(NmppsFFTSpec_32fcr** spec, int order);
602
603
604
605
606
618 //int nmppsFFTInv_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec, NmppsFFTSpec_32fcr* spec);
619
620
630 //int nmppsFFTInvInitAlloc_32fcr(NmppsFFTSpec_32fcr** spec, int order);
631
632
633
634
635
636
637
646 int nmppsFFTFree_32fcr(NmppsFFTSpec_32fcr* spec);
647
648 #ifdef __cplusplus
649 };
650 #endif
651
652 #endif // _FFT_32FCR_H_INCLUDED_

```

7.5 fftexp.h

```

1 #include "fft.h"
2 // #include "fftext.h"
3 // #include "nmtl/tmplx_spec.h"
4 #include "nmtl/tmplx.h"
5 #include <math.h>
6
7 #ifndef PI
8 #define PI 3.14159265359
9 #endif
10
11
12 template<int power> void expFFT(int arg, float* re, float* im)
13 {
14     float rad=-2.0*PI/power*arg;
15     *re=cosf((float)rad);
16     *im=sinf((float)rad);
17 }
18
19 template<int power> void expIFFT(int arg, float* re, float* im)
20 {
21     float rad=2.0*PI/power*arg;
22     *re=cosf(rad);
23     *im=sinf(rad);
24 }
25

```

```

26
27 template<int power> void expFFT(int arg, int amplitude, nm32sc* z)
28 {
29     float re;
30     float im;
31     expFFT<power>(arg,&re,&im);
32     z->re=floor(amplitude*re+0.5);
33     z->im=floor(amplitude*im+0.5);
34 }
35
36 template<int power> void expFFT127(int arg, int amplitude, nm32sc* z)
37 {
38     float re;
39     float im;
40     expFFT<power>(arg,&re,&im);
41     z->re=floor(amplitude*re+0.5);
42     z->im=floor(amplitude*im+0.5);
43     z->re= z->re > 127 ? 127:z->re;
44     z->im= z->im > 127 ? 127:z->im;
45 }
46
47 template<int power> void expIFFT127(int arg, int amplitude, nm32sc* z)
48 {
49     float re;
50     float im;
51     expIFFT<power>(arg,&re,&im);
52     z->re=floor(amplitude*re+0.5);
53     z->im=floor(amplitude*im+0.5);
54     z->re= z->re > 127 ? 127:z->re;
55     z->im= z->im > 127 ? 127:z->im;
56 }
57
58 template<int power> void expFFT127(int arg, int amplitude, cmplx<int>* z)
59 {
60     float re;
61     float im;
62     expFFT<power>(arg,&re,&im);
63     z->re=floor(amplitude*re+0.5);
64     z->im=floor(amplitude*im+0.5);
65     z->re= z->re > 127 ? 127:z->re;
66     z->im= z->im > 127 ? 127:z->im;
67 }
68
69 template<int power> void expIFFT127(int arg, int amplitude, cmplx<int>* z)
70 {
71     float re;
72     float im;
73     expIFFT<power>(arg,&re,&im);
74     z->re=floor(amplitude*re+0.5);
75     z->im=floor(amplitude*im+0.5);
76     z->re= z->re > 127 ? 127:z->re;
77     z->im= z->im > 127 ? 127:z->im;
78 }
79
80 template<int power> cmplx<int> expFFT127(int arg, int amplitude=128)
81 {
82     cmplx<int> z;
83     float re;
84     float im;
85     expFFT<power>(arg,&re,&im);
86     z.re=floor(amplitude*re+0.5);
87     z.im=floor(amplitude*im+0.5);
88     z.re= z.re > 127 ? 127:z.re;
89     z.im= z.im > 127 ? 127:z.im;
90     return z;
91 }
92
93
94 template<int power> cmplx<int> expFFT(int arg, int amplitude)
95 {
96     float re;
97     float im;
98     expFFT<power>(arg,&re,&im);
99     cmplx<int> z;
100    z.re=floor(amplitude*re+0.5);
101    z.im=floor(amplitude*im+0.5);
102    return z;
103 }
104
105 template<int power> cmplx<long long> expFFT(int arg, int amplitude)
106 {
107     float re;
108     float im;
109     expFFT<power>(arg, &re, &im);
110     cmplx<long long> z;
111     z.re = floor(amplitude*re + 0.5);
112     z.im = floor(amplitude*im + 0.5);

```

```

113     return z;
114 }
115
116
117 template<int power> cmplx<int> expIFFT(int arg, int amplitude)
118 {
119     float re;
120     float im;
121     expIFFT<power>(arg,&re,&im);
122     cmplx<int> z;
123     z.re=floor(amplitude*re+0.5);
124     z.im=floor(amplitude*im+0.5);
125     return z;
126 }
127
128
129
130 template<int power> cmplx<double> expFFT(int arg)
131 {
132     cmplx<double> z;
133     double im=-2.0*PI/power*arg;
134     z.re=cos(im);
135     z.im=sin(im);
136     return z;
137 }
138 template<int power> cmplx<double> expIFFT(int arg)
139 {
140     cmplx<double> z;
141     double im=2.0*PI/power*arg;
142     z.re=cos(im);
143     z.im=sin(im);
144     return z;
145 }
146
147 void load_wfifo(nm64s* wfifoData, int wfifoStep, int size);
148 void load_wfifo(cmplx<int>* wcoef, int wstep, int size);
149 void load_wfifo(nm32sc* wcoef, int wstep, int size);
150
151 void vsum_data(nm8s* data, nm32sc* afifo, int vr);
152 void vsum_data(nm16s* data, nm32sc* afifo, int vr);
153 void vsum_data(nm8s* data, cmplx<int>* afifo, int vr);

```

7.6 macros_fpu.h

```

1
2 #define ALL_FPU( instr )   "fpu 0 " instr "\n\t" \
3                                "fpu 1 " instr "\n\t" \
4                                "fpu 2 " instr "\n\t" \
5                                "fpu 3 " instr "\n\t"
6
7 // код для исправления аппаратной ошибки №5 процессора 6407
8 // замедляет выполнение процентов на 8, можно отключить для 6408
9 #if NMC_CORE_VERSION > NM6407
10 #define ALL_FPU_ANTI_MASK
11 #else
12 #define ALL_FPU_ANTI_MASK "fp0_lmask = fp0_lmask; \n\t" \
13                                "fp1_lmask = fp1_lmask; \n\t" \
14                                "fp2_lmask = fp2_lmask; \n\t" \
15                                "fp3_lmask = fp3_lmask; \n\t"
16 #endif
17
18 #define DUP(x) x,x
19
20 // Запрещаем прерывания и прочее в том же духе, деструктор возвращает обратно
21 class EnterHardMode
22 {
23     static int inst;
24 public:
25     EnterHardMode()
26     {
27         inst++;
28         asm volatile( "pswr clear 01e0h;           \n\t" // block interrupts
29 //                     ".int 0xf7990000          \n\t" // set fp_wait
30             );
31     }
32     ~EnterHardMode()
33     {
34         if (--inst==0)
35             asm volatile( "pswr set 01e0h;           \n\t" // allow interrupts"
36 //                     ".int 0xf7d90000          \n\t" // set fp_branch
37             );
38     }
39 };

```

7.7 malloc32.h

```

1 #ifndef MALLOC32_DEFINED
2 #define MALLOC32_DEFINED
3 #include <malloc.h>
4 #include "nmtype.h"
5
6 //##define void* malloc32 (unsigned size_int32 );
7 //##define malloc32 nmppsMalloc_32s
8 typedef void (*t_free_func)(void*);
9 #ifdef __cplusplus
10     extern "C" {
11 #endif
12
13
14 #define HEAP_0 1
15 #define HEAP_1 2
16 #define HEAP_2 4
17 #define HEAP_3 8
18
19
20 void* malloc0(unsigned int size);
21 void* malloc1(unsigned int size);
22 void* malloc2(unsigned int size);
23 void* malloc3(unsigned int size);
24
25 //void* malloc32 (unsigned size_int32, unsigned bank_mask);
26 //##define malloc32 (unsigned size_int32)
27 void free32(void* p);
28
29
30 #ifdef __NM
31 #define malloc32(size) malloc(size)
32
33 #else
34 #define malloc32(size) malloc((size)*2)
35 #endif
36
37 // universal malloc for nmc-gcc and nmcc (универсальный маллок для пмс-гсс и пмсс)
38 // size - размер выделяемой памяти в 32-битных словах
39 // heap_num - номер кучи, в которой будет выделена память (для пмсс от 0 до 3, для пмс-гсс от 0 до 15)
40 void* nmppMalloc(unsigned size, int heap_num);
41
42 typedef void (Free32Func)(void* );
43 typedef void* (Malloc32Func)(unsigned int );
44
45 #define NMPP_OPTIMIZE_ALLOC 1
46 #define NMPP_CUSTOM_ALLOC 2
47
48
49 #define ENABLE_HISTORY 1
50 #define ENABLE_RANDOM 1
51 enum MALLOC32_MODE
52     {MALLOC32_RING_MODE,MALLOC32_HISTORY_MODE,MALLOC32_PRIORITY_MODE,
53      MALLOC32_ROUTE_MODE, MALLOC32_LONG_ROUTE_MODE, MALLOC32_FIXED_MODE,
54      MALLOC32_RANDOM_MODE };
55 #define NMPPS_MALLOC_LIMIT 128
56 #define NMPPS_MALLOC_MAX_POS NMPPS_MALLOC_LIMIT-1
57
58 struct NmppsMallocSpec{
59     Malloc32Func* allocator[4];
60     enum MALLOC32_MODE mode;
61     uint32 random;    // bit-mask of available heaps
62     fseq32 priority; // heap allocation sequence in decedendng priority terminated by 0xF
63     uint32 status;   // error status
64     uint32 time;
65     uint32 timeBest;
66     uint32 routePos;
67     fseq64 route[NMPPS_MALLOC_LIMIT/16];
68     fseq64 bestRoute[NMPPS_MALLOC_LIMIT/16];
69     void* allocHistory[NMPPS_MALLOC_LIMIT];
70     void* freeHistory[NMPPS_MALLOC_LIMIT];
71     uint32 allocHistoryPos;
72     uint32 freeHistoryPos;
73     uint32 firstPass;
74 };
75
76 struct NmppsTmpSpec {
77     void* buffer0;
78     void* buffer1;
79 }
80 //nm1* nmppsMalloc_1 (unsigned sizeInt1) ;
81 //nm2s* nmppsMalloc_2s (unsigned sizeInt2) ;
82 //nm2u* nmppsMalloc_2u (unsigned sizeInt2) ;

```

```

83 //nm4s* nmppsMalloc_4s (unsigned sizeInt4) ;
84 //nm4u* nmppsMalloc_4u (unsigned sizeInt4) ;
85 //nm8s* nmppsMalloc_8s (unsigned sizeInt8) ;
86 //nm8u* nmppsMalloc_8u (unsigned sizeInt8) ;
87 //nm16s* nmppsMalloc_16s(unsigned sizeInt16);
88 //nm16u* nmppsMalloc_16u(unsigned sizeInt16);
89 //nm32s* nmppsMalloc_32s(unsigned sizeInt32);
90 //nm32u* nmppsMalloc_32u(unsigned sizeInt32);
91 //nm64s* nmppsMalloc_64s(unsigned sizeInt64);
92 //nm64u* nmppsMalloc_64u(unsigned sizeInt64);
93 //void nmppsFree32(void* buffer);
94
95 typedef struct {
96     void* pull;
97     nm8u* data;
98 }NmppsFrame_8u;
99
100 typedef struct {
101     void* pull;
102     nm8s* data;
103 }NmppsFrame_8s;
104
105 typedef struct {
106     void* pull;
107     nm16u* data;
108 }NmppsFrame_16u;
109
110 typedef struct {
111     void* pull;
112     nm16s* data;
113 }NmppsFrame_16s;
114
115 typedef struct {
116     void* pull;
117     nm32u* data;
118 }NmppsFrame_32u;
119
120 typedef struct {
121     void* pull;
122     nm32s* data;
123 }NmppsFrame_32s;
124
125 typedef struct {
126     void* pull;
127     nm64u* data;
128 }NmppsFrame_64u;
129
130 typedef struct {
131     void* pull;
132     nm64s* data;
133 }NmppsFrame_64s;
134
135 //NmppsFrame_32s* nmppsMallocFrame32 (unsigned sizeInt32, unsigned boundaryInt32);
136 nm8s* nmppsMallocFrame_8s (unsigned sizeInt8 , unsigned boundaryInt8 , NmppsFrame_8s * pFrame) ;
137 nm8u* nmppsMallocFrame_8u (unsigned sizeInt8 , unsigned boundaryInt8 , NmppsFrame_8u * pFrame) ;
138 nm16s* nmppsMallocFrame_16s(unsigned sizeInt16, unsigned boundaryInt16, NmppsFrame_16s* pFrame);
139 nm16u* nmppsMallocFrame_16u(unsigned sizeInt16, unsigned boundaryInt16, NmppsFrame_16u* pFrame);
140 nm32s* nmppsMallocFrame_32s(unsigned sizeInt32, unsigned boundaryInt32, NmppsFrame_32s* pFrame);
141 nm32u* nmppsMallocFrame_32u(unsigned sizeInt32, unsigned boundaryInt32, NmppsFrame_32u* pFrame);
142 nm64s* nmppsMallocFrame_64s(unsigned sizeInt64, unsigned boundaryInt64, NmppsFrame_64s* pFrame);
143 nm64u* nmppsMallocFrame_64u(unsigned sizeInt64, unsigned boundaryInt64, NmppsFrame_64u* pFrame);
144 void nmppsFreeFrame(void* pFrame);
145
146
147 /*
148 typedef struct ss {
149 void* pull;
150 void* data;
151 }NmppiFrame;
152
153 nm8s* nmppiMallocFrame_8s (unsigned width, unsigned height, unsigned border, NmppiFrame_8s * pFrame);
154 nm8u* nmppiMallocFrame_8u (unsigned width, unsigned height, unsigned border, NmppiFrame_8u * pFrame);
155 nm16s* nmppiMallocFrame_16s(unsigned width, unsigned height, unsigned border, NmppiFrame_16s* pFrame);
156 nm16u* nmppiMallocFrame_16u(unsigned width, unsigned height, unsigned border, NmppiFrame_16u* pFrame);
157 nm32s* nmppiMallocFrame_32s(unsigned width, unsigned height, unsigned border, NmppiFrame_32s* pFrame);
158 nm32u* nmppiMallocFrame_32u(unsigned width, unsigned height, unsigned border, NmppiFrame_32u* pFrame);
159 nm64s* nmppiMallocFrame_64s(unsigned width, unsigned height, unsigned border, NmppiFrame_64s* pFrame);
160 nm64u* nmppiMallocFrame_64u(unsigned width, unsigned height, unsigned border, NmppiFrame_64u* pFrame);
161 void nmppsFreeFrame32(void* buffer);
162 */
163
164
165
166 typedef struct {
167     void* buffer;
168     int status;
169     int mode;

```

```

170     fseq64 route;
171 } SpecTmp1;
172 //
173 //typedef struct {
174 // nm8s* buffer0;
175 // nm8s* buffer1;
176 // int status;
177 // int mode;
178 // fseq64 route;
179 //} Spec8s;
180
181 //int nmppsMallocCreate(int limit);
182 //int nmppsMallocDestroy();
183 //int nmppsMallocMode(WIPE_ENABLE|REC_HISTORY|REC_ROUTE);
184
185 //void* nmppsMallocSpec_8s(SpecTmp1* spec,int size, int mode);
186 void* nmppsMallocSpec1(SpecTmp1* spec,int sizeBuf0,int , int);
187 void* nmppsFreeSpec1 (SpecTmp1* spc);
188
189 int nmppsMallocResetPos();
190 int nmppsMallocResetRoute ();
191 void nmppsMallocSetRouteMode ();
192 void nmppsMallocSetRingMode ();
193 //void nmppsMallocSetRouteModeEx(seq64* heapSeq, int routeSize, int routeLimit);
194
195 //void nmppsMallocSetBigRouteMode(seq64* heapSeq, int heapCount);
196 //void nmppsMallocSetRandomMode(uint32 heapSet, int heapCount);
197 //void nmppsMallocSetPriorityMode (seq64 heapSeq, int heapCount);
198 //void nmppsMallocSetHistoryMode(seq64* addrSeq, void** allocSeq);
199 void nmppsMallocSetHistoryMode();
200 void nmppsMallocRecHistory (void** allocSeq, void** freeSeq);
201 //int nmppsMallocStopAddrRec ();
202 //void nmppsMallocStartRouteRec (seq64* heapSeq, int maxHeapCount);
203 //int nmppsMallocStopRouteRec ();
204 //void nmppsMallocSetBoundary32 (int size32 );
205 //void nmppsMallocSetFill32(int borderColor, int dataColor );
206 //int nmppsMallocCheckBoundary(void* );
207 //int nmppsMallocIsErrorStatus();
208 int nmppsMallocSuccess();
209 int nmppsMallocFail();
210 int nmppsMallocStatus();
211 void nmppsMallocResetStatus ();
212 //int nmppsMallocGetHistory (fseq64* heapSeq, int seqSize);
213 //int nmppsMallocSaveState();
214 //int nmppsMallocRestoreState();
215 //void nmppsMallocReset(seq64* currRoute,void** allocHistory,void** freeHistory, int size);
216
217 void nmppsMallocTimerStart();
218 void nmppsMallocTimerResume();
219 void nmppsMallocTimerStop();
220
221 //void nmppsMallocNextRoute(seq64* currRoute, int mode);
222 void nmppsMallocRandomRoute();
223 int nmppsMallocIncrementRoute0();
224 int nmppsMallocIncrementRoute(int routeLength, int heapMask );
225 int nmppsMallocWipe();
226 int nmppsMallocBetterRoute();
227 void nmppsMallocSetBestRoute(int historyEnable);
228 int nmppsMallocSetRoute(fseq64* route, int count); // implemented
229 void nmppsMallocSetShortRoute(fseq64 route);
230 void nmppsMallocSetRoute16(fseq64 route);
231 void nmppsMallocGetRoute16(fseq64* route);
232 extern struct NmppsMallocSpec nmppsMallocSpec;
233
234
235 void* nmppsMalloc32 (unsigned sizeInt32);
236 nm64s* nmppsMalloc_64s (unsigned nSize);
237 nm1* nmppsMalloc_1 (unsigned nSize) ;//{return (nm1*)nmppsMalloc_64s((nSize>>6) +1);}
238 nm2s* nmppsMalloc_2s (unsigned nSize) ;//{return (nm2s*)nmppsMalloc_64s((nSize>>5) +1);}
239 nm2u* nmppsMalloc_2u (unsigned nSize) ;//{return (nm2u*)nmppsMalloc_64s((nSize>>5) +1);}
240 nm4s* nmppsMalloc_4s (unsigned nSize) ;//{return (nm4s*)nmppsMalloc_64s((nSize>>4) +1);}
241 nm4u* nmppsMalloc_4u (unsigned nSize) ;//{return (nm4u*)nmppsMalloc_64s((nSize>>4) +1);}
242 nm8u* nmppsMalloc_8u (unsigned nSize) ;//{return (nm8u*)nmppsMalloc_64s((nSize>>3) +1);}
243 nm8s* nmppsMalloc_8s (unsigned nSize) ;//{return (nm8s*)nmppsMalloc_64s((nSize>>3) +1);}
244 nm16u* nmppsMalloc_16u (unsigned nSize) ;//{return (nm16u*)nmppsMalloc_64s((nSize>>2) +1);}
245 nm16s* nmppsMalloc_16s (unsigned nSize) ;//{return (nm16s*)nmppsMalloc_64s((nSize>>2) +1);}
246 nm32u* nmppsMalloc_32u (unsigned nSize) ;//{return (nm32u*)nmppsMalloc_64s((nSize>>1) +1);}
247 nm32s* nmppsMalloc_32s (unsigned nSize) ;//{return (nm32s*)nmppsMalloc_64s((nSize>>1) +1);}
248 nm64u* nmppsMalloc_64u (unsigned nSize) ;//{return (nm64u*)nmppsMalloc_64s(nSize);}
249 nm64sc* nmppsMalloc_64sc (unsigned nSize) ;//{return (nm64u*)nmppsMalloc_64s(nSize);}
250 nm32sc* nmppsMalloc_32sc (unsigned sizeCmplxInt32);
251 nm32fc* nmppsMalloc_32fc (unsigned sizeCmplxFloat);
252 nm32fcr* nmppsMalloc_32fcr(unsigned sizeCmplxFloat);
253 float* nmppsMalloc_32f (unsigned sizeFloat);
254 double* nmppsMalloc_64f (unsigned sizeDouble);
255

```

```

282 /*
283  _INLINE __ void nmppsMalloc_1 (nm1** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»6) +1,
284  hint);}
284  _INLINE __ void nmppsMalloc_2s(nm2s** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»5) +1,
285  hint);}
285  _INLINE __ void nmppsMalloc_2u(nm2u** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»5) +1,
286  hint);}
286  _INLINE __ void nmppsMalloc_4s(nm4s** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»4) +1,
287  hint);}
287  _INLINE __ void nmppsMalloc_4u(nm4u** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»4) +1,
288  hint);}
288  _INLINE __ void nmppsMalloc_8u(nm8u** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»3) +1,
289  hint);}
289  _INLINE __ void nmppsMalloc_8s(nm8s** pptr, int nSize, int hint ) { nmppsMalloc_64s((nm64s**)pptr, (nSize»3) +1,
290  hint);}
290  _INLINE __ void nmppsMalloc_16u(nm16u** pptr, int nSize, int hint) { nmppsMalloc_64s((nm64s**)pptr, (nSize»2) +1,
291  hint);}
291  _INLINE __ void nmppsMalloc_16s(nm16s** pptr, int nSize, int hint) { nmppsMalloc_64s((nm64s**)pptr, (nSize»2) +1,
292  hint);}
292  _INLINE __ void nmppsMalloc_32u(nm32u** pptr, int nSize, int hint) { nmppsMalloc_64s((nm64s**)pptr, (nSize»1) +1,
293  hint);}
293  _INLINE __ void nmppsMalloc_32s(nm32s** pptr, int nSize, int hint) { nmppsMalloc_64s((nm64s**)pptr, (nSize»1) +1,
294  hint);}
294  _INLINE __ void nmppsMalloc_64u(nm64u** pptr, int nSize, int hint) { nmppsMalloc_64s((nm64s**)pptr, nSize, hint);}
295 */
296 //*****
297
298
299
300
301
302
303
304
305
306
307
308 void nmppsFree(void* ptr);
309
310
311
312
313
314 #ifdef __cplusplus
315 };
316 #endif
317
318 #endif

```

7.8 metric.h

```

1 #include "nmotype.h"
2
3 #ifdef __cplusplus
4     extern "C" {
5 #endif
6
7 float nmppsMSD_32sc(const nm32sc* x0, const nm32sc* x1, int size);
8 float nmppsRMSD_32fc(const nm32fc *src1, const nm32fc *src2, int num);
9
10 void nmppsNormDiff_L1_32s (const nm32s* pSrc1, const nm32s* pSrc2, int len, float* pNorm);
11 void nmppsNormDiff_L2_32s (const nm32s* pSrc1, const nm32s* pSrc2, int len, float* pNorm);
12 void nmppsNormDiff_L2_32f(const float* pSrc1, const float* pSrc2, int len, float* pNorm);
13 void nmppsNormDiff_L2_32fcr(const nm32fc *src1, const nm32fc *src2, int num, float* pNorm);
14 void nmppsNormDiff_Inf_32s(const nm32s* pSrc1, const nm32s* pSrc2, int len, float* pNorm);
15 void nmppsNormDiff_Inf_32f(const float* pSrc1, const float* pSrc2, int len, float* pNorm);
16
17
18 #ifdef __cplusplus
19 };
20#endif

```

7.9 minrep.h

```
1 #define NMPP_MIN REP 1
```

7.10 multiheap.h

```

1 #ifndef MULTIHEAP_INCLUDED
2 #define MULTIHEAP_INCLUDED
3 typedef unsigned size_t32;
4 typedef unsigned size_t32;
5
6

```

```

7
8 #include "nmpp.h"
9
10
11 #define GUARD_BITS 0x600DB1D5
12 #define MAX_BUFFER32_SIZE 16*1024*1024/4
13 struct S_BufferInfo{
14     int      guardInfoBits0;
15     size_t32    size32Buffer;
16     S_BufferInfo* pPrevBuffer;
17     S_BufferInfo* pNextBuffer;
18     bool      isLocked;
19     int      guardInfoBits1;
20
21
22     int* DataBegin(){
23         return ((int*)this+sizeof(S_BufferInfo)/sizeof(int));
24     };
25
26     int* DataEnd(){
27         int *p=((int*)this+sizeof(S_BufferInfo)/sizeof(int)+size32Buffer);
28         return p;
29     }
30     int* EndGuardBits(){
31         return DataEnd()-2;
32     }
33     unsigned CheckGuardBits(){
34         if (guardInfoBits0!=GUARD_BITS){
35             return 0xBADE00F1;
36         }
37         if (guardInfoBits1!=GUARD_BITS){
38             return 0xBADE00F2;
39         }
40         if (size32Buffer){
41             int* pFinalGuardBits=EndGuardBits();
42             if ((*pFinalGuardBits)!=GUARD_BITS){
43                 return 0xBADE00F3;
44             }
45             if (*pFinalGuardBits+1)!=GUARD_BITS{
46                 return 0xBADE00F4;
47             }
48             if (size32Buffer<0)
49                 return 0xBADE00FF;
50         }
51         if (size32Buffer>MAX_BUFFER32_SIZE)
52             return 0xBADE00FE;
53     }
54
55     }
56     return 0;
57 }
58 }
59 }
60 };
61
62
63 class C_Heap{
64 public:
65     S_BufferInfo* pZeroBuffer;
66     int* pHeapEnd;
67     int      size32HeapAvail;
68     bool      isHeapLocked;
69     int      status;
70
71     C_Heap() {
72         pZeroBuffer = 0;
73         pHeapEnd   = 0;
74         size32HeapAvail=0;
75         isHeapLocked=false;
76         status =0;
77     }
78     C_Heap (void* addrHeap, size_t32 size32Heap){
79         Create(addrHeap,size32Heap);
80     }
81
82     void Create( void* addrHeap, size_t32 size32Heap){
83
84         pZeroBuffer= (S_BufferInfo*)addrHeap; // начальный адрес кучи
85         pZeroBuffer->pPrevBuffer=0;
86         pZeroBuffer->pNextBuffer=0;
87         pZeroBuffer->size32Buffer=0;
88         pZeroBuffer->isLocked=true;
89         pZeroBuffer->guardInfoBits0=GUARD_BITS;
90         pZeroBuffer->guardInfoBits1=GUARD_BITS;
91
92         pHeapEnd   = (int*)addrHeap+size32Heap; // конечный адрес кучи (следующий за последним
93         // байтом кучи)
94         size32HeapAvail=pHeapEnd-(int*)pZeroBuffer->DataEnd();
95     }
96
97     int IsMine(void* addr){
98         if ((int*)pZeroBuffer < (int*)addr && (int*)addr < pHeapEnd)
99
100

```

```

101         return 1;
102     return 0;
103 }
104
106 //int HeapAvail(){
107 //    return (size32FreeMem-sizeof(S_BufferInfo)/sizeof(int));
108 //}
109
110 //size_t32 TotalAvail(){
111 //    return size32HeapAvail;
112 //}
113
115 size_t32 AllocateMaxAvail(){
116     int size32Max=0;
117     S_BufferInfo* pBuffer=pZeroBuffer;
118     for ( ; pBuffer->pNextBuffer!=0; pBuffer=pBuffer->pNextBuffer) {
119         int size32Between=(int*)pBuffer->pNextBuffer - pBuffer->DataEnd();
120         if (size32Between>size32Max)
121             size32Max=size32Between;
122     }
123     // находимся на последнем буфере
124     int size32Between=(int*)pHeapEnd - pBuffer->DataEnd();
125     if (size32Between>size32Max)
126         size32Max=size32Between;
127     size32Max-=sizeof(S_BufferInfo)/sizeof(int);
128     return size32Max;
129 }
130
132 int* Allocate(size_t32 size32Buffer){
133     if (isHeapLocked)
134         return 0;
135     if (size32Buffer==0)
136         return 0;
137     size32Buffer+=(size32Buffer&1); // округляем вверх до четного размера
138     size32Buffer+=2; // добавляем 2 слова для защитного поля
139     S_BufferInfo* pBuffer=pZeroBuffer;
140     int size32Between=0;
141     for(; pBuffer->pNextBuffer!=0; pBuffer=pBuffer->pNextBuffer){
142         // проверяем влезает ли массив в промежуток между буферами
143         size32Between=(int*)pBuffer->pNextBuffer - pBuffer->DataEnd();
144         if (size32Between>=size32Buffer+sizeof(S_BufferInfo)/sizeof(int)){
145             // создаем новый буфер
146             S_BufferInfo* pAllocateBuffer=(S_BufferInfo*)pBuffer->DataEnd();
147             // устанавливаем связи в следующем буфере
148             pBuffer->pNextBuffer->pPrevBuffer=pAllocateBuffer;
149             // устанавливаем связи в новом буфере
150             pAllocateBuffer->pPrevBuffer=pBuffer;
151             pAllocateBuffer->pNextBuffer=pBuffer->pNextBuffer;
152             pAllocateBuffer->size32Buffer=size32Buffer;
153             pAllocateBuffer->isLocked=false;
154             // устанавливаем защитные биты
155             pAllocateBuffer->guardInfoBits0=GUARD_BITS;
156             pAllocateBuffer->guardInfoBits1=GUARD_BITS;
157             int* guardEndBits=pAllocateBuffer->EndGuardBits();
158             *guardEndBits = GUARD_BITS;
159             *(guardEndBits+1)=GUARD_BITS;
160             // устанавливаем связи в предыдущем буфере
161             pBuffer->pNextBuffer=pAllocateBuffer;
162             // уменьшаем свободный размер
163             size32HeapAvail-=(size32Buffer+sizeof(S_BufferInfo)/sizeof(int));
164
165             return pAllocateBuffer->DataBegin();
166         }
167     }
168     // дошли до последнего буфера (у которого ->pNextBuffer=0)
169     size32Between=(int*)pHeapEnd - pBuffer->DataEnd();
170     if (size32Between>=size32Buffer+sizeof(S_BufferInfo)/sizeof(int)){
171         // создаем новый буфер
172         S_BufferInfo* pAllocateBuffer=(S_BufferInfo*)pBuffer->DataEnd();
173         // устанавливаем связи в новом буфере
174         pBuffer->pNextBuffer=0;
175         pAllocateBuffer->pNextBuffer=0;
176         pAllocateBuffer->size32Buffer=size32Buffer;
177         pAllocateBuffer->isLocked=false;
178         // устанавливаем защитные биты
179         pAllocateBuffer->guardInfoBits0=GUARD_BITS;
180         pAllocateBuffer->guardInfoBits1=GUARD_BITS;
181         int* guardEndBits=pAllocateBuffer->EndGuardBits();
182         *guardEndBits = GUARD_BITS;
183         *(guardEndBits+1)=GUARD_BITS;
184         // устанавливаем связи в предыдущем буфере
185         pBuffer->pNextBuffer=pAllocateBuffer;
186         size32HeapAvail-=(size32Buffer+sizeof(S_BufferInfo)/sizeof(int));
187         // уменьшаем свободный размер
188         return pAllocateBuffer->DataBegin();
189     }
190     return 0;

```

```

191     }
192
193     int ReleaseBuffer(S_BufferInfo* pDelBuffer){
194         if (isHeapLocked)
195             return 0;
196         if (pDelBuffer->isLocked)
197             return 0;
198         if (pDelBuffer->size32Buffer==0)
199             return 0;
200
201         pDelBuffer->pPrevBuffer->pNextBuffer=pDelBuffer->pNextBuffer; // связываем ссылку предыдущего буфера
202         на следующий
203         if (pDelBuffer->pNextBuffer) // если есть следующий
204             pDelBuffer->pNextBuffer->pPrevBuffer=pDelBuffer->pPrevBuffer; // связываем ссылку следующего буфера на
205         предыдущий
206         size32HeapAvail+=(pDelBuffer->size32Buffer+sizeof(S_BufferInfo)/sizeof(int));
207         pDelBuffer->size32Buffer=0;
208         return 1;
209     }
210
211     int Release(void* p){
212         if (isHeapLocked)
213             return 0;
214         S_BufferInfo* pDelBuffer=(S_BufferInfo*)p-1;
215         return ReleaseBuffer(pDelBuffer);
216     }
217
218     void Lock(void* p){
219         S_BufferInfo* pBuffer=(S_BufferInfo*)p-1;
220         pBuffer->isLocked=true;
221     }
222
223
224     void LockAll(){
225         S_BufferInfo* pBuffer=pZeroBuffer->pNextBuffer;
226         while (pBuffer) {
227             pBuffer->isLocked=true;
228             pBuffer=pBuffer->pNextBuffer;
229         }
230     }
231
232     void UnlockAll(){
233         S_BufferInfo* pBuffer=pZeroBuffer->pNextBuffer;
234         while (pBuffer) {
235             pBuffer->isLocked=false;
236             pBuffer=pBuffer->pNextBuffer;
237         }
238     }
239
240
241     void Unlock(void* p){
242         S_BufferInfo* pBuffer=(S_BufferInfo*)p-1;
243         pBuffer->isLocked=false;
244         //numHeapLocked--;
245     }
246
247
248     void ReleaseAll(){
249         S_BufferInfo* pBuffer=pZeroBuffer->pNextBuffer;
250         if (!isHeapLocked)
251             return ;
252         while (pBuffer) {
253             if (!pBuffer->isLocked){
254                 ReleaseBuffer(pBuffer);
255             }
256             pBuffer=pBuffer->pNextBuffer;
257         }
258     }
259
260     void LockHeap(){
261         isHeapLocked=true;
262     }
263
264     void UnlockHeap(){
265         isHeapLocked=false;
266     }
267
268     int Check(){
269         if (status)
270             return status;
271
272         if (pZeroBuffer->pPrevBuffer){
273             status=0BADB00F0;
274             return status;
275         }
276         if (pZeroBuffer->size32Buffer){
277             status=0BADB00FF;
278             return status;
279         }
280         // if (status==pZeroBuffer->CheckGuardBits()){
281         //     return status;
282         // }
283
284         S_BufferInfo* pBuffer=pZeroBuffer->pNextBuffer;

```

```

285     int AllocatedMemory=0;
286     int MaxHeapSize=(int*)pHeapEnd-(int*)pZeroBuffer;
287
288     for(int i=0;i<100;i++){
289         if (pBuffer==0)
290             break;
291
292
293         if (!IsMine(pBuffer)){
294             status =0xBADB00FA;
295             return status;
296         }
297
298         if (status==pBuffer->CheckGuardBits()){
299             return status;
300         }
301
302         if (pBuffer->pPrevBuffer->pNextBuffer!=pBuffer){
303             status =0xBADB00FB;
304             return status;
305         }
306     }
307
308     AllocatedMemory+=sizeof(S_BufferInfo)/sizeof(int);
309     AllocatedMemory+=pBuffer->size32Buffer;
310     if (AllocatedMemory>MaxHeapSize){
311         status=0xBADB00FD;
312         return status;
313     }
314     if ((int)pBuffer&1){
315         status=0xBADB00FC;
316         return status;
317     }
318     pBuffer=pBuffer->pNextBuffer;
319 }
320
321 }
322 };
323
324
325 #define ALLOCATE_FORWARD 1
326 #define ALLOCATE_BACKWARD 2
327 #define ALLOCATE_RANDOM 4
328 #define ALLOCATE_CUSTOM 8
329 #define MAX_NUM_BANKS 6
330 class C_Allocator32 {
331     public:
332     void* Allocate();
333     int Release();
334 };
335
336 class C_MultiHeap: public C_Allocator32 {
337     unsigned m_z;
338     unsigned m_w;
339 public:
340     C_Heap pHeap[MAX_NUM_BANKS];
341     unsigned numHeaps;
342     unsigned numAllocateFails;
343     unsigned AllocateMode ;
344     void** pAllocateLegend;
345     unsigned idxAllocateLegend;
346     long long allocateHistory;
347     C_MultiHeap(int Mode=ALLOCATE_FORWARD) {
348         allocateHistory=0;
349         numHeaps=0;
350         numAllocateFails=0;
351         m_z=3141516;
352         m_w=3141516;
353         AllocateMode=Mode;
354         pAllocateLegend=0;
355     }
356
357     int Error(){
358         return numAllocateFails;
359     }
360     void Mode(int mode, void** legend=0){
361
362         idxAllocateLegend=0;
363         pAllocateLegend=legend;
364         AllocateMode=mode;
365     }
366
367     unsigned Rand()
368     {
369         m_z = 36969 * (m_z & 0xFFFF) + (m_z >> 16);
370         m_w = 18000 * (m_w & 0xFFFF) + (m_w >> 16);
371         return (m_z << 16) + m_w;
372     }

```

```

373     }
374     unsigned Rand(unsigned min, unsigned max){
375         unsigned r=Rand();
376         r=(r-min)%(max-min+1)+min;
377         return r;
378     }
379 }
380
381 C_Heap& operator [](int idxHeap){
382     return pHHeap[idxHeap];
383 }
384
385 int CreateHeap(void* addrHeap, size_t32 size32Heap){
386     if (numHeaps>MAX_NUM_BANKS)
387         return 0;
388
389     pHHeap[numHeaps].Create(addrHeap, size32Heap);
390     if (pHHeap[numHeaps].size32HeapAvail)
391         numHeaps++;
392     return 1;
393 }
394
395 void* Allocate(size_t32 size32Buffer){
396
397     void *p;
398     unsigned i;
399
400     switch (AllocateMode){
401         case ALLOCATE_FORWARD:
402             for(i=0; i<numHeaps; i++){
403                 p=pHHeap[i].Allocate(size32Buffer);
404                 if (p){
405                     allocateHistory<<=4;
406                     allocateHistory|=i;
407                     if (pAllocateLegend){
408                         //pAllocateLegend[idxAllocateLegend>>3]&=^(0xFF<<((idxAllocateLegend&7)<<4));
409                         //pAllocateLegend[idxAllocateLegend>>3]|=^(i<<((idxAllocateLegend&7)<<4));
410                         pAllocateLegend[idxAllocateLegend]=p;
411                         idxAllocateLegend++;
412                     }
413                 }
414             }
415             return p;
416         }
417         numAllocateFails++;
418         break;
419
420         case ALLOCATE_BACKWARD:
421             for(i=numHeaps-1; i>=0; i--){
422                 p=pHHeap[i].Allocate(size32Buffer);
423                 if (p){
424                     allocateHistory<<=4;
425                     allocateHistory|=i;
426                     if (pAllocateLegend){
427                         //pAllocateLegend[idxAllocateLegend>>3]&=^(0xFF<<((idxAllocateLegend&7)<<4));
428                         //pAllocateLegend[idxAllocateLegend>>3]|=^(i<<((idxAllocateLegend&7)<<4));
429                         pAllocateLegend[idxAllocateLegend]=p;
430                         idxAllocateLegend++;
431                     }
432                 }
433             }
434             return p;
435         }
436         numAllocateFails++;
437         break;
438
439         case ALLOCATE_RANDOM:{
440             unsigned mapAllocateFails=0;
441             unsigned mapNoMemory=0xFFFFFFFF<<(32-numHeaps);
442             while(mapAllocateFails!=mapNoMemory){
443                 i=Rand()%numHeaps;
444                 mapAllocateFails|=(1<<i);
445                 p=pHHeap[i].Allocate(size32Buffer);
446                 if (p){
447                     allocateHistory<<=4;
448                     allocateHistory|=i;
449                     if (pAllocateLegend){
450                         //pAllocateLegend[idxAllocateLegend>>3]&=^(0xFF<<((idxAllocateLegend&7)<<4));
451                         //pAllocateLegend[idxAllocateLegend>>3]|=^(i<<((idxAllocateLegend&7)<<4));
452                         pAllocateLegend[idxAllocateLegend]=p;
453                         idxAllocateLegend++;
454                     }
455                 }
456             }
457             return p;
458         }
459         numAllocateFails++;
460         break;
461     }

```

```

462     case ALLOCATE_CUSTOM:{
463         //unsigned mapAllocateFails=0;
464         //unsigned mapNoMemory=0xFFFFFFFF*(32-numHeaps);
465         //i=pAllocateLegend[idxAllocateLegend];
466         //idxAllocateLegend++;
467         //p=pHeap[i].Allocate(size32Buffer);
468         //if (p){
469             // return p;
470         //}
471         //numAllocateFails++;
472         return pAllocateLegend[idxAllocateLegend++];
473     }
474 }
475     return 0;
476 }
477
478 void* Allocate(size_t32 size32Buffer, int nPriorHeap0, int nPriorHeap1=-1,int nPriorHeap2=-1,int nPriorHeap3=-1,int
nPriorHeap4=-1,int nPriorHeap5=-1){
479
480     int* p;
481     p=pHeap[nPriorHeap0].Allocate(size32Buffer);
482     if (p) {
483         if (pAllocateLegend){
484             pAllocateLegend[idxAllocateLegend]=p;
485             idxAllocateLegend++;
486         }
487         return p;
488     }
489
490     if (nPriorHeap1!=-1){
491         p=pHeap[nPriorHeap1].Allocate(size32Buffer);
492         if (p) {
493             if (pAllocateLegend){
494                 pAllocateLegend[idxAllocateLegend]=p;
495                 idxAllocateLegend++;
496             }
497             return p;
498         }
499     }
500     if (nPriorHeap2!=-1){
501         p=pHeap[nPriorHeap2].Allocate(size32Buffer);
502         if (p) {
503             if (pAllocateLegend){
504                 pAllocateLegend[idxAllocateLegend]=p;
505                 idxAllocateLegend++;
506             }
507             return p;
508         }
509     }
510
511     if (nPriorHeap3!=-1){
512         p=pHeap[nPriorHeap3].Allocate(size32Buffer);
513         if (p) {
514             if (pAllocateLegend){
515                 pAllocateLegend[idxAllocateLegend]=p;
516                 idxAllocateLegend++;
517             }
518             return p;
519         }
520     }
521     if (nPriorHeap4!=-1){
522         p=pHeap[nPriorHeap4].Allocate(size32Buffer);
523         if (p) {
524             if (pAllocateLegend){
525                 pAllocateLegend[idxAllocateLegend]=p;
526                 idxAllocateLegend++;
527             }
528             return p;
529         }
530     }
531
532     if (nPriorHeap5!=-1){
533         p=pHeap[nPriorHeap5].Allocate(size32Buffer);
534         if (p) {
535             if (pAllocateLegend){
536                 pAllocateLegend[idxAllocateLegend]=p;
537                 idxAllocateLegend++;
538             }
539             return p;
540         }
541     }
542
543     numAllocateFails++;
544     return 0;
545 }
546 void* AllocateWith(size_t32 size32Buffer, void* addrInTheSameHeap){
547     for(int i=0; i<numHeaps; i++){

```

```

550     if (pHeap[i].IsMine(addrInTheSameHeap)){
551         void *p=pHeap[i].Allocate(size32Buffer);
552         if (p){
553             if (pAllocateLegend){
554                 pAllocateLegend[idxAllocateLegend]=p;
555                 idxAllocateLegend++;
556             }
557             return p;
558         }
559     }
560 }
561 return 0;
562 }
563
564 int Which(void *addr){
565     for(int i=0; i<numHeaps; i++){
566         if (pHeap[i].IsMine(addr))
567             return i;
568     }
569     return -1;
570 }
571
572 void Lock(void* addr){
573     S_BufferInfo* pBuffer=(S_BufferInfo*)addr-1;
574     pBuffer->isLocked=true;
575 }
576
577 int Unlock(void* addr){
578     S_BufferInfo* pBuffer=(S_BufferInfo*)addr-1;
579     pBuffer->isLocked=false;
580     return 0;
581 }
582
583 int LockAll(){
584     for(int i=0; i<numHeaps; i++){
585         pHeap[i].LockAll();
586     }
587     return 0;
588 }
589
590 int UnlockAll(){
591     for(int i=0; i<numHeaps; i++){
592         pHeap[i].UnlockAll();
593     }
594     return 0;
595 }
596
597 int Release(void* addr){
598     if (AllocateMode==ALLOCATE_CUSTOM)
599         return 0;
600     if (addr)
601         for(int i=0; i<numHeaps; i++){
602             if (pHeap[i].IsMine(addr)){
603                 return pHeap[i].Release(addr);
604             }
605         }
606     return 0;
607 }
608
609 void ReleaseAll(){
610     for(int i=0; i<numHeaps; i++){
611         pHeap[i].ReleaseAll();
612     }
613 }
614
615 void LockHeap(int idxHeap){
616     pHeap[idxHeap].LockHeap();
617 }
618 void UnlockHeap(int idxHeap){
619     pHeap[idxHeap].UnlockHeap();
620 }
621
622 int Check(){
623     for(int i=0; i<numHeaps; i++){
624         if (pHeap[i].Check())
625             return pHeap[i].status;
626     }
627     return 0;
628 }
629
630 }
631 };
632
633
634 template<class T> class C_BoxVec
635 {
636 public:
637     int    sizeData;
638     int    sizeBox;
639     int    nBorder;
640     T*    pBox;

```

```

641     T*      pData;
642     C_MultiHeap* pHeap;
643
644     C_BoxVec(){
645         sizeBox=0;
646         sizeData=0;
647         nBorder=0;
648         pBox=0;
649         pData=0;
650         pHeap=0;
651     }
652
653     C_BoxVec(T* Data, int SizeData, int SizeBorder=0){
654         pHeap=0;
655         Assign(Data, SizeData, SizeBorder);
656     }
657
658     C_BoxVec(C_MultiHeap& MultiHeap, int SizeData, int Border=0){
659         sizeData=SizeData;
660         nBorder =Border;
661         Allocate(MultiHeap);
662     }
663
664
665     T* Addr(int idx){
666         return nmppsAddr_(pData,idx);
667     }
668
669     int Assign (T* Data, int SizeData, int SizeBorder=0){
670         pData =Data;
671         pBox =nmppsAddr_(pData,-SizeBorder);
672         sizeData=SizeData;
673         sizeBox =SizeData+2*SizeBorder;
674     }
675     T* Allocate(C_MultiHeap& MultiHeap, int SizeData, int Border=0){
676         sizeData=SizeData;
677         nBorder =Border;
678         Allocate(MultiHeap);
679     }
680
681     T* Allocate(C_MultiHeap& MultiHeap){
682         pHeap=&MultiHeap;
683         pData=0;
684         sizeBox =sizeData+2*nBorder;
685         int sizeBox32=(int)nmppsAddr_((T*)0,sizeBox)/sizeof(int);
686         pBox =(T*)MultiHeap.Allocate(sizeBox32);
687         if (pBox){
688             pData=nmppsAddr_(pBox,(sizeBox-sizeData)/2);
689         }
690         return pData;
691     }
692
693     int Release(){
694         if (pBox)
695             if (pHeap){
696                 pHeap->Release(pBox);
697                 pBox=0;
698             }
699         return 0;
700     }
701
702     ~C_BoxVec() {
703         if (pHeap)
704             if (pBox)
705                 pHeap->Release(pBox);
706     }
707     //__INLINE__ T& operator [](int idx){
708     //    return *nmppsAddr_(pData,idx);
709     //}
710 };
711
712 #define BOX_nmppiFILL_NONE 0xF111DEAD
713 #define BOX_nmppiFILL_RAND 0xF1110123
714 #define BOX_nmppiFILL_00 0
715 #define BOX_nmppiFILL_01 1
716 #define BOX_nmppiFILL_FF -1
717 template<class T> class C_BoxImg
718 {
719
720     public:
721         int      nWidth;
722         int      nHeight;
723         int      sizeBox;
724         int      sizeData;
725         int      nBorder;
726         T*      pBox;
727         T*      pData;

```

```

728     C_MultiHeap* pHeap;
729
730     C_BoxImg(){
731         nWidth=0;
732         nHeight=0;
733         sizeBox=0;
734         sizeData=0;
735         nBorder=0;
736         pBox=0;
737         pData=0;
738         pHeap=0;
739     }
740
741
742     C_BoxImg(C_MultiHeap& MultiHeap, int Width, int Height, int BorderHeight=0, int FillMode=BOX_nmppiFILL_FF){
743         nWidth=Width;
744         nHeight=Height;
745         nBorder=BorderHeight;
746         pBox =(T*)0;
747         pData=(T*)0xcbcbcbcb;
748         pHeap=0;
749         if (Allocate(MultiHeap))
750             Fill(FillMode);
751     }
752     void Lock(){
753         S_BufferInfo* Info=((S_BufferInfo*)pBox)-1;
754         Info->isLocked=true;
755     }
756     void Unlock(){
757         S_BufferInfo* Info=((S_BufferInfo*)pBox)-1;
758         Info->isLocked=false;
759     }
760     void Fill(int FillMode){
761         if (FillMode==BOX_nmppiFILL_NONE)
762             return;
763         if (FillMode==BOX_nmppiFILL_RAND){
764             nmppsRandUniform_(pBox,sizeBox);
765             return;
766         }
767         nmppsSet_(pBox,FillMode,sizeBox);
768     }
769
770     C_BoxImg(T* Data, int Width, int Height, int BorderHeight=0){
771         pHeap =0;
772         pData =Data;
773         nWidth =Width;
774         nHeight =Height;
775         nBorder =BorderHeight;
776         pBox =nmppsAddr_(pData,-nBorder*nWidth);
777         sizeData=nWidth*nHeight;
778         sizeBox =sizeData+2*nBorder*nWidth;
779     }
780     void Init(T* Data, int Width, int Height, int BorderHeight=0){
781         pHeap =0;
782         pData =Data;
783         nWidth =Width;
784         nHeight =Height;
785         nBorder =BorderHeight;
786         pBox =nmppsAddr_(pData,-nBorder*nWidth);
787         sizeData=nWidth*nHeight;
788         sizeBox =sizeData+2*nBorder*nWidth;
789     }
790     T* Addr(int y, int x){
791         return nmppsAddr_(pData,y*nWidth+x);
792     }
793
794     T* Allocate(C_MultiHeap& MultiHeap, int Width, int Height, int BorderHeight=0){
795         nWidth=Width;
796         nHeight=Height;
797         nBorder=BorderHeight;
798         return Allocate(MultiHeap);
799     }
800 }
801
802     T* Allocate(C_MultiHeap& MultiHeap){
803         if (MultiHeap.Check())
804             return 0;
805         pHeap =&MultiHeap;
806         pData =0;
807         sizeData=nWidth*nHeight;
808         sizeBox =sizeData+2*nBorder*nWidth;
809         int sizeBox32=(int)nmppsAddr_((T*)0,sizeBox)/sizeof(int);
810         pBox=(T*)MultiHeap.Allocate(sizeBox32);
811         if (pBox)
812             pData=nmppsAddr_(pBox,(sizeBox-sizeData)/2);
813         return pData;
814     }

```

```

815     int Release(){
816         if (pHeap)
817             if (pBox){
818                 pHap->Release(pBox);
819                 pHeap=0;
820                 //pBox=0;
821                 //pData=0;
822             }
823         return 1;
824     }
825     __INLINE__ T* operator [] (int idx){
826         return nmppsAddr_(pData, idx);
827     }
828     ~C_BoxImg() {
829         if (pHeap)
830             if (pBox)
831                 pHap->Release(pBox);
832     }
833 }
834
835 };
836
837 #endif

```

7.11 Файл g:/nmpp/include/nmblas.h

nmblas (NeuroMatrix Basic Linear Algebra Subroutines)

```
#include <nmblas/nmblas_sgemm.h>
```

Перечисления

- enum nm_trans { nm_n =0 , nm_t =1 }

Функции

- double **nmblas_dasum** (const int N, const double *X, const int INCX)
takes the sum of the absolute values <>
- void **MullMatrix_f** (void *A, int pI, int ldA, void *B, int pK, int ldB, void *C, int pJ, int ldC, int plusC)
- void **nmblas_daxpy** (const int N, const double A, const double *X, const int INCX, double *Y, const int INCY)
constant times a vector plus a vector <>
- void **nmblas_dcopy** (const int N, const double *X, const int INCX, double *Y, const int INCY)
copies a vector, X, to a vector, Y
- double **nmblas_ddot** (const int N, const double *X, const int INCX, const double *Y, const int INCY)

forms the dot product of two vectors.

- double `nmblas_dnrm2` (const int N, const double *x, const int INCX)
returns the euclidean norm of a vector via the function name, so that SCNRM2 := sqrt(x**H*x)

- void `nmblas_drot` (const int N, double *X, const int INCX, double *Y, const int INCY, const double C, const double S)
applies a plane rotation <>

- void `nmblas_drotg` (double *A, double *B, double *C, double *S)
construct givens plane rotation <>

- void `nmblas_drotm` (const int N, double *X, const int INCX, double *Y, const int INCY, double *param)
Apply a Given's rotation constructed by DROTMG. <>

- void `nmblas_dscal` (const int N, const double ALPHA, double *X, const int INCX)
scales a vector by a constant <>

- double `nmblas_dsdot` (const int N, const float *X, const int INCX, const float *Y, const int INCY)
Compute the inner product of two vectors with double precision accumulation. <>

- void `nmblas_dswap` (const int N, double *X, const int INCX, double *Y, const int INCY)
interchanges two vectors <>

- double `nmblas_dznrm2` (const int N, const double *X, const int INCX)
returns the euclidean norm of a vector via the function name, so that DZNRM2 := sqrt(x**H*x) <>

- int `nmbblas_idamax` (const int N, const double *X, const int INCX)
finds the index of the first element having maximum absolute value.

- int `nmbblas_isamax` (const int N, const float *X, const int INCX)
finds the index of the first element having maximum absolute value.

- float `nmbblas_sasum` (const int N, const float *X, const int INCX)
takes the sum of the absolute values

- void `nmbblas_saxpy` (int N, const float A, const float *X, const int INCX, float *Y, const int INCY)
constant times a vector plus a vector

- float `nmbblas_scnrm2` (const int N, const float *X, const int INCX)
returns the euclidean norm of a vector via the function name, so that SCNRM2 := sqrt(x**H*x)

- void `nmbblas_scopy` (const int N, const float *X, const int INCX, float *Y, const int INCY)
copies a vector, X, to a vector, Y

- float `nmbblas_sdot` (const int N, const float *X, const int INCX, const float *Y, const int INCY)
SDOT forms the dot product of two vectors.

- float `nmbblas_sdsdot` (const int N, const float B, const float *X, const int INCX, const float *Y,
const int INCY)
Compute the inner product of two vectors with double precision accumulation. <>

- float `nmbblas_snrm2` (const int N, const float *X, const int INCX)

SNRM2 returns the euclidean norm of a vector via the function name, so that SNRM2 := $\sqrt{x'x}$.
<>

- void **nmblas_srot** (const int N, float *X, const int INCX, float *Y, const int INCY, const float C, const float S)
applies a plane rotation <>
- void **nmblas_srotm** (const int N, float *X, const int INCX, float *Y, const int INCY, float *param)
Apply a Given's rotation constructed by SROTMG. <>
- void **nmblas_sscal** (const int N, const float A, const float *X, const int INCX)
scales a vector by a constant <>
- void **nmblas_sswap** (const int N, const float *X, const int INCX, const float *Y, const int INCY)
interchanges two vectors <>
- void **nmblas_sgenv** (const enum nm_trans TRANS, const int M, const int N, const float ALPHA, const float *A, const int LDA, const float *X, const int INCX, const float BETA, float *Y, const int INCY)
performs one of the matrix-vector operations $y := \alpha A \cdot x + \beta y$, or $y := \alpha A^T \cdot x + \beta y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
- void **nmblas_dgemv** (const enum nm_trans TRANS, const int M, const int N, const double ALPHA, const double *A, const int LDA, const double *X, const int INCX, const double BETA, double *Y, const int INCY)
performs one of the matrix-vector operations $y := \alpha A \cdot x + \beta y$, or $y := \alpha A^T \cdot x + \beta y$, where alpha and beta are scalars, x and y are vectors and A is an m by n matrix.
- void **nmblas_sger** (const int M, const int N, const float ALPHA, const float *X, const int INCX, const float *Y, const int INCY, float *A, const int LDA)
perform the rank 1 operation $A := \alpha x y' + A$, where alpha is a scalar, x is an m element vector, y is an n element vector and A is an m by n matrix.

7.11.1 Подробное описание

nblas (NeuroMatrix Basic Linear Algebra Subroutines)

This class is used to demonstrate a number of section commands.

Автор

Alexandr Bolotnikov

Версия

1.0

Дата

2017-05-23T15:13:13

Предупреждения

Авторство

(c) RC Module Inc.

7.12 nblas.h

[См. документацию.](#)

```

1
16
18
22
24
25
26 #ifndef _NMBLAS_H_INCLUDED_
27 #define _NMBLAS_H_INCLUDED_
28
29 #ifdef __cplusplus
30     extern "C" {
31 #endif
32
33
54 double nblas_dasum(
55     const int N,
56     const double *X,
57     const int INCX
58 );
60
61 enum nm_trans{nm_n=0,nm_t=1};
62
63
89 void nblas_daxpy(
90     const int N,
91     const double A,
92     const double *X,
93     const int INCX,
94     double *Y,
95     const int INCY
96 );
98
99

```

```
123 void nmblas_dcopy(
124     const int N,
125     const double* X,
126     const int INCX,
127     double* Y,
128     const int INCY
129 );
131
156 double nmblas_ddot(
157     const int N,
158     const double* X,
159     const int INCX,
160     const double* Y,
161     const int INCY
162 );
164
184 double nmblas_dnrm2(
185     const int N,
186     const double *x,
187     const int INCX
188 );
190
217 void nmblas_drot(
218     const int N,
219     double *X,
220     const int INCX,
221     double *Y,
222     const int INCY,
223     const double C,
224     const double S
225 );
227
238 void nmblas_drotg(
239     double *A,
240     double *B,
241     double *C,
242     double *S
243 );
245
260 void nmblas_drotm(
261     const int N,
262     double *X,
263     const int INCX,
264     double *Y,
265     const int INCY,
266     double *param
267 );
269
290 void nmblas_dscal(
291     const int N,
292     const double ALPHA,
293     double*X,
294     const int INCX
295 );
297
325 double nmblas_dsdot(
326     const int N,
327     const float* X,
328     const int INCX,
329     const float* Y,
330     const int INCY
331 );
333
348 void nmblas_dswap(
349     const int N,
350     double *X,
351     const int INCX,
352     double *Y,
353     const int INCY
354 );
356
376 double nmblas_dznrm2(
377     const int N,
378     const double *X,
379     const int INCX
380 );
382
402 int nmblas_idamax(
403     const int N,
404     const double* X,
405     const int INCX
406 );
409
429 int nmblas_isamax(
430     const int N,
431     const float* X,
432     const int INCX
```

```
433 );
435
455 float nmblas_sasum(
456     const int N,
457     const float *X,
458     const int INCX
459 );
461
486 void nmblas_saxpy(
487     int N,
488     const float A,
489     const float *X,
490     const int INCX,
491     float *Y,
492     const int INCY
493 );
495
515 float nmblas_scnrm2(
516     const int N,
517     const float* X,
518     const int INCX
519 );
521
544 void nmblas_scopy(
545     const int N,
546     const float* X,
547     const int INCX,
548     float* Y,
549     const int INCY
550 );
552
576 float nmblas_sdot(
577     const int N,
578     const float *X,
579     const int INCX,
580     const float *Y,
581     const int INCY
582 );
584
603 float nmblas_sdsdot(
604     const int N,
605     const float B,
606     const float* X,
607     const int INCX,
608     const float* Y,
609     const int INCY
610 );
612
632 float nmblas_snrm2(
633     const int N,
634     const float* X,
635     const int INCX
636 );
638
653 void nmblas_srot(
654     const int N,
655     float *X,
656     const int INCX,
657     float *Y,
658     const int INCY,
659     const float C,
660     const float S
661 );
663
678 void nmblas_srotm(
679     const int N,
680     float *X,
681     const int INCX,
682     float *Y,
683     const int INCY,
684     float *param
685 );
687
707 void nmblas_sscal(
708     const int N,
709     const float A,
710     const float *X,
711     const int INCX
712 );
714
737 void nmblas_sswap(
738     const int N,
739     const float *X,
740     const int INCX,
741     const float *Y,
742     const int INCY
743 );
```

```

745
747
748 //defgroup sgemv nmblas_sgmv
788
789 void nmblas_sgmv(
790   const enum nm_trans      TRANS,
791   const int      M,
792   const int      N,
793   const float    ALPHA,
794   const float    * A,
795   const int      LDA,
796   const float    * X,
797   const int      INCX,
798   const float    BETA,
799   float          * Y,
800   const int      INCY
801 );
803
804 //defgroup dgemv nmblas_dgemv
844
845 void nmblas_dgemv(
846   const enum nm_trans      TRANS,
847   const int      M,
848   const int      N,
849   const double   ALPHA,
850   const double   * A,
851   const int      LDA,
852   const double   * X,
853   const int      INCX,
854   const double   BETA,
855   double         * Y,
856   const int      INCY
857 );
859
860 //defgroup sger nmblas_sger
902
903 void nmblas_sger(
904   const int      M,//row
905   const int      N,//coullum
906   const float    ALPHA,
907   const float    * X,
908   const int      INCX,
909   const float    * Y,
910   const int      INCY,
911   float          * A,
912   const int      LDA
913 );
915 #include<nmblas/nmblas_sgemm.h>
916
917 void NullMatrix_f(
918   void*  A,
919   int    pI,
920   int    ldA,
921   void*  B,
922   int    pK,
923   int    ldB,
924   void*  C,
925   int    pJ,
926   int    ldc,
927   int    plusC
928 );
929
930
931 #ifdef __cplusplus
932 };
933 #endif
934
935 #endif // _INIT_H_INCLUDED_

```

7.13 Файл g:/nmpp/include/nmblas/nmblas_sgemm.h

part of nmblas library, sgemm function implementation

7.13.1 Подробное описание

part of nmblas library, sgemm function implementation

single precision matrix multiplication implementation

$C = \text{alpha} * A * B + \text{beta} * C$

see blas\cblas docs for detailed description

limitations: M, N, K, ld* must be all even, all matrix rows must be even aligned

limitations: row major (C order, not Fortran) only

limitations: TransA is not supported yet

Автор

leshabirukov

Версия

1.0

Дата

2018-07-26

Аргументы

in	TransA	transpose first multiplicand (A) flag
in	TransB	transpose second multiplicand (B) flag
in	M	number of rows of the matrix A and C
in	N	number of columns of the matrix B and C
in	K	number of rows of the matrix B and of columns of the matrix C
in	alpha	specifies the scalar alpha
in	A	pointer to the first multiplicand
in	lda	A stride, first dimension of A holder
in	B	pointer to the second multiplicand
in	ldb	B stride, first dimension of B holder
in	beta	specifies the scalar beta
in,out	C	pointer to accumulator matrix
in	ldc	C stride, first dimension of C holder

Предупреждения

Авторство

(c) RC Module Inc.

7.14 nmblas_sgemm.h

[См. документацию.](#)

```

1
35
37
38 #ifdef __NM
39 #ifdef __GNUC__
40 #ifndef NMBLAS__SGEMM_H_
41 #define NMBLAS__SGEMM_H_
42
43 // aux macros
44 #define ALL_FPU( instr ) "fpu 0" instr "\n\t" \
45           "fpu 1" instr "\n\t" \
46           "fpu 2" instr "\n\t" \
47           "fpu 3" instr "\n\t"
48
49 // aux functions
50
51 // load old C for specific fpu
52 static inline __attribute__((always_inline)) void
53 loadCFromMemory( const float* pc, int ldc, const int fpu, int* dummy_to_link )
54 {
55     asm (
56         "fpu %4 rep vlen vreg7=[%1++%2];\n\t"
57         : "+m"(*dummy_to_link)
58         : "RA0"( pc + fpu*2 ), "RG0"(ldc), "m" (*(const float (*)[])pc), "i"(fpu) );
59 }
60
61 // load A for all fpus
62 static inline __attribute__((always_inline)) void
63 loadAFromMemory( const float* pa, int lda, const int vrNum, int* dummy_to_link )
64 {
65     asm (
66         "fpu 0 rep vlen vreg%3=[%1++%2];\n\t"
67 // All fpus got the same vector
68         "fpu 1 vreg%3=fpu 0 vreg%3;\n\t"
69         "fpu 2 vreg%3=fpu 1 vreg%3;\n\t"
70         "fpu 3 vreg%3=fpu 2 vreg%3;\n\t"
71         : "+m" (*dummy_to_link), "+RA0" (pa)
72         : "RG0"(lda), "i"(vrNum), "m"(*(const float (*)[])pa) );
73 }
74
75 static inline __attribute__((always_inline)) void
76 loadBAndMAAdd( const float* pb, const float* pb1, int ldb, const int vrNum, int* dummy_to_link )
77 {
78     asm (
79         "fpu 0 rep 1 vreg4=[%1++];\n\t"
80         "fpu 0 rep 1 vreg5=[%2++];\n\t"
81         "fpu 1 rep 1 vreg4=[%1++];\n\t"
82         "fpu 1 rep 1 vreg5=[%2++];\n\t"
83         "fpu 2 rep 1 vreg4=[%1++];\n\t"
84         "fpu 2 rep 1 vreg5=[%2++];\n\t"
85         "fpu 3 rep 1 vreg4=[%1++];\n\t"
86         "fpu 3 rep 1 vreg5=[%2++];\n\t"
87         ALL_FPU( ".matrix vreg7=vreg%3 *.retrieve (vreg4,vreg5) + vreg7;" )
88         : "+m" (*dummy_to_link), "+a" (pb), "+a" (pb1)
89         : "i"(vrNum), "m"(*(const float (*)[])pb), "m"(*(const float (*)[])pb1) );
90 }
91
92 // Same as loadBAndMAAdd, but without "+C"
93 static inline __attribute__((always_inline)) void
94 loadBAndMultiply( const float* pb, const float* pb1, int ldb, const int vrNum, int* dummy_to_link )
95 {
96     asm (
97         "fpu 0 rep 1 vreg4=[%1++];\n\t"
98         "fpu 0 rep 1 vreg5=[%2++];\n\t"
99         "fpu 1 rep 1 vreg4=[%1++];\n\t"
100        "fpu 1 rep 1 vreg5=[%2++];\n\t"
101        "fpu 2 rep 1 vreg4=[%1++];\n\t"
102        "fpu 2 rep 1 vreg5=[%2++];\n\t"
103        "fpu 3 rep 1 vreg4=[%1++];\n\t"
104        "fpu 3 rep 1 vreg5=[%2++];\n\t"
105        ALL_FPU( ".matrix vreg7=vreg%3 *.retrieve (vreg4,vreg5);"
106         : "+m" (*dummy_to_link), "+a" (pb), "+a" (pb1)
107         : "i"(vrNum), "m"(*(const float (*)[])pb), "m"(*(const float (*)[])pb1) );
108 }
109
110
111
112 static inline __attribute__((always_inline)) void
113 storeCToMemory( float* pc, int ldc, const int fpu, int* dummy_to_link )
114 {
115     asm (
116         "fpu %4 rep vlen [ar0+gr0]=vreg7;\n\t"

```

```

117      : "=>m"(*float (*[])(pc))
118      : "RA0"(pc), "RG0"(ldc), "m"(*dummy_to_link), "i"(fpu) );
119 }
120
121
122
123 static inline void
124 nmblas_sgemm( const enum nm_trans TransA,
125                 const enum nm_trans TransB,
126                 const int M,
127                 const int N,
128                 const int K,
129                 const float alpha,
130                 const float *A,
131                 const int lda,
132                 const float *B,
133                 const int ldb,
134                 const float _beta,
135                 float *C,
136                 const int ldc
137                 )
138 {
139     float beta= _beta;
140     if( TransA != nm_n || TransB != nm_n ){}
141
142     // Нижеследующие сравнения и деления должны быть локализованы строго до векторного кода,
143     // поскольку реализующие их интринсики испортят значения в векторных регистрах!
144     // { all float intrinsic calls must be here!
145     int beta0 = beta ==0.0f;
146     int alpha1 = alpha==1.0f;
147     int beta1;
148     if ( !beta0 ){
149         if ( !alpha1 && alpha !=0.0f )
150             beta /= alpha;
151         beta1 = beta ==1.0f;
152     }
153     // } all float intrinsic calls must be here!
154
155
156     const int I=M;
157     const int J=N;
158     int i, j, k;
159     int* dummy_to_link; // workaround to reflect dependence by vector registers
160
161     for(i=0; i<I; i+=32){
162         asm volatile(
163             "vlen= %0;\n\t"
164             :
165             : "g"( I-i-1 >= 31 ? 31 : I-i-1 ) );
166
167     for(j=0; j<J; j+=8){
168
169         k=0;
170         const float* pa;
171         const float* pb;
172         const float* pb1;
173         float bufScalar[2] __attribute__((aligned (8)));
174
175         asm("":="m"(*dummy_to_link),"=a"(dummy_to_link));
176
177         if ( !beta0 ){
178             // read C[i][j]
179             loadCFromMemory( C + i *ldc + j, ldc, 0, dummy_to_link );
180             loadCFromMemory( C + i *ldc + j, ldc, 1, dummy_to_link );
181             loadCFromMemory( C + i *ldc + j, ldc, 2, dummy_to_link );
182             loadCFromMemory( C + i *ldc + j, ldc, 3, dummy_to_link );
183             if ( !beta1 ){
184                 // C[i][j] *= beta
185                 bufScalar[0]=beta;
186                 bufScalar[1]=-beta;
187                 float* pbeta= bufScalar;
188                 asm (
189                     "fpu 0 rep 1 vreg4 = [%1++];\n\t"
190                     "fpu 1 vreg4 = fpu 0 vreg4;\n\t"
191                     "fpu 2 vreg4 = fpu 1 vreg4;\n\t"
192                     "fpu 3 vreg4 = fpu 2 vreg4;\n\t"
193                     ALL_FPU (".float vreg7=vreg7 * .retrieve (vreg4);"
194                     : "+m" (*dummy_to_link), "+a" (pbeta)
195                     : "m"(*bufScalar));
196                 }
197             }
198             else{
199                 pa = A + i*lda +k;
200                 pb = B + k*ldb +j;
201                 pb1 = B +(k+1)*ldb +j;
202
203                 asm("":="m"(*dummy_to_link));

```

```

204         loadAFromMemory( pa, lda, 0, dummy_to_link );
205
206         loadBAndMultiply( pb, pb1, ldb, 0, dummy_to_link );
207         k+=2;
208     }
209
210
211
212     for( ; k<K; k+=2){
213         //C[i][j] += A[i][k]*B[k][j];
214         pa = A + i*lda +k;
215         pb = B + k*ldb +j;
216         pb1 = B +(k+1)*ldb +j;
217
218         loadAFromMemory( pa, lda, 3, dummy_to_link );
219
220         loadBAndMAdd( pb, pb1, ldb, 3, dummy_to_link );
221
222         k+=2;
223         if( !( k<K ) )
224             break;
225         pa = A + i*lda +k;
226         pb = B + k*ldb +j;
227         pb1 = B +(k+1)*ldb +j;
228
229         loadAFromMemory( pa, lda, 0, dummy_to_link );
230
231         loadBAndMAdd( pb, pb1, ldb, 0, dummy_to_link );
232     }
233
234     if( !alpha1 ){
235         // C[i][j] *= alpha
236         bufScalar[0]=alpha;
237         bufScalar[1]=alpha;
238         float* palpha= bufScalar;
239         asm(
240             "fpu 0 rep 1 vreg4 = [%1++];\n\t"
241             "fpu 0 vreg4 = fpu 0 vreg4;\n\t"
242             "fpu 2 vreg4 = fpu 1 vreg4;\n\t"
243             "fpu 3 vreg4 = fpu 2 vreg4;\n\t"
244             ALL_FPU ( ".float vreg7= vreg7 * .retrieve (vreg4);"
245             : "r+m" (*dummy_to_link), "+a" (palpha)
246             : "m"(*bufScalar) );
247         }
248
249         // write C[i][j]
250         storeCToMemory( C + i * ldc +j+0, ldc, 0, dummy_to_link );
251         if( J-j<=2 )
252             break;
253         storeCToMemory( C + i * ldc +j+2, ldc, 1, dummy_to_link );
254         if( J-j<=4 )
255             break;
256         storeCToMemory( C + i * ldc +j+4, ldc, 2, dummy_to_link );
257         if( J-j<=6 )
258             break;
259         storeCToMemory( C + i * ldc +j+6, ldc, 3, dummy_to_link );
260     }
261 }
262 }
263
264
265
266 #endif /* NUBLAS_SGEMM_H */
267 #endif /* _GNUC_ */
268 #endif /* __NM__ */

```

7.15 nmchar.h

```

1 #ifndef NMCHAR
2 #define NMCHAR
3
4 #include <stdlib.h>
5 #include <string.h>
6 //extern unsigned int crc;
7 #ifndef inline
8 #define inline inline
9 #endif
10
11 class uint8ptr;
12 #define nmcharptr uint8ptr
13 class nmchar{
14 public:
15     unsigned int *adr;
16     int idx;

```

```

17     nmchar(){
18         adr=0;
19         idx=0;
20     }
21     nmchar(unsigned int*p, int offset) {
22         adr = p + (offset >> 2);
23         idx = offset & 3;
24     }
25
26     nmchar(nmchar& ch){
27         adr=ch.adr;
28         idx=ch.idx;
29 //        crc^=ch;
30     }
31     nmchar& operator = (nmchar ch){
32         unsigned char val=(ch);
33         (*adr) &= ~(0xFF<<(idx*8));
34         (*adr) |= (val) <<(idx*8);
35 //        crc^=ch;
36 //        adr=ch.adr;
37 //        idx=ch.idx;
38         return *this;
39     }
40     unsigned int operator + (nmchar& ch){
41         unsigned int a=(ch);
42         unsigned int b=(*this);
43         a+=b;
44 //        crc^=a;
45         return a;
46     }
47
48     nmchar& operator &= (unsigned int val) {
49         //(*adr) &= 0xFFFFFFFF & ((0xFF & val) << (idx * 8));
50         val = (~val) & 0xFF;
51         (*adr) &= ~(val << (idx * 8));
52         return *this;
53     }
54
55     inline nmchar& operator |= (unsigned int val) {
56         (*adr) |= ((0xFF & val) << (idx * 8));
57         return *this;
58     }
59
60     inline nmchar& operator = (unsigned int val){
61         (*adr) &= ~(0xFF<<(idx*8));
62         (*adr) |= (val&0xFF) <<(idx*8);
63 //        crc^=val;
64         return *this;
65     }
66     inline operator unsigned char(){
67         unsigned char ret=(*adr)<<(idx*8);
68         ret&=0xFF;
69 //        crc^=ret;
70         return ret;
71     }
72
73     inline uint8ptr operator &();
74
75 };
76
77
78
79 class uint8ptr {
80 public:
81     unsigned int *addr;
82     int idx;
83     nmchar arref;
84     inline uint8ptr (){
85         addr=0;
86         idx=0;
87     }
88     //uint8ptr(uint8ptr& p){
89     //    idx=p.idx;
90     //    addr=p.addr;
91     //}
92
93     //uint8ptr(uint8ptr& p){
94     //    idx=p.idx;
95     //    addr=p.addr;
96     //}
97
98     inline unsigned char* x86addr(){
99         return ((unsigned char*)addr)+idx;
100    }
101    inline uint8ptr (const void* p){
102        addr=(unsigned int*)p;
103        idx=0;

```

```

104    }
105   inline uint8ptr (const unsigned char* p){
106     addr=(unsigned int*)p;
107     idx=0;
108   }
109   //inline uint8ptr (unsigned char* p){
110   //  addr=(unsigned int*)p;
111   //  idx=0;
112   //}
113   inline uint8ptr (const char* p){
114     addr=(unsigned int*)p;
115     idx=0;
116   }
117
118   inline uint8ptr(const uint8ptr& p){
119     idx=p.idx;
120     addr=p.addr;
121   }
122   //uint8ptr(int* p){
123   //  addr=(unsigned*)p;
124   //  idx=0;
125   //}
126   inline uint8ptr(unsigned int*p){
127     addr=p;
128     idx=0;
129   }
130   inline uint8ptr(unsigned int*p,int offset){
131     addr=p+(offset>>2);
132     idx=offset&3;
133   }
134
135   //operator int(){
136   //  return int(addr);
137   //}
138
139   inline nmchar & operator [](int idx){
140
141     arref.adr = addr + (idx + idx) / 4;
142     arref.idx = (idx + idx) % 4;
143     //nmchar arref(addr+(idx+idx)/4,(idx+idx)%4);
144     return arref;
145   }
146
147   //uint8ptr& operator = (unsigned ptr){
148   //  addr=(unsigned*)ptr;
149   //  idx=0;
150   //  return (*this);
151   //}
152
153
154   inline uint8ptr& operator = (unsigned int* ptr){
155     addr=(unsigned*)ptr;
156     idx=0;
157     return (*this);
158   }
159
160   inline bool operator < (uint8ptr ptr){
161 #ifdef NM
162     if (addr<ptr.addr)
163       return 1;
164     if (addr==ptr.addr)
165       if (idx<ptr.idx)
166         return 1;
167     return 0;
168 #else
169     return ((unsigned long long)(addr) + idx < (unsigned long long)(ptr.addr) + ptr.idx);
170 #endif
171   }
172   inline bool operator > (uint8ptr ptr){
173 #ifdef NM
174     if (addr>ptr.addr)
175       return true;
176     if (addr==ptr.addr)
177       if (idx>ptr.idx)
178         return false;
179     return 0;
180 #else
181     return ((unsigned long long)(addr) + idx > (unsigned long long)(ptr.addr) + ptr.idx);
182 #endif
183   }
184
185   inline bool operator >= (uint8ptr ptr){
186 #ifdef NM
187     if (addr>ptr.addr)
188       return 1;
189     if (addr==ptr.addr)
190       if (idx>=ptr.idx)

```

```

191         return 1;
192     return 0;
193 #else
194     return ((unsigned long long)(addr) + idx >= (unsigned long long)(ptr.addr) + ptr.idx);
195 #endif
196 }
197
198
199     inline int operator - (uint8ptr ptr){
200
201
202 #ifdef __NM__
203     int a=(int)addr;
204     int b=(int)ptr.addr;
205     //a&=0xFFFFFFFF;
206     //b&=0xFFFFFFFF;
207     a<<=2;
208     b<<=2;
209     a+=idx;
210     b+=ptr.idx;
211     return a-b;
212 #else
213     //a+=idx;
214     //b+=ptr.idx;
215     return (unsigned long long)addr-(unsigned long long)ptr.addr+idx-ptr.idx;
216 #endif
217 }
218
219
220
221 // uint8ptr& operator = (unsigned char* ptr){
222 //     addr=(unsigned int*)ptr;
223 //     idx=0;
224 //     return (*this);
225 // }
226
227
228
229     inline uint8ptr& operator = (const uint8ptr& p){
230     addr=p.addr;
231     idx=p.idx;
232     return (*this);
233 }
234
235     inline unsigned int* ptr(){
236     if (idx==0)
237         return addr;
238     //}
239     else {
240         int g=1;
241         return (unsigned int*)-1;
242     }
243 }
244     inline nmchar& operator *(){
245
246     return (nmchar&)(*this);
247 }
248
249 /*
250 uint8ptr operator + (int idx){
251 uint8ptr tmp(*this);
252 tmp.addr=addr+(idx+idx)/4;
253 tmp.idx=(idx+idx)&0x3;
254 return tmp;
255 }
256
257 uint8ptr operator- (int idx){
258 uint8ptr tmp(*this);
259 tmp.addr=addr+(idx-idx)/4;
260 tmp.idx=(idx-idx)&0x3;
261 return tmp;
262 }
263
264 uint8ptr& operator+= (int idx){
265 addr=addr+(idx+idx)/4;
266 idx=(idx+idx)&0x3;
267 return *this;
268 }
269
270 uint8ptr& operator-= (int idx){
271 addr=addr+(idx-idx)/4;
272 idx=(idx-idx)&0x3;
273 return *this;
274 }
275 */
276     inline uint8ptr operator + (int idx){
277     uint8ptr tmp(*this);

```

```

278     tmp.addr=addr+((indx+idx)»2);
279     tmp.indx=(indx+idx)&0x3;
280     return tmp;
281 }
282
283 inline uint8ptr operator- (int idx){
284     uint8ptr tmp(*this);
285     tmp.addr=addr+((indx-idx)»2);
286     tmp.indx=(indx-idx)&0x3;
287     return tmp;
288 }
289
290
291 inline uint8ptr& operator+= (int idx){
292     addr=addr+((indx+idx)»2);
293     indx=(indx+idx)&0x3;
294     return *this;
295 }
296
297 inline uint8ptr& operator-= (int idx){
298     addr=addr+((indx-idx)»2);
299     indx=(indx-idx)&0x3;
300     return *this;
301 }
302
303 inline uint8ptr operator++ (int){
304     uint8ptr tmp(*this);
305     if (indx==3){
306         indx=0;
307         addr++;
308     }
309     else {
310         indx++;
311     }
312     return tmp;
313 }
314
315 inline uint8ptr& operator++ () {
316     if (indx==3){
317         indx=0;
318         addr++;
319     }
320     else {
321         indx++;
322     }
323     return *this;
324 }
325
326 //inline unsigned int operator== (unsigned int N){
327 //    return (((unsigned int)addr)==N);
328 //}
329
330 inline bool operator == (uint8ptr ptr){
331     if (addr==ptr.addr && indx==ptr.indx)
332         return 1;
333     return 0;
334 }
335
336 };
337
338 inline uint8ptr nmchar::operator &(){
339     uint8ptr p;
340     p.addr=addr;
341     p.indx=indx;
342     return p;
343 }
344
345 template <int Y,int X> class nmchar2D {
346 public:
347     uint8ptr arr;
348     unsigned int data[Y*X/4];
349
350     nmchar2D(){
351     }
352
353     uint8ptr& operator [] (int idx){
354         arr.addr=data+X*idx/4;
355         arr.indx=0;
356         return arr;
357     }
358     unsigned int* ptr(){
359         return data;
360     }
361 }
362
363 template<int N> class nmchar1D {
364 public:

```

```

365     nmchar deref;
366     unsigned int data[(N+3)/4];
367
368     nmchar1D(){
369 }
370
371     inline nmchar& operator [](int idx){
372         deref.adr=data+idx/4;
373         deref.idx=idx%4;
374
375         return deref;
376     }
377
378     inline operator uint8ptr(){
379         uint8ptr p;
380         p.addr=data;
381         p.idx=0;
382         return p;
383     }
384     inline unsigned int* ptr(){
385         return data;
386     }
387
388
389 };
390
391 void free(uint8ptr& p);
392 /**
393 // free(p.addr);
394 */
395
396 void nmchar_memcpy (uint8ptr dst, uint8ptr src, unsigned int len);
397 /**
398 //
399 //
400 // #ifdef __NM__
401 //     memcpy((void*)dst.x86addr(), (void*)src.x86addr(),len);
402 // #else
403 //     if (dst.idx==src.idx){
404 //         // copy n bytes to align pointers
405 //         if (dst.idx){
406 //             int n=4-dst.idx;
407 //             if (len<n)
408 //                 n=len;
409 //             for(int i=0; i<n; i++){
410 //                 *dst=*src;
411 //                 dst++;
412 //                 src++;
413 //             }
414 //             len-=n;
415 //         }
416 //         int words=len»2;
417 //         memcpy(dst.addr, src.addr, words);
418 //         dst.addr+=words;
419 //         src.addr+=words;
420 //         len%=4;
421 //         // copy rest bytes
422 //         for(int i=0;i<len;i++){
423 //             *dst=*src;
424 //             dst++;
425 //             src++;
426 //         }
427 //     }
428 //     else {
429 //         for(int i=0; i<len; i++){
430 //             *dst=*src;
431 //             src++;
432 //             dst++;
433 //         }
434 //     }
435 // }
436 // #endif
437 //}
438 //
439 void nmchar_memset (uint8ptr dst, int setvalue, unsigned int len);
440 /**
441 //
442 //
443 // #ifdef __NM__
444 //     memset((void*)dst.x86addr(), setvalue,len);
445 // #else
446 //
447 //     // copy n bytes to align pointers
448 //     if (dst.idx){
449 //         int n=4-dst.idx;
450 //         if (len<n)
451 //             n=len;

```

```

452 //      for(int i=0; i<n; i++){
453 //          *dst=setvalue;
454 //          dst++;
455 //      }
456 //      len-=n;
457 //  }
458 //  int words=len»2;
459 //
460 //  setvalue&=0xFF;
461 //  setvalue|=(setvalue«8);
462 //  setvalue|=(setvalue«16);
463 //  memset(dst.addr, setvalue, words);
464 //  dst.addr+=words;
465 //  len%==4;
466 //  // copy rest bytes
467 //  setvalue&=0xFF;
468 //  for(int i=0;i<len;i++){
469 //      *dst=setvalue;
470 //      dst++;
471 //  }
472 //
473 //
474 // #endif
475 //}
476 //
477
478 #undef inline
479 #endif

```

7.16 nmdef.h

```

1
2 #ifndef __NMDEF_H
3 #define __NMDEF_H
4
5 #ifdef __NM
6 #ifndef NMDEF
7 #define NMDEF
8 #endif
9 #endif
10
11 #ifdef NM6403
12 #ifndef NMDEF
13 #define NMDEF
14 #endif
15 #endif
16
17
18 #ifndef NULL
19 #define NULL 0
20 #endif
21 /*
22 #ifndef ADDR
23 #ifndef NM
24 #define ADDR(base,disp) \
25 ( sizeof(*(base)) == 1 ? nmppsAddr((base),(disp)): \
26 ((base)+(disp)) \
27 )
28 #else
29 #define ADDR(base,disp) ((base)+(disp))
30 #endif //NMDEF
31 #endif //ADDR
32 */
33 #ifndef MIN
34 #define MIN(a,b) ((a) > (b) ? (b) : (a))
35 #endif
36
37 #ifndef MAX
38 #define MAX(a,b) ((a) < (b) ? (b) : (a))
39 #endif
40
41 #ifndef ABS
42 #define ABS(a) ((a) < (0) ? -(a): (a))
43 #endif
44
45 #ifndef ROUND2INT
46 #define ROUND2INT(a) ((a) >= 0 ? (int((a)+0.5)) : (int((a)-0.5)))
47 #endif
48
49 #ifndef NM
50 #define GET_CHAR(pArray,Index,val)      (nmppsGetVal_((pArray),(Index),(val)))
51 #define GET_SHORT(pArray,Index,val)     (nmppsGetVal_((pArray),(Index),(val)))
52 #define SET_CHAR(pArray,Index,Char)    (nmppsPut_((pArray),(Index),(Char)))
53 #define SET_SHORT(pArray,Index,Short)  (nmppsPut_((pArray),(Index),(Short)))

```

```

54 #else
55 #define GET_CHAR(pArray,Index,val)      ((val)=(pArray)[(Index)])
56 #define GET_SHORT(pArray,Index,val)     ((val)=(pArray)[(Index)])
57 #define SET_CHAR(pArray,Index,Char)     ((pArray)[(Index)]=(Char))
58 #define SET_SHORT(pArray,Index,Short)   ((pArray)[(Index)]=(Short))
59#endif
60
61#define LONG2INT(size)    ((size)«1)
62
63#define long2INT(size)    ((size)«1)
64#define int2INT(size)     ((size))
65#define short2INT(size)   ((size)«1)
66#define char2INT(size)    ((size)«2)
67
68
69#define LONG2long(size)   ((size))
70#define LONG2int(size)    ((size)«1)
71#define LONG2short(size)  ((size)«2)
72#define LONG2char(size)   ((size)«3)
73
74#define long2long(size)   ((size))
75#define long2int(size)    ((size)«1)
76#define long2short(size)  ((size)«2)
77#define long2char(size)   ((size)«3)
78
79#define int2long(size)    ((size)«1)
80#define int2int(size)     ((size))
81#define int2short(size)   ((size)«1)
82#define int2char(size)    ((size)«2)
83
84#define short2long(size)  ((size)«2)
85#define short2int(size)   ((size)«1)
86#define short2short(size) ((size))
87#define short2char(size)  ((size)«1)
88
89#define char2long(size)   ((size)«3)
90#define char2int(size)    ((size)«2)
91#define char2short(size)  ((size)«1)
92#define char2char(size)   ((size))
93
94
95#endif

```

7.17 cArithmetic.h

```

1 ifndef _CARITHMETIC_H_INCLUDED_
2 define _CARITHMETIC_H_INCLUDED_
3
4 include "nmtype.h"
5
6
7
27 void nmppcDivC(nm64sc *pnSrcA, nm64s *pnSrcB, nm64sc *Dst);
29
49 void nmppcProdC(nm64sc *pnSrcA, nm64sc *pnSrcB, nm64sc *Dst); //ASM
51
52#endif

```

7.18 cfixpnt32.h

```

1 //*****
2 /* */ *
3 /*          Neuroprocessor NM3203 SDK v.1.2 */ *
4 /* */ *
5 /*          Заголовочный файл fixpoint32 функций */ *
6 /*          1999 (c) RC Module Inc., Moscow, Russia */ *
7 /* */ *
8 /* */ *
9 /* Date:28.12.99 */ *
10 //*****
11 #ifdef __cplusplus
12 extern "C" {
13#endif
14
15
16 //=====
42
43 int nmppcFixExp32( int nVal);

```

```

45      //=====
77 void nmppcFixSinCos32(int nArg, int* pnSin, int* pnCos);
80
81      //=====
107 int nmppcFixArcTan32(int nArg);
110
111
112      //=====
143 int nmppcDoubleToFix32(double arg);
146
147
148      //=====
149
150
177 double nmppcFix32toDouble(int arg);
179      //=====
205
206     unsigned int nmppcFixSqrt32( unsigned int nVal );
208
209      //=====
210
242     int nmppcFixMul32(int nX, int nY);
244
245      //=====
276
277     int nmppcFixDiv32(int nX, int nY);
279      //=====
315
316     int nmppcFixInv32(int nVal, int nFixpoint);
318
319      //=====
320
347     int nmppcTblFixArcSin32(int nArg);
349      //=====
350
377     int nmppcTblFixArcCos32(int nArg);
379
380
381      //=====
382
409     int nmppcTblFixCos32(int nArg);
411
412      //=====
413
440     int nmppcTblFixSin32(int nArg);
442
443      //=====
444
482     void nmppcFixDivMod32(int nDividend, int nDivisor, int* pnQuotient, int* pnReminder);
483     void nmppcFixDivPosMod32(unsigned int nDividend, unsigned int nDivisor, int* pnQuotient, int* pnReminder);
485 #ifndef __cplusplus
486     };
487 #endif
488

```

7.19 cfixpnt64.h

```

1 //*****
2 /*          Neuroprocessor NM6403 SDK v.1.2          */
3 /*          Заголовочный файл fixpoint64 функций        */
4 /*          1999 (c) RC Module Inc., Moscow, Russia       */
5 /*          */                                           */
6 /*          */                                           */
7 /*          */                                           */
8 /*          */                                           */
9 /*          Date:28.12.99                                */
10 //*****

```

```

11 #ifndef __cplusplus
12 extern "C" {
13 #endif
14
15
16 //=====
17
18 unsigned long nmppcFixSqrt64(unsigned long x);
19 //=====
20
21 long nmppcDoubleToFix64(double arg, int fixpoint);
22 //=====
23
24 double nmppcFix64.ToDouble(long arg , int fixpoint );
25 //=====
26
27 void nmppcFixDiv64(long* nDividend, long* nDivisor, int nFixpoint, long* nQuotient);
28 //    Функция деления A/B для целых чисел со знаком 2*abs(A)<abs(B)
29
30 //=====
31
32 void nmppcFixSinCos64(long nArg, long* pnSin, long *pnCos);
33 //=====
34
35 long nmppcFixArcTan64(long nArg);
36 //=====
37
38 long nmppcFix64Exp01(long nArg);
39
40 long nmppcFrExp(double nArg);
41
42 #ifndef __cplusplus
43     };
44 #endif
45
46 //=====
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

7.20 cInit.h

```

1 //-----
2 // $Workfile::: cInit. $
3 // Векторно-матричная библиотека
4 // Copyright (c) RC Module Inc.
5 // $Revision: 1.1 $      $Date: 2005/01/12 14:01:21 $
6 //-----
7 #ifndef _CINIT_H_INCLUDED_
8 #define _CINIT_H_INCLUDED_
9
10 //*****
11
12 #ifdef __cplusplus
13     extern "C" {
14 #endif
15
16 //*****
17
18 //*****
19
20
21
22
23 #ifdef __cplusplus
24     extern "C" {
25 #endif
26
27 //*****
28
29 //*****
30
31
32 int nmppcRandMinMaxDiv(int nMin, int nMax, int nDivisible);
33 int nmppcRandMinMax(int nMin, int nMax);
34 int nmppcRand();
35
36
37 #ifdef __cplusplus
38     };
39 #endif
40
41
42 #endif // _CINIT_H_INCLUDED_

```

7.21 cInteger.h

```

1 //***** ****
2 /**
3  *          Neuroprocessor NM6403 SDK v.1.2
4  */
5 /**
6  *          Заголовочный файл fixpoint64 функций
7  *          1999 (c) RC Module Inc., Moscow, Russia
8  */
9 /**
10 */
11
12 #ifdef __cplusplus
13 extern "C" {
14 #endif
15
16
17 //=====
18
44     unsigned int nmpcSqrt_64u(unsigned long long x);
45 //unsigned int nmpcSqrt_32u(unsigned int x);
46
47 //=====
48
49 //    Функция деления A/B для целых чисел со знаком 2*abs(A)<abs(B)
50 void IDivInv64(
51     long*      Dividend, // делимое
52     long*      Divisor, // делитель
53     const int   DivIterations, // кол-во итераций при пошаговом(побитовом) делении
54     long*      Quotient // частное
55 );
56 long Fix64Exp01(long arg);
57
58
59
60
61 #ifdef __cplusplus
62 }
63 #endif

```

7.22 nmplc.h

```
1 #include "./nmplc/nmplc.h"
```

7.23 nmplc.h

```

1
2 #ifndef INCLUDED_nmplc
3 #define INCLUDED_nmplc
4
5 #include "nmtype.h"
6 #include "nmdef.h"
7
8
9
43 #include "cInit.h"
44
45
55 #include "cInteger.h"
56
66 #include "cfixpnt64.h"
76 #include "cfixpnt32.h"
77
78 #include "cfixpnt64.h"
88 #include "cArithmetic.h"
89
90
91
92
93 #endif

```

7.24 filter.h

```
1 /*
```

```

2 #ifndef __cplusplus
3 extern "C" {
4 #endif
5 #include "nmptype.h"
6 #define NmppiFilterState nm64s
7
8 void nmppiFilter_8s16s( nm8s * pSrcImg, nm16s* pDstImg, int nWidth, int nHeight, NmppiFilterState* pKernel);
9 int nmppiFilterInit_8s16s(int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, NmppiFilterState* pKernel);
10 int nmppiFilterInitAlloc_8s16s(NmppiFilterState** ppState, int* pKernel, int kerWidth, int kerHeight, int imgWidth);
11
12 void nmppiFilter_16s32s( nm16s * pSrcImg, nm32s* pDstImg, int nWidth, int nHeight, NmppiFilterState* pKernel);
13 int nmppiFilterInit_16s32s(int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, NmppiFilterState* pKernel);
14 int nmppiFilterInitAlloc_16s32s(NmppiFilterState** ppState, int* pKernel, int kerWidth, int kerHeight, int imgWidth);
15
16 void nmppiFilterFree(NmppiFilterState* pState);
17
18
19
20
21
22 #ifdef __cplusplus
23 }
24 #endif
25 */

```

7.25 iArithmetics.h

```

1 #ifndef __IARITHMETICS_H
2 #define __IARITHMETICS_H
3 #ifndef ARITH
4 #include "iDef.h"
5
6
7 //*****
8
100 void nmppiAddConvertRShiftI(nm16s* pSrcDstImg, int nSrcDstStride, nm32s* pSrcImg, int nSrcStride, int nShift, int
nWidth, int nHeight);
102 //*****
103 //*****
104
106
197 void nmppiSubConvertRShiftI(nm16s* pSrcDstImg, int nSrcDstStride, nm32s* pSrcImg, int nSrcStride, int nShift, int
nWidth, int nHeight);
199 //*****
200 //*****
201
202
283 void nmppiAdd_16s(nm16s* pSrcDstImg, int nSrcDstStride, nm16s* pSrcImg, int nSrcStride, int nWidth, int nHeight);
285 //*****
286 //*****
287
288
371
372 void nmppiSub_16s(nm16s* pSrcDstImg, int nSrcDstStride, nm16s* pSrcImg, int nSrcStride, int nWidth, int nHeight);
373
375 //*****
376 //*****
377
471
472 void nmppiHalfsum(nm16s* pSrcMtr1, int nSrcStride1, nm16s* pSrcMtr2,
473     int nSrcStride2, nm16s* pDstMtr, int nDstStride, int nWidth, int nHeight);
474
476
477
534 void nmppiWAdd2I(RGB32_nm10s* pSrcImg1, int nMulVal1, RGB32_nm10s* pSrcImg2, int nMulVal2, int nAddVal,
    RGB32_nm10s* pDstImg, int nSize);
536
537
573 void nmppiRsh2(RGB32_nm10u* pSrcImg, RGB32_nm10u* pDstImg, int nSize);
575 #endif
576 #endif

```

7.26 iCellTexture.h

```

1 //-
2 //
3 // $Workfile::: CellTexture.h      $
4 //
5 // <Название библиотеки>
6 //
7 // Copyright (c) RC Module Inc.

```

```

8 //-
9 // $Revision: 1.1 $      $Date: 2005/02/10 12:36:38 $
10 //-
11 //-----
12 #ifndef __CELLTEXTURE_H
13 #define __CELLTEXTURE_H
14
15
16 //Класс определяет точку на плоскости
17 class RPoint {
18 public:
19     double x;
20     double y;
21
22
23
24     //virtual ~RPoint() {};
25
26
27     RPoint() : x(0), y(0)
28     {
29     }
30
31     RPoint(const RPoint &p) : x(p.x), y(p.y)
32     {
33     }
34     RPoint (double _x, double _y) :x(_x), y(_y)
35     {
36     }
37
38     RPoint &operator= (const RPoint &p)
39     {
40         x = p.x; y = p.y;
41         return (*this);
42     }
43
44
45
46
47
48
49
50
51
52
53 };
54
55
56
57
58 void nmppiCreateCellTexture(unsigned char *texture_img, int width, int height);
59 void nmppiCreateRandomCellTexture(unsigned char *texture_img, int width, int height);
60 void nmppiCreateComplexTexture(unsigned char *texture_img, int width, int height);
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110 //int PointInRectangle(RPoint& p, RPoint *borders);
111
112
113#endif

```

7.27 iConvert.h

```

1 //ifndef __ICONVERT_H
2 //define __ICONVERT_H
3 //
4 //include "nmtype.h"
5 //include "iDef.h"
6 //
7 //
8 //
9 //ifdef __cplusplus
10 //    extern "C" {
11 //endif
12 //
13 //
14 /**
15 //  */
16 //  \internal
17 //  \defgroup nmppiRShiftConvert nmppiRShiftConvert
18 //  \ingroup iInit
19 //  \brief
20 //      \ru Преобразование типов для элементов изображения.
21 //      \en Conversion of types for image elements.
22 //
23 //  \
24 //
25 //  \f[
26 //      pDst(x,y) = Convert(pSrcImg(x,y) >> nShift)
27 //      \f]
28 //
29 //  \f[ x = \overline{0 \ldots nWidth-1}, \quad
30 //      y = \overline{0 \ldots nHeight-1}
31 //  \f]
32 //
33 //  \ru Функция преобразует тип элементов путем отсечения верхних разрядов.

```

```

34 // \en The function converts a type of elements by means of most significant bits truncation.
35 //
36 // \
37 // \
38 // \param pSrcImg
39 //   \ru Указатель на первый элемент ROI для исходного изображения.
40 //   \en Pointer to the first ROI element for the source image.
41 // \
42 // \param nSrcStride \ru Межстроковое смещение для исходного изображения
43 //   (измеряется в 32-х разрядных словах).
44 //   \en Row-to-row shift for the source image \n
45 //   (measured in 32-bit words).
46 // \param pDstImg
47 //   \ru Указатель на первый элемент ROI для результирующего изображения.
48 //   \en Pointer to the first ROI element for the result image.
49 // \
50 // \param nDstStride \ru Межстроковое смещение для результирующего изображения
51 //   (измеряется в 32-х разрядных словах).
52 //   \en Row-to-row shift for the result image \n
53 //   (measured in 32-bit words).
54 // \param nShift
55 //   \ru Определяет число бит на которое предварительно сдвигается исходное
56 //   изображение.
57 //   \en Defines for how many bits the source image is preliminary shifted.
58 // \
59 // \param nWidth
60 //   \ru Ширина ROI (измеряется в пикселях).
61 //   \en ROI width (measured in pixels).
62 // \
63 // \param nHeight
64 //   \ru Высота ROI (измеряется в пикселях).
65 //   \en ROI height (measured in pixels).
66 // \
67 // \
68 // \return \e void
69 //
70 // \restr
71 //   \ru
72 //     - Указатели на первый элемент ROI должны быть выровнены
73 //       в памяти на границу 64-х разрядного слова;
74 //     - Ширина ROI должна быть кратна числу пикселей результирующего изображения
75 //       в 64-х разрядном слове;
76 //     - Межстроковые смещения должны быть кратны двум.
77 //     - Значение параметра nShift должно быть кратно 2.
78 // \en
79 //   - Pointers to the first ROI element should be aligned
80 //     in memory by the 64-bit word boundary;
81 //   - ROI width should be divisible by the number of pixels of the result image
82 //     in a 64-bit word;
83 //   - Row-to-row shifts should be divisible by two;
84 //   - nShift parameter value should be divisible by two.
85 //
86 // \par
87 // \xmlonly
88 // <testperf>
89 //   <param name="pSrcImg"> im0 im1 </param>
90 //   <param name="nSrcStride"> 128 </param>
91 //   <param name="nShift"> 2 </param>
92 //   <param name="pDstImg"> im0 im1 </param>
93 //   <param name="nDstStride"> 64 </param>
94 //   <param name="nHeight"> 128 </param>
95 //   <param name="nWidth"> 128 </param>
96 //   <size> nWidth*nHeight </size>
97 // </testperf>
98 // <testperf>
99 //   <param name="pSrcImg"> im0 </param>
100 //   <param name="nSrcStride"> 128 </param>
101 //   <param name="nShift"> 2 </param>
102 //   <param name="pDstImg"> im1 </param>
103 //   <param name="nDstStride"> 64 </param>
104 //   <param name="nHeight"> 128 </param>
105 //   <param name="nWidth"> 8 32 128 </param>
106 //   <size> nWidth*nHeight </size>
107 // </testperf>
108 // \endxmlonly
109 // /*
110 // /**
111 // \void nmpipiRShiftConvert(nm32s* pSrcImg, int nSrcStride, nm16s* pDstImg, int nDstStride, int nShift, int nWidth, int
112 //   nHeight);
113 // /**
114 // /**
115 // /**
116 // /**
117 // /**
118 // /**
119 // /**
120 // /**
121 // /**
122 // /**
123 // /**
124 // /**
125 // /**
126 // /**
127 // /**
128 // /**
129 // /**
130 // /**
131 // /**
132 // /**

```

```

133 //      \ru Преобразование типов для элементов изображения.
134 //      \en Conversion of types for image elements.
135 //
136 //      \-
137 //      \ru Функция конвертирует элементы RGB между 8-м разрядным представлением
138 //          и 10-ти разрядным.
139 //      \en Function converts RGB elements between 8-bits presentation and 10-bits.
140 //
141 //      \-
142 //      \
143 //      \param pSrcImg
144 //          \ru Указатель на первый элемент исходного изображения.
145 //          \en Pointer to the first element of the source image.
146 //          \
147 //      \param pDstImg
148 //          \ru Указатель на первый элемент результирующего изображения.
149 //          \en Pointer to the first element of the result image.
150 //          \
151 //      \param nSize
152 //          \ru Количество элементов в изображении.
153 //          \en Quantity of elements.
154 //
155 //      \
156 //      \return \e void
157 //
158 //      \restr
159 //          \ru
160 //              - Указатели на изображения должны быть выровнены
161 //                  в памяти на границу 64-х разрядного слова;
162 //          \en
163 //              - Pointers to the images should be aligned in memory
164 //                  by the 64-bit word boundary;
165 //
166 //      \par
167 //      \xmlonly
168 //          <testperf>
169 //              <param name="pSrcImg"> im0 im1 </param>
170 //              <param name="pDstImg"> im0 im1 </param>
171 //              <param name="nSize"> 128 </param>
172 //              <size> nSize </size>
173 //          </testperf>
174 //          <testperf>
175 //              <param name="pSrcImg"> im0 </param>
176 //              <param name="pDstImg"> im1 </param>
177 //              <param name="nSize"> 8 32 128 </param>
178 //              <size> nSize </size>
179 //          </testperf>
180 //      \endxmlonly
181 //      */
182 //      //! \{
183 //void nmppiConvertRGB32_8u10u(RGB32_nm8u* pSrcImg, RGB32_nm10u* pDstImg, int nSize);
184 //void nmppiConvertRGB32_10u8u(RGB32_nm10u* pSrcImg, RGB32_nm8u* pDstImg, int nSize);
185 //  //! \}
186 //
187 //
188 //
205 //
206 //  /**
207 //  \defgroup nmppiRGB32ToGray nmppiRGB32ToGray
208 //  \ingroup iInit
209 //  \brief
210 //      \ru Преобразование пикселей из RGB в яркость
211 //      \en
212 //
213 //      \
214 //      \param pRGB
215 //          \ru Вход, по 4 байта на пиксель. Порядок байтов B, G, R, 0.
216 //          \en
217 //      \param pDstGray
218 //          \ru Результат в виде 32 битных целых чисел, в которых
219 //              полезные данные занимают младшие 24 бита. Для получения
220 //              восьмибитовых пикселей необходимо вырезать биты 16..23,
221 //              например, с помощью
222 //          nmppsClipRShiftConvert_Add_32s(nm32s* pSrcVec, int nClipFactor,int nShift, nm64u* nAddValue,nm8s*
223 //          pDstVec, int nSize);
224 //              с параметрами nClipFactor=24, nShift=16.
225 //          \en
226 //      \param nSize
227 //          \ru Количество пикселей на входе и выходе. nSize=[64,128,192,...]
228 //          \en
229 //      \param pTmpBuf
230 //          \ru Временный массив размером nm32s[nSize] .
231 //          \en
232 //      \
233 //      \par
234 //      \xmlonly

```

```

235 //<testperf>
236 //<param name="pRGB"> im0 im1 </param>
237 //<param name="pDstGray"> im0 im1 </param>
238 //<param name="pTmpBuf"> im0 im1 </param>
239 //<param name="nSize"> 128 </param>
240 //<size> nSize </size>
241 //</testperf>
242 //<testperf>
243 //<param name="pRGB"> im0 </param>
244 //<param name="pDstGray"> im1 </param>
245 //<param name="pTmpBuf"> im2 </param>
246 //<param name="nSize"> 8 32 128 </param>
247 //<size> nSize </size>
248 //</testperf>
249 //\endxmlonly
250 //
251 // /**
252 // //! \{
253 //void nmpipiRGB32ToGray_8u32u(RGB32_nm8u* pRGB, nm32u* pDstGray, int nSize);
254 //void nmpipiRGB32ToGray_8u32s(RGB32_nm8u* pRGB, nm32s* pDstGray, int nSize);
255 //void nmpipiRGB32ToGray_8u8s(RGB32_nm8u* pRGB, nm8s* pDstGray, int nSize, void *pTmpBuf);
256 //
257 //void nmpipiRGB32ToGray_10s32s(RGB32_nm10s* pRGB, nm32s* pDstGray, int nSize);
258 //void nmpipiRGB32ToGray_10s32u(RGB32_nm10s* pRGB, nm32u* pDstGray, int nSize);
259 //void nmpipiRGB32ToGray_10s8s(RGB32_nm10s* pRGB, nm8s* pDstGray, int nSize, void *pTmpBuf);
260 // //! \}
261 //
262 //
263 //
264 //
265 //
266 //
267 // /**
268 // \internal
269 // \defgroup nmpipiGrayToRGB2424242424242424 nmpipiGrayToRGB242424242424242424
270 // \ingroup iInit
271 // \brief
272 // \ru преобразование из чернобелого формата в RGB24
273 // \en Gray to RGB24 conversion
274 // \
275 // \
276 // \param pSrcImg
277 // \ru Входное изображение
278 // \en Input image
279 // \
280 // \param pDstImg
281 // \ru Выходное изображение
282 // \en Output image
283 // \
284 // \param nSize
285 // \ru Размер изображение в пикселях
286 // \en Size of image in pixels
287 // \
288 // /**
289 // //! \{
290 //void nmpipiGrayToRGB2424242424242424(nm8u* pSrcImg, RGB24_nm8u* pDstImg, int nSize);
291 //void nmpipiRGB24ToGray(RGB24_nm8u* pRGB, nm8u* pDstGray, int nSize);
292 // //! \}
293 // #ifndef __cplusplus
294 // };
295 // #endif
296 // #endif
297 //

```

7.28 iDef.h

```

1 //-----
2 // $Workfile::: C_Frame.h      $
3 // FRC Library component
4 // Copyright (c) RC Module Inc.
5 // $Revision: 1.1 $   $Date: 2005/02/10 12:36:38 $
6 //-----
7
8
9
10
11
12
13
14
15
16
17
18
19 //-----
20
21 #ifndef _IDEF_H_
22 #define _IDEF_H_
23
24 // #ifdef _NM_
25 //typedef void RGB24_nm8u;
26 // #else

```

```

27 struct RGB24_nm8u
28 {
29     unsigned char nB:8;
30     unsigned char nG:8;
31     unsigned char nR:8;
32 };
33 //endif
34
35 struct RGB32_nm8u
36 {
37     unsigned int nB:8;
38     unsigned int nG:8;
39     unsigned int nR:8;
40     unsigned int nA:8;
41 };
42
43 struct RGB32_nm8s
44 {
45     int nB:8;
46     int nG:8;
47     int nR:8;
48     int nA:8;
49 };
50
51 struct RGB32_nm10u
52 {
53     unsigned int nB:10;
54     unsigned int nG:10;
55     unsigned int nR:10;
56     unsigned int nA:2;
57 };
58
59 struct RGB32_nm10s
60 {
61     int nB:10;
62     int nG:10;
63     int nR:10;
64     int nA:2;
65 };
66
67 //struct RGB64_nm16u
68 //{
69 //    unsigned int nB:16;
70 //    unsigned int nG:16;
71 //    unsigned int nR:16;
72 //    unsigned int nA:16;
73 //};
74
75 #endif

```

7.29 iDeinterlace.h

```

1 #include "nmpp.h"
2 #include "nmpli.h"
3
4 #ifdef __cplusplus
5     extern "C" {
6 #endif
7
8
19 void nmppiDeinterlaceSplit(nm8u* pSrcImg, int nSrcWidth, int nSrcHeight, nm8u* pDstEven, nm8u* pDstOdd);
20
28 void nmppiDeinterlaceBlend(nm8u* pSrcEven, nm8u* pSrcOdd, int nSrcWidth, int nSrcHeight, nm8u* pDst);
29
31
32 #ifdef __cplusplus
33     };
34 #endif

```

7.30 iFilter.h

```

1 //-
2 //-
3 // $Workfile:: iFiltration.h      $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.1 $    $Date: 2005/02/10 12:36:38 $

```

```

10 // -----
19 -----
20
21 #ifndef _IFILTER_H_INCLUDED_
22 #define _IFILTER_H_INCLUDED_
23
24 #define NmppiFilterState nm64s
25 #include "nmtype.h"
26
27 #ifdef __cplusplus
28     extern "C" {
29 #endif
30
31
32 struct S_nmppiFilterKernel {
33 #ifdef __NM__
34     nm32s* pDispArray;
35     nm32s* pWeightMatrix;
36 #else
37     nm32s* pDispArray;
38     nm32s* pWeightMatrix;
39     int nKerWidth;
40     int nKerHeight;
41 #endif
42 };
43
44
45 struct S_nmppiFilterKernel_32s32s {
46 #ifdef __NM__
47     nm32s* pDispArray;
48     nm32s* pWeightMatrix;
49     int nKerWidth;
50     int nKerHeight;
51 #else
52     nm32s* pDispArray;
53     nm32s* pWeightMatrix;
54     int nKerWidth;
55     int nKerHeight;
56 #endif
57 };
58
59 //*****
114
115 int nmppiSetFilter_8s8s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
116 int nmppiSetFilter_8s16s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
117 int nmppiSetFilter_8s32s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
118 int nmppiSetFilter_16s16s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
119 int nmppiSetFilter_16s32s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
120 int nmppiSetFilter_32s32s( int* pWeights, int nKerWidth, int nKerHeight, int nImgWidth, nm64s* pKernel);
122
123 int nmppiGetFilterKernelSize32_8s8s(int nKerWidth, int nKerHeight);
124 int nmppiGetFilterKernelSize32_8s16s(int nKerWidth, int nKerHeight);
125 int nmppiGetFilterKernelSize32_8s32s(int nKerWidth, int nKerHeight);
126 int nmppiGetFilterKernelSize32_16s16s(int nKerWidth, int nKerHeight);
127 int nmppiGetFilterKernelSize32_16s32s(int nKerWidth, int nKerHeight);
128 int nmppiGetFilterKernelSize32_32s32s(int nKerWidth, int nKerHeight);
129
130 //*****
131
132 void nmppiFilter_8s8s( nm8s * pSrcImg, nm8s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
133 void nmppiFilter_8s16s( nm8s * pSrcImg, nm16s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
134 void nmppiFilter_8s32s( nm8s * pSrcImg, nm32s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
135 void nmppiFilter_16s16s( nm16s * pSrcImg, nm16s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
136 void nmppiFilter_16s32s( nm16s * pSrcImg, nm32s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
137 void nmppiFilter_32s32s( nm32s * pSrcImg, nm32s* pDstImg, int nWidth, int nHeight, nm64s* pKernel);
138
139
140 //*****
141
142 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm8s* pSrcImg, nm8s* pDstImg, int nImgWidth,
143 //    int nImgHeight, nm64s* pKernel);
144 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm8s* pSrcImg, nm16s* pDstImg, int nImgWidth,
145 //    int nImgHeight, nm64s* pKernel);
146 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm8s* pSrcImg, nm32s* pDstImg, int nImgWidth,
147 //    int nImgHeight, nm64s* pKernel);
148 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm16s* pSrcImg, nm16s* pDstImg, int nImgWidth,
149 //    int nImgHeight, nm64s* pKernel);
150 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm16s* pSrcImg, nm32s* pDstImg, int nImgWidth,
151 //    int nImgHeight, nm64s* pKernel);
152 //void nmppiFilter_perf(int* pWeights, int nKerWidth, int nKerHeight, nm32s* pSrcImg, nm32s* pDstImg, int nImgWidth,
153 //    int nImgHeight, nm64s* pKernel);
154
155
156 //*****
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203 //*****
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297

```

```

304 //template <class nmbits_in, class nmbits_out> class CnmppiFIR
311 //{
312 //public:
313 // void (*pfFree32)(void*);    ///< Указатель на функции освобождения памяти (pKernel)
314 // int nKerWidth;           ///< Ширина окна коэффициентов КИХ фильтра
315 // int nKerHeight;          ///< Высота окна коэффициентов КИХ фильтра
316 // nm64s* pKernel;         ///< Указатель на внутреннюю структуру коэффициентов
317 // int nKernelSize;         ///< Размер памяти необходимый для хранения внутренней структуры коэффициентов
318 //
319 // //! Конструктор КИХ фильтра
320 // /*!
321 //   Выделяет область памяти под внутреннюю структуру коэффициентов
322 //   \param nKerWidth Ширина окна фильтра. nKerWidth=[3,5,7,...]
323 //   \param nKerHeight Высота окна фильтра. nKerHeight=[1,3,5,7,...]
324 //   \param malloc32_func указатель на функцию выделения динамической памяти 32-разрядными словами.
325 //   \param free32_func указатель на функцию динамического освобождения памяти
326 // */
327 // CnmppiFIR(int nKerWidth, int nKerHeight, void* (*malloc32_func)(unsigned),void (*free32_func)(void*));
328 //
329 //
330 // //! Загружает коэффициенты фильтра и инициализирует внутреннюю структуру хранения коэффициентов в
331 // pKernel
332 // /*!
333 //   \param pWeights коэффициенты фильтра
334 //   \param nImgWidth Ширина изображения к которому данный фильтр будет применен. Кратность согласно
335 //   входному типу.
336 //   \return указатель на внутреннюю структуру коэффициентов. 0- Если память под структуру не была выделена.
337 // */
338 // //! Функция одномерной фильтрации
339 // /*!
340 // \f[
341 //   pDstImg[y][x] = \sum\limits_{i=0}^{nKerHeight-1} \sum\limits_{j=0}^{nKerWidth-1}
342 //   \{pSrcImg[y+i-nKerHeight/2][x+j-nKerWidth/2] \cdot pWeights[i][j]\}, x=0 \dots nImgWidth-1, y=0 \dots
343 //   nImgHeight-1
344 // \f]
345 //   \param pSrcImg входное изображение
346 //   \param pDstImg выходное изображение
347 //   \param nImgWidth Ширина изображения к которому данный фильтр будет применен. Кратность согласно
348 //   входному типу.
349 //   \param nImgHeight Высота изображения (измеряется в пикселях).
350 //   \warning При вычислении первых и последних nKerHeight/2 строк производится выход за границы входного
351 //   массива pSrcImg .
352 //   Для корректного поведения функции необходимо дополнительные резервировать поля размером не менее
353 //   *nImgWidth*(nKerHeight/2+1) нулевых элементов перед началом и в конце массива pSrcImg.
354 // */
355 // //! Освобождает динамическую область памяти pKernel
356 // -CnmppiFIR();
357 //
358 //
359 //];
360 //
361 //
362 //
363 #ifdef __cplusplus
364 };
365#endif
366#endif
367
368
369#endif // _IFILTER_H_INCLUDED_
370
371

```

7.31 iFiltration.h

```

1 //-----
2 //-
3 // $Workfile::: iFiltration.h      $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.1 $    $Date: 2005/02/10 12:36:38 $
10 //-
11 //-----
12
13
14
15
16
17
18
19 //-----
20

```

```

21 #ifndef _IFILTRATION_H_INCLUDED_
22 #define _IFILTRATION_H_INCLUDED_
23
24 #ifdef __cplusplus
25     extern "C" {
26 #endif
27
28 #include "iFilter.h"
29 //*****
30
31 void nmppiMedian3x3( nm8s7b * pSrc, nm8s7b *pDst, int nWidth, int nHeight, void* pLTmp , void* pGTmp );
32
33
34
35
36 #ifdef __cplusplus
37 }
38#endif
39
40
41#endif // _IFILTRATION_H_INCLUDED_
42
43

```

7.32 iFloodFill.h

```

1 ****
2 *   RC Module
3 *   NeuroMatrix(r) NM6403 Software
4 *
5 *   Image Processing Library
6 *   (C-callable functions)
7 *
8 *   File:           FldFill.h
9 *   Contents:       FloodFill routines
10 *  Software design: Alex Ruzavin
11 *
12 *  Start date:    9may99
13 *  Release date:  23jun99
14 *
15 *
16 **** */
17 #ifndef __FLOODFILL_HEADER
18 #define __FLOODFILL_HEADER
19
20
21
22 typedef struct tagSegmentInfo
23 {
24     int xMin;
25     int yMin;
26     int xMax;
27     int yMax;
28     int N;          /* number of points */
29 } SegmentInfo;
30
31
32
33
80 int nmppiFloodFill( unsigned int * pSrcImage,SegmentInfo * pSegmentInfo,unsigned int * pSegmentImage, int nWidth,int
81             nHeight , unsigned int * pTmpBuff ) ;
82
83
84 typedef struct spot_struct{
85     int Xmin;
86     int Ymin;
87     int Xmax;
88     int Ymax;
89     int noPxl;
90     int dtSpot;
91 } spot_struct;
92
93
94 typedef struct ds_struct{
95     int nnSpot;
96     int nnPxl;
97     int dttSpot;
98 } ds_struct;
99
100
101
102 //extern "C" {
291
292 // extern "C" {

```

```

293 int FloodFill8(
294     void* src,
295     void* dst,
296     int nWidth,
297     int nHeight,
298     spot_struct* spot,
299     int lenSpot,
300     unsigned* pixels,
301     int mSpot,
302     int dtFull,
303     int dtSpot,
304     int lDiag,
305     int lDropSpot,
306     ds_struct* dropSpot,
307     int nPxIMin,
308     int nPxIMax,
309     int dXYmin,
310     int dXYmax
311 );
312 //};
314
315
316
317
318 #endif /* __FLOODFILL_HEADER */

```

7.33 iPlessy.h

```

1 #ifndef __PLESSY_H
2 #define __PLESSY_H
3
4 #ifdef __cplusplus
5     extern "C" {
6 #endif
7
8 //класс I_PlessyCornerDetector предназначен для поиска характерных точек на изображении по методу Plessy.
9 //На выходе выдаются координаты характерных точек с субпиксельной точностью.
10 class I_PlessyCornerDetector
11 {
12 public:
13     //Распределяет память для полей класса.
14     //w, h, ww - ширина, высота и step картинки соответственно.
15     virtual void Allocate(int w, int h, int ww) = 0;
16
17     //Освобождает память, выделенную под поля класса.
18     virtual void DeAllocate() = 0;
19
20     //Совсеменно ищет углы.
21     //w, h, ww - as above.
22     //Picture - обрабатываемая картинка.
23     //px, *py - массивы х и у координат углов (размер массивов определяется снаружи, максимальное
24     // значение - (w/2)*(h/2).
25     //nc - количество найденных углов.
26     virtual void FindCorners(unsigned char *Picture, int w, int h, int ww, float *px, float *py, int& nc) = 0;
27
28     //Освобождает интерфейс объекта
29     virtual void Release() = 0;
30
31     //Устанавливает threshold, по которому порождается Plessy-массив. Для 16s обычно ставится 32.0f.
32     //Для 32f обычно ставится 1/1024.
33     //Чем меньше порог, тем выше "качество" угла
34     virtual void SetThreshold(float _threshold)=0;
35 };
36
37 //Тип реализации (float и short соответственно)
38 enum PCD_TYPE {PCD_32f, PCD_16s};
39
40 //Создает объект и возвращает интерфейс I_PlessyCornerDetector
41 I_PlessyCornerDetector* CreatePlessyCornerDetector(PCD_TYPE type);
42
43 #ifdef __cplusplus
44 };
45 #endif
46 #endif

```

7.34 iPlessyDetector.h

```

1 #ifndef __PLESSYDETECTOR_H
2 #define __PLESSYDETECTOR_H
3

```

```

4 #include "iplessy.h"
5 #include "isubpixel2dimpl.h"
6
7
8 class C_PlessyCornerDetector : public I_PlessyCornerDetector
9 {
10 protected:
11     I_2DSubPixelMinPosition *SubPixelMinPosition;
12     short *fxi, *fyi;
13     short *Picture16;
14     float *SumSxxSyy, *Subxy;
15     float S9[9];
16     float threshold;
17     unsigned char *cres;
18
19     virtual void CountDer(unsigned char *Picture, int w, int h, int ww);
20     virtual void CountPlessy(int w, int h, int ww) {};
21 public:
22
23     C_PlessyCornerDetector();
24     ~C_PlessyCornerDetector();
25     virtual void Allocate(int w, int h, int ww);
26     virtual void DeAllocate();
27     virtual void FindCorners(unsigned char *Picture, int w, int h, int ww, float *px, float *py, int& nc);
28     virtual void Release();
29     virtual void SetThreshold(float _threshold)=0;
30 };
31
32
33
34 class C_PlessyCornerDetector_32f : public C_PlessyCornerDetector
35 {
36 protected:
37     float *fx, *fy;
38     float *sxx, *sxy, *syy;
39     float *Sxx, *Sxy, *Syy;
40     float *MulSxxSyy, *MulSxySxy;
41     float *cSubxy;
42     float threshold;
43
44     virtual void CountPlessy(int w, int h, int ww);
45     virtual void CountDer(unsigned char *Picture, int w, int h, int ww);
46 public:
47     C_PlessyCornerDetector_32f(){}
48     ~C_PlessyCornerDetector_32f(){}
49     virtual void Allocate(int w, int h, int ww);
50     virtual void DeAllocate();
51     virtual void SetThreshold(float _threshold);
52 };
53
54
55
56 class C_PlessyCornerDetector_16s : public C_PlessyCornerDetector
57 {
58 protected:
59     short *sxxi, *sxyi, *syyi;
60     short *Sxxi, *Sxyi, *Syyi;
61     short *Sxit, *Syyit;
62     short *SumSxxSyyi, *MulSxxSyyi, *MulSxySxyi, *Subxyi;
63     short *cSubxyi;
64     short threshold;
65
66     virtual void CountPlessy(int w, int h, int ww);
67
68 public:
69     C_PlessyCornerDetector_16s(){}
70     ~C_PlessyCornerDetector_16s(){}
71     virtual void Allocate(int w, int h, int ww);
72     virtual void DeAllocate();
73     virtual void SetThreshold(float _threshold);
74 };
75
76 #endif

```

7.35 iPrint.h

```

1 /**
2 /**
3 /**
4 /**
5 /**
6 /**
7 /**
8 /**

```

```

9 //-
19
20 #ifndef _IPRINT_H_INCLUDED_
21 #define _IPRINT_H_INCLUDED_
22
23
40 int nmppiPrint8x15( char *str, void* img, int imgWidth, int x, int y ,int FGcolor, int BGcolor);
42
53 char* hex2ascii(int value, char* str);
55
65 //char* hex2ascii(int value);
67
68 #endif // _IPRINT_H_INCLUDED_
69
70

```

7.36 iReodering.h

```

1 #ifndef __IREORDERING_H
2 #define __IREORDERING_H
3
4 #ifdef __cplusplus
5     extern "C" {
6 #endif
7
8
9 //*****
10
21 //*****
22
36 //*****
37
103
104 void nmppiSplitInt oBlocks8x8(
105     nm8s* pSrcImg,
106     nm8s* pDstBlockSeq,
107     int nWidth,
108     int nHeight
109 );
111
112 void nmppiSplitInt o2x2Blocks8x8(
113     nm8u* pSrcImg,
114     nm8u* pDstBlockSeq,
115     int nWidth,
116     int nHeight
117 );
118
119 void nmppiSplitInt o2x2Blocks8x8 or(
120     nm8u* pSrcImg,
121     nm8u* pDstBlockSeq,
122     nm8u* pXor,
123     int nWidth,
124     int nHeight
125 );
126
127
128 //*****
129
196 void nmppiMergeFromBlocks8x8(
197     nm8s* pSrcBlockSeq,
198     nm8s* pDstImg,
199     int nWidth,
200     int nHeight
201 );
203
204 #ifdef __cplusplus
205 };
206#endif
207
208#endif

```

7.37 iResample.h

```

1 //-----
2 //-
3 // $Workfile::: iResample. $
4 //-
5 // Векторно-матричная библиотека
6 //

```

```

7 // Copyright (c) RC Module Inc.
8 //
9 // $Revision: 1.1 $      $Date: 2005/02/10 12:36:38 $
10 //
11 //-----
12
13 #ifndef __I_Resample_H_INCLUDED_
14 #define __I_Resample_H_INCLUDED_
15
16 ///#include "nmpls.h"
17
18
19 #ifdef __cplusplus
20 extern "C" {
21 #endif
22
23 //*****
24
25
26
27 #ifdef __cplusplus
28 extern "C" {
29 #endif
30
31 //*****
32
33 //void nmppiResampleDown2X_8u8u(nm8u7b* pSrcImg, nm8u7b* pDstImg, int nSrcWidth, int nSrcHeight) ;
34 void nmppiResampleDown2X_8u8u(nm8u7b* pSrcImg, nm8u7b* pDstImg, int nSrcWidth, int nSrcHeight,nm64s*pKernel) ;
35 //void nmppiResampleDown2X_16u16u(nm16u15b* pSrcImg, nm16u15b* pDstImg, int nSrcWidth, int nSrcHeight);
36 void nmppiResampleDown2X_16u16u(nm16u15b* pSrcImg, nm16u15b* pDstImg, int nSrcWidth, int nSrcHeight, nm64s*
37 pKernel);
38 //void nmppiResampleDown2X_8u16u(nm8u* pSrcImg, nm16u* pDstImg, int nSrcWidth, int nSrcHeight, void* pTmpBuf);
39
40 //*****
41
42 void nmppiResampleDown2Y_8u8u (nm8u7b* pSrcImg, nm8u7b* pDstImg, int nSrcWidth, int nSrcHeight);
43 void nmppiResampleDown2Y_16u16u(nm16u15b* pSrcImg, nm16u15b* pDstImg, int nSrcWidth, int nSrcHeight);
44 void nmppiResampleDown2Y_8u16u_tmp(nm8u* pSrcImg, nm16u* pDstImg, int nSrcWidth, int nSrcHeight, void*
45 pTmpBuf);
46
47 //*****
48
49 void nmppiResampleDown2XY_8u8u(nm8u7b* pSrcImg, nm8u7b* pDstImg, int nSrcWidth, int nSrcHeight, void*
50 pTmpBuf);
51 //void nmppiResampleDown2XY_16u16u(nm16u15b* pSrcImg, nm16u15b* pDstImg, int nSrcWidth, int nSrcHeight, void*
52 pTmpBuf);
53 //void nmppiResampleDown2XY_8u16u(nm8u* pSrcImg, nm16u* pDstImg, int nSrcWidth, int nSrcHeight, void* pTmpBuf);
54
55 //void nmppiVResample3div2_RShift0(nm16s* pSrcImg, int nWidth, int nHeight, nm16s* pDstImg);
56
57
58 //__INLINE__ void nmppiCreateResampleDown2X_8u8u(nm64s** pKernel, int nHint) {
59 nmppsCreateResampleDown2_8u8u(pKernel,nHint);}
60 //__INLINE__ void nmppiCreateResampleDown2X_8u16u(nm64s** pKernel, int nHint) {
61 nmppsCreateResampleDown2_16u16u(pKernel,nHint);}
62 //__INLINE__ void nmppiCreateResampleDown2X_16u16u(nm64s** pKernel, int nHint) {
63 nmppsCreateResampleDown2_16u16u(pKernel,nHint);}
64 //__INLINE__ void nmppiCreateResampleDown2Y_8u16u(nm64s** pKernel, int nHint) {
65 nmppsCreateResampleDown2_16u16u(pKernel,nHint);}
66 //__INLINE__ void nmppiCreateResampleDown2XY_8u16u(nm64s** pKernel, int nHint){
67 nmppsCreateResampleDown2_16u16u(pKernel,nHint);}
68 //__INLINE__ void nmppiCreateResampleDown2XY_8u8u(nm64s** pKernel, int nHint) {
69 nmppsCreateResampleDown2_8u8u(pKernel,nHint);}
70 //__INLINE__ void nmppiCreateResampleDown2XY_16u16u(nm64s** pKernel,int nHint) {
71 nmppsCreateResampleDown2_16u16u(pKernel,nHint);}
72
73
74 #ifdef __cplusplus
75 };
76 #endif
77
78 #endif // __I_Resample_H_INCLUDED_

```

7.38 iSelect.h

```

1 #ifndef __ICOMPARE_H
2 #define __ICOMPARE_H
3
4
5 //*****
6
7 int nmppiCompareGtC(nm16s *pSrcImg, int nSrcStride, nm16s *pDstImg, int nDstStride, nm16s *pThreshold, int nWidth,
8     int nHeight);
9
10 #endif

```

7.39 isubpixel2d.h

```

1 #ifndef __SUBPIXEL2D_H

```

```

2 #define __SUBPIXEL2D_H
3
4
5 //класс I_2DSubPixelMinPosition предназначен для поиска субпиксельного минимума в массиве 3x3.
6 //»ояются две реализации - "параболическая" и "тригонометрическая"
7 class I_2DSubPixelMinPosition
8 {
9 public:
10
11    //»щет субпиксельный минимум
12    //S9 - массив 3x3 с минимумом в центре квадрата.
13    //dx, dy - координаты точки минимума относительно центра квадрата 3x3 (S9).
14    virtual void Find(float *S9, float& dx, float& dy) = 0;
15
16    //квебождает указатель
17    virtual void Release() = 0;
18
19 };
20
21 //«ип реализаций ("параболическая" и "тригонометрическая" соответственно)
22 enum SPP_TYPE {SPP_par, SPP_trg};
23
24
25 //—оздает объект и возвращает его интерфейс I_2DSubPixelMinPosition.
26 I_2DSubPixelMinPosition* Create2DSubPixelMinPosition(SPP_TYPE type);
27
28
29 #endif
30

```

7.40 isubpixel2dimpl.h

```

1 ifndef __SUBPIXEL2DIMPL_H
2 define __SUBPIXEL2DIMPL_H
3
4 include "isubpixel2d.h"
5
6
7 class C_2DSubPixelMinPosition : public I_2DSubPixelMinPosition
8 {
9 protected:
10
11 public:
12     C_2DSubPixelMinPosition();
13     ~C_2DSubPixelMinPosition();
14     virtual void Find(float *S9, float& dx, float& dy);
15     virtual void Release();
16 };
17
18
19
20 class C_2DTrigSubPixelMinPosition : public I_2DSubPixelMinPosition
21 {
22 protected:
23     float Teta[8];
24     float S9Interpolation(float x, float y, float *S9);
25
26 public:
27     C_2DTrigSubPixelMinPosition();
28     ~C_2DTrigSubPixelMinPosition();
29     virtual void Find(float *S9, float& dx, float& dy);
30     virtual void Release();
31 };
32
33
34 #endif

```

7.41 iSupport.h

```

1 //-
2 // $Workfile::: iSupport.h      $
3 // Neuro mtr processing library
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $   $Date: 2005/02/10 12:36:38 $
8 //
9 // $Revision: 1.1 $   $Date: 2005/02/10 12:36:38 $
10 //

```

```

19 //-----
20 #ifndef __IMALLOC_H
21 #define __IMALLOC_H
22 #include "nmtype.h"
23 /*
24  _ _INLINE_ _ int nmppsSizeOf_8s(nm8s* a, int nCount) {
25  return nCount»2;
26 }
27 _ _INLINE_ _ int nmppsSizeOf_8u(nm8u*, int nCount) {
28 return nCount»2;
29 }
30 _ _INLINE_ _ int nmppsSizeOf_32s(nm32s*, int nCount) {
31 return nCount;
32 }
33 _ _INLINE_ _ int nmppsSizeOf_32u(nm32u*, int nCount) {
34 return nCount;
35 }
36 */
37 */
38
39 // _ _INLINE_ _ int nmppsSizeOf_8s(nm8s*, int nCount){
40 // return nCount»2;
41 //}
42 // _ _INLINE_ _ int nmppsSizeOf_8u(nm8u*, int nCount){
43 // return nCount»2;
44 //}
45 // _ _INLINE_ _ int nmppsSizeOf_32s(nm32s*, int nCount){
46 // return nCount;
47 //}
48 // _ _INLINE_ _ int nmppsSizeOf_32u(nm32u*, int nCount){
49 // return nCount;
50 //}
51
52 template <class T> class C_Img
53 {
54 protected:
55
56 public:
57     int m_nBorder;
58     T* m_pContainer;
59     T* m_pData;
60     int m_nWidth;
61     int m_nHeight;
62     int m_nStride;
63     int m_nSize;
64     C_Img(int nWidth,int nHeight, int nStride, int nBorder, void* (*allocator32)(int))
65     {
66         m_nWidth=nWidth;
67         m_nHeight=nHeight;
68         m_nBorder=nBorder;
69         m_nStride=nStride;
70         m_nSize=m_nStride*m_nHeight;
71         m_pContainer=(T*)allocator32(nmppsSizeOf_(m_pData,m_nStride*(m_nHeight+2*m_nBorder)));
72         //nmppsMalloc_64s(&m_pContainer, m_nStride*(m_nHeight+2*m_nBorder), nHint);
73         m_pData=nmppsAddr_(m_pContainer,m_nStride*m_nBorder);
74     }
75     C_Img(T* pData, int nWidth,int nHeight, int nStride, int nBorder)
76     {
77         m_nWidth=nWidth;
78         m_nHeight=nHeight;
79         m_nBorder=nBorder;
80         m_nStride=nStride;
81         m_nSize=m_nStride*m_nHeight;
82         m_pContainer=0;
83         m_pData=pData;
84     }
85     ~C_Img()
86     {
87         if (m_pContainer)
88             nmppsFree(m_pContainer);
89     }
90     void Fill(T color)
91     {
92         nmppsSet_(m_pContainer,color,m_nSize+2*m_nBorder*m_nStride);
93     }
94 };
95 */
96 //*****
97 /**
98 * \internal
99 * \defgroup nmppiMalloc nmppiMalloc
100 * \ingroup iSupport
101 * \brief
102 * \ru Распределение памяти для изображений библиотеки.
103 * \en Memory allocation for library images.

```

```

106 \
107 \
108 \ru Начало и конец распределяемой памяти выравнивается на начало
109 64-х разрядного слова.
110 \en Begin and end of the allocated memory are being
111 aligned to 64-bit word.
112 \
113 \param nWidth
114 \ru Ширина изображения в пикселях.
115 \en Image width in pixels.
116 \
117 \param nHeight
118 \ru Высота изображения в пикселях.
119 \en Image height in pixels.
120 \
121 \param hint
122 \ru Номер банка памяти. Может принимать значения
123 MEM_LOCAL, MEM_GLOBAL.
124 \en Number of memory bank. Admissible values for memory bank
125 are MEM_LOCAL, MEM_GLOBAL.
126 \
127 \note \ru Память, распределенная с помощью функций nmppiMalloc должна
128 освобождаться только с помощью функции nmppiFree.
129 \en Memory allocated by function nmppiMalloc should be
130 deallocated by nmppiFree function only.
131 \
132 */
134 /*
135 void nmppiMalloc(nm1** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
136 void nmppiMalloc(nm2s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
137 void nmppiMalloc(nm2u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
138 void nmppiMalloc(nm4s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
139 void nmppiMalloc(nm4u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
140 void nmppiMalloc(nm8u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
141 void nmppiMalloc(nm8s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
142 void nmppiMalloc(nm16u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
143 void nmppiMalloc(nm16s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
144 void nmppiMalloc(nm32u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
145 void nmppiMalloc(nm32s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
146 void nmppiMalloc(nm64u** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
147 void nmppiMalloc(nm64s** pptr, int nWidth, int nHeight, int hint = MEM_LOCAL);
148 */
150 //*****
151 /*
152 /*
153 \defgroup nmppiFree nmppiFree
154 \ingroup iSupport
155 \brief
156 \ru Освобождение памяти для изображений.
157 \en Memory deallocation for images.
158 \
159 \note
160 \ru Данная функция должна вызываться только для
161 векторов, распределенных с помощью функций
162 nmppiMalloc.
163 \en This function should be called only for matrixis
164 allocated by nmppiMalloc functions.
165 \
166 */
168 //void nmppiFree(void* ptr);
170
171
181 __INLINE__ void nmppiReleaseObject(nm64s* pKernel);
183
184
185
186 #endif

```

7.42 nmpli.h

```
1 #include "./nmpli/nmpli.h"
```

7.43 nmpli.h

```

1
2 #ifndef INCLUDED_nmpli
3 #define INCLUDED_nmpli
4
5 #include "nmtypes.h"

```

```

6 #include "nmdef.h"
7 #include "nmpp.h"
8 #include "iDef.h"
9
10 #include "warpimg.h"
34 //*****
35
45 #include "iArithmetics.h"
46
47 //*****
48
58 #include "iFiltration.h"
59 //*****
60
71 #include "iConvert.h"
72
73
74 //*****
75
86 #include "iResample.h"
87
88
89 //*****
90
101 #include "iSelect.h"
102
103 //*****
104
114 //#include "iSupport.h"
115
116
117 #include "iFloodFill.h"
118
119
120 //#include "iCellTexture.h"
121
122
123 #include "iDeinterlace.h"
124
125
126 #include "iReodering.h"
127
128 //*****
129
138 #include "iPrint.h"
139
140 #include "filter.h"
141
142 //#
143
144 #

```

7.44 warpimg.h

```

1 /*
2 #include "nmpp.h"
3 #define nmppiAT_BUFFER 1
4 #define WARP_AT_BUFFER 2
5 template<class T> class C_WarpImg
6 {
7     bool isAllocated;
8     void (*pfFree32)(void*);
9 public:
10    unsigned   nWidth;
11    unsigned   nHeight;
12    unsigned   warpHeight;
13    T*        pWarp;
14    unsigned   nWarpSize;
15    T*        pImg;
16    unsigned   nImgSize;
17
18    C_WarpImg(unsigned width, unsigned height, unsigned border, void* (*malloc32)(unsigned ), void (*free32)(void*)){
19        nWidth=width;
20        nHeight=height;
21        nImgSize =width*height;
22        warpHeight=height+2*border;
23        nWarpSize=width*warpHeight;
24        pImg=0;
25        isAllocated=false;
26        pWarp=(T*)malloc32(nmppsSize32(pWarp,nWarpSize));
27        pfFree32=free32;
28        if (pWarp){
29            pImg=nmppsAddr_(pWarp,border*width);

```

```

30 isAllocated=true;
31 }
32 }
33
34 C_WarpImg(unsigned width, unsigned height, unsigned border, void* buffer, int mode=nmppiAT_BUFFER){
35 nWidth=width;
36 nHeight=height;
37 nImgSize =width*height;
38 warpHeight=height+2*border;
39 nWarpSize=width*warpHeight;
40
41 isAllocated=false;
42 if (mode==WARP_AT_BUFFER){
43 pWarp=(T*)buffer;
44 pImg=nmppsAddr_(pWarp,border*width);
45 }
46 if (mode==nmppiAT_BUFFER){
47 pImg =(T*)buffer;
48 pWarp=(T*)nmppsAddr_(pImg,-border*width);
49 }
50 }
51
52 T* addr(int x, int y){
53 return nmppsAddr_(pImg,y*nWidth+x);
54 }
55 T* end(){
56 return nmppsAddr_(pWarp,nWarpSize);
57 }
58
59 ~C_WarpImg(){
60 if (isAllocated){
61 pfFree32(pWarp);
62 }
63 }
64
65
66 };
67
68 */

```

7.45 mInit.h

```

1 //-----
2 // $Workfile::: mInit.h $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:03:24 $
8 //
9 // -----
10 // -----
11 #ifndef _MINIT_H_INCLUDED_
12 #define _MINIT_H_INCLUDED_
13
14 #ifdef __cplusplus
15     extern "C" {
16 #endif
17
18
19 // **** -----
20 // -----
21 // -----
22
23
24 #ifdef __cplusplus
25     extern "C" {
26 #endif
27
28
29 // **** -----
30 // -----
31 // -----
32 // -----
33 // -----
34 // -----
35 // -----
36 // -----
37 // -----
38 // -----
39 // -----
40 // -----
41 // -----
42
43 void nmppmCopyua_8s ( nm8s* pSrcMtr, int nSrcStride, int nSrcOffset, nm8s* pDstMtr, int nDstStride,int nHeight, int
44 nWidth);
45 void nmppmCopyua_16s( nm16s* pSrcMtr, int nSrcStride, int nSrcOffset, nm16s* pDstMtr, int nDstStride,int nHeight, int
46 nWidth);
47 void nmppmCopyua_32s( nm32s* pSrcMtr, int nSrcStride, int nSrcOffset, nm32s* pDstMtr, int nDstStride,int nHeight, int
48 nWidth);
49
50 void nmppmCopy_32fc(
51     double* SrcMtr,
52     int    nSrcStride,
53     double* DstMtr,
54     int    nDstStride,
55     int    nHeight,
56     int    nWidth
57

```

```

123 );
124
125
126 //*****
127
191 void nmppmCopyau_8s( nm8s* pSrcMtr, int nSrcStride, nm8s* pDstMtr, int nDstStride, int nDstOffset,int nHeight, int
192   nWidth),
192 void nmppmCopyau_16s( nm16s* pSrcMtr, int nSrcStride, nm16s* pDstMtr, int nDstStride, int nDstOffset,int nHeight, int
193   nWidth);
193 void nmppmCopyau_32s( nm32s* pSrcMtr, int nSrcStride, nm32s* pDstMtr, int nDstStride, int nDstOffset,int nHeight, int
194   nWidth);
195
196
197 //*****
198
254 void nmppmCopy_1 ( nm1* pSrcMtr, int nSrcStride, nm1* pDstMtr, int nDstStride, int nHeight, int nWidth);
255 void nmppmCopy_2s( nm2s* pSrcMtr, int nSrcStride, nm2s* pDstMtr, int nDstStride, int nHeight, int nWidth);
256 void nmppmCopy_4s( nm4s* pSrcMtr, int nSrcStride, nm4s* pDstMtr, int nDstStride, int nHeight, int nWidth);
257 void nmppmCopy_8s ( nm8s* pSrcMtr, int nSrcStride, nm8s* pDstMtr, int nDstStride, int nHeight, int nWidth);
258 void nmppmCopy_16s( nm16s* pSrcMtr, int nSrcStride, nm16s* pDstMtr, int nDstStride, int nHeight, int nWidth);
259 void nmppmCopy_32s( nm32s* pSrcMtr, int nSrcStride, nm32s* pDstMtr, int nDstStride, int nHeight, int nWidth);
260 void nmppmCopy_64s( nm64s* pSrcMtr, int nSrcStride, nm64s* pDstMtr, int nDstStride, int nHeight, int nWidth);
262
263
264 //*****
265
266
313 void MTR_Fill_8s(nm8s* pMtr, int8b nVal, int nMtrStride, int nMtrHeight, int nMtrWidth);
315
316
317 #ifdef __cplusplus
318 };
319#endif
320
321
322#endif // _MINIT_H_INCLUDED_

```

7.46 mInverse.h

```

1 //-
2 //-
3 // $Workfile::: mInverse.h $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 //-
18 //-
19
20 ifndef _MINVERSE_H_INCLUDED_
21 define _MINVERSE_H_INCLUDED_
22
23 //*****
24 #ifdef __cplusplus
25   extern "C" {
26#endif
27 //-
80 void MTR_fpResolve_Gauss(double* pSrcMtrA, double* pSrcVecB, double* pDstVecX, int nSize);
82
83
84 //*****
85
127 void MTR_fpResolve_PivotGauss(double* pSrcMtrAB, double* pDstVecX, int nSize);
129
130
131
143 void nmppmLUDecomp_64f (
144   const double * A,
145   double      * L,
146   double      * U,
147   int         N
148 );
150
151 //SLAY
152 // L*y = b; U*x = y
153
168 void nmppmLUResolve_64f (
169   const double      * L,
170   const double      * U,
171   const double      * b,
172   double            * x,
173   double            * y,
174   int               N

```

```

175 );
177
178 #ifndef __cplusplus
179 };
180 #endif
181
182
183 #endif // _MINVERSE_H_INCLUDED_

```

7.47 mMatrixVector.h

```

1 //-----
2 // $Workfile: mMatrixVec $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:03:24 $
8 //
9 //-----#
10 //-----#
11 #ifndef _MMATRIXVECTOR_H_INCLUDED_
12 #define _MMATRIXVECTOR_H_INCLUDED_
13
14 #ifdef __cplusplus
15     extern "C" {
16 #endif
17
18 //*****
19 //*****#
20 void nmppmMul_mm_2s64s ( nm2s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
21 nWidth2);
22 void nmppmMul_mm_4s64s ( nm4s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
23 nWidth2);
24 void nmppmMul_mm_8s8s ( nm8s* pSrcMtr1, int nHeight1, int nWidth1, nm8s* pSrcMtr2, nm8s* pDstMtr, int nWidth2);
25 void nmppmMul_mm_8s16s ( nm8s* pSrcMtr1, int nHeight1, int nWidth1, nm16s* pSrcMtr2, nm16s* pDstMtr, int
26 nWidth2);
27 void nmppmMul_mm_8s32s ( nm8s* pSrcMtr1, int nHeight1, int nWidth1, nm32s* pSrcMtr2, nm32s* pDstMtr, int
28 nWidth2);
29 void nmppmMul_mm_8s64s ( nm8s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
30 nWidth2);
31 void nmppmMul_mm_16s16s( nm16s* pSrcMtr1, int nHeight1, int nWidth1, nm16s* pSrcMtr2, nm16s* pDstMtr, int
32 nWidth2);
33 void nmppmMul_mm_16s32s( nm16s* pSrcMtr1, int nHeight1, int nWidth1, nm32s* pSrcMtr2, nm32s* pDstMtr, int
34 nWidth2);
35 void nmppmMul_mm_16s64s( nm16s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
36 nWidth2);
37 void nmppmMul_mm_32s32s( nm32s* pSrcMtr1, int nHeight1, int nWidth1, nm32s* pSrcMtr2, nm32s* pDstMtr, int
38 nWidth2);
39 void nmppmMul_mm_32s64s( nm32s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
40 nWidth2);
41 void nmppmMul_mm_64s64s( nm64s* pSrcMtr1, int nHeight1, int nWidth1, nm64s* pSrcMtr2, nm64s* pDstMtr, int
42 nWidth2);
43
44 void nmppmMul_mm_32f ( float* pSrcMtr1, int nHeight1, int nStride1,
45 float* pSrcMtr2, int nWidth1, int nStride2,
46 float* pDstMtr, int nWidth2, int nStrideDst, int bPlusDst );
47 void nmppmMul_mt_32f ( float* pSrcMtr1, int nHeight1, int nStride1,
48 float* pSrcMtr2, int nWidth1, int nStride2,
49 float* pDstMtr, int nWidth2, int nStrideDst, int bPlusDst );
50
51 void nmppmMul_mm_colmajor_8s8s (const nm8s* pSrcMtr1, int nHeight1, int nWidth1, const nm8s* pSrcMtr2, nm8s*
52 pDstMtr, int nWidth2);
53 void nmppmMul_mm_colmajor_8s16s (const nm8s* pSrcMtr1, int nHeight1, int nWidth1, const nm16s* pSrcMtr2, nm16s*
54 pDstMtr, int nWidth2);
55 void nmppmMul_mm_colmajor_8s32s (const nm8s* pSrcMtr1, int nHeight1, int nWidth1, const nm32s* pSrcMtr2, nm32s*
56 pDstMtr, int nWidth2);
57 void nmppmMul_mm_colmajor_8s64s (const nm8s* pSrcMtr1, int nHeight1, int nWidth1, const nm64s* pSrcMtr2, nm64s*
58 pDstMtr, int nWidth2);
59
60 void nmppmMul_mm_colmajor_16s16s(const nm16s* pSrcMtr1, int nHeight1, int nWidth1, const nm16s* pSrcMtr2, nm16s*
61 pDstMtr, int nWidth2);
62 void nmppmMul_mm_colmajor_16s32s(const nm16s* pSrcMtr1, int nHeight1, int nWidth1, const nm32s* pSrcMtr2, nm32s*
63 pDstMtr, int nWidth2);
64 void nmppmMul_mm_colmajor_16s64s(const nm16s* pSrcMtr1, int nHeight1, int nWidth1, const nm64s* pSrcMtr2, nm64s*
65 pDstMtr, int nWidth2);
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128

```

```

129 void nmppmMul_mm_colmajor_32s32s(const nm32s* pSrcMtr1, int nHeight1, int nWidth1, const nm32s* pSrcMtr2, nm32s*
    pDstMtr, int nWidth2);
130 void nmppmMul_mm_colmajor_32s64s(const nm32s* pSrcMtr1, int nHeight1, int nWidth1, const nm64s* pSrcMtr2, nm64s*
    pDstMtr, int nWidth2);
131
132
133 //*****
134
190 void nmppmMul_mv_8s64s( nm8s* pSrcMtr, nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int nWidth);
191 void nmppmMul_mv_16s64s( nm16s* pSrcMtr, nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int nWidth);
192 void nmppmMul_mv_32s64s( nm32s* pSrcMtr, nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int nWidth);
193
194
196 void nmppmMul_mv_colmajor_8s64s (const nm8s* pSrcMtr,const nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int
    nWidth);
197 void nmppmMul_mv_colmajor_16s64s(const nm16s* pSrcMtr,const nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int
    nWidth);
198 void nmppmMul_mv_colmajor_32s64s(const nm32s* pSrcMtr,const nm64s* pSrcVec, nm64s* pDstVec, int nHeight, int
    nWidth);
199
251 void nmppmMul_mv_8s16s_8xH( v8nm8s* pSrcMtr, v8nm16s* pSrcVec, nm16s* pDstVec, int nHeight);
252 void nmppmMul_mv_16s16s_8xH( v8nm16s* pSrcMtr, v8nm16s* pSrcVec, nm16s* pDstVec, int nHeight);
254 //*****
255
318 void nmppmMul_mv__AddC(v2nm32s* pSrcMtr, v2nm32s* pnSrcVec, int nAddVal, nm32s* pDstVec, int nHeight);
320
321
322 //*****
323
373 void MTR_ProdUnitV_16s_4xH ( v4nm16s* pSrcMtr, nm16s* pDstVec, int nHeight);
374 void MTR_ProdUnitV_16s_16xH( v16nm8s* pSrcMtr, nm16s* pDstVec, int nHeight);
376
377 //*****
378 //-----
421 void MTR_MulC_AddVsVc(int MuIN, nm32s* pSrcMtr, nm32s* pSrcVecStr, nm32s* pSrcVecCol, nm32s* pDstMtr, int
    nHeight, int nWidth);
423
424 #ifdef __cplusplus
425     };
426 #endif
427
428 #endif // _MMATRIXVECTOR_H_INCLUDED_

```

7.48 mMatrixVectorDev.h

```

1 //-
2 // $Workfile::: mMatrixVecDev.h $ $ 
3 // Векторно-матричная библиотека
4 // Copyright (c) RC Module Inc.
5 // $Revision: 1.2 $ $Date: 2005/06/23 15:15:15 $
6 //-
7 #ifndef _MMATRIXVECTORDEV_H_INCLUDED_
8 #define _MMATRIXVECTORDEV_H_INCLUDED_
9
10 //-
11 //-
12
13 #ifndef _MMATRIXVECTORDEV_H_INCLUDED_
14 #define _MMATRIXVECTORDEV_H_INCLUDED_
15
16
22 void nmppmMul_mv_( nm64sc *pSrcMtr, nm64sc *pSrcVec, nm64sc *pDstVec, int nHeight, int nWidth, void*tmp);
24
73 //void MTR_ProdSelfV( nm64sc *pSrcVec, nm64sc *pDstMtr, int nSize);
74 void MTR_ProdSelfV( nm64sc *pSrcVec, nm64sc *pDstMtr, int nSize, void* pTmp);
75 //void MTR_ProdSelfV( nm32sc *pSrcVec, nm64sc *pDstMtr, int nSize); //pc version is not available
77
147 void nmppmMul_mv__Zero( nm64sc *pSrcMtr, nm64sc *pSrcVec, nm64sc *pDstVec, int nStart, int nQuantity, int nHeight,
    int nWidth, void *tmp);
149
150 //-
151 // \defgroup nmppsSub8_Abs nmppsSub8_Abs
152 // \ingroup mMatrixVector
153 // \{
154 //-
155 //void nmppsSub8_Abs(nm32s* pSrcVec, nm32s* SrcN8, nm32s* pDstVec, int nSize);
156 //\todo void MTR_SubV8_Abs(nm32s* pSrcMtr,int nSrcStride,nm32s* pSrcVec8,nm32s* pDstMtr,int pDstStride,int
    nHeight);
157 // \}
158
159 //-
160 // \defgroup nmppsSub16_Abs nmppsSub16_Abs
161 // \ingroup MatrixVector
162 // \{

```

```

163 //-----
164 //void nmppsSub16_Abs(nm8s* pSrcVec1, nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
165 //void nmppsSub16_Abs(nm32s* pSrcVec1, nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
166 //void nmppsSub16_Abs(nm32s* pSrcVec1, nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
167 // \todo void MTR_SubV16_Abs(nm8s* pSrcMtr,int nSrcStride, nm8s* pSrcVec16, nm8s* pDstMtr,int nDstStride, int
168 // \todo void MTR_SubV16_Abs(nm32s* pSrcMtr,int nSrcStride, nm32s* pSrcVec16, nm32s* pDstMtr,int nDstStride, int
169 // nHeight);
170 // \}
171 #endif

```

7.49 mStat.h

```

1 //-----
2 // $Workfile:: vStat.h $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2004/11/22 13:50:02 $
8 //
9 // -----
10 //
11 //
12 #ifndef _MSTAT_H_INCLUDED_
13 #define _MSTAT_H_INCLUDED_
14
15 #ifdef __cplusplus
16     extern "C" {
17 #endif
18
19     unsigned nmppmCrc_32u(const unsigned int* pSrcVec, int height, int width, int stride);
20
21     INLINE unsigned nmppmCrc_32s(const nm32s * pSrcVec, int height, int width, int stride) { return
22         nmppmCrc_32u((const unsigned*)pSrcVec, height, width, stride); }
23     INLINE unsigned nmppmCrc_64u(const nm64u * pSrcVec, int height, int width, int stride) { return
24         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<1,stride<1); }
25     INLINE unsigned nmppmCrc_64s(const nm64s * pSrcVec, int height, int width, int stride) { return
26         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<1,stride<1); }
27     INLINE unsigned nmppmCrc_16u(const nm16u * pSrcVec, int height, int width, int stride) { return
28         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<1,stride<1); }
29     INLINE unsigned nmppmCrc_16s(const nm16s * pSrcVec, int height, int width, int stride) { return
30         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<1,stride<1); }
31     INLINE unsigned nmppmCrc_8u (const nm8s * pSrcVec, int height, int width, int stride) { return
32         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<2,stride>2); }
33     INLINE unsigned nmppmCrc_8s (const nm8s * pSrcVec, int height, int width, int stride) { return
34         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<2,stride>2); }
35     INLINE unsigned nmppmCrc_4u (const nm4s * pSrcVec, int height, int width, int stride) { return
36         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<3,stride>3); }
37     INLINE unsigned nmppmCrc_4s (const nm4s * pSrcVec, int height, int width, int stride) { return
38         nmppmCrc_32u((const unsigned*)pSrcVec, height, width<3,stride>3); }
39
40     unsigned nmppmCrcAcc_32u (const unsigned int* pSrcVec, int height, int width, int stride, unsigned int* crc);
41
42
43
44
45     INLINE unsigned nmppsCrc_32f(nm32f* pSrcVec, int numBitsToClear, int nSize) { return
46         nmppsCrcMask_32u((unsigned*)pSrcVec,(-1<<numBitsToClear),nSize<1); }
47
48
49
50     INLINE unsigned nmppmCrcAcc_64s(nm64s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
51         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize<1, crcAccumulator); }
52     INLINE unsigned nmppmCrcAcc_32s(nm32s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
53         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize, crcAccumulator); }
54     INLINE unsigned nmppmCrcAcc_16s(nm16s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
55         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize<1, crcAccumulator); }
56     INLINE unsigned nmppmCrcAcc_8s (nm8s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
57         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize>2, crcAccumulator); }
58     INLINE unsigned nmppmCrcAcc_64u(nm64u* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
59         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize<1, crcAccumulator); }
60
61
62     INLINE unsigned nmppmCrcAcc_16u(nm16u* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
63         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize>1, crcAccumulator); }
64     INLINE unsigned nmppmCrcAcc_8u (nm8u* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
65         nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize>2, crcAccumulator); }
66
67
68     unsigned nmppmCrcAcc_32f(const nm32f* pSrcVec, int numBitsToClear, int nSize, unsigned int* crcAccumulator) ;
69
70     unsigned nmppmCrcAcc_64f(const nm64f* pSrcVec, int numBitsToClear, int nSize, unsigned int* crcAccumulator) ;
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97

```

```

98 //|
99
100
101
102
103
104 #ifndef __cplusplus
105     };
106 #endif
107
108 #endif // _MSTAT_H_INCLUDED_

```

7.50 mSupport.h

```

1 //-----
2 // $Workfile::: mSupport.h      $
3 // <Название библиотеки>
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.2 $   $Date: 2005/06/23 15:15:15 $
8 //
9 // ****
10 //
11 //-----
12 #ifndef __MSUPPORT_H
13 #define __MSUPPORT_H
14
15 #include "nmpp.h"
16
17 //*****
18 /*
19 * \if Russian
20 *     \defgroup mSupport Функции поддержки
21 * \endif
22 * \if English
23 *     \defgroup mSupport Support functions
24 * \endif
25 * \ingroup mtr
26 */
27 /*
28 * \if Russian
29 *     \defgroup mSupport Функции поддержки
30 * \endif
31 * \if English
32 *     \defgroup mSupport Support functions
33 * \endif
34 * \ingroup mtr
35 */
36 */
37
38 //*****
39 /*
40 */
41 /*
42 */
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92 */
93
94 //*****
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180

```

```

181 __INLINE nm2s* MTR_ Addr_2s (nm2s* pMTR, int nWidth, int nY, int nX) { return (nm2s*)
182 __INLINE nm4s* MTR_ Addr_4s (nm4s* pMTR, int nWidth, int nY, int nX) { return (nm4s*)
183 __INLINE nm8s* MTR_ Addr_8s (nm8s* pMTR, int nWidth, int nY, int nX) { return (nm8s*)
184 __INLINE nm16s* MTR_ Addr_16s(nm16s* pMTR, int nWidth, int nY, int nX) { return
185 __INLINE nm32s* MTR_ Addr_32s(nm32s* pMTR, int nWidth, int nY, int nX) { return
186 __INLINE nm64s* MTR_ Addr_64s(nm64s* pMTR, int nWidth, int nY, int nX) { return
187 __INLINE nm2u* MTR_ Addr_2u (nm2u* pMTR, int nWidth, int nY, int nX) { return (nm2u*)
188 __INLINE nm4u* MTR_ Addr_4u (nm4u* pMTR, int nWidth, int nY, int nX) { return (nm4u*)
189 __INLINE nm8u* MTR_ Addr_8u (nm8u* pMTR, int nWidth, int nY, int nX) { return (nm8u*)
190 __INLINE nm16u* MTR_ Addr_16u(nm16u* pMTR, int nWidth, int nY, int nX) { return
191 __INLINE nm32u* MTR_ Addr_32u(nm32u* pMTR, int nWidth, int nY, int nX) { return
192 __INLINE nm64u* MTR_ Addr_64u(nm64u* pMTR, int nWidth, int nY, int nX) { return
193 __INLINE nm1* MTR_SetVal_1 (nm1* pMtr, int nWidth, int nY, int nX, int1b nVal) { nmppsPut_1 (pMtr,
194 /*-----*/
195 __INLINE void MTR_SetVal_2s (nm2s* pMtr, int nWidth, int nY, int nX, int2b nVal) { nmppsPut_2s (pMtr,
196 __INLINE void MTR_SetVal_4s (nm4s* pMtr, int nWidth, int nY, int nX, int4b nVal) { nmppsPut_4s (pMtr,
197 __INLINE void MTR_SetVal_8s (nm8s* pMtr, int nWidth, int nY, int nX, int8b nVal) { nmppsPut_8s (pMtr,
198 __INLINE void MTR_SetVal_16s(nm16s* pMtr, int nWidth, int nY, int nX, int16b nVal) { nmppsPut_16s(pMtr,
199 __INLINE void MTR_SetVal_32s(nm32s* pMtr, int nWidth, int nY, int nX, int32b nVal) { nmppsPut_32s(pMtr,
200 __INLINE void MTR_SetVal_64s(nm64s* pMtr, int nWidth, int nY, int nX, int64b nVal) { nmppsPut_64s(pMtr,
201 __INLINE void MTR_SetVal_2u (nm2u* pMtr, int nWidth, int nY, int nX, uint2b nVal) { nmppsPut_2u (pMtr,
202 __INLINE void MTR_SetVal_4u (nm4u* pMtr, int nWidth, int nY, int nX, uint4b nVal) { nmppsPut_4u (pMtr,
203 __INLINE void MTR_SetVal_8u (nm8u* pMtr, int nWidth, int nY, int nX, uint8b nVal) { nmppsPut_8u (pMtr,
204 __INLINE void MTR_SetVal_16u(nm16u* pMtr, int nWidth, int nY, int nX, uint16b nVal) { nmppsPut_16u(pMtr,
205 __INLINE void MTR_SetVal_32u(nm32u* pMtr, int nWidth, int nY, int nX, uint32b nVal) { nmppsPut_32u(pMtr,
206 __INLINE void MTR_SetVal_64u(nm64u* pMtr, int nWidth, int nY, int nX, uint64b nVal) { nmppsPut_64u(pMtr,
207 __INLINE void MTR_GetVal_1 (nm1* pMtr, int nWidth, int nY, int nX, int1b* nVal) { nmppsGetVal_1 (pMtr,
208 /*-----*/
209 __INLINE void MTR_GetVal_2s (nm2s* pMtr, int nWidth, int nY, int nX, int2b* nVal) { nmppsGetVal_2s (pMtr,
210 __INLINE void MTR_GetVal_4s (nm4s* pMtr, int nWidth, int nY, int nX, int4b* nVal) { nmppsGetVal_4s (pMtr,
211 __INLINE void MTR_GetVal_8s (nm8s* pMtr, int nWidth, int nY, int nX, int8b* nVal) { nmppsGetVal_8s (pMtr,
212 __INLINE void MTR_GetVal_16s(nm16s* pMtr, int nWidth, int nY, int nX, int16b* nVal) { nmppsGetVal_16s(pMtr,
213 __INLINE void MTR_GetVal_32s(nm32s* pMtr, int nWidth, int nY, int nX, int32b* nVal) { nmppsGetVal_32s(pMtr,
214 __INLINE void MTR_GetVal_64s(nm64s* pMtr, int nWidth, int nY, int nX, int64b* nVal) { nmppsGetVal_64s(pMtr,
215 __INLINE void MTR_GetVal_2u (nm2u* pMtr, int nWidth, int nY, int nX, uint2b* nVal) { nmppsGetVal_2u (pMtr,
216 __INLINE void MTR_GetVal_4u (nm4u* pMtr, int nWidth, int nY, int nX, uint4b* nVal) { nmppsGetVal_4u (pMtr,
217 __INLINE void MTR_GetVal_8u (nm8u* pMtr, int nWidth, int nY, int nX, uint8b* nVal) { nmppsGetVal_8u (pMtr,
218 __INLINE void MTR_GetVal_16u(nm16u* pMtr, int nWidth, int nY, int nX, uint16b* nVal){ nmppsGetVal_16u(pMtr, nY*nWidth+nX,nVal);
219 __INLINE void MTR_GetVal_32u(nm32u* pMtr, int nWidth, int nY, int nX, uint32b* nVal){ nmppsGetVal_32u(pMtr, nY*nWidth+nX,nVal);
220 __INLINE void MTR_GetVal_64u(nm64u* pMtr, int nWidth, int nY, int nX, uint64b* nVal){ nmppsGetVal_64u(pMtr, nY*nWidth+nX,nVal);
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
```

```

327 //*****
328 //***** MTR_GetCol( nm64sc *pSrcMtr, nm64sc *pDstVec, int nCol, int nHeight, int nWidth);
329
381 void MTR_GetCol( nm64sc *pSrcMtr, nm64sc *pDstVec, int nCol, int nHeight, int nWidth);
382 //void MTR_GetCol( nm16sc *pSrcMtr, nm16sc *pDstVec, int nCol, int nHeight, int nWidth); //pc version is not available!
384
385 #endif

```

7.51 nmplm.h

```
1 #include "./nmplm/nmplm.h"
```

7.52 nmplm.h

```

1
2 #ifndef INCLUDED_nmplm
3 #define INCLUDED_nmplm
4
5 #include "nmtpe.h"
6 #include "nmdef.h"
7 #include "nmpp.h"
8
9
33 //*****
34
44 #include "mSupport.h"
45
46 //-----
47 //-----
54 //-----
55
56 #include "mInit.h"
57
58 //*****
59
69 #include "mMatrixVector.h"
70 #include "mMatrixVectorDev.h"
71 //*****
72
82 #include "mInverse.h"
83
84 #include "mStat.h"
85
86 //#include "profile\profdata.h"
87
88
89
90#endif

```

7.53 fft_old.h

```

1 //*****
2 /*          RC Module Inc., Moscow, Russia          */
3 /*          NeuroMatrix(r) NM6403 Software          */
4 /*
5 /*  Fast Fourie Transform Library
6 /*  (C-callable functions)
7 /*
8 /* $Workfile:: sFFT.h
9 /* Contents:      Header file of FFT routines
10 /*
11 /*
12 /* Author:      S.Mushkaev
13 /*
14 /* Version      1.0
15 /* Start Date: 03.07.2001
16 /* Release $Date: 2005/07/13 14:19:56 $
17 /*
18 /*
19 //*****
20
21 // LIBRARY nmfft.lib
22
23
24

```

```

25 #ifndef _SFFT_H_INCLUDED_
26 #define _SFFT_H_INCLUDED_
27
28
29 #include "nmtype.h"
30
31
32 /* \mainpage Введение
33 *
34 * \htmlinclude intro.html
35 * \section intro_sec Introduction
36 *
37 * This is the introduction.
38 *
39 * \section install_sec Installation
40 *
41 * \subsection step1 Step 1: Opening the box
42 *
43 * etc...
44
45 */
46
47
48
49 //*****
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68 //*****
69
70
71 /*
72 #ifndef _NMCMPLX_H_INCLUDED_
73 struct nm32sc
74 {
75 int re;//Real;
76 int im;//Imag;
77 nm32sc():
78 re(0),im(0){}; //Real(0),Imag(0){}
79 nm32sc(int _Real,int _Imag):
80 re(_Real),im(_Imag){};//Real(_Real),Imag(_Imag){}
81 };
82 #endif // _NMCMPLX_H_INCLUDED_
83 */
84 //#include "nmpp.h"
85
86 // The functions listed below are forward and inversed FFT routines for
87 // 256,512,1024 or 2048-point compex data, represented as arrays of nm32sc type.
88 // Each complex number is stored in 64-bit word.
89 // The lower 32-bits is real part of complex number.
90 // The higher 32-bits is imaginary part of complex number;
91 // The admissible input range of data depends on dimension of array,
92 // mode of calculation accuracy and on/off mode of intermediate and final scaling down (shift normalization) of results.
93 // This range guarantee against overflow during calculation process.
94 // The table of ranges you may find in "FFT Library Programmer's manual"
95 //
96 // The mode of calculation accuracy tells how sine-cosine coeffecients are represented if fixed-point format.
97 // When the 7-bit accuracy mode is used, output shape accuracy approaches of the maximum,
98 // but output is reduced by around 2%.
99 // If the 6-bit accuracy mode is used, then output range corresponds to result of Fourie transform
100 // based on floating-point arithmetic, but output is less precise.
101 // The mode of calulation accuracy may be set or switched by appropriate ***Set6bit() or ***Set7bit() function
102 // NOTE: At least on time the accuracy setting function must be called before FFT routine executing.
103 #include "malloc32.h"
104
105
106 #ifdef __cplusplus
107     extern "C" {
108 #endif
109 //===== Forward FFT 256
110 =====
111
112
113
114 void FFT_Fwd256Set6bit(); // Sets 6-bit accuracy of sin-cosine coefficients
115 void FFT_Fwd256Set7bit(); // Sets 7-bit accuracy of sin-cosine coefficients
116
117
118 // The performance of the FFT_Fwd256 routine depends on memory allocation for
119 // input,output and temporary buffers.
120 // For the maximum speed performance it is recommended
121 // to use the following configuration:
122 //      GSrcBuffer: Global SRAM
123 //      LDstBuffer: Local SRAM
124 //      LBuffer : Local SRAM
125 //      GBuffer : Global SRAM
126

```

```

146 //|
147 //| For this configuration the following results were achieved:
148 //|   3994 clocks - full operation (0.1 ms at 40MHz CPU)
149 //|   3662 clocks - without final normalization (0.092ms at 40MHz CPU)
150 //|
151 //| If you are not going to use this routine as a C callable function,
152 //| you can reduce the number of instructions removing all stack operations. In this case
153 //| the total execution time can be reduced by around 50 clocks.
154
155
156
157
251 void FFT_Fwd256(
252     nm32sc* GSrcBuffer,    // Source buffer :long[256]
253     nm32sc* LDstBuffer,   // Result FFT :long[256]
254     void* LBuffer,        // Temp buffer :long[256*3]
255     void* GBuffer,        // Temp buffer :long[256*2]
256     int ShiftR=-1         // Shift normalization by default it means ShiftR=14 at 7 bit precision and ShiftR=12 at 6
      bit precision
257 );
258 #include "time.h"
259 #include "malloc32.h"
260
261
262
263
264
265
266 // int nmppsFFT256FwdInitAlloc(Malloc32Func* allocate, Free32Func* free, NmppsFFTSpec* spec);
267 // void nmppsFFT256FwdOptimize(void* src, void* dst, uint64* allocOrder);
268 // void nmppsFFT256Fwd(nm32sc* src, nm32sc* dst, NmppsFFTSpec* spec);
269 //
270 // void nmppsFFTFree(NmppsFFTSpec* spec );
271
272
273
274 /*
275 struct s_fft_fwd256_settings {
276 int8x8* dataSinCos0;
277 int sizeSinCos0; // in int64
278 int bitsSinCos0;
279 int shift0;
280 int8x8* dataSinCos1;
281 int sizeSinCos1; // in int64
282 int bitsSinCos1;
283 int shift1;
284 }.*;
285 */
286 /*
287
288 s_fft_fwd256_settings s_fft256_default={0,1,1,1,0,1,1,1};
289
290 void FFT_Fwd256_(
291 int32x2* pSrc,
292 int32x2* pDst,
293 int64* tmp0,
294 int64* tmp1,
295 s_fft_fwd256_settings* s=&s_fft256_default
296 );
297 */
298
299
300
312 void FFT_Inv256Set6bit(); // Sets 6-bit accuracy of sin-cosine coefficients
313
321 void FFT_Inv256Set7bit(); // Sets 7-bit accuracy of sin-cosine coefficients
322
446 void FFT_Inv256(
447     nm32sc* GSrcBuffer, // Source buffer :long[256]
448     nm32sc* GDstBuffer, // Result FFT :long[256]
449     void* LBuffer,      // Temp buffer :long[256*3]
450     void* GBuffer,      // Temp buffer :long[256*2]
451     int ShiftR1=8,      // Intermediate shift normalization
452     int ShiftR2=-1       // Final shift normalization
453           // by default it means ShiftR2=14 at 7 bit precision
454           // and ShiftR2=12 at 6 bit precision
455 );
456
457
458 //=====
472 void FFT_Fwd512Set6bit(); // Sets 6-bit accuracy of sin-cosine coefficients
480 void FFT_Fwd512Set7bit(); // Sets 7-bit accuracy of sin-cosine coefficients
481
482
577 void FFT_Fwd512(
578     nm32sc* GSrcBuffer, // Source buffer :long[512]
579     nm32sc* GDstBuffer, // Result FFT :long[512]

```

```

580     void*    LBuffer,   // Temp buffer :long[512*3]
581     void*    GBuffer,   // Temp buffer :long[512*3]
582     int      ShiftR=-1 // Right shift normalization
583 );
584 /**
585  * \en The performance of the FFT_Fwd512 routine depends on memory allocation for
586  * input,output and temporary buffers.
587  * For the maximum speed performance it is recommended
588  * to use the following configuration:
589  *   GSrcBuffer: Global SRAM
590  *   GDstBuffer: Local SRAM
591  *   LBuffer : Local SRAM
592  *   GBuffer : Global SRAM
593  *
594  * For this configuration the following results were achieved:
595  *   8766 clocks - full operation (0.22 ms at 40MHz CPU)
596  *   8180 clocks - without normalization (0.2 ms at 40MHz CPU)
597  *
598  * If you are not going to use this routine as a C callable function,
599  * you can reduce the number of instructions removing all stack operations. In this case
600  * the total execution time can be reduced by around 50 clocks.
601 //=====
614 void FFT_Inv512Set6bit();// Sets 6-bit accuracy of sin-cosine coefficients
622 void FFT_Inv512Set7bit();// Sets 7-bit accuracy of sin-cosine coefficients
623
745 void FFT_Inv512(
746     nm32sc* GSrcBuffer, // Source buffer :long[512]
747     nm32sc* LDstBuffer, // Result FFT :long[512]
748     void*   LBuffer,   // Temp buffer :long[512*3]
749     void*   GBuffer,   // Temp buffer :long[512*3]
750     int     ShiftR1=9, // First shift normalization
751     int     ShiftR2=-1 // Final shift normalization
752 );
753 /**
754  * \en The performance of the FFT_Fwd256 routine depends on memory allocation for
755  * input,output and temporary buffers.
756  * For the maximum speed performance it is recommended
757  * to use the following configuration:
758  *   GSrcBuffer: Global SRAM
759  *   LDstBuffer: Local SRAM
760  *   LBuffer : Local SRAM
761  *   GBuffer : Global SRAM
762  *
763  * For this configuration the following results were achieved:
764  *   9407 clocks - full operation (0.24ms at 40MHz CPU)
765  *   8199 clocks - without normalization (0.2ms at 40MHz CPU)
766  *
767  * If you are not going to use this routine as a C callable function,
768  * you can reduce the number of instructions removing all stack operations. In this case
769  * the total execution time can be reduced by around 50 clocks.
770
771 //=====
784 void FFT_Fwd1024Set6bit();// Sets 6-bit accuracy of sin-cosine coefficients
793 void FFT_Fwd1024Set7bit();// Sets 7-bit accuracy of sin-cosine coefficients
794
886 void FFT_Fwd1024(
887     nm32sc* GSrcBuffer, // Source buffer :long[1024]
888     nm32sc* LDstBuffer, // Result FFT :long[1024]
889     void*   LBuffer,   // Temp buffer :long[1024*3]
890     void*   GBuffer,   // Temp buffer :long[1024]
891     int     ShiftR=-1 // Right shift normalization
892 );
893 /**
894  * \en The performance of the FFT_Fwd1024 routine depends on memory allocation for
895  * input,output and temporary buffers.
896  * For the maximum speed performance it is recommended
897  * to use the following configuration:
898  *   GSrcBuffer: Global SRAM
899  *   LDstBuffer: Local SRAM
900  *   LBuffer : Local SRAM
901  *   GBuffer : Global SRAM
902  *
903  * For this configuration the following results were achieved:
904  *   20041 clocks - full operation (0.5ms at 40MHz CPU)
905  *   18900 clocks - without normalization (0.47ms at 40MHz CPU)
906  *
907  * If you are not going to use this routine as a C callable function,
908  * you can reduce the number of instructions removing all stack operations. In this case
909  * the total execution time can be reduced by around 50 clocks.
910 //=====
923 void FFT_Inv1024Set6bit();// Sets 6-bit accuracy of sin-cosine coefficients
931 void FFT_Inv1024Set7bit();// Sets 7-bit accuracy of sin-cosine coefficients
932
1056 void FFT_Inv1024(

```

```

1057     nm32sc* GSrcBuffer, // Source buffer :long[1024]
1058     nm32sc* GDstBuffer, // Result FFT :long[1024]
1059     void*    LBuffer,   // Temp buffer :long[1024*3]
1060     void*    GBuffer,   // Temp buffer :long[1024*3]
1061     int      ShiftR1=10, // First Right shift normalization
1062     int      ShiftR2=-1 // Final Right shift normalization
1063 );
1064
1065 // \en The performance of the FFT_Fwd2048 routine depends on memory allocation for
1066 // input,output and temporary buffers.
1067 // For the maximum speed performance it is recommended
1068 // to use the following configuration:
1069 //     GSrcBuffer: Global SRAM
1070 //     LDstBuffer: Local SRAM
1071 //     LBuffer : Local SRAM
1072 //
1073 // For this configuration the following results were achieved:
1074 //     49800 clocks - full operation (1.25ms at 40 MHz CPU)
1075 //     47624 clocks - without normalization (1.2ms at 40 MHz CPU)
1076 //
1077 // If you are not going to use this routine as a C callable function,
1078 // you can reduce the number of instructions removing all stack operations. In this case
1079 // the total execution time can be reduced by around 50 clocks.
1080
1081 //===== FFT2048 =====
1082 // Forward FFT 2048
1095 void FFT_Fwd2048Set6bit(); // Sets 6-bit accuracy of sin-cosine coefficients
1103 void FFT_Fwd2048Set7bit(); // Sets 7-bit accuracy of sin-cosine coefficients
1104
1177 void FFT_Fwd2048(
1178     nm32sc* GSrcBuffer, // Source buffer :long[2048]
1179     nm32sc* GDstBuffer, // Result FFT :long[2048]
1180     void*    LBuffer,   // Temp buffer :long[2048*4]
1181     int      ShiftR=-1 // Right shift normalization
1182 );
1183 // \en The performance of the FFT_Fwd2048 routine depends on memory allocation for
1184 // input,output and temporary buffers.
1185 // For the maximum speed performance it is recommended
1186 // to use the following configuration:
1187 //     GSrcBuffer: Global SRAM
1188 //     LDstBuffer: Local SRAM
1189 //     LBuffer : Local SRAM
1190 //
1191 // For this configuration the following results were achieved:
1192 //     49800 clocks - full operation (1.25ms at 40 MHz CPU)
1193 //     47624 clocks - without normalization (1.2ms at 40 MHz CPU)
1194 //
1195 // If you are not going to use this routine as a C callable function,
1196 // you can reduce the number of instructions removing all stack operations. In this case
1197 // the total execution time can be reduced by around 50 clocks.
1198
1199 //===== Inversed FFT 2048 =====
1212 void FFT_Inv2048Set6bit(); // Sets 6-bit accuracy of sin-cosine coefficients
1220 void FFT_Inv2048Set7bit(); // Sets 7-bit accuracy of sin-cosine coefficients
1221
1344 void FFT_Inv2048(
1345     nm32sc* GSrcBuffer, // Source buffer :long[2048]
1346     nm32sc* LDstBuffer, // Result FFT :long[2048]
1347     void*    LBuffer,   // Temp buffer :long[2048*4]
1348     void*    GBuffer,   // Temp buffer :long[2048*4]
1349     int      ShiftR1=11, // First Right shift normalization
1350     int      ShiftR2=-1 // Final Right shift normalization
1351 );
1352 // \en The performance of the FFT_Fwd256 routine depends on memory allocation for
1353 // input,output and temporary buffers.
1354 // For the maximum speed performance it is recommended
1355 // to use the following configuration:
1356 //     GSrcBuffer: Global SRAM
1357 //     LDstBuffer: Local SRAM
1358 //     LBuffer : Local SRAM
1359 //     GBuffer : Global SRAM
1360 //
1361 // For this configuration the following results were achieved:
1362 //     52160 clocks - full operation (1.3 ms at 40MHz CPU)
1363 //     47780 clocks - without both normalizations (1.2ms at 40MHz CPU)
1364 //
1365 // If you are not going to use this routine as a C callable function,
1366 // you can reduce the number of instructions removing all stack operations. In this case
1367 // the total execution time can be reduced by around 50 clocks.#include "nmfft.h"
1368
1369 //===== FFT4096 =====
1375 // Forward FFT 4096
1376
1429 void FFT_Fwd4096(

```

```

1430     nm32sc* GSrcBuffer,    // Source buffer :long[4096]
1431     nm32sc* GDstBuffer,   // Result FFT  :long[4096]
1432     void*    LBuffer,      // Temp buffer :long[4096*2]
1433     void*    GBuffer       // Temp buffer :long[4096*3]
1434 );
1435
1436 //===== Inversed FFT 4096
1437 =====
1438 void FFT_Inv4096(
1439     nm32sc* GSrcBuffer,    // Source buffer :long[4096]
1440     nm32sc* GDstBuffer,   // Result FFT  :long[4096]
1441     void*    LBuffer,      // Temp buffer :long[4096*2]
1442     void*    GBuffer       // Temp buffer :long[4096*3]
1443 );
1444
1445 //===== FFT8192
1446 =====
1447 // Forward FFT 8192
1448
1449 void FFT_Fwd8192(
1450     nm32sc* LSrcBuffer,    // Source buffer :long[8192]
1451     nm32sc* GDstBuffer,   // Result FFT  :long[8192]
1452     void*    LBuffer,      // Temp buffer :long[8192]
1453     void*    GBuffer       // Temp buffer :long[8192*3]
1454 );
1455
1456 //===== Inversed FFT 8192
1457 =====
1458 void FFT_Inv8192(
1459     nm32sc* LSrcBuffer,    // Source buffer :long[8192]
1460     nm32sc* GDstBuffer,   // Result FFT  :long[8192]
1461     void*    LBuffer,      // Temp buffer :long[8192]
1462     void*    GBuffer       // Temp buffer :long[8192*3]
1463 );
1464
1465
1466
1467 #ifdef __cplusplus
1468 };
1469 #endif
1470
1471 #endif
1472

```

7.54 fftext.h

```

1 //*****RC Module Inc., Moscow, Russia*****
2 //** NeuroMatrix(r) NM6403 Software */
3 //** Fast Fourie Transform Library */
4 //** (C-callable functions) */
5 //** $Workfile:: FFTText.h */
6 //** Contents: Header file of FFT extended routines */
7 //** Author: S.Mushkaev */
8 //** Version 1.0 */
9 //** Start Date: 03.07.2001 */
10 //** Release $Date: 2005/02/10 11:47:59 $*/
11 //** */
12 //** */
13 //** */
14 //** */
15 //** */
16 //** */
17 //** */
18 //** */
19 //***** */
20
21 // LIBRARY pcfft.lib
22
23
24
25 // The functions declared below are used for modeling and researching FFT calculations on PC.
26 // using fixed and floating-point arithmetic on PC.
27 #ifndef _FFTEXT_H_INCLUDED_
28 #define _FFTEXT_H_INCLUDED_
29
30 #define FFT7BIT 1
31 #define FFT6BIT 2
32
33 #ifndef __NM__
34
35 #include "nm1l.h"
36 #include "nmtype.h"
37 #include "crtdbg2.h"
38

```

```

39
40
41 class Cplx_float
42 {
43 public:
44     double Re;
45     double Im;
46 } ;
47
48 class Wi_4096_fixed
49 {
50 public:
51     char Re;
52     char Im;
53 } ;
54
55
56
57 //*****
58 // Computing FFT-256 using fixed-point arithmetic
59 // This is the C equivalent of the FFT_Fwd256 Assembly Code
60
61 int FFT_Fwd256(
62     vec<cmplx<int> > &X,
63     vec<cmplx<int> > &Y,
64     vec<cmplx<char> >   &W1_256,
65     vec<cmplx<char> >   &W2_256,
66     int             Shr1,
67     int             Shr2
68 );
69 // Computing FFT-256 using floating-point arithmetic
70
71 int FFT256(
72     vec<cmplx<double> >    &X,    // Source buffer :long[256]
73     vec<cmplx<double> >    &Y    // Result FFT   :long[256]
74 );
75 // Initialization FFT table of char coefficients
76 void MakeTable256W1W2(
77     vec<cmplx<char> >   &W1_256,
78     vec<cmplx<char> >   &W2_256,
79     double Ampl
80 );
81
82
83 //*****
84 // Computing FFT-1024 using fixed-point arithmetic
85 // This is the C equivalent of the FFT_Fwd512 Assembly Code
86
87 int FFT_Fwd512(
88     vec<cmplx<int> > &X,    // input array
89     vec<cmplx<int> > &Y,    // output array
90     vec<cmplx<char> >   &W1_512, // table of coefficients
91     vec<cmplx<char> >   &W2_512, // table of coefficients
92     int Shr1,                // 1-st shiftr normalization
93     int Shr2                 // 2-nd shiftr normalization
94 );
95 // Computing FFT-512 using floating-point arithmetic
96
97 int FFT512(
98     vec<cmplx<double> >    &X,    // Source buffer :long[512]
99     vec<cmplx<double> >    &Y    // Result FFT   :long[512]
100 );
101 // Initialization FFT table of char coefficients
102 void MakeTable512W1W2(
103     vec<cmplx<char> >   &W1_512,
104     vec<cmplx<char> >   &W2_512,
105     double Ampl
106 );
107
108 //*****
109 // Computing FFT-1024 using fixed-point arithmetic
110 // This is the C equivalent of the FFT_Fwd1024 Assembly Code
111
112 int FFT_Fwd1024(
113     vec<cmplx<int> > &X,    // input array
114     vec<cmplx<int> > &Y,    // output array
115     vec<cmplx<char> >   &W1_1024, // table of coefficients
116     vec<cmplx<char> >   &W2_1024, // table of coefficients
117     int Shr1,                // 1-st shiftr normalization
118     int Shr2                 // 2-nd shiftr normalization
119 );
120 // Computing FFT-1024 using floating-point arithmetic
121
122 int FFT1024(
123     vec<cmplx<double> >    &X,    // Source buffer :long[1024]
124     vec<cmplx<double> >    &Y    // Result FFT   :long[1024]
125 );
126 // Initialization table of char coefficients
127 void MakeTable1024W1W2(
128     vec<cmplx<char> >   &W1_1024,
129     vec<cmplx<char> >   &W2_1024,
130     double Ampl
131 );
132
133 //*****
134 // Computing FFT-2048 using fixed-point arithmetic
135

```

```

136 // This is the C equivalent of the FFT_Fwd2048 Assembly Code
137 int FFT_Fwd2048(
138     vec<cmplx<int>> &X,    // input array
139     vec<cmplx<int>> &Y,    // output array
140     vec<cmplx<char>> &W1_2048, // table of coefficients
141     vec<cmplx<char>> &W2_2048, // table of coefficients
142     int Shr1,                // 1-st shiftr normalization
143     int Shr2                 // 2-nd shiftr normalization
144 );
146 // Computing FFT-2048 using floating-point arithmetic
147 int FFT2048(
148     vec<cmplx<double>> &X,    // Source buffer :long[1024]
149     vec<cmplx<double>> &Y    // Result FFT   :long[1024]
150 );
152 // Initialization table of char coefficients
153 void MakeTable2048W1W2(
154     vec<cmplx<char>> &W1_2048,
155     vec<cmplx<char>> &W2_2048,
156     double Ampl
157 );
158 ****
160 // Computing FFT-4096 using floating-point arithmetic
161 void FFT_Fwd4096_float1(
162     Cpx_float *X,    // Source buffer :long[4096]
163     Cpx_float *Y // Result FFT   :long[4096]
164 );
165 ****
167 // Computing IFFT-256 using fixed-point arithmetic
168 // This is the C equivalent of the FFT_Fwd256 Assembly Code
169 int FFT_Inv256(
170     vec<cmplx<int>> &X,
171     vec<cmplx<int>> &Y,
172     vec<cmplx<char>> &IW1_256,
173     vec<cmplx<char>> &IW2_256,
174     int Shr1,
175     int Shr2
176 );
178 // Computing IFFT-256 using floating-point arithmetic
179 int IFFT256(
180     vec<cmplx<double>> &X, // Source buffer :long[256]
181     vec<cmplx<double>> &Y // Result FFT   :long[256]
182 );
184 // Initialization table of char coefficients
185 void IMakeTable256W1W2(
186     vec<cmplx<char>> &IW1_256,
187     vec<cmplx<char>> &IW2_256,
188     double Ampl
189 );
190 ****
192 // Computing IFFT-512 using fixed-point arithmetic
193 // This is the C equivalent of the FFT_Inv512 Assembly Code
194 int FFT_Inv512(
195     vec<cmplx<int>> &X,    // input array
196     vec<cmplx<int>> &Y,    // output array
197     vec<cmplx<char>> &IW1_512, // table of coefficients
198     vec<cmplx<char>> &IW2_512, // table of coefficients
199     int Shr1,                // 1-st shiftr normalization
200     int Shr2                 // 2-nd shiftr normalization
201 );
203 // Computing FFT-512 using floating-point arithmetic
204 int IFFT512(
205     vec<cmplx<double>> &X, // Source buffer :long[512]
206     vec<cmplx<double>> &Y // Result IFFT   :long[512]
207 );
209 // Initialization table of char coefficients
210 void IMakeTable512W1W2(
211     vec<cmplx<char>> &IW1_512,
212     vec<cmplx<char>> &IW2_512,
213     double Ampl
214 );
215 ****
217 // Computing FFT-1024 using fixed-point arithmetic
218 // This is the C equivalent of the FFT_Fwd1024 Assembly Code
219 int FFT_Inv1024(
220     vec<cmplx<int>> &X,    // input array
221     vec<cmplx<int>> &Y,    // output array
222     vec<cmplx<char>> &IW1_1024, // table of coefficients
223     vec<cmplx<char>> &IW2_1024, // table of coefficients
224     int Shr1,                // 1-st shiftr normalization
225     int Shr2                 // 2-nd shiftr normalization
226 );
228 // Computing IFFT-1024 using floating-point arithmetic
229 int IFFT1024(
230     vec<cmplx<double>> &X, // Source buffer :long[1024]
231     vec<cmplx<double>> &Y // Result IFFT   :long[1024]
232 );
234 // Initialization table of char coefficients

```

```

235 void IMakeTable1024W1W2(
236     vec<cmplx<char>> &IW1_1024,
237     vec<cmplx<char>> &IW2_1024,
238     double Ampl);
239 //*****
240
242 // Computing IFFT-2048 using fixed-point arithmetic
243 // This is the C equivalent of the FFT_Inv2048 Assembly Code
244 int FFT_Inv2048(
245     vec<cmplx<int>> &X,           // input array
246     vec<cmplx<int>> &Y,           // output array
247     vec<cmplx<char>> &IW1_2048, // table of coefficients
248     vec<cmplx<char>> &IW2_2048, // table of coefficients
249     int Shr1,                  // 1-st shiftr normalization
250     int Shr2                  // 2-nd shiftr normalization
251 );
253 // Computing IFFT-2048 using floating-point arithmetic
254 int IFFT2048(
255     vec<cmplx<double>> &X, // Source buffer :long[1024]
256     vec<cmplx<double>> &Y // Result IFFT :long[1024]
257 );
259 // Initialization table of char coefficients
260 void IMakeTable2048W1W2(
261     vec<cmplx<char>> &IW1_2048,
262     vec<cmplx<char>> &IW2_2048,
263     double Ampl
264 );
265 //*****
267 // Computing IFFT-4096 using floating-point arithmetic
268 void FFT_Inv4096_float1(
269     Cpx_float *X, // Source buffer :long[4096]
270     Cpx_float *Y // Result IFFT :long[4096]
271 );
272 //*****
273 // Radix-2 FFT-256 Functions
274
275 // Computing IFFT-256 using floating-point arithmetic by radix-2 method
276 int IFFT256_radix2(
277     vec<cmplx<double>> &X, // Source buffer
278     vec<cmplx<double>> &Y // Result IFFT
279 );
280 // Computing FFT-256 using floating-point arithmetic by radix-2 method
281 int FFT256_radix2(
282     vec<cmplx<double>> &X, // Source buffer :long[256]
283     vec<cmplx<double>> &Y // Result FFT :long[256]
284 );
285 // Computing FFT-256 using fixed-point arithmetic by radix-2 method
286 int FFT256_int_radix2(
287     vec<cmplx<int>> &X,
288     vec<cmplx<int>> &Y);
289 // Computing FFT-256 using fixed-point arithmetic by radix-2 method
290 int IFFT256_int_radix2(
291     vec<cmplx<int>> &X,
292     vec<cmplx<int>> &Y
293 );
294
295
296 __INLINE__ void CmplxPack8to64(
297     vec<cmplx<char>> &W,
298     vec<nmint64s> &WRePacked,
299     vec<nmint64s> &WImPacked)
300 {
301     ASSERT(W.size==WRePacked.size*8);
302     ASSERT(W.size==WImPacked.size*8);
303
304     int64 PackageIm=0,PackageRe=0;
305     int64 ByteMask=0xFF;
306     for(int i=0,k=0;i<WRePacked.size;i++,k+=8)
307     {
308         for(int j=7;j>=0;j--)
309         {
310             PackageRe<=8;
311             PackageRe=PackageRe|(W[k+j].re& ByteMask);
312
313             PackageIm<=8;
314             PackageIm=PackageIm|(W[k+j].im& ByteMask);
315         }
316         WRePacked[i]=PackageRe;
317         WImPacked[i]=PackageIm;
318     }
319 }
320
321 __INLINE__ void OutTable(vec<nmint64s> &Table,nm8s*TableName,nm8s* Section)
322 __INLINE__ void OutTable(vec<nmint64s> &Table,char*TableName,char* Section)
323 __INLINE__ void OutTable(nmvec64s &Table,char*TableName,char* Section)
324 {
325 */

```

```

326 cout << "data" << ""<< Section << ""<< endl;
327
328 cout << "global " << TableName << ":long["<< dec << Table.size<"]=(" << endl;
329 cout << Table << ')';<< endl;
330
331 cout << "end " << ""<< Section << ""<< endl;
332 */
333 }
334
335
336
337 #endif // NM6403
338 #endif //_NMFFTEXT_H_INCLUDED_

```

7.55 fftref.h

```

1 #ifndef FFTREF_H_INCLUDED
2 #define FFTREF_H_INCLUDED
3 #include "nmtype.h"
4
5 void nmppsDFT16Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
6 void nmppsDFT16Fwd2x8_RefFloat(const nm32sc* src, nm32sc* dst);
7 void nmppsFFT16Fwd2x8_RefInt(const nm32sc* src, nm32sc* dst);
8 void nmppsFFT16Fwd8x2_RefFloat(const nm32sc* src, nm32sc* dst);
9 void nmppsFFT16Fwd2x8Time_RefFloat(const nm32sc* src, nm32sc* dst);
10 void nmppsFFT16Fwd2x8Freq_RefFloat(const nm32sc* src, nm32sc* dst);
11 void nmppsFFT16Fwd242_RefFloat(const nm32sc* src, nm32sc* dst);
12 void nmppsFFT16Fwd242_RefFloat(const nm32sc* src, nm32fc* dst);
13 void nmppsFFT16Fwd242_RefInt(const nm32sc* src, nm32sc* dst);
14 void nmppsFFT16Fwd242_RefNmc(const nm32sc* src, nm32sc* dst);
15
16
17
18 void nmppsDFT32Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
19 void nmppsFFT32Fwd282_RefFloat(const nm32sc* src, nm32sc* dst);
20
21 void nmppsDFT64Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
22 void nmppsFFT64Fwd88_RefFloat(const nm32sc* src, nm32sc* dst);
23
24 void nmppsDFT128Fwd_RefFlt (const nm32sc* src, nm32sc* dst);
25 void nmppsFFT128Fwd882_RefFlt(const nm32sc* src, nm32sc* dst);
26 void nmppsFFT128Fwd828_RefFlt(const nm32sc* src, nm32sc* dst);
27 void nmppsFFT128Fwd828_RefInt(const nm32sc* src, nm32sc* dst);
28 extern "C" {
29     void nmppsFFT128Fwd828_RefNmc(const nm32sc* src, nm32sc* dst);
30 }
31 void nmppsDFT256Fwd_RefFloat(const nm32sc* src, nm32sc* dst);
32 void nmppsFFT256Fwd2882_RefFloat(const nm32sc* src, nm32sc* dst);
33
34 /*
35 template <int rep, int nb_bits, int sb_bits> load_wfifo(void *wfifo) {
36
37 }
38 template<int rep> void load_data_vsum_data_vr(void *data, int vr) {
39
40 }
41
42 template<int rep> void save_afifo(void *dst) {
43
44 }*/
45 #endif

```

7.56 nmpls.h

```
1 #include "./nmpls/nmpls.h"
```

7.57 nmpls.h

```

1
2 #ifndef INCLUDED_nmpls
3 #define INCLUDED_nmpls
4
5 ///#include "nmtype.h"
6 ///#include "nmdef.h"
7 ///#include "nmpp.h"
8 #include "sfir.h"

```

```

9
32 //*****
33
34
35 #include "sCorrelation.h"
36 //*****
37
38 ///#include "sFiltration.h"
39
40
41 //*****
42
43 #include "sResample.h"
44
45 //*****
46 #include "fft.h"
47
48 #include "fft_32fc.h"
49
50 //*****
51
52 #endif

```

7.58 sConvolution.h

7.59 sCorrelation.h

```

1 //-----
2 // $Workfile::: sConvolution. $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:09:07 $
8 //
9 //-----
10 //
11 //
12 //
13 //-----
14
15 #ifndef _SCONVOLUTION_H_INCLUDED_
16 #define _SCONVOLUTION_H_INCLUDED_
17 #include "nmtype.h"
18
19 //*****
20
21 void nmppsXCorr_32s_32s(nm32s* pSrcVec, int nSrcVecSize,nm32s* pKernel, int nKernelSize, nm32s* pDstVec, void*
22 pTmpBuf);
23 void nmppsXCorr_32s_16s32s(nm16s* pSrcVec, int nSrcVecSize,nm32s* pKernel, int nKernelSize, nm32s* pDstVec, void*
24 pTmpBuf);
25 void nmppsXCorr_32s_8s32s(nm8s* pSrcVec, int nSrcVecSize,nm32s* pKernel, int nKernelSize, nm32s* pDstVec, void*
26 pTmpBuf);
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93 #endif

```

7.60 sFiltration.h

```

1 //-----
2 // $Workfile::: sFiltrat. $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:09:07 $
8 //
9 //-----
10 //
11 //
12 //
13 //-----
14
15 #ifndef _SFILTRATION_H_INCLUDED_
16 #define _SFILTRATION_H_INCLUDED_
17
18 #include "nmtype.h"
19
20
21
22
23
24
25
26

```

```

27
28
35 int SIG_Median_V9(nm8s* pVec);
36 int SIG_Median_V9(nm16s* pVec);
37 int SIG_Median_V9(nm32s* pVec);
38 int64b SIG_Median_V9(nm64s* pVec);
40
41 //*****
42
78 int nmpcMedian3_32u(int a,int b, int c);
79 uint32b nmpcMedian3_32u(uint32b a, uint32b b, uint32b c);
81
82 #endif // _SFILTRATION_H_INCLUDED_
83

```

7.61 sfir.h

```

1 //-----
2 // $Workfile::: sFiltrat $
3 // Векторно-матрична библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:09:07 $
8 //
9 //-----
10 //-----
11 //-----
12
13 #ifndef _SFIR_H_INCLUDED_
14 #define _SFIR_H_INCLUDED_
15
16 #include "nmtype.h"
17
18 #ifdef __cplusplus
19     extern "C" {
20
21 #ifndef _SFIR_H_INCLUDED_
22 #define _SFIR_H_INCLUDED_
23
24 #include "nmtype.h"
25
26 #ifdef __cplusplus
27     extern "C" {
28
29
30 //*****
31 #define NmppsFIRState void
32
33 void nmppsFIR_8s ( nm8s* pSrc, nm8s* pDst, int srcSize, NmppsFIRState* pState);
34 void nmppsFIR_8s16s ( nm8s* pSrc, nm16s* pDst, int srcSize, NmppsFIRState* pState);
35 void nmppsFIR_8s32s ( nm8s* pSrc, nm32s* pDst, int srcSize, NmppsFIRState* pState);
36 void nmppsFIR_16s ( nm16s* pSrc, nm16s* pDst, int srcSize, NmppsFIRState* pState);
37 void nmppsFIR_16s32s( nm16s* pSrc, nm32s* pDst, int srcSize, NmppsFIRState* pState);
38 void nmppsFIR_32s ( nm32s* pSrc, nm32s* pDst, int srcSize, NmppsFIRState* pState);
39
40
41 int nmppsFIRInit_8s (NmppsFIRState* pState, int* pTaps, int tapsLen);
42 int nmppsFIRInit_8s16s (NmppsFIRState* pState, int* pTaps, int tapsLen);
43 int nmppsFIRInit_8s32s (NmppsFIRState* pState, int* pTaps, int tapsLen);
44 int nmppsFIRInit_16s (NmppsFIRState* pState, int* pTaps, int tapsLen);
45 int nmppsFIRInit_16s32s(NmppsFIRState* pState, int* pTaps, int tapsLen);
46 int nmppsFIRInit_32s (NmppsFIRState* pState, int* pTaps, int tapsLen);
47
48 int nmppsFIRInit_Alloc_8s (NmppsFIRState** ppState, int* pTaps, int tapsLen);
49 int nmppsFIRInit_Alloc_8s16s (NmppsFIRState** ppState, int* pTaps, int tapsLen);
50 int nmppsFIRInit_Alloc_8s32s (NmppsFIRState** ppState, int* pTaps, int tapsLen);
51 int nmppsFIRInit_Alloc_16s (NmppsFIRState** ppState, int* pTaps, int tapsLen);
52 int nmppsFIRInit_Alloc_16s32s(NmppsFIRState** ppState, int* pTaps, int tapsLen);
53 int nmppsFIRInit_Alloc_32s (NmppsFIRState** ppState, int* pTaps, int tapsLen);
54
55
56 int nmppsFIRGetSize_8s (int tapsLen);
57 int nmppsFIRGetSize_8s16s (int tapsLen);
58 int nmppsFIRGetSize_8s32s (int tapsLen);
59 int nmppsFIRGetSize_16s (int tapsLen);
60 int nmppsFIRGetSize_16s32s(int tapsLen);
61 int nmppsFIRGetSize_32s (int tapsLen);
62
63 void nmppsFIRFree(NmppsFIRState* pState);
64
65
66
67 #ifdef __cplusplus
68     }
69
70
71
72 #endif // _SFILTRATION_H_INCLUDED_
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132 #ifdef __cplusplus
133     };
134
135
136
137 #endif // _SFILTRATION_H_INCLUDED_
138

```

7.62 sResample.h

```

1 //-----
2 // $Workfile::: sResampl $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2005/01/12 14:09:07 $
8 //
9 //-----#
10 //-----#
11 //-----#
12 #include "nmtype.h"
13 #ifndef _S_RESAMPLE_H_INCLUDED_
14 #define _S_RESAMPLE_H_INCLUDED_
15
16 #ifdef __cplusplus
17     extern "C" {
18 #endif
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65 void nmppsResampleDown2_8u8u(nm8u7b* pSrcVec, nm8u7b* pDstVec, int nSrcVecSize, nm64s* pKernel);
66 void nmppsResampleDown2_16u16u(nm16u15b* pSrcVec, nm16u15b* pDstVec, int nSrcVecSize, nm64s* pKernel);
67
68 //\param pTmpBuf
69 // \ru Временный буффер размера 2*nSize.
70 // \en Temporary buffer with size 2*nSize.
71
72 //void nmppsResampleDown2(nm8u* pSrcVec, nm16u* pDstVec, int nSrcVecSize, nm64s* pKernel, void* pTmpBuf);
73 void nmppsResampleDown2_8u16u(nm8u* pSrcVec,nm16u* pDstVec,void* pTmpBuf,int nSize);
74
75
76 //*****
77
78 void nmppsResampleUp3Down2_8s16s(nm8s* pSrcVec, nm16s* pDstVec, int nSrcVecSize, nm64s* pKernel);
79
80 //*****
81
82 void nmppsCreateResampleUp3Down2_8s16s(nm64s** pKernel, int nHint);
83 void nmppsCreateResampleDown2_8u8u(nm64s** pKernel, int nHint);
84 void nmppsCreateResampleDown2_16u16u(nm64s** pKernel, int nHint);
85
86 //*****
87
88 void nmppsSetResampleUp3Down2_8s16s(nm64s* pKernel);
89 void nmppsSetResampleDown2_8u8u(nm64s* pKernel);
90 void nmppsSetResampleDown2_16u16u(nm64s* pKernel);
91
92 //*****
93
94 void nmppsDecimate16_8s(nm8s* pSrcVec,nm8s* pDstVec,int nDstSize);
95 void nmppsDecimate16_16s(nm16s* pSrcVec,nm16s* pDstVec,int nDstSize);
96 void nmppsDecimate16_32s(nm32s* pSrcVec,nm32s* pDstVec,int nDstSize);
97 void nmppsDecimate16_64s(nm64s* pSrcVec,nm64s* pDstVec,int nDstSize);
98
99 //*****
100
101 void nmppsResampleUp3Down2_perf(nm8s* pSrcVec, nm16s* pDstVec, int nSrcVecSize, nm64s* pKernel);
102 void nmppsResampleDown2_perf_8u(nm8u7b* pSrcVec, nm8u7b* pDstVec, int nSrcVecSize, nm64s* pKernel);
103 void nmppsResampleDown2_perf_16u(nm16u15b* pSrcVec, nm16u15b* pDstVec, int nSrcVecSize, nm64s* pKernel);
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323

```

```
324
325 };
326 #endif
327
328
329
330
331 #ifdef __cplusplus
332     };
333 #endif
334
335
336 #endif //
```

7.63 nmplv.h

```
1 #include "./nmplv/nmplv.h"
```

7.64 nmplv.h

```
1
2 #ifndef INCLUDED_nmplv
3 #define INCLUDED_nmplv
4
5
6
7
8
9
10
11
12
13 //-----
14
15 //-----
16
17 //-----
18
19 //-----
20
21 //-----
22
23 //-----
24
25 //-----
26
27
28 //-----
29
30
31 //-----
32
33
34
35 //-----
36
37
38 #include "nmtype.h"
39 #include "nmdef.h"
40
41
42 //-----
43
44
45 //-----
46
47
48 //-----
49
50
51 //-----
52
53
54 //-----
55
56
57 //-----
58
59
59 //-----
60
61
62 //-----
63
64
65 //-----
66
67
68 //-----
69
70
71 //-----
72
73
74 //-----
75
76
77 //-----
78
79
79 //-----
80
81
82 //-----
83
84
85
86
87
88
89
89 //-----
90
91
92 //-----
93
94
95 //-----
96
97
98 //-----
99
100
101
102
103
104
105
106
107
108 //*****
109
110
111 //-----
112
113 //-----
114
115
116 //-----
117
118
119 //-----
120
121 //-----
122
123 //*****
124
125
126 //-----
127
128 //-----
129
130
131 //-----
132
133
134 //-----
135
136
137 //-----
138
139
140 //-----
141
142
143 //-----
144
145
146 //-----
147
148
149 //*****
150
151
152 //-----
153
154
155 //-----
156
157
158 //-----
159
160
161 //-----
162
163
164 //-----
165
166
167 //-----
168
169
170 //-----
171
172 //-----
173
174
175 //*****
176
177
178 //-----
179
180
181 //-----
182
183
184 //-----
185
186
187 //-----
188
189 //*****
190
191
192 //-----
193
194
195 //-----
196
197
198 //-----
199
200
201 //-----
202
203
204 //-----
205
206
207 //-----
208
209
210 //-----
211
212 //-----
213
214 //*****
215
216
217 //-----
218
219
220 //-----
221
222
223 //-----
224
225
226 //-----
227
228 //*****
229
230
231 //-----
232
233
234 //-----
235
236
237 //-----
238
239
240 //-----
241
242 //*****
243
244
245 //-----
246
247
248 //-----
249
250
251 //-----
252
253 //-----
254
255 #endif
```

7.65 nmtl.h

```
1 #include "nmtl/nmtl.h"
```

7.66 nmtl.h

```
1 #include "./nmtl/nmtl.h"
```

7.67 nmtl.h

```
1 #include "tmatrix.h"
2 #include "tnmint.h"
3 #include "tnmvec.h"
4 #include "tnmmtr.h"
5 #include "tcmplx.h"
6 #include "tnmivec.h"
7
8
9
10
11 //-----
12 //-----
13 //-----
14 //-----
```

7.68 sElementary.h

```
1 //-----
2 // $Workfile::: sElementary.h      $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 //-----
8 //-----
9 //-----
10 //-----
11
12 #ifndef __SARITHM_H_INCLUDED__
13 #define __SARITHM_H_INCLUDED__
14
15 #ifdef __cplusplus
16 extern "C" {
17 #endif
18 //-----
19
20 void nmppsSin_32f(const nm32f* pSrcVec, nm32f* pDstVec, int nSize);
21 void nmppsCos_32f(const nm32f* pSrcVec, nm32f* pDstVec, int nSize);
22
23 void nmppsDiv_32f(const nm32f *pSrcVec1, const nm32f *pSrcVec2, nm32f *pDstVec, int nSize);
24
25 void nmppsExp_32f(const nm32f *pSrcVec, nm32f *pDstVec, int nSize);
26 void nmppsExp_64f(const nm64f *pSrcVec, nm64f *pDstVec, int nSize);
27
28 void nmppsLn_32f(const nm32f *pSrcVec, nm32f *pDstVec, int nSize);
29 void nmppsLn_64f(const nm64f *pSrcVec, nm64f *pDstVec, int nSize);
30
31 void nmppsPowx_64f(const nm64f *pSrcVec, nm64f *pDstVec, nm32u Deg, int nSize);
32
33 void nmppsSqrt_32f(const nm32f* pSrcVec, nm32f* pDstVec, int nSize);
34
35 #ifdef __cplusplus
36 }
37 #endif
38 //-----
```

7.69 vArithmetics.h

```
1 //-----
2 // $Workfile::: vArithmetics.h      $
3 //-----
```

```

5 // Векторно-матричная библиотека
6 //
7 // Copyright (c) RC Module Inc.
8 //
9 // $Revision: 1.2 $      $Date: 2005/06/23 15:15:15 $
10 //
11 //-----
12
13 #ifndef _VARITHM_H_INCLUDED_
14 #define _VARITHM_H_INCLUDED_
15
16 #include "malloc32.h"
17
18 #ifdef __cplusplus
19     extern "C" {
20 #endif
21
22 #ifndef _VARITHM_H_INCLUDED_
23
24 #include "malloc32.h"
25
26 #ifdef __cplusplus
27     extern "C" {
28 #endif
29
30 void nmppsAbs_4s (const nm4s* pSrcVec, nm4s* pDstVec, int nSize);
31 void nmppsAbs_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
32 void nmppsAbs_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
33 void nmppsAbs_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
34 void nmppsAbs_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
35
36
37 void nmppsAbsl_4s (const nm4s* pSrcVec, nm4s* pDstVec, int nSize);
38 void nmppsAbsl_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
39 void nmppsAbsl_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
40 void nmppsAbsl_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
41 void nmppsAbsl_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
42
43
44 //*****
45
46 void nmppsNeg_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
47 void nmppsNeg_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
48 void nmppsNeg_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
49 void nmppsNeg_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
50
51
52 //*****
53 void nmppsAddC_8s (const nm8s* pSrcVec, int8b nVal, nm8s* pDstVec, int nSize);
54 void nmppsAddC_16s(const nm16s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
55 void nmppsAddC_32s(const nm32s* pSrcVec, int32b nVal, nm32s* pDstVec, int nSize);
56 void nmppsAddC_64s(const nm64s* pSrcVec, int64b nVal, nm64s* pDstVec, int nSize);
57
58
59 //*****
60 void nmppsAddC_p64s(const nm64s* pSrcVec, int64b* pnVal, nm64s* pDstVec, int nSize);
61
62
63 //*****
64 void nmppsAddC_32fcr(const nm32fc* pSrcVec, nm32fc* pDstVec, float nVal, int nSize);
65
66
67 //*****
68 void nmppsAdd_4s (const nm4s* pSrcVec1, const nm4s* pSrcVec2, nm4s* pDstVec, int nSize);
69 void nmppsAdd_8s (const nm8s* pSrcVec1, const nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
70 void nmppsAdd_16s(const nm16s* pSrcVec1, const nm16s* pSrcVec2, nm16s* pDstVec, int nSize);
71 void nmppsAdd_32s(const nm32s* pSrcVec1, const nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
72 void nmppsAdd_64s(const nm64s* pSrcVec1, const nm64s* pSrcVec2, nm64s* pDstVec, int nSize);
73
74
75 void nmppsAdd_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, nm32f* pDstVec, int nSize);
76
77
78 void nmppsAddC_32f (const nm32f* pSrcVec, nm32f* pDstVec, float C, int nSize);
79
80
81 void nmppsAddEx_64s (const nm64s* pSrcVec1, int srcStep1, const nm64s* pSrcVec2, int srcStep2, nm64s* pDstVec, int dstStep, int nSize );
82
83
84 void nmppsAdd4V_16s(
85     nm16s** Vectors,           // array of pointers to buffers :nm8s* Any [NumberOfBuffer]
86     nm16s* pDstVec,           // result buffer          :long Local [VecSize/4]
87     int nSize                 // buffer size in 8-bit elements:nSize  =[256,512,..]
88 );
89
90
91 //*****
92
93 void nmppsAdd_AddC_32s(nm32s* pSrcVec1, nm32s* pSrcVec2, int nVal, nm32s* pDstVec, int nSize);
94
95
96 void nmppsSubC_4s (const nm4s* pSrcVec, int4b nVal, nm4s* pDstVec, int nSize);
97 void nmppsSubC_8s (const nm8s* pSrcVec, int8b nVal, nm8s* pDstVec, int nSize);
98 void nmppsSubC_16s(const nm16s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
99 void nmppsSubC_32s(const nm32s* pSrcVec, int32b nVal, nm32s* pDstVec, int nSize);
100 void nmppsSubC_64s(const nm64s* pSrcVec, int64b nVal, nm64s* pDstVec, int nSize);

```

```

696
750 void nmppsSubC_32f (const nm32f* pSrcVec, nm32f* pDstVec, float C, int nSize);
752
753 //*****
754
808 void nmppsSubCRev_8s (const nm8s* pSrcVec, int8b nVal, nm8s* pDstVec, int nSize);
809 void nmppsSubCRev_16s(const nm16s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
810 void nmppsSubCRev_32s(const nm32s* pSrcVec, int32b nVal, nm32s* pDstVec, int nSize);
811 void nmppsSubCRev_64s(const nm64s* pSrcVec, int64b nVal, nm64s* pDstVec, int nSize);
812 //void nmppsSubCRev_64s(nm64s* pSrcVec, int64b* pnVal, nm64s* pDstVec, int nSize);
814
815 //*****
816
870 void nmppsSubCRev_32f (const nm32f* pSrcVec, nm32f* pDstVec, float C, int nSize);
872
926 void nmppsSub_4s (const nm4s* pSrcVec1, nm4s* pSrcVec2, nm4s* pDstVec, int nSize);
927 void nmppsSub_8s (const nm8s* pSrcVec1, nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
928 void nmppsSub_16s(const nm16s* pSrcVec1, nm16s* pSrcVec2, nm16s* pDstVec, int nSize);
929 void nmppsSub_32s(const nm32s* pSrcVec1, nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
930 void nmppsSub_64s(const nm64s* pSrcVec1, nm64s* pSrcVec2, nm64s* pDstVec, int nSize);
932
986 void nmppsSub_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, nm32f* pDstVec, int nSize);
988 //*****
989
1054 void nmppsAbsDiff_8s (const nm8s* pSrcVec1, nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
1055 void nmppsAbsDiff_16s(const nm16s* pSrcVec1, nm16s* pSrcVec2, nm16s* pDstVec, int nSize);
1056 void nmppsAbsDiff_32s(const nm32s* pSrcVec1, nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
1057 void nmppsAbsDiff_64s(const nm64s* pSrcVec1, nm64s* pSrcVec2, nm64s* pDstVec, int nSize);
1059
1060
1115 void nmppsAbsDiff_32f(const nm32f* pSrcVec1, nm32f* pSrcVec2, nm32f* pDstVec, int nSize);
1117 //*****
1118
1186 void nmppsAbsDiff1_8s(nm8s* pSrcVec1, nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
1188
1189 //*****
1190
1243 void nmppsMulC_8s (const nm8s* pSrcVec, int8b nVal, nm8s* pDstVec, int nSize);
1244 void nmppsMulC_8s16s (const nm8s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
1245 void nmppsMulC_16s (const nm16s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
1246 void nmppsMulC_16s32s(const nm16s* pSrcVec, int32b nVal, nm32s* pDstVec, int nSize);
1247 void nmppsMulC_32s (const nm32s* pSrcVec, int32b nVal, nm32s* pDstVec, int nSize);
1248 void nmppsMulC_32s64s(const nm32s* pSrcVec, int64b nVal, nm64s* pDstVec, int nSize);
1249 void nmppsMulC_64s (const nm64s* pSrcVec, int64b nVal, nm64s* pDstVec, int nSize);
1250
1251 void nmppsMulC_2s16s (const nm2s* pSrcVec, int16b nVal, nm16s* pDstVec, int nSize);
1253
1254
1298 void nmppsMulC_32f(const nm32f* pSrcVec, nm32f* pDstVec, float C, int nSize);
1300
1301
1356 void nmppsMul_Mul_Add_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, const nm32f* pSrcVec3, const nm32f*
    pSrcVec4, nm32f* pDstVec, int nSize);
1357 void nmppsMul_Mul_Add_32fcr(const nm32fcr* pSrcVec1, const nm32fcr* pSrcVec2, const nm32fcr* pSrcVec3, const
    nm32fcr* pSrcVec4, nm32fcr* pDstVec, int nSize);
1358 void nmppsMul_Mul_Add_64f(const nm64f* pSrcVec1, const nm64f* pSrcVec2, const nm64f* pSrcVec3, const nm64f*
    *pSrcVec4, nm64f* pDstVec, int nSize);
1360
1409 void nmppsMul_Add_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, const nm32f* pSrcVecAdd, nm32f* pDstVec, int
    nSize);
1410 void nmppsMul_Add_64f(const nm64f* pSrcVec1, const nm64f* pSrcVec2, const nm64f* pSrcVecAdd, nm64f* pDstVec, int
    nSize);
1411 void nmppsMul_Add_32fcr(const nm32fcr* pSrcVec1, const nm32fcr* pSrcVec2, const nm32fcr* pSrcVecAdd, nm32fcr*
    pDstVec, int nSize);
1413
1468 void nmppsMul_ConjMul_Add_32fcr(const nm32fcr* pSrcVec1, const nm32fcr* pSrcVec2, const nm32fcr* pSrcVec3, const
    nm32fcr* pSrcVec4, nm32fcr* pDstVec, int nSize);
1470
1519 void nmppsMulC_AddC_32f(const nm32f* pSrcVec, float nMulC, float nAddC, nm32f* pDstVec, int nSize);
1521
1576 void nmppsMulC_AddV_AddC_32f(nm32f* pSrcVec, float nMulC, nm32f* pVecAdd, float nAddC, nm32f* pDstVec, int
    nSize);
1578
1621 void nmppsMul_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, nm32f* pDstVec, int nSize);
1622 void nmppsMul_32fcr(const nm32fcr* pSrcVec1, const nm32fcr* pSrcVec2, nm32fcr* pDstVec, int nSize);
1624
1667 void nmppsConjMul_32fcr(const nm32fcr* pSrcVec1, const nm32fcr* pSrcVec2, nm32fcr* pDstVec, int nSize);
1669
1724 void nmppsMul_Mul_Sub_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, const nm32f* pSrcVec3, const nm32f*
    pSrcVec4, nm32f* pDstVec, int nSize);
1726
1771 void nmppsMulC_AddV_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, nm32f* pDstVec, float C, int nSize);
1773 //*****
1774
1775
1776 void nmppsMulC_AddC_2s16s(const nm2s* pSrcVec, int32b nMulC, int nAddC, nm16s* pDstVec, int nSize);
1777

```

```

1778 //*****
1779
1838 void nmppsMul_AddC_64s(const nm64s* pSrcVec1,const nm64s* pSrcVec2, const nm64s* pnVal, nm64s* pDstVec, int
nSize);
1840
1841 //*****
1842
1901 void nmppsMul_AddC_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, float nValueAddC, nm32f* pDstVec, int nSize);
1903 //*****
1904
1963 void nmppsMulC_AddC_32s(const nm32s* pSrcVec, int nMulVal, int nAddVal, nm32s* pDstVec, int nSize);
1965
1966
1967
1968
1993 void nmppsMulC_AddC_2x32s(int32x2* dataSparseSrc, int32x2* mulArg, int32x2* addArg, int32x2 *dataSparseDst, int
size, int stepSparseSrc, int stepSparseDst);
1995
1996
2023 void nmppsRShiftC_MulC_AddC_2x32s(int32x2* dataSparseSrc, int32x2* preshiftArg, int32x2* mulArg, int32x2*
addArg, int32x2 *dataSparseDst, int size, int stepSparseSrc, int stepSparseDst);
2025
2026
2027 //*****
2028
2093 void nmppsMulC_AddV_AddC_32s(nm32s* pSrcVec1, int nMulVal, nm32s* pSrcVec2, int nAddVal, nm32s* pDstVec, int
nSize);
2095
2096 //*****
2097
2155 void nmppsSumN_8s16s(nm8s ** ppSrcVec, nm16s* pDstVec, int nSize, int nNumberOfVectors);
2156 void nmppsSumN_16s (nm16s ** ppSrcVec, nm16s* pDstVec, int nSize, int nNumberOfVectors);
2158
2159
2160 void nmppsSum4_16s(
2161     nm16s** Vectors,           // array of pointers to buffers :nm8s* Any [NumberOfBuffer]
2162     nm16s* pDstVec,           // result buffer          :long Local [VecSize/4]
2163     int      nSize            // buffer size in 8-bit elements:nSize  =[256,512,..]
2164 );
2165
2166 //*****
2167
2256 void nmppsDivC_32s(nm32s* pSrcVec, int nDivisor, nm32s* pDstVec, int nSize, void* pTmpBuf1, void* pTmpBuf2);
2258
2259 //*****
2260
2308 void nmppsSum_8s (const nm8s* pSrcVec, int nSize, int32b *pnRes);
2309 void nmppsSum_16s(const nm16s* pSrcVec, int nSize, int64b *pnRes);
2310 void nmppsSum_32s(const nm32s* pSrcVec, int nSize, int64b *pnRes);
2311 void nmppsSum_64s(const nm64s* pSrcVec, int nSize, int64b *pnRes);
2313
2314 //*****
2315
2365 void nmppsSum_1 (const nm1* pSrcVec, int nSize, int32b* pnRes, void* pTmpBuf);
2367
2422
2423
2424
2425 /*
2426 void nmppsDotProd_8s8sm (nm8s* pSrcVec1, nm8s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2427 void nmppsDotProd_8s16sm(nm8s* pSrcVec1, nm16s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2428 void nmppsDotProd_8s32sm(nm8s* pSrcVec1, nm32s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2429 void nmppsDotProd_8s64s(nm8s* pSrcVec1, nm64s* pSrcVec2, int nSize, int64b* pnRes);
2430
2431 void nmppsDotProd_16s16sm(nm16s* pSrcVec1, nm16s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2432 void nmppsDotProd_16s32sm(nm16s* pSrcVec1, nm32s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2433 void nmppsDotProd_16s64s(nm16s* pSrcVec1, nm64s* pSrcVec2, int nSize, int64b* pnRes);
2434
2435 void nmppsDotProd_32s32sm(nm32s* pSrcVec1, nm32s* pSrcVec2, int nSize, int64b* pnRes, SpecTmp1* spec);
2436 void nmppsDotProd_32s64s(nm32s* pSrcVec1, nm64s* pSrcVec2, int nSize, int64b* pnRes);
2437
2438 void nmppsDotProd_64s64s(nm64s* pSrcVec1, nm64s* pSrcVec2, int nSize, int64b* pnRes);
2439 */
2440 int nmppsDotProd_8s8sm (const nm8s* pSrcVec1, const nm8s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2441 int nmppsDotProd_8s16sm (const nm8s* pSrcVec1, const nm16s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2442 int nmppsDotProd_8s32sm (const nm8s* pSrcVec1, const nm32s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2443 int nmppsDotProd_16s16sm(const nm16s* pSrcVec1, const nm16s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2444 int nmppsDotProd_16s32sm(const nm16s* pSrcVec1, const nm32s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2445 int nmppsDotProd_32s32sm(const nm32s* pSrcVec1, const nm32s* pSrcVec2, int nSize, int64b* pnRes, nm64s* tmp);
2447
2500 void nmppsDotProd_8s64s (const nm8s* pSrcVec1, const nm64s* pSrcVec2, int nSize, int64b* pnRes);
2501 void nmppsDotProd_16s64s (const nm16s* pSrcVec1, const nm64s* pSrcVec2, int nSize, int64b* pnRes);
2502 void nmppsDotProd_32s64s (const nm32s* pSrcVec1, const nm64s* pSrcVec2, int nSize, int64b* pnRes);
2503 void nmppsDotProd_64s64s (const nm64s* pSrcVec1, const nm64s* pSrcVec2, int nSize, int64b* pnRes);
2505
2511 //void nmppsDotProd_16sc(nm16sc *pSrcVec1, nm64sc *pSrcVec2, int nSize, nm64sc *pnRes); //pc version is not available
2512 void nmppsDotProd_64sc(nm64sc *pSrcVec1, nm64sc *pSrcVec2, int nSize, nm64sc *pnRes);

```

```

2514
2515 //*****
2516 ****
2517 void nmppsWeightedSum_8s16s(nm8s* pSrcVec1,int nW1,nm8s* pSrcVec2,int nW2, nm16s* pDstVec, int nSize);
2518 void nmppsWeightedSum_16s32s(nm16s* pSrcVec1,int nW1,nm16s* pSrcVec2,int nW2, nm32s* pDstVec, int nSize);
2519 void nmppsWeightedSum_32s64s(nm32s* pSrcVec1,nm64s nW1,nm32s* pSrcVec2,nm64s nW2, nm64s* pDstVec, int nSize);
2520
2521 #ifdef __cplusplus
2522     };
2523 #endif
2524
2525 #endif // _VECARITM_H_INCLUDED_

```

7.70 vArithmeticsDev.h

```

1 //-----
2 //-
3 // $Workfile::: vArithmeticsDev.h      $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.5 $      $Date: 2005/07/13 14:19:56 $
10 //-
11 //-----
12
13 #ifndef _VARITHMETICSDEV_H_INCLUDED_
14 #define _VARITHMETICSDEV_H_INCLUDED_
15
16 #ifdef __cplusplus
17     extern "C" {
18 #endif
19
20
26 void nmppsAdd_64sc(
27     const nm64sc*pSrcVec1,
28     const nm64sc*pSrcVec2,
29     nm64sc*pDstVec,
30     int nSize);
32
38 void nmppsSub_64sc(nm64sc*pSrcVec1, nm64sc*pSrcVec2, nm64sc*pDstVec, int nSize);
40
41
47 void nmppsMulC_64sc(nm64sc*pSrcVec, nm64s*pnVal, nm64sc*pDstVec, int nSize);
49
50 #ifdef __cplusplus
51     };
52 #endif
53
54
55 #endif

```

7.71 vBitwise.h

```

1 //-----
2 //-
3 // $Workfile::: vBitwis $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.1 $      $Date: 2004/11/22 13:50:02 $
10 //-
19 //-----
20 #ifndef _VBITWISE_H_INCLUDED_
21 #define _VBITWISE_H_INCLUDED_
22
23 #ifdef __cplusplus
24     extern "C" {
25 #endif
26
27
28 // #include "Vector.h"
29
84 void nmppsNot_2u (const nm2u* pSrcVec, nm2u* pDstVec, int nSize);
85 void nmppsNot_4u (const nm4u* pSrcVec, nm4u* pDstVec, int nSize);

```

```

86 void nmppsNot_8u (const nm8u* pSrcVec, nm8u* pDstVec, int nSize);
87 void nmppsNot_16u(const nm16u* pSrcVec, nm16u* pDstVec, int nSize);
88 void nmppsNot_32u(const nm32u* pSrcVec, nm32u* pDstVec, int nSize);
89 void nmppsNot_64u(const nm64u* pSrcVec, nm64u* pDstVec, int nSize);
91
92 //*****
93
145 void nmppsAndC_4u (const nm4u* pSrcVec, uint4b nVal, nm4u* pDstVec, int nSize);
146 void nmppsAndC_8u (const nm8u* pSrcVec, uint8b nVal, nm8u* pDstVec, int nSize);
147 void nmppsAndC_16u(const nm16u* pSrcVec, uint16b nVal, nm16u* pDstVec, int nSize);
148 void nmppsAndC_32u(const nm32u* pSrcVec, uint32b nVal, nm32u* pDstVec, int nSize);
149 void nmppsAndC_64u(const nm64u* pSrcVec, uint64b nVal, nm64u* pDstVec, int nSize);
151
152 void nmppsAndC_p64u(nm64u* pSrcVec, nm64u* pnVal, nm64u* pDstVec, int nSize);
153
154 //*****
155
207 void nmppsAnd_1 (const nm1* pSrcVec1, const nm1* pSrcVec2, nm1* pDstVec, int nSize);
208 void nmppsAnd_2u (const nm2u* pSrcVec1, const nm2u* pSrcVec2, nm2u* pDstVec, int nSize);
209 void nmppsAnd_4u (const nm4u* pSrcVec1, const nm4u* pSrcVec2, nm4u* pDstVec, int nSize);
210 void nmppsAnd_8u (const nm8u* pSrcVec1, const nm8u* pSrcVec2, nm8u* pDstVec, int nSize);
211 void nmppsAnd_16u(const nm16u* pSrcVec1, const nm16u* pSrcVec2, nm16u* pDstVec, int nSize);
212 void nmppsAnd_32u(const nm32u* pSrcVec1, const nm32u* pSrcVec2, nm32u* pDstVec, int nSize);
213 void nmppsAnd_64u(const nm64u* pSrcVec1, const nm64u* pSrcVec2, nm64u* pDstVec, int nSize);
215
275 void nmppsAnd4V_64u(nm64u* pSrcVec1, nm64u* pSrcVec2, nm64u* pSrcVec3, nm64u* pSrcVec4, nm64u* pDstVec, int
nSize);
277
278 // numVecs>2
279 void nmppsAndNV_64u(nm64u** pSrcVecs, int numVecs, nm64u* pDstVec, int nSize);
280 //*****
281
333 void nmppsAndNotV_64u(nm64u* pSrcVec1, nm64u* pSrcVec2, nm64u* pDstVec, int nSize);
335
336 //*****
337
389 void nmppsOrC_8u (const nm8u* pSrcVec, uint8b nVal, nm8u* pDstVec, int nSize);
390 void nmppsOrC_16u(const nm16u* pSrcVec, uint16b nVal, nm16u* pDstVec, int nSize);
391 void nmppsOrC_32u(const nm32u* pSrcVec, uint32b nVal, nm32u* pDstVec, int nSize);
392 void nmppsOrC_64u(const nm64u* pSrcVec, uint64b nVal, nm64u* pDstVec, int nSize);
394
395 //*****
396
448 void nmppsOr_1 (const nm1* pSrcVec1, const nm1* pSrcVec2, nm1* pDstVec, int nSize);
449 void nmppsOr_2u (const nm2u* pSrcVec1, const nm2u* pSrcVec2, nm2u* pDstVec, int nSize);
450 void nmppsOr_4u (const nm4u* pSrcVec1, const nm4u* pSrcVec2, nm4u* pDstVec, int nSize);
451 void nmppsOr_8u (const nm8u* pSrcVec1, const nm8u* pSrcVec2, nm8u* pDstVec, int nSize);
452 void nmppsOr_16u(const nm16u* pSrcVec1, const nm16u* pSrcVec2, nm16u* pDstVec, int nSize);
453 void nmppsOr_32u(const nm32u* pSrcVec1, const nm32u* pSrcVec2, nm32u* pDstVec, int nSize);
454 void nmppsOr_64u(const nm64u* pSrcVec1, const nm64u* pSrcVec2, nm64u* pDstVec, int nSize);
456
512 void nmppsOr3V_64u(nm64u* pSrcVec1, nm64u* pSrcVec2, nm64u* pSrcVec3, nm64u* pDstVec, int nSize);
514
574 void nmppsOr4V_64u(nm64u* pSrcVec1, nm64u* pSrcVec2, nm64u* pSrcVec3, nm64u* pSrcVec4, nm64u* pDstVec, int
nSize);
576
577 // numVecs>2
578 void nmppsOrNV_64u(nm64u** pSrcVecs, int numVecs, nm64u* pDstVec, int nSize);
579
580 //*****
581
636 //-----
638 void nmppsXorC_8u (const nm8u* pSrcVec, uint8b nVal, nm8u* pDstVec, int nSize);
639 void nmppsXorC_16u(const nm16u* pSrcVec, uint16b nVal, nm16u* pDstVec, int nSize);
640 void nmppsXorC_32u(const nm32u* pSrcVec, uint32b nVal, nm32u* pDstVec, int nSize);
641 void nmppsXorC_64u(const nm64u* pSrcVec, uint64b nVal, nm64u* pDstVec, int nSize);
643
644 //*****
645
697 void nmppsXor_4u (const nm4u* pSrcVec1, const nm4u* pSrcVec2, nm4u* pDstVec, int nSize);
698 void nmppsXor_8u (const nm8u* pSrcVec1, const nm8u* pSrcVec2, nm8u* pDstVec, int nSize);
699 void nmppsXor_16u(const nm16u* pSrcVec1, const nm16u* pSrcVec2, nm16u* pDstVec, int nSize);
700 void nmppsXor_32u(const nm32u* pSrcVec1, const nm32u* pSrcVec2, nm32u* pDstVec, int nSize);
701 void nmppsXor_64u(const nm64u* pSrcVec1, const nm64u* pSrcVec2, nm64u* pDstVec, int nSize);
703
704 //*****
705
764 void nmppsMaskV_64u(nm64u* pSrcVec1, nm64u* pSrcVec2, nm64u* pMaskVec, nm64u* pDstVec, int nSize);
766
767 //*****
768
840 void nmppsRShiftC_8s (const nm8s* pSrcVec, int nShift, nm8s* pDstVec, int nSize);
841 void nmppsRShiftC_16s(const nm16s* pSrcVec, int nShift, nm16s* pDstVec, int nSize);
842 void nmppsRShiftC_32s(const nm32s* pSrcVec, int nShift, nm32s* pDstVec, int nSize);
843 void nmppsRShiftC_64s(const nm64s* pSrcVec, int nShift, nm64s* pDstVec, int nSize);
845
846 //*****

```

```

847 void nmppsRShiftC_4u (const nm4u* pSrcVec, int nShift, nm4u* pDstVec, int nSize);
848 void nmppsRShiftC_8u (const nm8u* pSrcVec, int nShift, nm8u* pDstVec, int nSize);
849 void nmppsRShiftC_16u (const nm16u* pSrcVec, int nShift, nm16u* pDstVec, int nSize);
850 void nmppsRShiftC_32u (const nm32u* pSrcVec, int nShift, nm32u* pDstVec, int nSize);
851 void nmppsRShiftC_64u (const nm64u* pSrcVec, int nShift, nm64u* pDstVec, int nSize);
852
853
854
1000 void nmppsRShiftC_AddC_8u (const nm8u *pSrcVec, int nShift, uint8b nAddVal, nm8u *pDstVec, int nSize);
1001 void nmppsRShiftC_AddC_16u (const nm16u *pSrcVec, int nShift, uint16b nAddVal, nm16u *pDstVec, int nSize);
1002 void nmppsRShiftC_AddC_32u (const nm32u *pSrcVec, int nShift, uint32b nAddVal, nm32u *pDstVec, int nSize);
1003
1004 //*****
1005 //*****
1006 //*****
1007
1008 void nmppsFwdShiftBitstream(const nm64u* pSrcVec, nm64u* pDstVec, nm64u* pnBits, int nBits, int nSize);
1009
1010 #ifdef __cplusplus
1011 }
1012 #endif
1093 #endif // _VBITWISE_H_INCLUDED_

```

7.72 vInit.h

```

1 // Векторно-матричная библиотека
2 // Copyright (c) RC Module Inc.
3 // $Revision: 1.1 $      $Date: 2004/11/22 13:50:02 $
4 // -----
5 #ifndef _VINIT_H_INCLUDED_
6 #define _VINIT_H_INCLUDED_
7
8 //-----
17 #ifndef _VINIT_H_INCLUDED_
18 #define _VINIT_H_INCLUDED_
19
20
21 #ifdef __cplusplus
22     extern "C" {
23 #endif
24
72
73 void nmppsSet_8s ( int8b val, nm8s* pDst, int len);
74 void nmppsSet_16s ( int16b val, nm16s* pDst, int len);
75 void nmppsSet_32s ( int32b val, nm32s* pDst, int len);
76 void nmppsSet_64s ( int64b val, nm64s* pDst, int len);
77 void nmppsSet_64sc(int64sc val, nm64sc* pDst, int len);
78 //void nmppsSet_64sp( int64b* pVal,nm64s* pDst, int len);
79 //__INLINE__ void nmppsSet_64s(nm64s* pDst, int64b val, int len) {nmppsSet_64sp((nm64s*) pDst, (int64b*)&val,
80 //len);}
80
81 __INLINE__ void nmppsSet_8u ( uint8b val, nm8u* pDst, int len) {nmppsSet_8s ( (int8b)val, (nm8s*) pDst, len);}
82 __INLINE__ void nmppsSet_16u(uint16b val, nm16u* pDst, int len) {nmppsSet_16s((int16b)val, (nm16s*) pDst, len);}
83 __INLINE__ void nmppsSet_32u(uint32b val, nm32u* pDst, int len) {nmppsSet_32s((int32b)val, (nm32s*) pDst, len);}
84 __INLINE__ void nmppsSet_64u(uint64b val, nm64u* pDst, int len) {nmppsSet_64s((int64b)val, (nm64s*) pDst, len);}
85
86 //__INLINE__ void nmppsSet_16sc (nm16sc val, nm16sc* pDst, int len);
87 //__INLINE__ void nmppsSet_32sc (nm32sc val, nm32sc* pDst, int len);
88
89
90 //__INLINE__ void nmppsSet_64up(nm64u* pDst, uint64b* val, int len){nmppsSet_64sp((nm64s*) pDst, (int64b*)val,
91 //len);}
92
93
94
95
96 //*****
97
128
133 //void nmppsRandUniform_32u(nm32u* pDstVec, int nSize, unsigned nRandomize );
134 void nmppsRandUniform_64s(nm64s* pDstVec, int nSize);
135 __INLINE__ void nmppsRandUniform_8s (nm8s* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
136 //nSize>3);}
136 __INLINE__ void nmppsRandUniform_16s(nm16s* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
137 //nSize>2);}
137 __INLINE__ void nmppsRandUniform_32s(nm32s* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
138 //nSize>1);}
138 __INLINE__ void nmppsRandUniform_8u (nm8u* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
139 //nSize>3);}
139 __INLINE__ void nmppsRandUniform_16u(nm16u* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
140 //nSize>2);}
140 __INLINE__ void nmppsRandUniform_32u(nm32u* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
141 //nSize>1);}


```

```

141 __INLINE__ void nmppsRandUniform_64u(nm64u* pDstVec, int nSize) {nmppsRandUniform_64s((nm64s*)pDstVec,
142   nSize );}
143 // __INLINE__ void nmppsRandUniform_64s(nm64s* pDstVec, int nSize, unsigned nRandomize = 1)
144   {nmppsRandUniform_32u((nm32u*)pDstVec, nSize<<1, nRandomize);}
145
146
147
148 //*****
149
150
151
152
153
154
155 //*****
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191 void nmppsRandUniform_64f(nm64f* pDstVec, int nSize, double low, double hi);
192 void nmppsRand_32f(nm32f* pDstVec, int nSize, float low, float hi);
193
194 void nmppsRandUniform_32f_integer(nm32f *pDstVec, int nSize, int hi, int low);
195
196
197
198
199 //*****
200
201
202
203 int nmppsRandUniform2_32s(int nMin, int nMax, int nDivisible);
204 int nmppsRandUniform3_32s(int nMin, int nMax);
205 int nmppsRandUniform();
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228 void nmppsRamp_8s(nm8s* pVec, int8_t nOffset, int8_t nSlope, int nSize);
229 void nmppsRamp_16s(nm16s* pVec, int16_t nOffset, int16_t nSlope, int nSize);
230 void nmppsRamp_32s(nm32s* pVec, int32_t nOffset, int32_t nSlope, int nSize);
231 void nmppsRamp_64s(nm64s* pVec, int64_t nOffset, int64_t nSlope, int nSize);
232
233 //*****
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295 void nmppsConvert_1s2s (const nm1* pSrcVec, nm2s* pDstVec, int nSize);
296 void nmppsConvert_1u2u (const nm1* pSrcVec, nm2u* pDstVec, int nSize);
297 void nmppsConvert_1u4u (const nm1* pSrcVec, nm4u* pDstVec, int nSize);
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512

```

```

513 void nmppsConvert_32f32s_ceiling (const nm32f* pSrcVec, nm32s* pDstVec, int scale, int nSize);
514 void nmppsConvert_32f32s_floor (const nm32f* pSrcVec, nm32s* pDstVec, int scale, int nSize);
515 void nmppsConvert_32f32s_truncate(const nm32f* pSrcVec, nm32s* pDstVec, int scale, int nSize);
517
543 void nmppsMerge_32f(const nm32f* pSrcVec1, const nm32f* pSrcVec2, nm32f* pDstVec, int nSize);
545
568 void nmppsConvertRisc_8u32u(const nm8u* pSrcVec, nm32u* pDstVec, int nSize);
569 void nmppsConvertRisc_32u8u(const nm32u* pSrcVec, nm8u* pDstVec, int nSize);
571
572 //*****
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619 void nmppsCopy_2s (const nm2s* pSrcVec, nm2s* pDstVec, int nSize);
620 void nmppsCopy_4s (const nm4s* pSrcVec, nm4s* pDstVec, int nSize);
621 void nmppsCopy_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
622 void nmppsCopy_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
623 void nmppsCopy_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
624 void nmppsCopy_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
625 void nmppsCopy_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
626
627 __INLINE__ void nmppsCopy_4u (const nm4u* pSrcVec, nm4u* pDstVec, int nSize) { nmppsCopy_4s (( nm4s*)
628     pSrcVec, (nm4s*) pDstVec, nSize);}
629 __INLINE__ void nmppsCopy_8u (const nm8u* pSrcVec, nm8u* pDstVec, int nSize) { nmppsCopy_8s (( nm8s*)
629     pSrcVec, (nm8s*) pDstVec, nSize);}
630 __INLINE__ void nmppsCopy_16u(const nm16u* pSrcVec, nm16u* pDstVec, int nSize) { nmppsCopy_16s(( nm16s*)
630     pSrcVec, (nm16s*) pDstVec, nSize);}
631 __INLINE__ void nmppsCopy_32u(const nm32u* pSrcVec, nm32u* pDstVec, int nSize) { nmppsCopy_32s(( nm32s*)
631     pSrcVec, (nm32s*) pDstVec, nSize);}
632 __INLINE__ void nmppsCopy_64u(const nm64u* pSrcVec, nm64u* pDstVec, int nSize) { nmppsCopy_64s(( nm64s*)
632     pSrcVec, (nm64s*) pDstVec, nSize);}
633 __INLINE__ void nmppsCopy_64sc(const nm64sc *pSrcVec, nm64sc *pDstVec, int nSize) { nmppsCopy_64s(( nm64s*)
633     pSrcVec, (nm64s*) pDstVec, 2*nSize);}

634
635
636 void nmppsCopy_32f(const float* pSrcVec, float* pDstVec, int size);
637
638 void nmppsCopyOddToOdd_32f(const float* pSrcVec, float* pDstVec, int size);
639
708 void nmppsCopyEvenToOdd_32f(const float* pSrcVec, float* pDstVec, int size);
710
711
729 void nmppsCopyOddToEven_32f(const float* pSrcVec, float* pDstVec, int size);
731
732
750 void nmppsCopyEvenToEven_32f(const float* pSrcVec, float* pDstVec, int size);
752
753
776 void nmppsCopyRisc_32f(const float* pSrcVec, float* pDstVec, int nSize);
778
779 //*****
780
845 void nmppsCopyua_8s(const nm8s* pSrcVec, int nSrcOffset, nm8s* pDstVec, int nSize);
847
848 //*****
849
890 void nmppsSwap_64s(nm64s* pSrcVec1, nm64s* pSrcVec2, int nSize);
892
893
894
895
896 #ifndef __cplusplus
897 };
898 #endif
899
900 #endif // _INIT_H_INCLUDED_

```

7.73 vSelect.h

```

1 //-----
2 // $Workfile::: vSelect.h $
3 // Векторно-матричная библиотека
4 // Copyright (c) RC Module Inc.
5 // $Revision: 1.2 $    $Date: 2005/07/13 14:19:56 $
6 // author: S. Mushkaev
7 //-----
8
9
10
11
12
13
14 #ifndef VSELECT_H
15 #define VSELECT_H
16
17 #include "malloc32.h"

```

```

18 #ifdef __cplusplus
19     extern "C" {
20 #endif
21
26
91 void nmppsMax_8s7b(const nm8s7b* pSrcVec, int nSize, int8b* n.MaxValue);
103 void nmppsMax_16s15b(const nm16s15b* pSrcVec, int nSize, int16b* n.MaxValue);
116 void nmppsMax_32s31b(const nm32s31b* pSrcVec, int nSize, int* n.MaxValue);
117 void nmppsMax_64s63b(const nm64s63b* pSrcVec, int nSize, int64b *n.MaxValue);
118
119 int nmppsMax_8sm (const nm8s* srcVec, int size, int8b* maxValue, nm16s* tmp);
120 int nmppsMax_16sm(const nm16s* srcVec, int size, int16b* maxValue, nm32s* tmp);
121 int nmppsMax_32sm(const nm32s* srcVec, int size, int32b* maxValue, nm64s* tmp);
122
123
125 //*****
126 //*****
127
128
129 void nmppsMin_8s7b(const nm8s7b* pSrcVec, int nSize, int8b* n.MinValue);
130 void nmppsMin_16s15b(const nm16s15b* pSrcVec, int nSize, int16b* n.MinValue);
131 void nmppsMin_32s31b(const nm32s31b* pSrcVec, int nSize, int* n.MinValue);
132 void nmppsMin_64s63b(const nm64s63b* pSrcVec, int nSize, int64b *n.MinValue);
133
134
135 int nmppsMin_8sm (const nm8s* srcVec, int size, int8b* minValue, nm16s* tmp);
136 int nmppsMin_16sm(const nm16s* srcVec, int size, int16b* minValue, nm32s* tmp);
137 int nmppsMin_32sm(const nm32s* srcVec, int size, int32b* minValue, nm64s* tmp);
138
139 //*****
140
141 void nmppsMaxIndx_8s(nm8s7b *pSrcVec, int nSize, int* nIndex, int8b *n.MaxValue, void *pLTmpBuf, void *pGTmpBuf,
142     int nSearchDir);
143 void nmppsMaxIndx_16s(nm16s15b *pSrcVec, int nSize, int* nIndex, int16b *n.MaxValue, void *pLTmpBuf, void
144     *pGTmpBuf, int nSearchDir);
145 void nmppsMaxIndx_32s(nm32s31b *pSrcVec, int nSize, int* nIndex, int32b *n.MaxValue, void *pLTmpBuf, void
146     *pGTmpBuf, int nSearchDir);
147
148 //*****
149
150 void nmppsMinIndx_8s(nm8s7b *pSrcVec, int nSize, int* nIndex, int8b *n.MinValue, void *pLTmpBuf, void *pGTmpBuf,
151     int nSearchDir);
152 void nmppsMinIndx_16s(nm16s15b *pSrcVec, int nSize, int* nIndex, int16b *n.MinValue, void *pLTmpBuf, void *pGTmpBuf,
153     int nSearchDir);
154 void nmppsMinIndx_32s(nm32s31b *pSrcVec, int nSize, int* nIndex, int32b *n.MinValue, void *pLTmpBuf, void *pGTmpBuf,
155     int nSearchDir);
156
157 //*****
158
159 int nmppsMinIndxV9_32s(int* pSrcVec, int nStride, int* nPos);
160 int nmppsMinIndxV16_32s(int* pSrcVec, int nStride, int* nPos);
161 int nmppsMinIndxV25_32s(int* pSrcVec, int nStride, int* nPos);
162 int nmppsMinIndxV256_32s(int* pSrcVec, int nStride, int* nPos);
163 int nmppsMinIndxV1024_32s(int* pSrcVec, int nStride, int* nPos);
164
165 //*****
166
167 int nmppsFirstZeroIndx_32s(int* pSrcVec, int nSize);
168 //*****
169
170 int nmppsFirstNonZeroIndx_32s(int* pSrcVec, int nSize);
171
172 //*****
173
174 int nmppsLastZeroIndx_32s(int* pSrcVec, int nSize);
175
176 //*****
177
178 void nmppsMinEvery_8s(nm8s7b* pSrcVec1, nm8s7b* pSrcVec2, nm8s7b* pDstMinVec, int nSize);
179 void nmppsMinEvery_16s(nm16s15b* pSrcVec1, nm16s15b* pSrcVec2, nm16s15b* pDstMinVec, int nSize);
180 void nmppsMinEvery_32s(nm32s31b* pSrcVec1, nm32s31b* pSrcVec2, nm32s31b* pDstMinVec, int nSize);
181 void nmppsMinEvery_64s(nm64s63b* pSrcVec1, nm64s63b* pSrcVec2, nm64s63b* pDstMinVec, int nSize);
182
183 //*****
184
185 // numVecs > 2
186 void nmppsMinNV_8s(nm8s7b** pSrcVecs, int numVecs, nm8s7b* pDstMinVec, int nSize);
187 void nmppsMinNV_16s(nm16s15b** pSrcVecs, int numVecs, nm16s15b* pDstMinVec, int nSize);
188 void nmppsMinNV_32s(nm32s31b** pSrcVecs, int numVecs, nm32s31b* pDstMinVec, int nSize);

```

```

699 void nmppsMinNV_64s(nm64s63b** pSrcVecs, int numVecs, nm64s63b* pDstMinVec, int nSize);
700
701 //*****
702
763 void nmppsMaxEvery_8s(nm8s7b* pSrcVec1, nm8s7b* pSrcVec2, nm8s7b* pDstMaxVec, int nSize);
764 void nmppsMaxEvery_16s(nm16s15b* pSrcVec1, nm16s15b* pSrcVec2, nm16s15b* pDstMaxVec, int nSize);
765 void nmppsMaxEvery_32s(nm32s31b* pSrcVec1, nm32s31b* pSrcVec2, nm32s31b* pDstMaxVec, int nSize);
766 void nmppsMaxEvery_64s(nm64s63b* pSrcVec1, nm64s63b* pSrcVec2, nm64s63b* pDstMaxVec, int nSize);
767
768
770 //*****
771
848 void nmppsMinCmpLtV_16s(nm16s15b* pSrcVec1, nm16s15b* pSrcVec2, nm16s15b* pDstMin, nm16s15b* pDstSignMask,
    int nSize);
850
851
852 //*****
853
911 void nmppsCmpLto_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
912 void nmppsCmpLto_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
913 void nmppsCmpLto_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
914 void nmppsCmpLto_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
916
932 void nmppsCmpLteC_v2nm32f(const v2nm32f* pSrcVec, const v2nm32f* C, nm1* evenFlags, nm1* oddFlags, int step, int
    nSize);
934
950 void nmppsCmpLtC_v2nm32f(const v2nm32f* pSrcVec, const v2nm32f* C, nm1* evenFlags, nm1* oddFlags, int step, int
    nSize);
952
953 //*****
954
1023
1026 void nmppsCmpEq0_8u7b(nm8u7b* pSrcVec, nm1* pDstVec, int nSize, int nTrueFlag);
1030 void nmppsCmpEq0_16u15b(nm16u15b* pSrcVec, nm1* pDstVec, int nSize, int nTrueFlag);
1034 void nmppsCmpEq0_32u31b(nm32u31b* pSrcVec, nm1* pDstVec, int nSize, int nTrueFlag);
1035
1036 //void nmppsCmpNe0_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
1037 //void nmppsCmpNe0_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
1038 //void nmppsCmpNe0_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
1039 //void nmppsCmpNe0_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
1040
1042
1043 //*****
1044
1102 void nmppsCmpGt0_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
1103 void nmppsCmpGt0_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
1104 void nmppsCmpGt0_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
1105 void nmppsCmpGt0_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
1107
1172 void nmppsMinMaxEvery_8s(nm8s* pSrcVec1, nm8s* pSrcVec2, nm8s* pDstMin, nm8s* pDstMax, int nSize);
1173 void nmppsMinMaxEvery_16s(nm16s* pSrcVec1, nm16s* pSrcVec2, nm16s* pDstMin, nm16s* pDstMax, int nSize);
1174 void nmppsMinMaxEvery_32s(nm32s* pSrcVec1, nm32s* pSrcVec2, nm32s* pDstMin, nm32s* pDstMax, int nSize);
1176
1177
1178 //*****
1179
1240 void nmppsClipPowC_8s(nm8s* pSrcVec, int nClipFactor, nm8s* pDstVec, int nSize);
1241 void nmppsClipPowC_16s(nm16s* pSrcVec, int nClipFactor, nm16s* pDstVec, int nSize);
1242 void nmppsClipPowC_32s(nm32s* pSrcVec, int nClipFactor, nm32s* pDstVec, int nSize);
1243 void nmppsClipPowC_64s(nm64s* pSrcVec, int nClipFactor, nm64s* pDstVec, int nSize);
1245
1246 //*****
1247
1323 void nmppsClipCC_32s(nm32s30b* pSrcVec, int30b nNegThresh, int30b nPosThresh, nm32s30b* pDstVec, int nSize);
1325
1394 void nmppsThreshold_Lt_Gt_32f(nm32f* pSrcVec, nm32f* pDstVec, float min, float max, int nSize);
1396
1397 //*****
1398
1494 void nmppsClipRShiftConvertAddC_16s8s(nm16s* pSrcVec, int nClipFactor, int nShift, int8b nAddValue, nm8s* pDstVec,
    int nSize);
1495 void nmppsClipRShiftConvertAddC_32s8s(nm32s* pSrcVec, int nClipFactor, int nShift, int8b nAddValue, nm8s* pDstVec,
    int nSize);
1497
1498 //extern "C" nm64u VEC_TBL_Diagonal_01h_G[8];
1499 //extern "C" nm64u VEC_TBL_Diagonal_01h_L[8];
1500
1551 //void nmppsClipConvertAddC_16s8s(nm16s* pSrcVec, int nClipFactor, int8b nAddValue, nm8s* pDstVec, int nSize,
    nm64u* weights); //=VEC_TBL_Diagonal_01h_G
1552 typedef nm64u NmppsWeightState;
1553 void nmppsClipConvertAddCIInitAlloc_16s8s(NmppsWeightState** ppState);
1554 void nmppsClipConvertAddC_16s8s(nm16s* pSrcVec, int nClipFactor, int8b nAddValue, nm8s* pDstVec, int nSize,
    NmppsWeightState* pState); //=VEC_TBL_Diagonal_01h_G
1555 void nmppsClipConvertAddCFree(NmppsWeightState* pState);
1557
1558
1559 //*****

```

```

1650
1651 //void nmppsClipRShiftConvert _AddC_Ext_(v8nm16s* pSrcVec, v8nm32s* pnClipFactor, v8nm32s* pnShift, v8nm8s*
1652   pnAdd, v8nm8s* pDstVec, int nSize);
1653 void nmppsClipRShiftConvert _AddC_Ext_(v8nm16s* pSrcVec, v8nm32s* pnClipFactor, v8nm32s* pnShift, v8nm32s*
1654   pnAdd, v8nm8s* pDstVec, int nSize);
1655 //void nmppsClipRShiftConvert _AddC_Ext_(v16nm16s* pSrcVec, v16nm32s* pnClipFactor, v16nm32s* pnShift, v16nm4s*
1656   pnAdd,
1657   v16nm4s* pDstVec, int nSize);
1658
1659 //*****
1660
1661 void nmppsCmpEqC_16u15b(nm16u15b* pSrcVec, uint15b nCmpVal, nm16s* pDstVec, int nSize, int16b nTrueFlag);
1671 void nmppsCmpEqC_8u7b( nm8u7b* pSrcVec, uint7b nCmpVal, nm8s* pDstVec, int nSize, int8b nTrueFlag);
1718 void nmppsCmpEqC_4u3b( nm4u3b* pSrcVec, uint3b nCmpVal, nm4s* pDstVec, int nSize, int4b nTrueFlag);
1720
1721 //*****
1723
1768
1769 void nmppsCmpNe0_8s (const nm8s* pSrcVec, nm8s* pDstVec, int nSize);
1770 void nmppsCmpNe0_16s(const nm16s* pSrcVec, nm16s* pDstVec, int nSize);
1771 void nmppsCmpNe0_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
1772 void nmppsCmpNe0_64s(const nm64s* pSrcVec, nm64s* pDstVec, int nSize);
1773
1775
1820
1821 void nmppsCmpEq0_32s(const nm32s* pSrcVec, nm32s* pDstVec, int nSize);
1822
1824
1825
1873 void nmppsCmpNeC_8s (const nm8s* pSrcVec, int8b nCmpVal, nm8s* pDstVec, int nSize);
1874 void nmppsCmpNeC_16s (const nm16s* pSrcVec, int16b nCmpVal, nm16s* pDstVec, int nSize);
1875 void nmppsCmpNeC_32s (const nm32s* pSrcVec, int32b nCmpVal, nm32s* pDstVec, int nSize);
1876 void nmppsCmpNeC_64s (const nm64s* pSrcVec, int64b nCmpVal, nm64s* pDstVec, int nSize);
1878
1939 void nmppsCmpNeC_8u7b (nm8u7b* pSrcVec, uint7b nCmpVal, nm8s* pDstVec, int nSize, int8b nTrueFlag);
1940 void nmppsCmpNeC_16u15b(nm16u15b* pSrcVec, uint15b nCmpVal, nm16s* pDstVec, int nSize, int16b nTrueFlag);
1942
1943 int nmppsCmpNeC_8s8um (const nm8s* pSrcVec, int8b nCmpVal, nm8u* pDstVec, int nSize, struct NmppsTmpSpec
1944 *spec);
1944 int nmppsCmpNeC_16s8um (const nm16s* pSrcVec, int16b nCmpVal, nm8u* pDstVec, int nSize, struct
1945 NmppsTmpSpec *spec);
1945 int nmppsCmpNeC_32s8um (const nm32s* pSrcVec, int32b nCmpVal, nm8u* pDstVec, int nSize, struct
1946 NmppsTmpSpec *spec);
1946 int nmppsCmpNeC_64s8um (const nm64s* pSrcVec, int64b nCmpVal, nm8u* pDstVec, int nSize, struct
1947 NmppsTmpSpec *spec);
1948
1949
1950
1951
1952 //spec.mode=ONE_TIME_ALLOC|ROUTE_ALLOC|OPTIMIZE_ROUTE;
1953
1954
1955 //int nmppsCmpLtc_8s8um (const nm8s* pSrcVec, int8b nCmpVal, nm8s* pDstVec, int nSize, struct
1956 NmppsTmpSpec *spec);
1956 //int nmppsCmpLtc_16s8um (const nm16s* pSrcVec, int16b nCmpVal, nm16s* pDstVec, int nSize, struct
1957 NmppsTmpSpec *spec);
1957 //int nmppsCmpLtc_32s8um (const nm32s* pSrcVec, int32b nCmpVal, nm32s* pDstVec, int nSize, struct
1958 NmppsTmpSpec *spec);
1958 //int nmppsCmpLtc_64s (const nm64s* pSrcVec, int64b nCmpVal, nm64s* pDstVec, int nSize, Tmp2BuffSpec *spec);
1959 //
1960 int nmppsCmpLtc_8s8um (const nm8s* pSrcVec, int8b nCmpVal, nm8u* pDstVec, int nSize, struct NmppsTmpSpec
1961 *spec);
1961 int nmppsCmpLtc_16s8um (const nm16s* pSrcVec, int16b nCmpVal, nm8u* pDstVec, int nSize, struct NmppsTmpSpec
1962 *spec);
1962 int nmppsCmpLtc_32s8um (const nm32s* pSrcVec, int32b nCmpVal, nm8u* pDstVec, int nSize, struct NmppsTmpSpec
1963 *spec);
1963 int nmppsCmpLtc_64s8um (const nm64s* pSrcVec, int64b nCmpVal, nm8u* pDstVec, int nSize, struct NmppsTmpSpec
1964 *spec);
1964 /*
1965 */
1966 void nmppsCmpGtC_8s (nm8s* pSrcVec, int8b nCmpVal, nm8s* pDstVec, int nSize, Tmp2BuffSpec *spec);
1967 void nmppsCmpGtC_16s (nm16s* pSrcVec, int16b nCmpVal, nm16s* pDstVec, int nSize, Tmp2BuffSpec *spec);
1968 void nmppsCmpGtC_32s (nm32s* pSrcVec, int32b nCmpVal, nm32s* pDstVec, int nSize, Tmp2BuffSpec *spec);
1969 void nmppsCmpGtC_64s (nm64s* pSrcVec, int64b nCmpVal, nm64s* pDstVec, int nSize, Tmp2BuffSpec *spec);
1970 */
2008 void nmppsCmpLtc_8s7b (const nm8s7b * pSrcVec, int8b nCmpVal,nm8s* pDstVec, int nSize);
2009 void nmppsCmpLtc_16s15b (const nm16s15b* pSrcVec, int16b nCmpVal, nm16s* pDstVec, int nSize);
2010 void nmppsCmpLtc_32s31b (const nm32s31b* pSrcVec, int32b nCmpVal, nm32s* pDstVec, int nSize);
2011 void nmppsCmpLtc_64s63b (const nm64s63b* pSrcVec, int64b nCmpVal, nm64s* pDstVec, int nSize);
2013
2051 void nmppsCmpGtC_8s7b (const nm8s7b * pSrcVec, int8b nCmpVal, nm8s * pDstVec, int nSize);
2052 void nmppsCmpGtC_16s15b (const nm16s15b* pSrcVec, int16b nCmpVal, nm16s* pDstVec, int nSize);
2053 void nmppsCmpGtC_32s31b (const nm32s31b* pSrcVec, int32b nCmpVal, nm32s* pDstVec, int nSize);

```

```

2054 void nmppsCmpGtC_64s63b (const nm64s63b* pSrcVec, int64b nCmpVal, nm64s* pDstVec, int nSize);
2056
2071 void nmppsCmpGteC_v2nm32f(const v2nm32f* pSrcVec, const v2nm32f* C, nm1* evenFlags, nm1* oddFlags, int step, int
nSize);
2073
2088 void nmppsCmpGtC_v2nm32f(const v2nm32f* pSrcVec, const v2nm32f* C, nm1* evenFlags, nm1* oddFlags, int step, int
nSize);
2090
2091 //*****
2092
2146 void nmppsCmpEqV_16u15b(nm16u15b* pSrcVec1, nm16u15b* pSrcVec2, nm16s* pDstVec, int nSize, int16b nTrueFlag);
2147 void nmppsCmpEqV_8u7b(nm8u7b* pSrcVec1, nm8u7b* pSrcVec2, nm8s* pDstVec, int nSize, int8b nTrueFlag);
2149
2193 void nmppsCmpNe_8s (const nm8s* pSrcVec1, const nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
2194 void nmppsCmpNe_16s (const nm16s* pSrcVec1, const nm16s* pSrcVec2, nm16s* pDstVec, int nSize);
2195 void nmppsCmpNe_32s (const nm32s* pSrcVec1, const nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
2196 void nmppsCmpNe_64s (const nm64s* pSrcVec1, const nm64s* pSrcVec2, nm64s* pDstVec, int nSize);
2198
2199 int nmppsCmpNe_8s8um (const nm8s* pSrcVec1, const nm8s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2200 int nmppsCmpNe_16s8um(const nm16s* pSrcVec1, const nm16s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2201 int nmppsCmpNe_32s8um(const nm32s* pSrcVec1, const nm32s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2202 int nmppsCmpNe_64s8um(const nm64s* pSrcVec1, const nm64s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2203
2204
2241 void nmppsCmpLt_8s7b (const nm8s* pSrcVec1,const nm8s* pSrcVec2, nm8s* pDstVec, int nSize);
2242 void nmppsCmpLt_16s15b(const nm16s* pSrcVec1,const nm16s* pSrcVec2, nm16s* pDstVec, int nSize);
2243 void nmppsCmpLt_32s31b(const nm32s* pSrcVec1,const nm32s* pSrcVec2, nm32s* pDstVec, int nSize);
2244 void nmppsCmpLt_64s63b(const nm64s* pSrcVec1,const nm64s* pSrcVec2, nm64s* pDstVec, int nSize);
2246 int nmppsCmpLt_8s8um (const nm8s* pSrcVec1, const nm8s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2247 int nmppsCmpLt_16s8um(const nm16s* pSrcVec1, const nm16s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2248 int nmppsCmpLt_32s8um(const nm32s* pSrcVec1, const nm32s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2249 int nmppsCmpLt_64s8um(const nm64s* pSrcVec1, const nm64s* pSrcVec2, nm8u* pDstVec, int nSize, struct
NmppsTmpSpec* spec);
2250
2251 //*****
2252
2306 void nmppsCmpNeV_16u(nm16u15b* pSrcVec1, nm16u15b* pSrcVec2, nm16s* pDstVec, int nSize, int16b nTrueFlag);
2307 void nmppsCmpNeV_8u(nm8u7b* pSrcVec1, nm8u7b* pSrcVec2, nm8s* pDstVec, int nSize, int8b nTrueFlag);
2309
2310
2311 void nmppsCmpNeV_8s8u(nm8s* src1, nm8s* src2, nm8u* dst, int nSize, int8b nTrueFlag);
2312
2313 //*****
2314
2315
2369 void Vec_ClipRShiftConvert_AddC( nm32s* pSrcVec, nm8u* pDstVec, int nSize);
2371
2372 #ifdef __cplusplus
2373 };
2374 #endif
2375
2376 #endif

```

7.74 vStat.h

```

1 //-----
2 // $Workfile:: vStat.h $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $      $Date: 2004/11/22 13:50:02 $
8 //
9 // -----
10 // -----
11
12 #ifndef _VSTAT_H_INCLUDED_
13 #define _VSTAT_H_INCLUDED_
14
15 #ifdef __cplusplus
16     extern "C" {
17 #endif
18
19 //-----
20
21
22 #ifndef _VSTAT_H_INCLUDED_
23 #define _VSTAT_H_INCLUDED_
24
25 #ifdef __cplusplus
26     extern "C" {
27 #endif
28
29     unsigned nmppsCrc_32u      (const unsigned int* pSrcVec, int nSize);
30     unsigned nmppsCrcAcc_32u   (const unsigned int* pSrcVec, int nSize, unsigned int* crcAccumulator);

```

```

71
72
73
74 --INLINE-- unsigned nmppsCrc_64s(nm64s* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«1); }
75 --INLINE-- unsigned nmppsCrc_32s(nm32s* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize ); }
76 --INLINE-- unsigned nmppsCrc_16s(nm16s* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«1); }
77 --INLINE-- unsigned nmppsCrc_8s(nm8s* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«2); }
78 --INLINE-- unsigned nmppsCrc_4s (nm4s* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«3); }
79 --INLINE-- unsigned nmppsCrc_64u(nm64u* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«1);
80 }
81 // --INLINE-- unsigned nmppsCrc_32f(nm32f* pSrcVec, int numBitsToClear, int nSize) { return
82     nmppsCrcMask_32u((unsigned*)pSrcVec,(-1«numBitsToClear),nSize«1); }
83
84 --INLINE-- unsigned nmppsCrc_16u(nm16u* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«1);
85 }
86 --INLINE-- unsigned nmppsCrc_8u (nm8u* pSrcVec, int nSize) { return nmppsCrc_32u((unsigned*)pSrcVec, nSize«2); }
87
88 --INLINE-- unsigned nmppsCrcAcc_64s(nm64s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
89     nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize«1, crcAccumulator); }
90 --INLINE-- unsigned nmppsCrcAcc_32s(nm32s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
91     nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize, crcAccumulator); }
92 --INLINE-- unsigned nmppsCrcAcc_16s(nm16s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
93     nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize«1, crcAccumulator); }
94 --INLINE-- unsigned nmppsCrcAcc_8s (nm8s* pSrcVec, int nSize, unsigned int* crcAccumulator) { return
95     nmppsCrcAcc_32u((unsigned*)pSrcVec, nSize«2, crcAccumulator); }
96
97 unsigned nmppsCrcAcc_32f(const nm32f* pSrcVec, int numBitsToClear, int nSize, unsigned int* crcAccumulator) ;
98
99 unsigned nmppsCrcAcc_64f(const nm64f* pSrcVec, int numBitsToClear, int nSize, unsigned int* crcAccumulator) ;
100
101
102
103
104 ////////////////////////////////////////////////////////////////////////
105
106 int nmppsSadV16_16s(nm16s* pSrc1,nm16s* pSrc2);
107 int nmppsSadV16_8s(nm8s* pSrc1,nm8s* pSrc2);
108
109
110 #ifdef __cplusplus
111 };
112 #endif
113
114 #endif // _VSTAT_H_INCLUDED_

```

7.75 vSupport.h

```

1 -----
2 //
3 /// $Workfile:: vSupport. $
4 /// Векторно-матричная библиотека
5 /// Copyright (c) RC Module Inc.
6 ///
7 /// $Revision: 1.2 $      $Date: 2004/12/21 14:36:01 $
8 ///
9 -----
10 -----
11 -----
12
13
14 -----
15 #ifndef __VSUPPORT_H
16 #define __VSUPPORT_H
17 //#include <stdlib.h>
18 #include "nmtype.h"
19 #ifdef __cplusplus
20     extern "C" {
21 #endif
22
23
24
25 ////////////////////////////////////////////////////////////////////////

```

```

26
73 /*
74 --INLINE-- nm1* nmppsAddr_1*(nm1* pVec, int nIndex){ return (nm1*)((int*)pVec+(nIndex>>5));}
75 --INLINE-- nm2s* nmppsAddr_2s(nm2s* pVec, int nIndex){ return (nm2s*)((int*)pVec+(nIndex>>4));}
76 --INLINE-- nm4s* nmppsAddr_4s(nm4s* pVec, int nIndex){ return (nm4s*)((int*)pVec+(nIndex>>3));}
77
78 nm8s* nmppsAddr_(const nm8s* pVec, int nIndex);
79 nm16s* nmppsAddr_16s(nm16s* pVec, int nIndex);
80 nm32s* nmppsAddr_32s(nm32s* pVec, int nIndex);
81
82 --INLINE-- nm64s* nmppsAddr_64s(nm64s* pVec, int nIndex) {return pVec+nIndex;}
83 #ifdef NM
84 --INLINE-- nm8u* nmppsAddr_(const nm8u* pVec, int nIndex) {return (nm8u*)nmppsAddr_8s((nm8s*) pVec,
85 --INLINE-- nm16u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return (nm16u*)nmppsAddr_16s((nm16s*) pVec,
86 #else
87 --INLINE-- nm8u* nmppsAddr_(const nm8u* pVec, int nIndex) {return (nm8u*)pVec+nIndex;}
88 --INLINE-- nm16u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return pVec+nIndex;}
89 #endif
90 --INLINE-- nm32u* nmppsAddr_32u(nm32u* pVec, int nIndex) {return pVec+nIndex;}
91 --INLINE-- nm64u* nmppsAddr_64u(nm64u* pVec, int nIndex) {return pVec+nIndex;}
92 */
93 /*
94 --INLINE-- nm64u* nmppsAddr_1 (nm1* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>6);}
95 --INLINE-- nm64u* nmppsAddr_2s(nm2s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>5);}
96 --INLINE-- nm64u* nmppsAddr_4s(nm4s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>4);}
97 --INLINE-- nm64u* nmppsAddr_8s(nm8s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>3);}
98 --INLINE-- nm64u* nmppsAddr_16s(nm16s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>2);}
99 --INLINE-- nm64u* nmppsAddr_32s(nm32s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>1);}
100 --INLINE-- nm64u* nmppsAddr_64s(nm64s* pVec, int nIndex) {return (nm64u*)pVec+nIndex;}
101
102 --INLINE-- nm64u* nmppsAddr_8u(nm8u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>3);}
103 --INLINE-- nm64u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>2);}
104 --INLINE-- nm64u* nmppsAddr_32u(nm32u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>1);}
105 --INLINE-- nm64u* nmppsAddr_64u(nm64u* pVec, int nIndex) {return (nm64u*)pVec+nIndex;}
106 */
107
108 --INLINE-- nm1 * nmppsAddr_1 (const nm1* pVec, int nIndex) {return (nm1*)((nm64u*)pVec+(nIndex>>6));}
109 inline nm2s * nmppsAddr_2s (const nm2s* pVec, int nIndex) {return (nm2s*)((nm64u*)pVec+(nIndex>>5));}
110 inline nm4s * nmppsAddr_4s (const nm4s* pVec, int nIndex) {return (nm4s*)((nm64u*)pVec+(nIndex>>4));}
111 --INLINE-- nm8s * nmppsAddr_8s (const nm8s* pVec, int nIndex) {return (nm8s*)((nm64u*)pVec+(nIndex>>3));}
112 --INLINE-- nm16s * nmppsAddr_16s(const nm16s* pVec, int nIndex) {return (nm16s*)((nm64u*)pVec+(nIndex>>2));}
113 --INLINE-- nm32s * nmppsAddr_32s(const nm32s* pVec, int nIndex) {return (nm32s*)((nm64u*)pVec+(nIndex>>1));}
114 --INLINE-- nm32f * nmppsAddr_32f(const nm32f* pVec, int nIndex) {return (nm32f*)((nm64u*)pVec+(nIndex>>1));}
115 --INLINE-- nm64s * nmppsAddr_64s(const nm64s* pVec, int nIndex) {return (nm64s*)((nm64u*)pVec+nIndex);}
116 --INLINE-- nm64f * nmppsAddr_64f(const nm64f* pVec, int nIndex) {return (nm64f*)((nm64u*)pVec+nIndex);}
117
118
119 inline nm2u * nmppsAddr_2u (const nm2u* pVec, int nIndex) {return (nm2u*)((nm64u*)pVec+(nIndex>>5));}
120 inline nm4u * nmppsAddr_4u (const nm4u* pVec, int nIndex) {return (nm4u*)((nm64u*)pVec+(nIndex>>4));}
121 --INLINE-- nm8u * nmppsAddr_8u (const nm8u* pVec, int nIndex) {return (nm8u*)((nm64u*)pVec+(nIndex>>3));}
122 --INLINE-- nm16u * nmppsAddr_16u(const nm16u* pVec, int nIndex) {return (nm16u*)((nm64u*)pVec+(nIndex>>2));}
123 --INLINE-- nm32u * nmppsAddr_32u(const nm32u* pVec, int nIndex) {return (nm32u*)((nm64u*)pVec+(nIndex>>1));}
124 --INLINE-- nm64u * nmppsAddr_64u(const nm64u* pVec, int nIndex) {return (nm64u*)((nm64u*)pVec+nIndex);}
125
126
127
128
129 *****
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169 void nmppsPut_1 (nm1* pVec, int nIndex, int1b nVal);
170 void nmppsPut_2s(nm2s* pVec, int nIndex, int2b nVal);
171 void nmppsPut_4s(nm4s* pVec, int nIndex, int4b nVal);
172 void nmppsPut_8s(nm8s* pVec, int nIndex, int8b nVal);
173 void nmppsPut_16s(nm16s* pVec, int nIndex, int16b nVal);
174 --INLINE-- void nmppsPut_32s(nm32s* pVec, int nIndex, int32b nVal) {pVec[nIndex]=nVal;}
175 --INLINE-- void nmppsPut_64s(nm64s* pVec, int nIndex, int64b nVal) {pVec[nIndex]=nVal;}
176
177 --INLINE-- void nmppsPut_2u(nm2u* pVec, int nIndex, uint2b nVal)
178 {nmppsPut_2s((nm2s*)pVec,nIndex,(int2b)nVal);}
179 --INLINE-- void nmppsPut_4u(nm4u* pVec, int nIndex, uint4b nVal)
180 {nmppsPut_4s((nm4s*)pVec,nIndex,(int4b)nVal);}
181 --INLINE-- void nmppsPut_8u(nm8u* pVec, int nIndex, uint8b nVal)
182 {nmppsPut_8s((nm8s*)pVec,nIndex,(int8b)nVal);}
183 --INLINE-- void nmppsPut_16u(nm16u* pVec, int nIndex, uint16b nVal)
184 {nmppsPut_16s((nm16s*)pVec,nIndex,(int16b)nVal);}
185 --INLINE-- void nmppsPut_32u(nm32u* pVec, int nIndex, uint32b nVal) {pVec[nIndex]=nVal;}
186 --INLINE-- void nmppsPut_64u(nm64u* pVec, int nIndex, uint64b nVal) {pVec[nIndex]=nVal;}
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212 void nmppsGetVal_1 (const nm1* pVec, int nIndex, int1b *nVal);
213 void nmppsGetVal_2s (const nm2s* pVec, int nIndex, int2b *nVal);

```

```

224 void nmppsGetVal_4s (const nm4s* pVec, int nIndex, int4b *nVal);
225 void nmppsGetVal_8s (const nm8s* pVec, int nIndex, int8b *nVal);
226 void nmppsGetVal_16s(const nm16s* pVec, int nIndex, int16b *nVal);
227 --INLINE-- void nmppsGetVal_32s(const nm32s* pVec, int nIndex, int32b* nVal) { *nVal=pVec[nIndex];}
228 --INLINE-- void nmppsGetVal_64s(const nm64s* pVec, int nIndex, int64b* nVal) { *nVal=pVec[nIndex];}
229
230 void nmppsGetVal_2u (const nm2u* pVec, int nIndex, uint2b *nVal);
231 void nmppsGetVal_4u (const nm4u* pVec, int nIndex, uint4b *nVal);
232 void nmppsGetVal_8u (const nm8u* pVec, int nIndex, uint8b *nVal);
233 void nmppsGetVal_16u(const nm16u* pVec, int nIndex, uint16b *nVal);
234 --INLINE-- void nmppsGetVal_32u(const nm32u* pVec, int nIndex, uint32b* nVal) { *nVal=pVec[nIndex];}
235 --INLINE-- void nmppsGetVal_64u(const nm64u* pVec, int nIndex, uint64b* nVal) { *nVal=pVec[nIndex];}
236
237
238
239 ////////////////////////////////////////////////////////////////////////
240
241 int2b nmppsGet_2s (const nm2s* pVec, int nIndex);
242 int4b nmppsGet_4s (const nm4s* pVec, int nIndex);
243 int8b nmppsGet_8s (const nm8s* pVec, int nIndex);
244 int16b nmppsGet_16s(const nm16s* pVec, int nIndex);
245 --INLINE-- int32b nmppsGet_32s(const nm32s* pVec, int nIndex) {return pVec[nIndex];}
246
247 uint1b nmppsGet_1 (const nm1* pVec, int nIndex);
248 uint2b nmppsGet_2u (const nm2u* pVec, int nIndex);
249 uint4b nmppsGet_4u (const nm4u* pVec, int nIndex);
250 uint8b nmppsGet_8u (const nm8u* pVec, int nIndex);
251 uint16b nmppsGet_16u(const nm16u* pVec, int nIndex);
252 --INLINE-- uint32b nmppsGet_32u(const nm32u* pVec, int nIndex) {return pVec[nIndex];}
253
254 *
255 --INLINE-- unsigned nmppsSize32_8s(nm8s* ,unsigned size) {return size/4;}
256 --INLINE-- unsigned nmppsSize32_16s(nm16s* ,unsigned size) {return size/2;}
257 --INLINE-- unsigned nmppsSize32_32s(nm32s* ,unsigned size) {return size; }
258 --INLINE-- unsigned nmppsSize32_64s(nm64s* ,unsigned size) {return size*2;}
259
260 --INLINE-- unsigned nmppsSize32_8u(nm8u* ,unsigned size) {return size/4;}
261 --INLINE-- unsigned nmppsSize32_16u(nm16u* ,unsigned size) {return size/2;}
262 --INLINE-- unsigned nmppsSize32_32u(nm32u* ,unsigned size) {return size; }
263 --INLINE-- unsigned nmppsSize32_64u(nm64u* ,unsigned size) {return size*2;}
264
265 *
266
267 #ifdef __cplusplus
268     };
269 #endif
270
271 #endif

```

7.76 vSupport_.h

```

1 ////////////////////////////////////////////////////////////////////////
2 //-----
3 // $Workfile::: vSupport. $
4 //-----
5 // Векторно-матричная библиотека
6 //-----
7 // Copyright (c) RC Module Inc.
8 //-----
9 // $Revision: 1.2 $ $Date: 2004/12/21 14:36:01 $
10 //-----
11 //-----
12
13
14 ////////////////////////////////////////////////////////////////////////
15 #ifndef __VSUPPORT_H
16 #define __VSUPPORT_H
17 //#include <stdlib.h>
18 #include "nmtype.h"
19 #ifdef __cplusplus
20     extern "C" {
21 #endif
22
23
24
25 ////////////////////////////////////////////////////////////////////////
26
27 *
28 --INLINE-- nm1* nmppsAddr_1*(nm1* pVec, int nIndex){ return (nm1*)((int*)pVec+(nIndex>>5));}
29 --INLINE-- nm2s* nmppsAddr_2s(nm2s* pVec, int nIndex){ return (nm2s*)((int*)pVec+(nIndex>>4));}
30 --INLINE-- nm4s* nmppsAddr_4s(nm4s* pVec, int nIndex){ return (nm4s*)((int*)pVec+(nIndex>>3));}
31
32 nm8s* nmppsAddr_(const nm8s* pVec, int nIndex);
33 nm16s* nmppsAddr_16s(nm16s* pVec, int nIndex);
34 nm32s* nmppsAddr_32s(nm32s* pVec, int nIndex);

```

```

81
82     _ __INLINE__ nm64s* nmppsAddr_64s(nm64s* pVec, int nIndex) {return pVec+nIndex;}
83 #ifndef NM
84     _ __INLINE__ nm8u* nmppsAddr_(const nm8u* pVec, int nIndex) {return (nm8u*)nmppsAddr_8s((nm8s*) pVec,
85     nIndex);}
85     _ __INLINE__ nm16u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return (nm16u*)nmppsAddr_16s((nm16s*) pVec,
86     nIndex);}
86 #else
87     _ __INLINE__ nm8u* nmppsAddr_(const nm8u* pVec, int nIndex) {return (nm8u*)pVec+nIndex;}
88     _ __INLINE__ nm16u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return pVec+nIndex;}
89 #endif
90     _ __INLINE__ nm32u* nmppsAddr_32u(nm32u* pVec, int nIndex) {return pVec+nIndex;}
91     _ __INLINE__ nm64u* nmppsAddr_64u(nm64u* pVec, int nIndex) {return pVec+nIndex;}
92 */
93 /*
94     _ __INLINE__ nm64u* nmppsAddr_1 (nm1* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>6);}
95     _ __INLINE__ nm64u* nmppsAddr_2s(nm2s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>5);}
96     _ __INLINE__ nm64u* nmppsAddr_4s(nm4s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>4);}
97     _ __INLINE__ nm64u* nmppsAddr_8s(nm8s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>3);}
98     _ __INLINE__ nm64u* nmppsAddr_16s(nm16s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>2);}
99     _ __INLINE__ nm64u* nmppsAddr_32s(nm32s* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>1);}
100    _ __INLINE__ nm64u* nmppsAddr_64s(nm64s* pVec, int nIndex) {return (nm64u*)pVec+nIndex;}
101
102   _ __INLINE__ nm64u* nmppsAddr_8u(nm8u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>3);}
103   _ __INLINE__ nm64u* nmppsAddr_16u(nm16u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>2);}
104   _ __INLINE__ nm64u* nmppsAddr_32u(nm32u* pVec, int nIndex) {return (nm64u*)pVec+(nIndex>>1);}
105   _ __INLINE__ nm64u* nmppsAddr_64u(nm64u* pVec, int nIndex) {return (nm64u*)pVec+nIndex;}
106 */
107
108   _ __INLINE__ nm1 * nmppsAddr_1 (const nm1* pVec, int nIndex) {return (nm1*)((nm64u*)pVec+(nIndex>>6));}
109   _ __INLINE__ nm2s * nmppsAddr_2s (const nm2s* pVec, int nIndex) {return (nm2s*)((nm64u*)pVec+(nIndex>>5));}
110   _ __INLINE__ nm4s * nmppsAddr_4s (const nm4s* pVec, int nIndex) {return (nm4s*)((nm64u*)pVec+(nIndex>>4));}
111   _ __INLINE__ nm8s * nmppsAddr_8s (const nm8s* pVec, int nIndex) {return (nm8s*)((nm64u*)pVec+(nIndex>>3));}
112   _ __INLINE__ nm16s * nmppsAddr_16s (const nm16s* pVec, int nIndex) {return (nm16s*)((nm64u*)pVec+(nIndex>>2));}
113   _ __INLINE__ nm32s * nmppsAddr_32s (const nm32s* pVec, int nIndex) {return (nm32s*)((nm64u*)pVec+(nIndex>>1));}
114   _ __INLINE__ nm32f* nmppsAddr_32f (const nm32f* pVec, int nIndex) {return (nm32f*)((nm64u*)pVec+(nIndex>>1));}
115   _ __INLINE__ nm64s * nmppsAddr_64s (const nm64s* pVec, int nIndex) {return (nm64s*)((nm64u*)pVec+nIndex);}
116   _ __INLINE__ nm64f* nmppsAddr_64f (const nm64f* pVec, int nIndex) {return (nm64f*)((nm64u*)pVec+nIndex); }
117
118
119   _ __INLINE__ nm2u * nmppsAddr_2u (const nm2u* pVec, int nIndex) {return (nm2u*)((nm64u*)pVec+(nIndex>>5));}
120   _ __INLINE__ nm4u * nmppsAddr_4u (const nm4u* pVec, int nIndex) {return (nm4u*)((nm64u*)pVec+(nIndex>>4));}
121   _ __INLINE__ nm8u * nmppsAddr_8u (const nm8u* pVec, int nIndex) {return (nm8u*)((nm64u*)pVec+(nIndex>>3));}
122   _ __INLINE__ nm16u * nmppsAddr_16u (const nm16u* pVec, int nIndex) {return (nm16u*)((nm64u*)pVec+(nIndex>>2));}
123   _ __INLINE__ nm32u * nmppsAddr_32u (const nm32u* pVec, int nIndex) {return (nm32u*)((nm64u*)pVec+(nIndex>>1));}
124   _ __INLINE__ nm64u * nmppsAddr_64u (const nm64u* pVec, int nIndex) {return (nm64u*)((nm64u*)pVec+nIndex); }
125
126
127
128 //*****
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169 void nmppsPut_1 (nm1* pVec, int nIndex, int1b nVal);
170 void nmppsPut_2s(nm2s* pVec, int nIndex, int2b nVal);
171 void nmppsPut_4s(nm4s* pVec, int nIndex, int4b nVal);
172 void nmppsPut_8s(nm8s* pVec, int nIndex, int8b nVal);
173 void nmppsPut_16s(nm16s* pVec, int nIndex, int16b nVal);
174 _ __INLINE__ void nmppsPut_32s(nm32s* pVec, int nIndex, int32b nVal) {pVec[nIndex]=nVal;}
175 _ __INLINE__ void nmppsPut_64s(nm64s* pVec, int nIndex, int64b nVal) {pVec[nIndex]=nVal;}
176
177 _ __INLINE__ void nmppsPut_2u(nm2u* pVec, int nIndex, uint2b nVal)
178     {nmppsPut_2s((nm2s*)pVec,nIndex,(int2b)nVal);}
179 _ __INLINE__ void nmppsPut_4u(nm4u* pVec, int nIndex, uint4b nVal)
180     {nmppsPut_4s((nm4s*)pVec,nIndex,(int4b)nVal);}
181 _ __INLINE__ void nmppsPut_8u(nm8u* pVec, int nIndex, uint8b nVal)
182     {nmppsPut_8s((nm8s*)pVec,nIndex,(int8b)nVal);}
183 _ __INLINE__ void nmppsPut_16u(nm16u* pVec, int nIndex, uint16b nVal)
184     {nmppsPut_16s((nm16s*)pVec,nIndex,(int16b)nVal);}
185 _ __INLINE__ void nmppsPut_32u(nm32u* pVec, int nIndex, uint32b nVal) {pVec[nIndex]=nVal;}
186 _ __INLINE__ void nmppsPut_64u(nm64u* pVec, int nIndex, uint64b nVal) {pVec[nIndex]=nVal;}
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222 void nmppsGetVal_1 (const nm1* pVec, int nIndex, int1b *nVal);
223 void nmppsGetVal_2s (const nm2s* pVec, int nIndex, int2b *nVal);
224 void nmppsGetVal_4s (const nm4s* pVec, int nIndex, int4b *nVal);
225 void nmppsGetVal_8s (const nm8s* pVec, int nIndex, int8b *nVal);
226 void nmppsGetVal_16s (const nm16s* pVec, int nIndex, int16b *nVal);
227 _ __INLINE__ void nmppsGetVal_32s(const nm32s* pVec, int nIndex, int32b* nVal) { *nVal=pVec[nIndex];}
228 _ __INLINE__ void nmppsGetVal_64s(const nm64s* pVec, int nIndex, int64b* nVal) { *nVal=pVec[nIndex];}
229
230 void nmppsGetVal_2u (const nm2u* pVec, int nIndex, uint2b *nVal);
231 void nmppsGetVal_4u (const nm4u* pVec, int nIndex, uint4b *nVal);
232 void nmppsGetVal_8u (const nm8u* pVec, int nIndex, uint8b *nVal);

```

```

233 void nmppsGetVal_16u(const nm16u* pVec, int nIndex, uint16b *nVal);
234 #ifndef _VTRANSFORM_H_INCLUDED_
235 #define _VTRANSFORM_H_INCLUDED_
236
237 //*****
238
239 //*****
240
271 int2b nmppsGet_2s (const nm2s* pVec, int nIndex);
272 int4b nmppsGet_4s (const nm4s* pVec, int nIndex);
273 int8b nmppsGet_8s (const nm8s* pVec, int nIndex);
274 int16b nmppsGet_16s(const nm16s* pVec, int nIndex);
275 #ifndef _VTRANSFORM_H_INCLUDED_
276
277 uint1b nmppsGet_1 (const nm1* pVec, int nIndex);
278 uint2b nmppsGet_2u (const nm2u* pVec, int nIndex);
279 uint4b nmppsGet_4u (const nm4u* pVec, int nIndex);
280 uint8b nmppsGet_8u (const nm8u* pVec, int nIndex);
281 uint16b nmppsGet_16u(const nm16u* pVec, int nIndex);
282 #ifndef _VTRANSFORM_H_INCLUDED_
283
284
285 /*
286 #ifndef _VTRANSFORM_H_INCLUDED_
287
288 #ifndef _VTRANSFORM_H_INCLUDED_
289
290
291
292
293
294
295 */
296
297 #ifdef __cplusplus
298 };
299 #endif
300
301 #endif

```

7.77 vTransform.h

```

1 //-
2 //-
3 // $Workfile:: vTransform. $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.1 $      $Date: 2004/11/22 13:50:02 $
10 //-
19 //-----
20 #ifndef _VTRANSFORM_H_INCLUDED_
21 #define _VTRANSFORM_H_INCLUDED_
22
23 #ifdef __cplusplus
24     extern "C" {
25 #endif
26
79 void nmppsQSort_32s(nm32s* pSrcDstVec, int nSize);
81
82 //*****
83
168 void nmppsRemap_32u(nm32u* pSrcVec, nm32u* pDstVec, nm32s* pRemapTable, int nDstVecSize);
169 void nmppsRemap_8u(nm8u* pSrcVec, nm8u* pDstVec, nm32s* pRemapTable, int nSrcVecSize, int nDstVecSize, void*
    pTmpBuf1, void* pTmpBuf2);
171
172
173
174
213 void nmppsSplit_(v4nm16s* pSrcVec, nm16s** pDst4Vec, int nSize);
215
216
235 void nmppsSplitTmp_8s(const nm8s* src, nm8s* dst1, nm8s* dst2, int size, nm8s* tmpSizeOfDst);
237
254 void nmppsSplit_8s (const nm8s* src, nm8s* dst1, nm8s* dst2, int size);
255 void nmppsSplit_16s(const nm16s* src, nm16s* dst1, nm16s* dst2, int size);
256 void nmppsSplit_32s(const nm32s* src, nm32s* dst1, nm32s* dst2, int size);
258
273 void nmppsMerge_8s (const nm8s* src0, const nm8s* src1, nm8s* dst, int sizeSrc);
274 void nmppsMerge_16s(const nm16s* src0, const nm16s* src1, nm16s* dst, int sizeSrc);
275 void nmppsMerge_32s(const nm32s* src0, const nm32s* src1, nm32s* dst, int sizeSrc);
277
294 void nmppsSplit_32fcr(const nm32fcr* pSrcVec, nm32fcr* pDstVec1, nm32fcr* pDstVec, int sizeSrc);
296

```

```

313 void nmppsDecimate_16s(nm16s* pSrc, int startPos, int step, nm16s* pDst, int nSizeDst);
314 void nmppsDecimate_32s(nm32s* pSrc, int startPos, int step, nm32s* pDst, int nSizeDst);
316
317 #ifndef __cplusplus
318 };
319 #endif
320
321 #endif // _VTRANSFORM_H_INCLUDED_

```

7.78 nmpp-cpp.h

```

1 #ifndef NMPP_CPP_DEFINED
2 #define NMPP_CPP_DEFINED
3 //inline void nmppsSet(nm4s *pSrcVec, int val, int nSize) {
4 //    nmppsSet_4s(pSrcVec, val, nSize);
5 //}
6 //inline void nmppsSet(nm8s *pSrcVec, int val, int nSize) {
7 //    nmppsSet_8s(pSrcVec, val, nSize);
8 //}
9
10 //inline void nmppsSet(nm16s *pSrcVec, int val, int nSize) {
11 //    nmppsSet_16s(pSrcVec, val, nSize);
12 //}
13 inline void nmppsSet(nm8s *pSrcVec, int val, int nSize) {
14     nmppsSet_8s((nm8s*)pSrcVec, val, nSize);
15 }
16
17 inline void nmppsSet(nm16s *pSrcVec, int val, int nSize) {
18     nmppsSet_16s((nm16s*)pSrcVec, val, nSize);
19 }
20
21
22 inline void nmppsSet(nm32s *pSrcVec, int val, int nSize) {
23     nmppsSet_32s(pSrcVec, val, nSize);
24 }
25
26 inline void nmppsSet(nm64s *pSrcVec, int val, int nSize){
27     nmppsSet_64s(pSrcVec, val, nSize);
28 }
29
30
31
32
33 //inline void nmppsPut_1 (nm1* pVec, int nIndex, int1b nVal) nmppsPut_1 (nm1* pVec, int nIndex, int1b nVal);
34 //inline void nmppsPut_2s (nm2s* pVec, int nIndex, int2b nVal) nmppsPut_2s(nm2s* pVec, int nIndex, int2b nVal);
35 //inline void nmppsPut_4s (nm4s* pVec, int nIndex, int4b nVal) nmppsPut_4s(nm4s* pVec, int nIndex, int4b nVal);
36 inline void nmppsPut(nm8s* pVec, int nIndex, int8b nVal) {nmppsPut_8s((nm8s*) pVec, nIndex, nVal);}
37 //inline void nmppsPut(nm8s7b* pVec, int nIndex, int8b nVal) {nmppsPut_8s((nm8s*) pVec, nIndex, nVal);}
38 //inline void nmppsPut(nm16s* pVec, int nIndex, int16b nVal) {nmppsPut_16s((nm16s*) pVec, nIndex, nVal);}
39 inline void nmppsPut(nm16s15b* pVec, int nIndex, int16b nVal) {nmppsPut_16s((nm16s*) pVec, nIndex, nVal);}
40 //inline void nmppsPut(nm8s7b* pVec, int nIndex, int32b nVal) {pVec[nIndex]=nVal;}
41 inline void nmppsPut(nm32s* pVec, int nIndex, int32b nVal) {pVec[nIndex]=nVal;}
42 inline void nmppsPut(nm64s* pVec, int nIndex, int64b nVal) {pVec[nIndex]=nVal;}
43
44
45
46 #endif

```

7.79 nmpp.h

```

1
18 #include "./nmplv/nmplv.h"
19
31 #include "./nmplm/nmplm.h"
32
40 #include "./nmpls/nmpls.h"
41
45 #include "./nmpli/nmpli.h"
46
50 #include "./nmplc/nmplc.h"
51
68 #include "./malloc32.h"
69 #include "./metric.h"
70 #include "./nmplv/vStat.h"
71
72

```

7.80 nmshort.h

```

1 #ifndef NMSHORT_DEFINED
2 #define NMSHORT_DEFINED
3
4 #include <stdlib.h>
5 #include <string.h>
6 //extern unsigned int crc;
7 class uint16ptr;
8 class nmshort{
9 public:
10    unsigned int *adr;
11    int idx;
12    __inline__ nmshort(){
13        //adr=0;
14        //idx=0;
15    }
16    __inline__ nmshort(nmshort& ch){
17        adr=ch.adr;
18        idx=ch.idx;
19        //    crc^=ch;
20    }
21    __inline__ nmshort& operator = (nmshort ch){
22        unsigned char val=(ch);
23        (*adr) &= ~(0xFFFF «(idx*16));
24        (*adr) |= (val) «(idx*16);
25        //    crc^=ch;
26        //adr=ch.adr;
27        //idx=ch.idx;
28        return *this;
29    }
30    __inline__ unsigned int operator + (nmshort& ch){
31        unsigned int a=(ch);
32        unsigned int b=(*this);
33        a+=b;
34        //    crc^=a;
35        return a;
36    }
37
38    __inline__ nmshort& operator = (unsigned int val){
39        (*adr) &= ~(0xFFFF «(idx*16));
40        (*adr) |= (val&0xFFFF) «(idx*16);
41        //    crc^=val;
42        return *this;
43    }
44    __inline__ operator unsigned char(){
45        unsigned char ret=((*adr)»(idx*4));
46        ret&=0xFF;
47        //    crc^=ret;
48        return ret;
49    }
50
51    __inline__ uint16ptr operator &();
52
53 };
54
55
56
57 class uint16ptr {
58 public:
59    unsigned int *addr;
60    int idx;
61    nmshort arref;
62    __inline__ uint16ptr (){
63        addr=0;
64        idx=0;
65    }
66    //uint16ptr(uint16ptr& p){
67    //    idx=p.idx;
68    //    addr=p.addr;
69    //}
70
71    //uint16ptr(uint16ptr& p){
72    //    idx=p.idx;
73    //    addr=p.addr;
74    //}
75
76    __inline__ unsigned char* x86addr(){
77        return ((unsigned char*)addr)+idx;
78    }
79    __inline__ uint16ptr (void* p){
80        addr=(unsigned int*)p;
81        idx=0;
82    }
83    __inline__ uint16ptr (unsigned short* p){
84        addr=(unsigned int*)p;
85        idx=0;

```

```

86      }
87      --INLINE -- uint16ptr (short* p){
88          addr=(unsigned int*)p;
89          idx=0;
90      }
91
92      --INLINE -- uint16ptr(const uint16ptr& p){
93          idx=p.idx;
94          addr=p.addr;
95      }
96      //uint16ptr(int* p){
97      //    addr=(unsigned*)p;
98      //    idx=0;
99      //}
100     --INLINE -- uint16ptr(unsigned int*p){
101         addr=p;
102         idx=0;
103     }
104     --INLINE -- uint16ptr(unsigned int*p,int offset){
105         addr=p+((offset>>1);
106         idx=offset&1;
107     }
108
109     //operator int(){
110     //    return int(addr);
111     //}
112
113     --INLINE -- nmshort& operator [](int idx){
114
115         arref.adr=addr+(idx+idx)/2;
116         arref.idx=(idx+idx)%2;
117         return arref;
118     }
119
120     //uint16ptr& operator = (unsigned ptr){
121     //    addr=(unsigned*)ptr;
122     //    idx=0;
123     //    return (*this);
124     //}
125
126
127     --INLINE -- uint16ptr& operator = (unsigned int* ptr){
128         addr=(unsigned*)ptr;
129         idx=0;
130         return (*this);
131     }
132
133     --INLINE -- bool operator < (uint16ptr ptr){
134         if (addr<ptr.addr)
135             return 1;
136         if (addr==ptr.addr)
137             if (idx<ptr.idx)
138                 return 1;
139         return 0;
140     }
141
142     --INLINE -- bool operator >= (uint16ptr ptr){
143         if (addr>ptr.addr)
144             return 1;
145         if (addr==ptr.addr)
146             if (idx>=ptr.idx)
147                 return 1;
148         return 0;
149     }
150
151
152     --INLINE -- int operator - (uint16ptr ptr){
153         int a=(int)addr;
154         int b=(int)ptr.addr;
155
156 #ifdef NM
157     //a&=0x0FFFFFF;
158     //b&=0x0FFFFFF;
159     a<<=1;
160     b<<=1;
161 #endif
162         a+=idx;
163         b+=ptr.idx;
164         return a-b;
165     }
166
167
168 // uint16ptr& operator = (unsigned char* ptr){
169 //     addr=(unsigned int*)ptr;
170 //     idx=0;
171 //     return (*this);
172 // }

```

```

173
174
175
176     __INLINE __ uint16ptr& operator = (const uint16ptr& p){
177         addr=p.addr;
178         idx=p.idx;
179         return (*this);
180     }
181
182     __INLINE __ unsigned int* ptr(){
183         if (idx==0)
184             return addr;
185         //}
186         else {
187             int g=1;
188             return (unsigned int*)-1;
189         }
190     }
191     __INLINE __ nmshort& operator *(){
192
193         return (nmshort&)(*this);
194     }
195
196 /*
197 uint16ptr operator + (int idx){
198 uint16ptr tmp(*this);
199 tmp.addr=addr+(idx+idx)/2;
200 tmp.idx=(idx+idx)&0x1;
201 return tmp;
202 }
203
204 uint16ptr operator- (int idx){
205 uint16ptr tmp(*this);
206 tmp.addr=addr+(idx-idx)/2;
207 tmp.idx=(idx-idx)&0x1;
208 return tmp;
209 }
210
211 uint16ptr& operator+= (int idx){
212 addr=addr+(idx+idx)/2;
213 idx=(idx+idx)&0x1;
214 return *this;
215 }
216
217 uint16ptr& operator-= (int idx){
218 addr=addr+(idx-idx)/2;
219 idx=(idx-idx)&0x1;
220 return *this;
221 }
222 */
223     __INLINE __ uint16ptr operator + (int idx){
224         uint16ptr tmp(*this);
225         tmp.addr=addr+((idx+idx)»1);
226         tmp.idx=(idx+idx)&0x1;
227         return tmp;
228     }
229
230     __INLINE __ uint16ptr operator- (int idx){
231         uint16ptr tmp(*this);
232         tmp.addr=addr+((idx-idx)»1);
233         tmp.idx=(idx-idx)&0x1;
234         return tmp;
235     }
236
237
238     __INLINE __ uint16ptr& operator+= (int idx){
239         addr=addr+((idx+idx)»1);
240         idx=(idx+idx)&0x1;
241         return *this;
242     }
243
244     __INLINE __ uint16ptr& operator-= (int idx){
245         addr=addr+((idx-idx)»1);
246         idx=(idx-idx)&0x1;
247         return *this;
248     }
249
250     __INLINE __ uint16ptr operator++ (int){
251         uint16ptr tmp(*this);
252         if (idx==1){
253             idx=0;
254             addr++;
255         }
256         else {
257             idx++;
258         }
259         return tmp;

```

```

260     }
261
262     --INLINE-- uint16ptr& operator++ (){
263         if (idx==1){
264             idx=0;
265             addr++;
266         }
267         else {
268             idx++;
269         }
270         return *this;
271     }
272
273     --INLINE-- unsigned int operator== (unsigned int N){
274         return ((unsigned int)addr)==N;
275     }
276
277     --INLINE-- bool operator == (uint16ptr ptr){
278         if (addr==ptr.addr && idx==ptr.idx)
279             return 1;
280         return 0;
281     }
282 }
283 };
284
285 --INLINE-- uint16ptr nmshort::operator &(){
286     uint16ptr p;
287     p.addr=adr;
288     p.idx=idx;
289     return p;
290 }
291
292 template <int Y,int X> class nmshort2D {
293 public:
294     uint16ptr arr;
295     unsigned int data[Y*X/2];
296
297     nmshort2D(){
298     }
299
300     uint16ptr& operator [](int idx){
301         arr.addr=data+X*idx/2;
302         arr.idx=0;
303         return arr;
304     }
305     unsigned int* ptr(){
306         return data;
307     }
308 };
309 */
310 template<int N> class nmshort1D {
311 public:
312     nmshort deref;
313     unsigned int data[(N+3)/4];
314
315     nmshort1D(){
316     }
317
318     --INLINE-- nmshort& operator [](int idx){
319         addr=data+idx/4;
320         idx=idx%4;
321         return deref;
322     }
323
324     --INLINE-- operator uint16ptr(){
325         uint16ptr p;
326         p.addr=data;
327         p.idx=0;
328         return p;
329     }
330     --INLINE-- unsigned int* ptr(){
331         return data;
332     }
333
334
335 };
336 */
337 --INLINE-- void free(uint16ptr& p){
338     free(p.addr);
339 }
340
341 --INLINE-- void nmshort_memcpy (uint16ptr dst, uint16ptr src, unsigned int len){
342
343
344 #ifdef __NM
345     memcpy((void*)dst.x86addr(), (void*)src.x86addr(),len);
346 #else

```

```

347     if (dst.idx==src.idx){
348         // copy n bytes to align pointers
349         if (dst.idx){
350             int n=4-dst.idx;
351             if (len<n)
352                 n=len;
353             for(int i=0; i<n; i++){
354                 *dst=*src;
355                 dst++;
356                 src++;
357             }
358             len-=n;
359         }
360         int words=len>>2;
361         memcpy(dst.addr, src.addr, words);
362         dst.addr+=words;
363         src.addr+=words;
364         len%-=4;
365         // copy rest bytes
366         for(int i=0;i<len;i++){
367             *dst=*src;
368             dst++;
369             src++;
370         }
371     }  

372     else {
373         for(int i=0; i<len; i++){
374             *dst=*src;
375             src++;
376             dst++;
377         }
378     }
379
380 #endif
381 }
382
383 __INLINE__ void nmshort_memset (uint16ptr dst, int setvalue, unsigned int len){
384
385
386 #ifdef __NM_
387     memset((void*)dst.x86addr(), setvalue,len);
388 #else
389
390     // copy n bytes to align pointers
391     if (dst.idx){
392         int n=4-dst.idx;
393         if (len<n)
394             n=len;
395         for(int i=0; i<n; i++){
396             *dst=setvalue;
397             dst++;
398         }
399         len-=n;
400     }
401     int words=len>>2;
402
403     setvalue&=0xFF;
404     setvalue|=(setvalue<<8);
405     setvalue|=(setvalue<<16);
406     memset(dst.addr, setvalue, words);
407     dst.addr+=words;
408     len%-=4;
409     // copy rest bytes
410     setvalue&=0xFF;
411     for(int i=0;i<len;i++){
412         *dst=setvalue;
413         dst++;
414     }
415
416
417 #endif
418 }
419
420 #endif

```

7.81 nmtlio.h

```

1
2 // Header file of template of class mtr
3 // // Header file of template of class mtr
4 // // Header file of template of class mtr
5 #ifndef __TNMTLIO_H_INCLUDED__
6 #define __TNMTLIO_H_INCLUDED__
7
8

```

```

9 #include "nmtl.h"
10 #include <iomanip>
11 #ifndef NM6403
12 #include <iostream>
13 #endif
14 #include <stdio.h>
15 #include <crtdbg.h>
16
17
18 // Class of matrixes
19 // ****
20 // ****
21 // ****
22
23 // ****
24 // ****
25 // ****
26 // ****
27 // ****
28 // ****
29 // ****
30 // ****
31 // ****
32 // ****
33 // ****
34 // ****
35 // ****
36 // ****
37 // ****
38 // ****
39 // ****
40 // ****
41 // ****
42 // ****
43 // ****
44 // ****
45 // ****
46 // ****
47 // ****
48 // ****
49 // ****
50 // ****
51 // ****
52 // ****
53 // ****
54 // ****
55 // ****
56 // ****
57 // ****
58 // ****
59
60
61
62
63 #ifndef NM6403
64     _INLINE __ ostream& operator<< (ostream& s, mtr<unsigned char>& mtr)
65 {
66     // char str[255];
67     s << "\n";
68     for(int y=0;y<mtr.m_height-1;y++)
69     {
70         s << "\t{ ";
71         for(int x=0;x<mtr.m_width-1;x++)
72         {
73             s << int(mtr[y][x]) << ", ";
74         }
75         s << int(mtr[y][mtr.m_width-1]) << " },\n";
76     }
77     //Last string
78     s << "\t{ ";
79     for(int x=0;x<mtr.m_width-1;x++)
80     {
81         s << int(mtr[mtr.m_height-1][x]) << ", ";
82     }
83     s << int( mtr[mtr.m_height-1][mtr.m_width-1]) << " }\n};\n";
84
85     return s;
86 }
87
88     _INLINE __ ostream& operator<< (ostream& s, mtr<int>& mtr)
89 {
90     // char str[255];
91     s << "\n";
92     for(int y=0;y<mtr.m_height-1;y++)
93     {
94         s << "\t{ ";
95         for(int x=0;x<mtr.m_width-1;x++)
96         {
97             //sprintf(str,"%d",mtr[y][x]);
98             //s << dbl << ", ";
99             s << mtr[y][x] << ", ";
100        }
101        //sprintf(dbl,"%20.20f",mtr[y][mtr.m_width-1]);
102        //s << dbl << " },\n";
103        s << mtr[y][mtr.m_width-1] << " },\n";
104    }
105    //Last string
106    s << "\t{ ";
107    for(int x=0;x<mtr.m_width-1;x++)
108    {
109        s << mtr[mtr.m_height-1][x] << ", ";
110    }
111    s << mtr[mtr.m_height-1][mtr.m_width-1] << " }\n};\n";
112
113    return s;
114 }
115
116     _INLINE __ ostream& operator<< (ostream& s, const __int64& y)
117 {
118
119     int hi,lo;
120     hi=int(y>>32);
121     lo=int(y);
122     s << "0x" << setw(8) << setiosflags(ios::hex|ios::uppercase|ios::internal) << setfill('0') << hi
123     << setw(8) << setiosflags(ios::hex|ios::uppercase|ios::internal) << setfill('0') << lo ;
124
125     return s;
126
127
128     _INLINE __ ostream& operator<< (ostream& s, mtr<__int64>& mtr)
129 {
130     // char str[255];

```

```

131     s <"{\n";
132     for(int y=0;y<mtr.m_height-1;y++)
133     {
134         s < "\t{ ";
135         for(int x=0;x<mtr.m_width-1;x++)
136         {
137             //sprintf(str,"%d",mtr[y][x]);
138             //s< dbl < ", ";
139             s< mtr[y][x] < ", ";
140         }
141         //sprintf(dbl,"%20.20f",mtr[y][mtr.m_width-1]);
142         //s< dbl < " }\n";
143         s< mtr[y][mtr.m_width-1] < " },\n";
144     }
145     //Last string
146     s < "\t{ ";
147     for(int x=0;x<mtr.m_width-1;x++)
148     {
149         s< mtr[mtr.m_height-1][x] < ", ";
150     }
151     s< mtr[mtr.m_height-1][mtr.m_width-1] < " }\n};\n";
152
153     return s;
154 }
155
156
157
158 #ifndef WIN32
159 inline ostream& operator<< (ostream& s, mtr<double>& mtr)
160 {
161     char dbl[255];
162     s <"{\n";
163     for(int y=0;y<mtr.m_height-1;y++)
164     {
165         s < "\t{ ";
166         for(int x=0;x<mtr.m_width-1;x++)
167         {
168             sprintf(dbl,"%20.20f",mtr[y][x]);
169             s< dbl < ", ";
170             sprintf(dbl,"%20.20f",mtr[y][mtr.m_width-1]);
171             s< dbl < " }\n";
172         }
173         //Last string
174         s < "\t{ ";
175         for(int x=0;x<mtr.m_width-1;x++)
176         {
177             sprintf(dbl,"%20.20f",mtr[mtr.m_height-1][x]);
178             s< dbl < ", ";
179         }
180         sprintf(dbl,"%20.20f",mtr[mtr.m_height-1][mtr.m_width-1]);
181         s< dbl < " }\n};\n";
182
183     return s;
184 }
185
186 /*
187 template <class T> cmplx<T> sqrt(const cmplx<T>& x)
188 {
189
190     cmplx<T> Result;
191     Result.re=sqrt(double_(x.re));
192     return Result;
193 }
194
195
196 template <class T> ostream& operator<< (ostream& s, const cmplx<T>& y)
197 {
198     return s< '[' < y.re < ',' < y.im < ']';
199 }
200 */
201
202 #ifdef WIN32
203 inline ostream& operator<< (ostream& s, const nmint<__int64>& y)
204 {
205
206     int hi,lo;
207     hi=int(y.m_value>>32);
208     lo=int(y.m_value);
209     s< "0x" < setw(8) < setiosflags(ios::hex|ios::uppercase|ios::internal) < setfill('0') < hi
210     < setw(8) < setiosflags(ios::hex|ios::uppercase|ios::internal) < setfill('0') < lo ;
211
212     return s;
213
214
215 inline ostream& operator<< (ostream& s, const nmint<int>& y)
216 {
217     unsigned int x=(__int32)y.m_value;
218     s< "0x" < setw(8) < setiosflags(ios::hex|ios::uppercase|ios::internal) < setfill('0') < setiosflags(ios::uppercase) < x;
219
220     return s;
221 }
```

```

219 }
220
221 //INLINE __ ostream& operator<>(ostream& s, const nmint<short>& y)
222 {//
223     unsigned __ int16 x=(__ int16)y.m_value;
224     s<< "0x" << setw(4) << setiosflags(ios::hex|ios::uppercase|ios::internal) << setfill('0') << x ;
225     return s;
226 }
227
228 //INLINE __ ostream& operator<>(ostream& s, const nmint<char>& y)
229 {//
230     unsigned __ int8 x=y.m_value;
231     s<< "0x" << setw(2) << setiosflags(ios::hex|ios::uppercase|ios::internal) << setfill('0') << x ;
232     return s;
233 }
234 #endif
235
236
237
238 #endif
239
240 #ifdef WIN32
241 template <class T> __INLINE __ ostream& operator<>(ostream& s, nmmtr<T>& mtr)
242 {
243     s << "{\n";
244     for(int y=0;y<mtr.m_height-1;y++)
245     {
246         s << "\t{ ";
247         for(int x=0;x<mtr.m_width-1;x++)
248         {
249             s<< (mtr[y][x]) << ", ";
250         }
251         s << (mtr[y][mtr.m_width-1]) << " },\n";
252     }
253
254     s << "\t{ ";
255     for(int x=0;x<mtr.m_width-1;x++)
256     {
257         s<< (mtr[mtr.m_height-1][x]) << ", ";
258     }
259     s << (mtr[mtr.m_height-1][mtr.m_width-1]) << " }\n";
260     s << "};\n";
261     return s;
262 }
263
264
265
266 //INLINE __ ostream& AsmArray (ostream& s, nmmtr64s& mtr)
267 {//
268     s << " long[" << dec << mtr.m_width << "*" << mtr.m_height << "]=(\n";
269     for(int y=0;y<mtr.m_height-1;y++)
270     {
271         for(int x=0;x<mtr.m_width;x++)
272         {
273             int hi,lo;
274             hi=int(mtr[y][x].m_value>>32);
275             lo=int(mtr[y][x].m_value);
276             s << "0"
277                 << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << hi << " "
278                 << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << lo << "hl,";
279         }
280         s << "\n";
281     }
282
283     for(int x=0;x<mtr.m_width-1;x++)
284     {
285         int hi,lo;
286         hi=int(mtr[mtr.m_height-1][x].m_value>>32);
287         lo=int(mtr[mtr.m_height-1][x].m_value);
288         s << "0"
289             << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << hi << " "
290             << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << lo << "hl,";
291     }
292
293     {
294         int hi,lo;
295         hi=int(mtr[mtr.m_height-1][mtr.m_width-1].m_value>>32);
296         lo=int(mtr[mtr.m_height-1][mtr.m_width-1].m_value);
297         s << "0"
298             << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << hi << " "
299             << hex << setw(8) << setfill('0') << setiosflags(ios::uppercase) << lo << "hl";
300     }
301     s << ")\n";
302
303     return s;
304 }
305 }
```

```

306     _ASERTE(mtr.m_width%8==0);
307     nmmtr64s mClone(_int64*mtr.m_data,mtr.m_height,mtr.m_width/8,mtr.m_stride/8);
308     AsmArray(s,mClone);
309     return s;
310 }
311
312 template<class T1, class T2> void DotMul(nmmtr<T1>& mSrcMtr1, nmmtr<T2>& mSrcMtr2, nmmtr<T2>& mDstMtr)
313 {
314     for(int y=0; y< mSrcMtr1.m_height; y++)
315         for(int x=0; x<mSrcMtr1.m_width; x++)
316         {
317             nmint<T2> res=mSrcMtr1[y][x]*mSrcMtr2[y][x];
318             mDstMtr[y][x]=res;
319         }
320     }
321
322 template<class T1, class T2> void GetSum(nmmtr<T1>& mSrcMtr1, nmint<T2>& nResSum)
323 {
324     nResSum=0;
325     for(int y=0; y< mSrcMtr1.m_height; y++)
326         for(int x=0; x<mSrcMtr1.m_width; x++)
327             nResSum+=nmint<T2>(mSrcMtr1[y][x].m_value);
328     }
329
330 #endif
331 #endif

```

7.82 tcmplx.h

```

1 // Header file of template class of complex numbers
2 //////////////////////////////////////////////////////////////////
3 //////////////////////////////////////////////////////////////////
4 //////////////////////////////////////////////////////////////////
5 //////////////////////////////////////////////////////////////////
6
7 #ifndef _TCMPLX_H_INCLUDED_
8 #define _TCMPLX_H_INCLUDED_
9
10 #ifdef __cplusplus
11
12 #include <assert.h>
13 #include <math.h>
14
15
16
17
18 //*****
19 //*****
20
21
22 template <class T> class cmplx
23 {
24 public:
25     T re;
26     T im;
27
28     cmplx()
29         :re(0),im(0){}
30     cmplx(T _re)
31         :re(_re),im(0){}
32     cmplx(T _re,T _im)
33         :re(_re),im(_im){}
34     cmplx(const cmplx<T>& c)
35         :re(c.re),im(c.im){}
36     explicit cmplx(cmplx<long long>& c)
37         :re(c.re),im(c.im){}
38     //----- Z=Y -----
39 /*
40 cmplx<T>& operator= ( cmplx<T>& y)
41 {
42     re=y.re;
43     im=y.im;
44     return (*this);
45 }
46
47 cmplx<T>& operator= (const cmplx<T>& y)
48 {
49     re=y.re;
50     im=y.im;
51     return (*this);
52 }

```

```

83 //----- Z=Y -----
84 cmplx<T>& operator= (const double y)
85 {
86     re=y;
87     im=0;
88     return (*this);
89 }
90 */
91 //----- Z=Y -----
92 cmplx<T>& operator= (const int y)
93 {
94     re=y;
95     im=0;
96     return (*this);
97 }
98 */
99
100 cmplx<T>& operator+= (const cmplx<T>& y);
101 cmplx<T> operator+ (const cmplx<T>& y) const;
102 cmplx<T>& operator-= (const cmplx<T>& y);
103 cmplx<T> operator- (const cmplx<T>& y) const;
104 //----- Z*=Y -----
105 cmplx<T>& operator*= (const cmplx<T>& y)
106 {
107     cmplx<T> x(*this);
108     re=x.re*y.re-x.im*y.im;
109     im=x.re*y.im+x.im*y.re;
110     return (*this);
111 }
112 //----- Z=X*Y -----
113 template <class P> cmplx<P> operator* (const cmplx<P>& y) const
114 {
115     cmplx<P> z;
116     z.re=re*y.re-im*y.im;
117     z.im=re*y.im+im*y.re;
118     return z;
119 }
120 cmplx<T>& operator/= (const cmplx<T>& y);
121 cmplx<T> operator/ (const cmplx<T>& y) const;
122 cmplx<T> operator- (void) const;
123 //----- Z»=y -----
124 cmplx<T>& operator»= (const int y)
125 {
126     re»=y;
127     im»=y;
128     return (*this);
129 }
130
131 //----- Z=X»y -----
132 cmplx<T> operator» (const int y) const
133 {
134     cmplx<T> z(*this);
135     z»=y;
136     return z;
137 }
138 //----- Z«=y -----
139 cmplx<T>& operator«= (const int y)
140 {
141     re«=y;
142     im«=y;
143     return (*this);
144 }
145 //----- Z=X«y -----
146 cmplx<T> operator« (const int y) const
147 {
148     cmplx<T> z(*this);
149     z«=y;
150     return z;
151 }
152
153 bool operator== (const cmplx<T>& y) const
154 {
155     return ((re==y.re)&&(im==y.im));
156 }
157
158 bool operator!= (const cmplx<T>& y) const
159 {
160     return ((re!=y.re)||((im!=y.im)));
161 }
162
163 cmplx<T> conjg () const;
164 T real () const;
165 void real (const T& Re);
166 T imag () const;
167 void imag (const T& Im);
168
169 };

```

```

170
171 //----- Z=Y -----
172 /*----- Z=Y -----
173 */
174 template <class T> cmplx<T>::operator cmplx<double> ()
175 {
176     cmplx<Double> DblComplex;
177     DblComplex.re=re;
178     DblComplex.im=im;
179     return (DblComplex);
180 }
181 */
182 //----- Z+=Y -----
183 template <class T> cmplx<T>& cmplx<T>::operator+=(const cmplx<T>& y)
184 {
185     re+=y.re;
186     im+=y.im;
187     return (*this);
188 }
189 //----- Z=X+Y -----
190 template <class T> cmplx<T> cmplx<T>::operator+(const cmplx<T>& y) const
191 {
192     cmplx<T> z(*this);
193     z+=y;
194     return z;
195 }
196 //----- Z-=Y -----
197 template <class T> cmplx<T>& cmplx<T>::operator-=(const cmplx<T>& y)
198 {
199     re-=y.re;
200     im-=y.im;
201     return (*this);
202 }
203 //----- Z=X-Y -----
204 template <class T> cmplx<T> cmplx<T>::operator-=(const cmplx<T>& y) const
205 {
206     cmplx<T> z(*this);
207     z-=y;
208     return z;
209 }
210 //----- Z=-Y -----
211 template <class T> cmplx<T> cmplx<T>::operator-=(void) const
212 {
213     cmplx<T> z(*this);
214     z.re=-z.re;
215     z.im=-z.im;
216     return z;
217 }
218 //----- Z/=Y -----
219 template <class T> cmplx<T>& cmplx<T>::operator/=(const cmplx<T>& y)
220 {
221     //__ASSERT((y.re==(T)0)||((y.im==(T)0));
222     //cmplx<T> Conjug;
223     //Conjug=y.conjg();
224     if (y.im===(T)0)
225     {
226         re/=y.re;
227         im/=y.re;
228     }
229     else
230     {
231         (*this)*=y.conjg();
232         T denum=y.re*y.re+y.im*y.im;
233         re/=denum;
234         im/=denum;
235     }
236     return (*this);
237 }
238 //----- Z=X/y -----
239 template <class T> cmplx<T> cmplx<T>::operator/=(const cmplx<T>& y) const
240 {
241     cmplx<T> z(*this);
242     z/=y;
243     return z;
244 }
245 //***** Comparison operators *****
246 /*----- Comparison operators -----
247 -----*/
248
249 //----- Z=conjg(X) -----
250 template <class T> cmplx<T> cmplx<T>::conjg() const
251 {
252     cmplx<T> z(*this);
253     z.im=-z.im;
254     return z;

```

```
257 }
258 template <class T> T cmplx<T>::real()const
259 { return re; }
260 template <class T> void cmplx<T>::real(const T& Re)
261 { re=Re; }
262 template <class T> T cmplx<T>::imag()const
263 { return im; }
264 template <class T> void cmplx<T>::imag(const T& Im)
265 { im=Im; }
266 /*
267 template <class T> void Out(const cmplx<T>& x)
268 {
269 printf("[";
270 Out(x.re);
271 Out(x.im);
272 printf("]");
273 }
274 */
275 /*
276 ****
277 /* Type conversion */
278 ****
279 /*
280 template <class T> INLINE void double_(cmplx<T> &X, double* &pDstVec)
281 {
282 double_(X.re,pDstVec);
283 double_(X.im,pDstVec);
284 }
285 */
286 /*
287
288
289 template <class T> double abs2(const cmplx<T>& x)
290 {
291     return (double)(x.re*x.re+x.im*x.im);
292 }
293
294 template <class T, class T2> void abs2(const cmplx<T>& x, T2 &res)
295 {
296     res =(T2)(x.re*x.re+x.im*x.im);
297 }
298
299 template <class T> double abs(const cmplx<T>& x)
300 {
301     return (double)(sqrt(x.re*x.re+x.im*x.im));
302 }
303
304 template <class T> cmplx<T> exp(const cmplx<T> &arg)
305 {
306     cmplx<T> Res(arg);
307     T ExpRe;
308     ExpRe=exp(arg.re);
309     Res.re=ExpRe*cos(arg.im);
310     Res.im=ExpRe*sin(arg.im);
311     return Res;
312 }
313 }
314 #endif
315
316 #endif
```

7.83 tcmplx_spec.h

```
1 // Header file of template class of complex numbers // //
2 // // // // //
3 // // // // //
4 // // // // //
5 // // // // //
6 // // // // //
7 // // // // //
8 #ifndef _TCMPLX_CPEC_H_INCLUDED_
9 #define _TCMPLX_CPEC_H_INCLUDED_
10
11 #include <nmtl.h>
12
13
14
15 //----- Z»=y -----
16 template<>
17 cmplx<double>& cmplx<double>::operator»= (const int Shr);
18 /**
19 // double n=pow((double)2,Shr);
20 // re/=n;
21 // im/=n;
22 // return (*this);
```

```

23 //}
24 /*
25 //----- Z=X»y -----
26 cmplx<double> cmplx<double>::operator» (const int y) const
27 {
28 cmplx<T> z(*this);
29 z»=y;
30 return z;
31 }/*
32 //----- Z«=y -----
33 template<>
34 cmplx<double>& cmplx<double>::operator«= (const int Shl);
35 /**
36 // double n=pow((double)2,Shl);
37 // re*=n;
38 // im*=n;
39 // return (*this);
40 /**
41 /*
42 //----- Z=X«y -----
43 cmplx<double> cmplx<double>::operator« (const int y) const
44 {
45 cmplx<T> z(*this);
46 z«=y;
47 return z;
48 }
49 */
50 //----- Z=Y -----
51 /*
52 template <class T> cmplx<T>::operator cmplx<double> ()
53 {
54 cmplx<Double> DblComplex;
55 DblComplex.re=re;
56 DblComplex.im=im;
57 return (DblComplex);
58 }
59 */
60
61 #endif

```

7.84 tfixpoint.h

```

1 #ifndef _tfixpoint
2 #define _tfixpoint
3 // #include <math.h>
4 // #define STAT_MINMAX
5 // #define STAT_FIXPOINT
6 // #ifdef __NM__
7 // #define _ASSE RTE(a)
8 // #else
9 // #include <crtdbg.h>
10 // #endif
11
12 #define FIX_TYPE __int64
13 #define ROUND2FIXED(a) ((a) >= 0 ? (((a)+0.5)) : (((a)-0.5)))
14
15 #define FIXPOINT_ONE (T(1)<point>)
16
17 #define DOUBLE2FIX64(a,point) (ROUND2FIXED((a)*((__int64(1))<point>)))
18 #define DOUBLE2FIX32(a,point) (ROUND2FIXED((a)*(int(1)<point>)))
19
20
21
22
23 // #define FIX_POINT_ONE (1<FIX_POINT>)
24 #define FLT2FIX(f) (DOUBLE2FIX64((f),(FIX_POINT)))
25
26 #define MULI(a,i) ((a).m_value*(i))
27 #define MULC(a,c) ((a.m_value*c)>FIX_POINT)
28 #define MULX(a,n) (((a).m_value*(n).m_value)>FIX_POINT)
29
30 #define ADDC(a,c) (a.m_value+c)
31 #define ADDN(a,n) (a.m_value+n.m_value)
32 #define SUBC(a,c) (a.m_value-c)
33 #define SUBN(a,n) (a.m_value-n.m_value)
34
35 // #define SHR(a,shift) (((a)+((1)<<((shift)-1)))>shift)
36 // #define FIX_RSH(a,shift) ((a)>>shift)
37 #define SHR(a,shift) ((a)>>shift)
38
39
40 // #ifndef _ASERTE
41 // #define _ASERTE(a)
42 // #endif

```

```

43
44 #define DBL2FIX DOUBLE2FIX64
45
46 #ifndef __NM__
47 #define INLINE
48 #else
49 #define INLINE __INLINE__
50#endif
51
52 template<class T, int point> class tfixpoint{
53 public:
54     T m_value; // m_value
55     //----- 0 operands constructors -----
56     INLINE tfixpoint(void){}
57     //----- 1 operands constructors -----
58
59     INLINE tfixpoint(const int val)
60     {
61         m_value=T(val)«point;
62         //__ASERTE(m_value<(T(1)«32));
63         //__ASERTE(m_value>(-(T(1)«32)));
64     }
65
66     INLINE tfixpoint(const float val)
67     {
68         m_value=ROUND2FIXED(val*FIXPOINT_ONE);
69         //__ASERTE(m_value<(T(1)«32));
70         //__ASERTE(m_value>(-(T(1)«32)));
71     }
72
73     INLINE tfixpoint(const double val)
74     {
75         m_value=ROUND2FIXED(val*FIXPOINT_ONE);
76         //__ASERTE(m_value<(T(1)«32));
77         //__ASERTE(m_value>(-(T(1)«32)));
78     }
79
80     //tfixpoint(const tfixpoint<T,point> val)
81     //{
82     //    m_value=val.m_value;
83     //}
84
85     INLINE tfixpoint<T,point>& operator=(const int val)
86     {
87         m_value=T(val)«point;
88         return (*this);
89     }
90
91     INLINE tfixpoint<T,point>& operator=(const float val)
92     {
93         m_value=ROUND2FIXED(val*FIXPOINT_ONE);
94         return (*this);
95     }
96
97     tfixpoint<T,point>& operator=(const double val)
98     {
99         m_value=ROUND2FIXED(val*FIXPOINT_ONE);
100        return (*this);
101    }
102
103    INLINE tfixpoint<T,point>& operator=(const tfixpoint<T,point>& val)
104    {
105        m_value=val.m_value;
106        return (*this);
107    }
108 //endif
109
110
111 // tfixpoint<T,point>& operator=(const tfixpoint<T,point>& val)
112 // {
113 //     m_value=val.m_value;
114 //     return (*this);
115 // }
116 //
117
118 // template <class T2> tfixpoint<T,point>& operator=(const T2& value)
119 // {
120 //     m_value=value;
121 //     return (*this);
122 // }
123
124
125     INLINE tfixpoint<T,point>& operator=(const tfixpoint<T,point>& val)
126     {
127 #ifdef STAT_FIXPOINT
128     stat.subs++;
129#endif

```

```

130     m_value-=val.m_value;
131     return (*this);
132 }
133
134
135     INLINE tfixpoint<T,point>& operator+=(const tfixpoint<T,point>& val)
136 {
137 #ifdef STAT_FIXPOINT
138     stat.adds++;
139 #endif
140     m_value+=val.m_value;
141     return (*this);
142 }
143
144     INLINE tfixpoint<T,point> operator-(const tfixpoint<T,point>& val) const
145 {
146     tfixpoint<T,point> Res(*this);
147     Res-=val;
148     return Res;
149 }
150
151     INLINE tfixpoint<T,point> operator+(const tfixpoint<T,point>& val) const
152 {
153     tfixpoint<T,point> Res(*this);
154     Res+=val;
155     return Res;
156 }
157
158     INLINE tfixpoint<T,point>& operator++(int)
159 {
160 #ifdef STAT_FIXPOINT
161     stat.adds++;
162 #endif
163     m_value+=FIXPOINT_ONE;
164     return (*this);
165 }
166
167     INLINE tfixpoint<T,point>& operator--(int)
168 {
169 #ifdef STAT_FIXPOINT
170     stat.subs++;
171 #endif
172     m_value-=FIXPOINT_ONE;
173     return (*this);
174 }
175
176
177
178
179
180     template<class T2, int point2> INLINE tfixpoint<T,point>& operator*=(const tfixpoint<T2,point2> val)
181 {
182 #ifdef STAT_FIXPOINT
183     stat.muls++;
184 #endif
185     m_value*=val.m_value;
186     m_value+=(T(1)<<(point2-1));
187     m_value>>=point2;
188     return (*this);
189 }
190
191     INLINE tfixpoint<T,point>& operator*=(const int val)
192 {
193 #ifdef STAT_FIXPOINT
194     stat.muls++;
195 #endif
196     m_value*=val;
197     return (*this);
198 }
199
200     INLINE tfixpoint<T,point>& operator*=(const float val)
201 {
202 #ifdef STAT_FIXPOINT
203     stat.muls++;
204 #endif
205     m_value=ROUND2FIXED(m_value*val);
206     return (*this);
207 }
208
209     INLINE tfixpoint<T,point>& operator*=(const double val)
210 {
211 #ifdef STAT_FIXPOINT
212     stat.muls++;
213 #endif
214     //m_value*=val;
215     m_value=ROUND2FIXED(m_value*val);
216     return (*this);

```

```

217     }
218
219     template <int point2> INLINE tfixpoint<T,point> operator*  (const tfixpoint<T,point2> val) const
220 {
221     tfixpoint<T,point> Res(*this);
222     Res*=val;
223     return Res;
224 }
225
226     INLINE tfixpoint<T,point> operator*  (const int val) const
227 {
228     tfixpoint<T,point> Res(*this);
229     Res*=val;
230     return Res;
231 }
232
233 /*
234 INLINE tfixpoint<T,point> operator*  (const float val) const
235 {
236     tfixpoint<T,point> Res(*this);
237     Res*=val;
238     return Res;
239 }
240
241 INLINE tfixpoint<T,point> operator*  (const double val) const
242 {
243     tfixpoint<T,point> Res(*this);
244     Res*=val;
245     return Res;
246 }
247 */
248     INLINE tfixpoint<T,point>& operator/= (const tfixpoint<T,point> val)
249 {
250 #ifdef STAT_FIXPOINT
251     stat.divs++;
252 #endif
253     long long val64=m_value;
254     val64<=point;
255     val64/=val.m_value;
256     m_value=val64;
257     return (*this);
258 }
259
260     INLINE tfixpoint<T,point> operator/  (const tfixpoint<T,point> val) const
261 {
262     tfixpoint<T,point> Res(*this);
263     Res/=val;
264     return Res;
265 }
266
267     INLINE tfixpoint<T,point>& operator»= (const int y)
268 {
269     m_value»=y;
270     return (*this);
271 }
272
273     INLINE tfixpoint<T,point> operator»  (const int y) const
274 {
275     tfixpoint<T,point> z(*this);
276     z»=y;
277     return z;
278 }
279
280     INLINE tfixpoint<T,point>& operator«= (const int y)
281 {
282     m_value«=y;
283     return (*this);
284 }
285
286     INLINE tfixpoint<T,point> operator«  (const int n) const
287 {
288     tfixpoint<T,point> Res(*this);
289     Res«=n;
290     return Res;
291 }
292
293     INLINE tfixpoint<T,point>& operator^= (tfixpoint<T,point> &val)
294 {
295     m_value^=val.m_value;
296     return (*this);
297 }
298     INLINE tfixpoint<T,point> operator^  (tfixpoint<T,point> &val) const
299 {
300     tfixpoint<T,point> Res(*this);
301     Res^=val;
302     return Res;
303 }

```

```

304
305
306     INLINE tfixpoint<T,point>      operator- ()const
307 {
308 #ifdef STAT_FIXPOINT
309     stat.subs++;
310#endif
311     tfixpoint<T,point> Res;
312     Res.m_value=-m_value;
313     return Res;
314 }
315
316     bool operator> (const tfixpoint<T,point>& y)const
317 { return m_value>y.m_value;}
318     bool operator>= (const tfixpoint<T,point>& y)const
319 { return m_value>=y.m_value;}
320     bool operator< (const tfixpoint<T,point>& y)const
321 { return m_value<y.m_value;}
322     bool operator<= (const tfixpoint<T,point>& y)const
323 { return m_value<=y.m_value;}
324     bool operator== (const tfixpoint<T,point>& y)const
325 { return m_value==y.m_value;}
326     bool operator!= (const tfixpoint<T,point>& y)const
327 { return m_value!=y.m_value;}
328
329     float flt(){
330     float res=m_value;
331     res/=FIXPOINT_ONE;
332     return res;
333 }
334 };
335
336 //template class tfixpoint<float>;
337 /*
338 template <class T, int point> INLINE tfixpoint<T,point> pow (tfixpoint<T,point> a, tfixpoint<T,point> b) { return
339     pow(a.fl(), b.fl());}
340 template <class T, int point> INLINE tfixpoint<T,point> fabs(tfixpoint<T,point> a) { return fabs(a.fl());}
341 template <class T, int point> INLINE tfixpoint<T,point> log (tfixpoint<T,point> a) { return log (a.fl());}
342 template <class T, int point> INLINE tfixpoint<T,point> sqrt(tfixpoint<T,point> a) { return sqrt(a.fl());}
343 template <class T, int point> INLINE tfixpoint<T,point> cos (tfixpoint<T,point> a) { return cos (a.fl());}
344 */
345
346 //template class<int point> fixpoint32<point>; public ;
347
348 #define fixpoint64(point) tfixpoint<__int64,point>
349 #define fixpoint32(point) tfixpoint<int,point>
350
351 //typedef template <int point> fixpoint32<int,point>;
352
353 #define FIXPOINT64(val,point) (ROUND2FIXED((val)*(__int64(1))<>point))
354
355 double __INLINE__ cnv2double(double x){
356     return x;
357 }
358
359 float __INLINE__ cnv2float(float x){
360     return x;
361 }
362
363 template <class T, int point> double __INLINE__ cnv2double(tfixpoint<T,point>& x){
364     return x.fl();
365 }
366 template <class T, int point> float __INLINE__ cnv2float(tfixpoint<T,point>& x){
367     return x.fl();
368 }
369
370#endif

```

7.85 tfixpoint.h

```
1 #include "./nmply/nmtl/tfixpoint.h"
```

7.86 tfixpointmath.h

```
1 #ifndef _tfixpoint_MATH
2 #define _tfixpoint_MATH
3 #include <math.h>
4
```

```

5 template <class T, int point> __INLINE__ tfixpoint<T,point> pow (tfixpoint<T,point> a, tfixpoint<T,point> b) { return
6     pow(a.ft(), b.ft());}
7 template <class T, int point> __INLINE__ tfixpoint<T,point> fabs(tfixpoint<T,point> a) { return fabs(a.ft());}
8 template <class T, int point> __INLINE__ tfixpoint<T,point> log (tfixpoint<T,point> a) { return log (a.ft());}
9 template <class T, int point> __INLINE__ tfixpoint<T,point> sqrt(tfixpoint<T,point> a) { return sqrt(a.ft());}
10 template <class T, int point> __INLINE__ tfixpoint<T,point> cos (tfixpoint<T,point> a) { return cos (a.ft());}
11 template <class T, int point> __INLINE__ tfixpoint<T,point> sin (tfixpoint<T,point> a) { return sin (a.ft());}
12
13
14 #endif

```

7.87 tmatrix.h

```

1 // Header file of template of class mtr
2 //////////////////////////////////////////////////////////////////
3 //////////////////////////////////////////////////////////////////
4 //////////////////////////////////////////////////////////////////
5 #ifndef _TMATRIX_H_INCLUDED_
6 #define _TMATRIX_H_INCLUDED_
7
8 //////////////////////////////////////////////////////////////////
9 //////////////////////////////////////////////////////////////////
10 //////////////////////////////////////////////////////////////////
11 #include "crtdbg2.h"
12
13 #include <stdio.h>
14 #include "tvector.h"
15 #include <string.h>
16
17 //////////////////////////////////////////////////////////////////
18 //////////////////////////////////////////////////////////////////
19 // Class of matrixes
20 //////////////////////////////////////////////////////////////////
21
22 //***** *****
23 //***** *****
24 //***** *****
25 //***** *****
26
27 //////////////////////////////////////////////////////////////////
28 //////////////////////////////////////////////////////////////////
29 //////////////////////////////////////////////////////////////////
30 //////////////////////////////////////////////////////////////////
31 //////////////////////////////////////////////////////////////////
32 //////////////////////////////////////////////////////////////////
33 //////////////////////////////////////////////////////////////////
34 //////////////////////////////////////////////////////////////////
35 //////////////////////////////////////////////////////////////////
36 //////////////////////////////////////////////////////////////////
37 //////////////////////////////////////////////////////////////////
38 //////////////////////////////////////////////////////////////////
39 //////////////////////////////////////////////////////////////////
40 //////////////////////////////////////////////////////////////////
41 //////////////////////////////////////////////////////////////////
42 //////////////////////////////////////////////////////////////////
43 //////////////////////////////////////////////////////////////////
44 //////////////////////////////////////////////////////////////////
45 //////////////////////////////////////////////////////////////////
46 //////////////////////////////////////////////////////////////////
47 //////////////////////////////////////////////////////////////////
48 //////////////////////////////////////////////////////////////////
49 //////////////////////////////////////////////////////////////////
50 //////////////////////////////////////////////////////////////////
51 //////////////////////////////////////////////////////////////////
52 //////////////////////////////////////////////////////////////////
53 //////////////////////////////////////////////////////////////////
54 //////////////////////////////////////////////////////////////////
55 //////////////////////////////////////////////////////////////////
56 //////////////////////////////////////////////////////////////////
57 //////////////////////////////////////////////////////////////////
58
59 #ifndef GetVec
60 #define GetVec getvec
61 #endif
62
63 #ifndef GetCol
64 #define GetCol getcol
65 #endif
66
67 #ifndef Min
68 #define Min min
69 #endif
70
71
72 #ifndef Max
73 #define Max max
74 #endif
75
76
77 #ifndef MinPos
78 #define MinPos minpos
79 #endif
80
81 #ifndef MaxPos
82 #define MaxPos maxpos
83 #endif
84
85
86
87 template<class T> class mtr
88 {
89 protected:
90     T* m_container;
91 public:
92     int m_border;
93     int m_stride;
94     int m_height,m_width,m_size;
95     int m_x0;
96     int m_y0;
97     T *m_data;
98
99     explicit mtr()
100    {
101         m_height=0;
102         m_width=0;
103         m_size=0;
104         m_container=0;
105         m_data=0;
106         m_stride=0;

```

```

107    }
108
109 // Setting (0,0) index for element [y][x]
110 void origin(int y,int x){
111     m_x0=x;
112     m_y0=y;
113 }
114
115 // Setting value
116 inline void setval(int y, int x, const T& val)
117 {
118     ASSERTE(y>=-m_border);
119     ASSERTE(y<m_height+m_border);
120     m_data[y*m_stride+x]=val;
121 }
122
123 // extended setting value
124 inline void setvalx(int y, int x, const T& val)
125 {
126     ASSERTE(y+m_y0>=-m_border);
127     ASSERTE(y+m_y0<m_height+m_border);
128     m_data[(y+m_y0)*m_stride+x+m_x0]=val;
129 }
130
131 // getting value
132 inline T getval(int y,int x)const
133 {
134     ASSERTE(y>=-m_border);
135     ASSERTE(y<m_height+m_border);
136     T val=m_data[y*m_stride+x];
137     return val;
138 }
139
140 // extended getting value
141 inline T getvalx(int y,int x)const
142 {
143     ASSERTE(y+m_y0>=-m_border);
144     ASSERTE(y+m_y0<m_height+m_border);
145     T val=m_data[(y+m_y0)*m_stride+x+m_x0];
146     return val;
147 }
148
149 // constructor
150 explicit mtr(int nHeight, int nWidth, int nBorder=0)
151 {
152     m_border=nBorder;
153     m_height=nHeight;
154     m_width=nWidth;
155     m_stride=m_width;
156     m_size=m_width*m_height;
157     m_container=new T[m_size+2*m_stride*m_border];
158     m_data=m_container+m_border*m_stride;
159     ASSERTE(m_container!=NULL);
160 }
161
162 // resizing of matrix dimensions
163 void resize(int nHeight, int nWidth, int nBorder=0)
164 {
165     if (m_container)
166         delete m_container;
167     m_border=nBorder;
168     m_height=nHeight;
169     m_width=nWidth;
170     m_size=m_width*m_height;
171     m_stride=m_width;
172     m_container=new T[m_size+2*m_stride*m_border];
173     m_data=m_container+m_border*m_stride;
174 }
175
176 // constructor for matrix duplication
177 /*explicit*/ mtr(const mtr<T>& matr)
178 {
179     m_height=matr.m_height;
180     m_width=matr.m_width;
181     m_size=matr.m_size;
182     m_stride=m_width;
183     m_border=matr.m_border;
184     m_container=new T[m_size+2*m_stride*m_border];
185     m_data=m_container+m_border*m_stride;
186
187     for(int y=0;y<m_height;y++){
188         for(int x=0;x<m_width;x++){
189             T val=matr.getval(y,x);
190             setval(y,x,val);
191         }
192     };
193 */

```

```

194 ..\include\tmatrix.h(162) : error C2535: '__thiscall mtr<T2>::mtr<T2>(const class mtr<T2> &)' : member function
      already defined or declared
195 ..\include\tmatrix.h(143) : see declaration of 'mtr<T>::mtr<T>'
196 ..\include\tmatrix.h(692) : see reference to class template instantiation 'mtr<T>' being compiled
197 template <class T2> mtr(const mtr<T2>& matr)
198 {
199     m_height=matr.m_height;
200     m_width=matr.m_width;
201     m_size=matr.m_size;
202     m_stride=m_width;
203     m_border=matr.m_border;
204     m_container=new T[m_size+2*m_border*m_stride];
205     m_data=m_container+m_border*m_stride;
206
207     for(int y=0;y<m_height;y++){
208         for(int x=0;x<m_width;x++){
209             T val=matr.getval(y,x);
210             setval(y,x,val);
211         }
212     }
213 };
214 */
215 // constructor of matrix over existing data
216 void assign(T* Data,int nHeight,int nWidth,int nStride=0){
217     m_height=nHeight;
218     m_width=nWidth;
219     if(nStride==0)
220         m_stride=m_width;
221     else
222         m_stride=nStride;
223     m_size=m_height*m_width;
224     m_data=Data;
225     m_container=0;
226     m_border=0;
227 }
228 explicit mtr(T* Data,int nHeight,int nWidth,int nStride=0)
229 {
230     assign(Data,nHeight,nWidth,nStride);
231 }
232
233 // destructor
234 ~mtr()
235 {
236     if(m_container)
237         delete m_container;
238 }
239
240 // assignment
241
242 mtr<T>& operator=(mtr<T>& matr)
243 {
244     ASSERTE(matr.m_height==m_height);
245     ASSERTE(matr.m_width==m_width);
246     T* pSrcData=matr.m_data;
247     T* pDstData=m_data;
248     for(int y=0;y<m_height;y+=pSrcData+=matr.m_stride, pDstData+=m_stride)
249         for(int x=0;x<m_width;x++)
250             pDstData[x]=pSrcData[x];
251     return *this;
252 }
253
254 // filling of matrix with constant value
255 mtr<T>& operator=(T& val)
256 {
257     for(int y=0;y<m_height;y++)
258         getvec(y)=val;
259     return *this;
260 }
261
262 // #ifdef _DEBUG
263 // inline vec<T> operator[](int row)
264 // {
265 //     ASSERTE(row<m_height+m_border);
266 //     ASSERTE(row>=-m_border);
267 //     return vec<T>(m_data+row*m_stride,m_width,m_border);
268 // }
269 // return pointer to first element in row
270 inline T* operator[](int row) const
271 {
272     ASSERTE(row<m_height+m_border);
273     ASSERTE(row>=-m_border);
274     return m_data+row*m_stride;
275 }
276
277 // getting value by linear index
278 inline T& index(int idx)
279 {

```

```

280     ASERTE(idx>=0);
281     ASERTE(idx<m_size);
282     ASERTE(m_width==m_stride);
283     return(m_data[idx]);
284 }
285
286 //#else
287 // inline T* operator[](int row)
288 // {
289 //     return (m_data+row*m_stride);
290 // }
291 //#endif
292
293 // inplace multiplication by constant value
294 mtr<T>& operator*=(const T val)
295 {
296     for(int y=0;y<m_height;y++)
297         getvec(y)*=val;
298     return *this;
299 }
300
301 // multiplication by constant value
302 mtr<T> operator*(const T& val) const
303 {
304     mtr<T> res(*this);
305     return res*=val;
306 }
307
308 // multiplication by vector
309 vec<T> operator*(const vec<T>& vect)
310 {
311     ASERTE(m_width==vect.m_size);
312     vec<T> res(m_height);
313     for(int y=0;y<m_height;y++)
314         res[y]=getvec(y)*vect;
315     return vec<T>(res);
316 }
317
318 // mtr<T> operator*(const mtr<T>& matr)
319 //
320 //     ASERTE(m_width==matr.m_height);
321 //     mtr<T> res(m_height,matr.m_width);
322 //     for(int row=0;row<res.m_height;row++)
323 //         for(int col=0;col<res.m_width;col++)
324 //     {
325 //         T sum(0);
326 //         for(int i=0;i<m_width;i++)
327 //             sum+=m_data[m_width*row+i]*matr.m_data[matr.m_width*i+col];
328 //         res.m_data[row*res.m_width+col]=sum;
329 //     }
330 //     return res;
331 // }
332
333 // multiplication by matrix
334 mtr<T> operator*(const mtr<T>& matr)
335 {
336     ASERTE(m_width==matr.m_height);
337     mtr<T> res(m_height, matr.m_width);
338     res.reset();
339     for(int y=0;y<m_height;y++)
340     {
341         vec<T>& thisvec=getvec(y);
342         vec<T>& resvec=res.getvec(y);
343         for(int x=0;x<m_width;x++)
344             resvec+=matr.getvec(x)*thisvec[x];
345     }
346     return res;
347 }
348
349 // inplace adding of value to all elements of matrix
350 mtr<T>& operator+=(const T &val)
351 {
352     for(int y=0;y<m_height;y++)
353         getvec(y)+=val;
354     return (*this);
355 }
356
357 // inplace summation of matrixes
358 mtr<T>& operator+=(const mtr<T> &matr)
359 {
360     ASERTE (matr.m_size==m_size);
361     for(int y=0;y<m_height;y++)
362         getvec(y)+=matr.getvec(y);
363     return (*this);
364 }
365
366 // summation of matrices

```

```

367     mtr<T> operator+ (const mtr<T>& matr) const
368     {
369         ASERTE (matr.m_size==m_size);
370         mtr<T> res(*this);
371         res+=matr;
372         return res;
373     }
374
375     //inplace subtraction of matrix
376     mtr<T>& operator-= (const mtr<T> &matr)
377     {
378         ASERTE (matr.m_size==m_size);
379         for(int y=0;y<m_height;y++)
380             getvec(y)=matr.getvec(y);
381         return (*this);
382     }
383
384     // inplace subtraction of value from all elements of matrix
385     mtr<T>& operator-= (const T &val)
386     {
387         for(int y=0;y<m_height;y++)
388             getvec(y)=val;
389         return (*this);
390     }
391
392     // subtraction of matrixes
393     mtr<T> operator- (const mtr<T>& matr) const
394     {
395         ASERTE (matr.m_size==m_size);
396         mtr <T> res(*this);
397         res-=matr;
398         return res;
399     }
400
401     // sign inversion
402     mtr<T> operator- ()const
403     {
404         mtr<T> res(m_height,m_width);
405         for(int y=0;y<m_height;y++)
406             res.getvec(y)=-getvec(y);
407         return res;
408     }
409
410     // inplace division
411     mtr<T>& operator/=(const T val)
412     {
413         ASERTE(val!=0);
414         for(int y=0;y<m_height;y++)
415             getvec(y)/=val;
416         return (*this);
417     }
418
419     // division
420     mtr<T> operator/ (const T val) const
421     {
422         mtr<T> res(*this);
423         res/=val;
424         return res;
425     }
426
427     // inplace shift
428     mtr<T>& operator>=(const int shr)
429     {
430         ASERTE(shr>=0);
431         for(int y=0;y<m_height;y++)
432             getvec(y)>=shr;
433         return (*this);
434     }
435
436     // shift
437     mtr<T> operator>> (const int shr) const
438     {
439         ASERTE(shr>=0);
440         mtr<T> res(*this);
441         res>>=shr;
442         return res;
443     }
444
445     // inplace left shift
446     mtr<T>& operator<=(const int shl)
447     {
448         ASERTE(shl>=0);
449         for(int y=0; y<m_height; y++)
450             getvec(y)<=shl;
451         return (*this);
452     }
453

```

```

454 // left shift
455 mtr<T> operator<< (const int shl) const
456 {
457     ASSERTE(shl>=0);
458     mtr<T> res(*this);
459     res<=shl;
460     return res;
461 }
462
463 // inplace AND with value
464 mtr<T>& operator&= (const T &val)
465 {
466     for(int y=0;y<m_height;y++)
467         getvec(y)&=val;
468     return (*this);
469 }
470
471
472 // inplace AND with matrix
473 mtr<T>& operator&= (const mtr<T> &matr)
474 {
475     ASSERTE (matr.m_size==m_size);
476     for(int y=0;y<m_height;y++)
477         getvec(y)&=matr.getvec(y);
478     return (*this);
479 }
480
481 // AND
482 mtr<T> operator& (const mtr<T>& matr) const
483 {
484     ASSERTE (matr.m_size==m_size);
485     mtr <T> res(*this);
486     res&=matr;
487     return res;
488 }
489
490 // inplace OR
491 mtr<T>& operator|= (const mtr<T> &matr)
492 {
493     ASSERTE (matr.m_size==m_size);
494     for(int y=0;y<m_height;y++)
495         getvec(y)|=matr.getvec(y);
496     return (*this);
497 }
498
499 // OR
500 mtr<T> operator| (const mtr<T>& matr) const
501 {
502     ASSERTE (matr.m_size==m_size);
503     mtr <T> res(*this);
504     res|=matr;
505     return res;
506 }
507
508 // inplace XOR
509 mtr<T>& operator^= (const mtr<T> &matr)
510 {
511     ASSERTE (matr.m_size==m_size);
512     for(int y=0;y<m_height;y++)
513         getvec(y)^=matr.getvec(y);
514     return (*this);
515 }
516
517 // inplace XOR with constant value
518 mtr<T>& operator^= (const T &val)
519 {
520     for(int y=0;y<m_height;y++)
521         getvec(y)^=val;
522     return (*this);
523 }
524
525 // XOR
526 mtr<T> operator^ (const mtr<T>& matr) const
527 {
528     ASSERTE (matr.m_size==m_size);
529     mtr <T> res(*this);
530     res^=matr;
531     return res;
532 }
533
534 // XOR
535 mtr<T> operator^ (const T& val) const
536 {
537     mtr <T> res(*this);
538     res^=val;
539     return res;
540 }

```

```

541 // NOT
542 mtr<T> operator ~ ()
543 {
544     mtr <T> res(*this);
545     T* r= res.m_data;
546     T* s=m_data;
547     for(int y=0;y<m_height;y++, r+=res.m_stride, s+=m_stride)
548         for(int x=0; x<m_width;x++)
549             r[x]=~s[x];
550
551     return res;
552 }
553
554
555
556
557 void set (const T val)
558 {
559     T* p=m_container;
560     for(int y=0; y<m_height; y++,p+=m_stride){
561         for(int x=0; x<m_width; x++){
562             p[x]=val;
563         }
564     }
565 }
566
567
568 /*
569 void SetBorder(const T val, int nBorderWidth)
570 {
571     T* p=m_container;
572     for(int y=0; y<m_border+nBorderWidth; y++,p+=m_stride){
573         for(int x=0; x<m_width; x++){
574             p[x]=val;
575         }
576     }
577
578     p=addr(nHeight-1-nBorderWidth,0);
579     for(int y=0; y<m_border+nBorderWidth; y++,p+=m_stride){
580         for(int x=0; x<m_width; x++){
581             p[x]=val;
582         }
583     }
584 }*/
585 // void set(const T val, const T border)
586 // {
587 //     T* p0;
588 //     T* p1;
589 //     for(p0=m_container,p1=m_data+m_size;p0<m_data;p0++,p1++)
590 //     {
591 //         *p0=border;
592 //         *p1=border;
593 //     }
594 //     for(int i=0;i<m_size;i++)
595 //         m_data[i]=val;
596 // }
597 /*
598
599 template<class T> void SetData(T* Data)
600 {
601     for(int i=0;i<m_size;i++)
602         m_data[i]=Data[i];
603 }
604
605 template<class T> void GetData(T* Data)
606 {
607     for(int i=0;i<m_size;i++)
608         Data[i]=(T)m_data[i].value;
609 }
610
611 mtr<T>& operator*=(const mtr<T>& matr)
612 {
613     ASSERTE(m_width==m_height);
614     ASSERTE(m_width==matr.m_width);
615     ASSERTE(m_height==matr.m_height);
616     vec<T> res(m_height);
617     for(int row=0;row<m_height;row++)
618         res[row]=*this)[row]*vec;
619     return res;
620 }
621
622
623 void InitConst(T &Value)
624 {
625     int m_size=m_height*m_width;
626     for(int i=0;i<m_size;i++)
627         m_data[i]=Value;

```

```

628 }
629 // void InitConst(T &Value)
630 // {
631 //     int m_size=m_height*m_width;
632 //     for(int i=0;i<m_size;i++)
633 //         m_data[i]=Value;
634 // }
635 // }
636
637 //THIS FUNCTIONS WAS INSERTED FROM OLD VERSION OF THIS FILE!!!!!!!!!!!!!!!
638 mtr<T> transpose()
639 {
640     mtr<T> Z(m_width,m_height);
641     for(int row=0;row<m_height;row++)
642         for(int col=0;col<m_width;col++)
643             Z[col][row]=(*this)[row][col];
644     return Z;
645 }
646
647 mtr<T>& diag(T val)
648 {
649     reset();
650     int size=(m_height<m_width)?(m_height):(m_width);
651     for(int i=0; i<size; i++)
652         (*this)[i][i]=val;
653     return (*this);
654 }
655
656
657
658
659
660 // template<class T> void SetMatrix(mtr<T> &matr)
661 // {
662 //     int m_size=m_height*m_width;
663 //     for(int i=0;i<m_size;i++)
664 //         m_data[i]=matr.m_data[i];
665 // }
666
667 inline T* addr(int y,int x)
668 {
669     ASSERTE(y<m_height+m_border);
670     ASSERTE(y>=m_border);
671     ASSERTE(x<m_width+m_border);
672     ASSERTE(x>=m_border);
673     T* address=m_data+m_stride*y+x;
674     return address;
675 }
676
677 void reset()
678 {
679 //     ASSERTE(m_container);
680 //     if (m_container){
681 //         memset(m_container,0,(m_size+2*m_border*m_width)*sizeof(T));
682 //     }else{
683 //         memset(m_data,0,m_size*sizeof(T));
684 //     }
685 }
686
687 int sum()
688 {
689     int summ=0;
690     for(int y=0;y<m_height;y++)
691     {
692         summ+=getvec(y).sum();
693     }
694     return summ;
695 }
696 }
697
698
699 vec<T> getvec(int y) const
700 {
701     ASSERTE(y>=0 && y<m_height);
702     return vec<T>(m_data+y*m_stride,m_width,m_border);
703 }
704
705 vec<T> getcol(int x) const
706 {
707     ASSERTE(x>=0 && x<m_width);
708     vec<T> vect(m_height);
709     for(int i=0; i<m_height; i++)
710         vect[i]=(*this)[i][x];
711     return vect;
712 }
713
714 }

```

```

715 /*
716 T min()
717 {
718 T m=m_data[0];
719 T* row=m_data;
720 for(int y=0; y<m_height; y++, row+=m_stride)
721 for(int x=0; x<m_width; x++)
722 if (row[x]<m)
723 m=row[x];
724 return m;
725 }
726
727 T max()
728 {
729 T m=m_data[0];
730 T* row=m_data;
731 for(int y=0; y<m_height; y++, row+=m_stride)
732 for(int x=0; x<m_width; x++)
733 if (row[x]>m)
734 m=row[x];
735 return m;
736 }
737 */
738 T minpos(int &ypos, int &xpos){
739     ypos=0;
740     xpos=0;
741     T m=m_data[0];
742     T* row=m_data;
743     for(int y=0; y<m_height; y++, row+=m_stride)
744         for(int x=0; x<m_width; x++)
745             if (row[x]<m){
746                 m=row[x];
747                 ypos=y;
748                 xpos=x;
749             }
750     return m;
751 }
752
753 T maxpos(int &ypos, int &xpos){
754     ypos=0;
755     xpos=0;
756     T m=m_data[0];
757     T* row=m_data;
758     for(int y=0; y<m_height; y++, row+=m_stride)
759         for(int x=0; x<m_width; x++)
760             if (row[x]>m){
761                 m=row[x];
762                 ypos=y;
763                 xpos=x;
764             }
765     return m;
766 }
767
768 void CopyTo(T* pData){
769     T* row=m_data;
770     for(int y=0; y<m_height; y++, row+=m_stride, pData+=m_width){
771         memcpy(pData, row, sizeof(T)*m_width);
772     }
773 }
774
775 void CopyFrom(T* pData){
776     T* row=m_data;
777     for(int y=0; y<m_height; y++, row+=m_stride, pData+=m_width){
778         memcpy(row, pData, sizeof(T)*m_width);
779     }
780 }
781
782 template <class T2> void ConvertTo(T2* pData){
783     T* row=m_data;
784     for(int y=0; y<m_height; y++, row+=m_stride, pData+=m_width){
785         for(int x=0; x<m_width; x++){
786             pData[x]=row[x];
787         }
788     }
789 }
790
791 template <class T2> void ConvertFrom(T2* pData){
792     T* row=m_data;
793     for(int y=0; y<m_height; y++, row+=m_stride, pData+=m_width){
794         for(int x=0; x<m_width; x++){
795             row[x]=pData[x];
796         }
797     }
798 }
799
800 */

```

```

802 inline T max()
803 {
804     T Max=m_data[0];
805     T* row=m_data;
806     for(int y=0; y<m_height; y++, row+=m_stride)
807     for(int x=0; x<m_width; x++)
808         if (row[x]<min)
809             Max=row[x];
810     return Max;
811 }
812 */
813 //THIS FUNCTIONS WAS INSERTED FROM OLD VERSION OF THIS FILE!!!!!!!!!!!!!!!
814
815 };
816
817
818 /*
819 template <class T1, class T2> convert(mtr<T1>& pSrcMtr1,mtr<T2>& pSrcMtr2)
820 {
821     for(int i=0;i<pSrcMtr1.m_size;i++)
822         pSrcMtr2.m_data[i]=pSrcMtr1.m_data[i];
823 }
824 */
825
826 template<class T> mtr<T> DirectProduct(vec<T>& vSrc1,vec<T>& vSrc2)
827 {
828     mtr<T> res(vSrc1.m_size, vSrc2.m_size);
829     for(int y=0; y<res.m_height; y++){
830         res.getvec(y)=vSrc2*vSrc1[y];
831     }
832     return res;
833
834 }
835
836
837 #ifndef __NM__
838
839 template<class T1, class T2> void DotMul(mtr<T1>& mSrcMtr1, mtr<T2>& mSrcMtr2, mtr<T2>& mDstMtr )
840 {
841     for(int y=0; y< mSrcMtr1.m_height; y++)
842         for(int x=0; x<mSrcMtr1.m_width; x++)
843         {
844             T2 res=mSrcMtr1[y][x]*mSrcMtr2[y][x];
845             mDstMtr[y][x]=res;
846         }
847 }
848
849 template<class T1, class T2> void GetSum(mtr<T1>& mSrcMtr1, T2& nResSum)
850 {
851     nResSum=0;
852     for(int y=0; y< mSrcMtr1.m_height; y++)
853         for(int x=0; x<mSrcMtr1.m_width; x++)
854             nResSum+=T2 (mSrcMtr1[y][x]);
855 }
856 template<class T1, class T2> void Convert(mtr<T1>& mSrcMtr, mtr<T2>& mDstMtr){
857     ASSERTE(mSrcMtr.m_height==mDstMtr.m_height);
858     ASSERTE(mSrcMtr.m_width ==mDstMtr.m_width);
859     for(int y=0; y<mSrcMtr.m_height; y++){
860         for(int x=0; x<mSrcMtr.m_width; x++){
861             mDstMtr[y][x]=(T2)mSrcMtr[y][x];
862         }
863     }
864 }
865 #endif
866 #endif

```

7.88 tmatrix_spec.h

```

1
2 //////////////////////////////////////////////////////////////////
3 // Header file of template of class mtr
4 //////////////////////////////////////////////////////////////////
5 #ifndef _TMATRIX_SPEC_H_INCLUDED_
6 #define _TMATRIX_SPEC_H_INCLUDED_
7
8
9 //include "tnmtl.h"
10
11 //THIS FUNCTIONS WAS INSERTED FROM OLD VERSION OF THIS FILE!!!!!!!!!!!!!!!
12 template<>
13 mtr<double>& mtr<double>::operator<= (const int Shl)
14 {
15     int m_size=m_height*m_width;
16     //double n=pow(2,Shl);
17     double n=1<<Shl;

```

```

18     for(int i=0;i<m_size;i++)
19         m_data[i]*=n;
20     return (*this);
21 }
22 //THIS FUNCTIONS WAS INSERTED FROM OLD VERSION OF THIS FILE!!!!!!!!!!!!!!!
23
24
25 #endif

```

7.89 tnmcube.h

```

1 //////////////////////////////////////////////////////////////////
2 // Header file of template of class nmmtr
3 //////////////////////////////////////////////////////////////////
4 //////////////////////////////////////////////////////////////////
5 #ifndef _TCUBE_H_INCLUDED_
6 #define _TCUBE_H_INCLUDED_
7
8
9 #include "crtdbg.h"
10 // #ifdef __cplusplus
11
12
13 // #include "tmatrix.h"
14 #include <string.h>
15
16
17
18 template<class T> class tmtr
19 {
20 protected:
21     T* m_container;
22 public:
23     int m_stride;
24     int m_height, m_width;
25     T *m_data;
26
27     tmtr(tmtr<T>& mtr)
28     {
29         m_stride = mtr.m_stride;
30         m_height = mtr.m_height;
31         m_width = mtr.m_width;
32
33         m_container = new T[m_height*m_stride];
34         m_data = m_container;
35         memcpy(m_data, mtr.m_data, m_height*m_stride);
36     }
37
38
39 // constructor
40 // explicit
41 tmtr(int nHeight, int nWidth, int nStride = 0) :m_height(nHeight), m_width(nWidth)
42 {
43
44     nStride == 0? m_stride = nWidth: m_stride = nStride;
45     m_container = new T[m_height*m_stride];
46     m_data = m_container;
47 }
48
49 // explicit
50 tmtr(T* Data, int nHeight, int nWidth, int nStride=0) :m_data(Data), m_height(nHeight), m_width(nWidth)
51 {
52     nStride == 0 ? m_stride = m_width : m_stride = nStride;
53     _ASSERTE(m_stride >= m_width);
54     m_container = 0;
55 };
56
57 T* ref(int y, int x) {
58     return m_data + y*m_stride + x;
59 }
60 tmtr()
61 {
62     if (m_container)
63         delete m_container;
64 }
65
66 inline T* operator[](int row) const
67 {
68
69     //_ASSERTE(row < m_height + PADDING);
70     //_ASSERTE(row >= -PADDING);
71     return m_data + row*m_stride;
72 }
73 void fill(T& value) {
74     T* p = m_data;

```

```

75     for (int y = 0; y < m_height; y++, p += m_stride)
76         for (int x = 0; x < m_width; x++)
77             p[x] = value;
78     }
79 }
80
82 //-
83 // Class of nmmtres
84 //-
85
86
87 //-----
88 //-----
89 //-----
90 //-----
91 //-----
92
93 template<class T> class tcube
94 {
95 protected:
96     T*    m_container;
97 public:
98     int    m_height;
99     int    m_width;
100    int    m_depth;
101    int    m_ystride;
102    int    m_wstride;
103    int    m_cstride;
104    T*    m_data;
105
106
107 tcube(tcube<T>& cube)
108 {
109     m_height= cube.m_height;
110     m_width= cube.m_width;
111     m_depth= cube.m_depth;
112     m_ystride= cube.m_ystride;
113     m_wstride= cube.m_wstride;
114     m_cstride= cube.m_cstride;
115
116
117     m_container = new T[m_height*m_ystride];
118     m_data = m_container;
119     memcpy(m_data, cube.m_data, m_height*m_ystride);
120 }
121
122 explicit
123 tcube(int Height, int Width, int Depth, int WStride=0, int CStride=0):m_height(Height), m_width(Width),
124 m_depth(Depth)
125 {
126     WStride == 0 ? m_wstride = m_width: m_wstride=WStride;
127     CStride == 0 ? m_cstride = m_depth: m_cstride=CStride;
128     _ASSERTE(m_width<=m_wstride);
129     m_ystride = m_cstride*m_wstride;
130     m_container=new T[m_height*m_ystride];
131     m_data=m_container;
132 }
133
134 explicit
135 tcube(T* data, int Height, int Width, int Depth, int WStride = 0, int CStride = 0) :m_data (data), m_height(Height),
136 m_width(Width), m_depth(Depth)
137 {
138     WStride == 0 ? m_wstride = m_width : m_wstride = WStride;
139     CStride == 0 ? m_cstride = m_depth : m_cstride = CStride;
140     m_ystride = m_cstride*m_wstride;
141     m_container = 0;
142 }
143
144 tcube<T>& operator =(tcube<T>& cube)
145 {
146     _ASSERTE(cube.m_height == m_height);
147     _ASSERTE(cube.m_width == m_width);
148     _ASSERTE(cube.m_depth == m_depth);
149     for (int y = 0; y < m_height; y++)
150         for (int x = 0; x < m_width; x++)
151             for (int c = 0; c < m_depth; c++)
152                 (*this)[y][x][c] = cube[y][x][c];
153
154     return *this;
155 }
156 inline tmtr<T> operator [] (int y) const
157 {
158     // ASSERTE(y>=-PADDING);
159     // ASSERTE(y<m_height+PADDING);
160     return tmtr<T>(m_data+y*m_ystride, m_width, m_depth, m_cstride);
161 }
162 tcube()
163 {
164     if (m_container) {
165         _ASSERTE(m_container != (T*)-1);
166         delete m_container;
167     }
168 }
169
170
171

```

```

172     m_container = (T*)-1;
173 }
174 }
175 }
176 inline T* data() {
177     return m_data;
178 }
179 }
180 void printChannel(int ch) {
181     char str[32];
182     int maxlen = 0;
183     for (int h = 0; h < m_height; h++) {
184         for (int w = 0; w < m_width; w++) {
185             float v = (*this)[h][w][ch];
186             sprintf(str, "%-.2f ", v);
187             int len = strlen(str);
188             if (len > maxlen)
189                 maxlen = len;
190         }
191     }
192     char spaces[] = "          ";
193     for (int h = 0; h < m_height; h++) {
194         for (int w = 0; w < m_width; w++) {
195             float v = (*this)[h][w][ch];
196             sprintf(str, "%-.2f ", v);
197             int len = strlen(str);
198             //strncat(str,spaces,maxlen-len);
199             printf(spaces + 10 - (maxlen - len));
200             printf(str);
201         }
202     }
203     printf("\n");
204 }
205 }
206 }
207 }
208 }
209 }
210 void fill(T value)
211 {
212     T* layer = m_data;
213     for (int y = 0; y < m_height; y++, layer += m_ystride) {
214         tmr<float> mtr(layer, m_width, m_depth, m_cstride);
215         mtr.fill(value);
216     }
217 }
218 }
219 };
220
221 #endif

```

7.90 tnmivec.h

```

1 //-----
2 // $Workfile: tnmivec. $
3 // Векторно-матричная библиотека
4 //
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.2 $      $Date: 2005/05/03 13:22:40 $
8 //
9 //-----
10 //-
11 #ifndef _Tnmivec_H_INCLUDED_
12 #define _Tnmivec_H_INCLUDED_
13
14 #ifdef __cplusplus
15
16 #ifdef ENABLE_ASSERTE
17 #include <crtdbg.h>
18 #define ASSERTE _ASSERTE
19 #else
20 #define ASSERTE(expression)
21 #endif
22
23 #include "tnmint.h"
24
25 //*****
26 //*****
27 //*****
28 //*****
29 //*****
30 //*****
31 //*****
32 //*****
33 //*****
34 //*****
35 //*****
36 //*****
37 //*****
38 //*****
39 //*****
40 //*****
41 //*****
42 //*****
43 //*****
44 //*****
45 //*****
46 //*****
47 //*****
48 //*****
49 //*****
50 //*****
51 //*****
52 //*****
53 //*****
54 //*****
55 //*****
56 //*****
57 template<class T> class nmivec

```

```

58 {
59     protected:
60     cmplx<nmint<T>> * m_container;
61     bool m_flag_new;
62 public:
63     int m_size;
64     cmplx<nmint<T>> * m_data;
65
66     nmivec(cmplx<nmint<T>>* Data, int Size):m_data(Data),m_size(Size),m_flag_new(false){};
67
68     nmivec(int Size,int Reserve=0):m_size(Size),m_flag_new(true)
69     {
70         m_container=new cmplx<nmint<T>>[m_size+2*Reserve];
71         m_data=m_container+Reserve;
72     }
73
74     //nmivec(T * Data, int Size):m_data(Data),m_size(Size),m_flag_new(false){};
75
76
77     nmivec(const nmivec<T> &vec)
78     {
79         m_flag_new=true;
80         m_size=vec.m_size;
81         m_data=m_container=new T[m_size];
82         memcpy(m_data,vec.m_data,m_size*sizeof(T));
83     };
84
85
86     ~nmivec()
87     {
88         if (m_flag_new)
89             delete []m_container;
90     }
91
92     nmivec<T>& operator= (nmivec<T>& vec)
93     {
94         ASSERTE(vec.m_size==m_size);
95         memcpy(m_data,vec.m_data,m_size*sizeof(T));
96         return *this;
97     }
98 }
99
100 #ifndef NMTL_DISABLE_INDEX_CHECK
101     inline cmplx<nmint<T>>& operator [] (int idx)
102     {
103         ASSERTE(idx>=0);
104         ASSERTE(idx<m_size);
105         cmplx<nmint<T>>* res=(cmplx<nmint<T>>*)(m_data+idx);
106         return *res;
107     }
108 #else
109     inline cmplx<nmint<T>>& operator [] (int idx)
110     {
111         return (cmplx<nmint<T>>*)(m_data+idx);
112     }
113 #endif
114
115     template <class T2> nmivec<T2>& operator*= (const cmplx<nmint<T2>> val)
116     {
117         for(int idx=0;idx<m_size;idx++)
118             m_data[idx]*=val.m_value;
119         return (*this);
120     }
121
122     template <class T2> nmivec<T2> operator* (const cmplx<nmint<T2>>& val)const
123     {
124         nmivec<T2> res(m_size);
125         for(int idx=0;idx<m_size;idx++)
126             res.m_data[idx]=m_data[idx]*val.m_value;
127         return res;
128     }
129     template <class T2> nmint<T2> operator* (const nmivec<T2>& vec)const
130     {
131         ASSERTE(m_size==vec.m_size);
132         nmint<T2> res;
133         for (int idx=0;idx<m_size;idx++)
134             res.m_value+=m_data[idx]*vec.m_data[idx];
135         return res;
136     }
137
138     nmivec<T>& operator+= (const cmplx<nmint<T>> &val)
139     {
140         for(int idx=0;idx<m_size;idx++)
141             m_data[idx]+=val.m_value;
142         return (*this);
143     }
144

```

```

145     nmvec<T>& operator+=(const nmvec<T> &vec)
146     {
147         ASSERTE (vec.m_size==m_size);
148         for(int idx=0;idx<m_size;idx++)
149             m_data[idx]+=vec.m_data[idx];
150         return (*this);
151     }
152
153     nmvec<T> operator+ (const cmplx<nmint<T> >& val) const
154     {
155         nmvec<T> res(*this);
156         res+=val;
157         return res;
158     }
159
160     nmvec<T> operator+ (const nmvec<T>& vec) const
161     {
162         ASSERTE (vec.m_size==m_size);
163         nmvec<T> res(*this);
164         res+=vec;
165         return res;
166     }
167
168     nmvec<T>& operator-= (const nmvec<T> &vec)
169     {
170         ASSERTE (vec.m_size==m_size);
171         for(int idx=0;idx<m_size;idx++)
172             m_data[idx]-=vec.m_data[idx];
173         return (*this);
174     }
175
176     nmvec<T> operator- (const nmvec<T>& vec) const
177     {
178         ASSERTE (vec.m_size==m_size);
179         nmvec<T> res(*this);
180         res-=vec;
181         return res;
182     }
183
184     nmvec<T> operator- () const
185     {
186         nmvec<T> res(*this);
187         for(int idx=0;idx<m_size;idx++)
188             m_data[idx]=-m_data[idx];
189         return res;
190     }
191
192     nmvec<T>& operator/=(const cmplx<nmint<T> > val)
193     {
194         ASSERTE(val.m_value!=0);
195         for(int idx=0;idx<m_size;idx++)
196             m_data[idx]/=val.m_value;
197         return (*this);
198     }
199
200     nmvec<T> operator/ (const T val) const
201     {
202         nmvec<T> res(*this);
203         res/=val;
204         return res;
205     }
206
207     nmvec<T>& operator>=(const int shr)
208     {
209         ASSERTE(shr>=0);
210         if (shr==0) return(*this);
211         for(int idx=0;idx<m_size;idx++)
212             m_data[idx]>=shr;
213         return (*this);
214     }
215
216     nmvec<T> operator> (const int shr) const
217     {
218         ASSERTE(shr>=0);
219         nmvec<T> res(*this);
220         res>=shr;
221         return res;
222     }
223
224     nmvec<T>& operator<=(const int shl)
225     {
226         ASSERTE(shl>=0);
227         if (shl==0) return(*this);
228         for(int idx=0;idx<m_size;idx++)
229             m_data[idx]<=shl;
230         return (*this);
231     }

```

```

232     nmcvec<T> operator<< (const int shl) const
233 {
234     ASSERTE(shl>=0);
235     nmcvec<T> res(*this);
236     res<<=shl;
237     return res;
238 }
239
240
241 template<class T2> void SetData(T2* Data)
242 {
243     for(int i=0;i<m_size;i++)
244         m_data[i]=Data[i];
245 }
246
247 template<class T2> void GetData(T2* Data)
248 {
249     for(int i=0;i<m_size;i++)
250         Data[i]=(T2)m_data[i].value;
251 }
252
253     void SetConst(const cmplx<nmint<T>> val)
254 {
255     for (int idx=0;idx<m_size;idx++)
256         m_data[idx]=val;
257 }
258 }
259 */
260
261 template <class T> void InitInc2(nmcvec<T>& A,cmplx<nmint<T>> StartValue=0,cmplx<nmint<T>> Increment=1)
262 {
263 for (int idx=0;idx<A.m_size;idx++,StartValue+=Increment)
264 A[idx]=StartValue;
265 }
266 */
267 //bool operator== (const nmcvec<T>& vec) const
268 //bool operator!= (const nmcvec<T>& vec) const;
269
270 #ifdef __NM__
271 #else
272 typedef nmcvec<char> nmvec8sc;
273 typedef nmcvec<short> nmvec16sc;
274 #endif
275 typedef nmcvec<int> nmvec32sc;
276 typedef nmcvec<long long> nmvec64sc;
277
278 #endif
279
280 #endif

```

7.91 tnmint.h

```

1 //-----
2 //
3 // $Workfile:: tnmint. $
4 //
5 // Векторно-матричная библиотека
6 //
7 // Copyright (c) RC Module Inc.
8 //
9 // $Revision: 1.2 $      $Date: 2005/01/21 19:22:37 $
10 //
11 //-----
12
13 #ifndef _TNMINT_H_INCLUDED_
14 #define _TNMINT_H_INCLUDED_
15
16 #ifdef __cplusplus
17 #include "tnmvecpack.h"
18
19
20
21
22 #ifndef _TNMINT_C_
23 #define _TNMINT_C_
24
25 /*
26 #define CHECK_OVERFLOW_ON() _putenv("nmpssCheckOverflow_=On")
27 #define CHECK_OVERFLOW_OFF() _putenv("nmpssCheckOverflow_=")
28 #define ENABLE_CHECK_OVERFLOW() _putenv("nmpssCheckOverflow_=On")
29 #define DISABLE_CHECK_OVERFLOW() _putenv("nmpssCheckOverflow_=")
30 */
31
32 /*
33 #define CHECK_OVERFLOW_ON() _putenv("nmpssCheckOverflow_=On")
34 #define CHECK_OVERFLOW_OFF() _putenv("nmpssCheckOverflow_=")
35 #define ENABLE_CHECK_OVERFLOW() _putenv("nmpssCheckOverflow_=On")
36 #define DISABLE_CHECK_OVERFLOW() _putenv("nmpssCheckOverflow_=")
37 */
38 */
39 extern int nmpssCheckOverflow_;
40 #define CHECK_OVERFLOW_ON() nmpssCheckOverflow_=1
41 #define CHECK_OVERFLOW_OFF() nmpssCheckOverflow_=0

```

```

42 #define ENABLE_CHECK_OVERFLOW() nmpssCheckOverflow_=1
43 #define DISABLE_CHECK_OVERFLOW() nmpssCheckOverflow_=0
44
45 #define CHECK_OVERFLOW() (nmpssCheckOverflow_==1) //Getting check overflow status
46 */
47
48 //extern int nmpssCheckOverflow_;
49 #define CHECK_OVERFLOW_ON()
50 #define CHECK_OVERFLOW_OFF()
51 #define ENABLE_CHECK_OVERFLOW()
52 #define DISABLE_CHECK_OVERFLOW()
53
54 #define CHECK_OVERFLOW() 0
55
56
57 //-----
58 //-----
59
60 //-----
61
62
63 template <class T>class nmint
64 {
65
66 public:
67     T m_value; // m_value
68     T *m_pvalue;
69     int m_disp;
70     //----- 0 operands constructors -----
71     nmint(void):m_value(0){}
72     //----- 1 operands constructors -----
73
74     nmint(const T val)
75     {
76         m_value=val;
77     }
78
79
80     nmint<T>& operator= (const T& val)
81     {
82         m_value=val;
83         return (*this);
84     }
85
86     nmint<T>& operator= (T& val)
87     {
88         m_value=val;
89         return (*this);
90     }
91
92 /*
93 nmint<T>& operator= (const nmint<T>& val)
94 {
95     m_value=val.m_value;
96     return (*this);
97 }
98
99 template <class T2> nmint<T>& operator= (const T2& value)
100 {
101     m_value=value;
102     return (*this);
103 }
104 */
105
106     nmint<T>& operator-= (const nmint<T>& val)
107     {
108         /*
109 if (CHECK_OVERFLOW())
110 {
111     __int64 tmp=m_value;
112     tmp-= val.m_value;
113     (tmp<=(64-N))>=(64-N);
114     if(tmp+val.m_value!=m_value)
115         OverflowMessage("Subtraction Overflow ( Res=A-B )",m_value,val.m_value,tmp,N,N,N);
116     m_value = tmp;
117     return (*this);
118 }
119 else
120 */
121     m_value-=val.m_value;
122     // (m_value<=(64-N))>=(64-N);
123     return (*this);
124     //}
125 }
126
127
128     nmint<T>& operator+= (const nmint<T>& val)
129     {
130 //     if (CHECK_OVERFLOW())

```

```

131 // {
132 //     __int64 tmp=m_value;
133 //     tmp += val.m_value;
134 //     (tmp<=(64-N))>=(64-N);
135 //     if(tmp-val.m_value!=m_value)
136 //         OverflowMessage("Summation Overflow ( Res=A+B )",m_value,val.m_value,tmp,N,N,N);
137 //     m_value = tmp;
138 //     return (*this);
139 // }
140 // else
141 // {
142 //     m_value+=val.m_value;
143 //     (m_value<=(64-N))>=(64-N);
144 //     return (*this);
145 // }
146
147     m_value+=val.m_value;
148     return (*this);
149 }
150
151 nmint<T> operator- (const nmint<T>& val) const
152 {
153     nmint<T> Res(*this);
154     Res-=val;
155     return Res;
156 }
157
158 nmint<T> operator+ (const nmint<T>& val) const
159 {
160     nmint<T> Res(*this);
161     Res+=val;
162     return Res;
163 }
164
165 void operator++ (int)
166 {
167     (*this)+=1;
168 }
169
170 void operator-- (int)
171 {
172     (*this)-=1;
173 }
174
175
176
177
178 nmint<T>& operator*= (const nmint<T>& val)
179 {
180     m_value*=val.m_value;
181     return (*this);
182 }
183
184 template <class T2> nmint<T2> operator* (const nmint<T2>& val) const
185 {
186     nmint<T2> Res;
187     Res.m_value=m_value*val.m_value;
188     return Res;
189 }
190
191
192 inline nmint<T>& operator/= (const nmint<T>& val)
193 {
194     m_value/=val.m_value;
195 }
196
197 inline nmint<T> operator/ (const nmint<T>& val) const
198 {
199     nmint<T> Res(*this);
200     Res/=val;
201     return Res;
202 }
203
204
205 inline nmint<T>& operator»= (const int y)
206 {
207     m_value»=y;
208     return (*this);
209 }
210
211 inline nmint<T> operator» (const int y) const
212 {
213     nmint<T> z(*this);
214     z»=y;
215     return z;
216 }
217

```

```

218     inline nmint<T>& operator<= (const int y)
219     {
220         m_value<=y;
221         return (*this);
222     }
223
224     inline nmint<T> operator< (const int n) const
225     {
226         nmint<T> Res(*this);
227         Res<=n;
228         return Res;
229     }
230
231     inline nmint<T>& operator^= (nmint<T> &val)
232     {
233         m_value^=val.m_value;
234         return (*this);
235     }
236     inline nmint<T> operator^ (nmint<T> &val) const
237     {
238         nmint<T> Res(*this);
239         Res^=val;
240         return Res;
241     }
242
243     nmint<T> operator- () const
244     {
245         nmint<T> Res;
246         Res.m_value=-m_value;
247         return Res;
248     }
249
250     bool operator> (const nmint<T>& y) const
251     { return m_value>y.m_value; }
252     bool operator>= (const nmint<T>& y) const
253     { return m_value>=y.m_value; }
254     bool operator< (const nmint<T>& y) const
255     { return m_value<y.m_value; }
256     bool operator<= (const nmint<T>& y) const
257     { return m_value<=y.m_value; }
258     bool operator== (const nmint<T>& y) const
259     { return m_value==y.m_value; }
260     bool operator!= (const nmint<T>& y) const
261     { return m_value!=y.m_value; }
262 };
263
264
265
266 /*
267
268 template <int N> void Round(double& X,nmint<T> &Y)
269 {
270     Y=X+0.5;
271 }
272
273
274
275 // Conversion of to different types
276 template<int N> inline double double_(const nmint<T> &x)
277 {
278     return double(x.m_value);
279 }
280
281
282 template<int N> inline int int_(const nmint<T> &x)
283 {
284     return int(x.m_value);
285 }
286 template<int N> inline short int shortint_(const nmint<T> &x)
287 {
288     return short int(x.m_value);
289 }
290 */
291
292 #ifdef __NM__
293 #else
294     typedef nmint<char> nmint8s;
295     typedef nmint<short> nmint16s;
296 #endif
297
298     typedef nmint<int> nmint32s;
299     typedef nmint<long long> nmint64s;
300
301 #endif
302
303 #endif

```

7.92 tnmmtr.h

```

1 // Header file of template of class nmmt
2 //////////////////////////////////////////////////////////////////
3 //////////////////////////////////////////////////////////////////
4 //////////////////////////////////////////////////////////////////
5 #ifndef _TNMMTR_H_INCLUDED_
6 #define _TNMMTR_H_INCLUDED_
7
8 //////////////////////////////////////////////////////////////////
9 //////////////////////////////////////////////////////////////////
10
11
12 #include "tnmvec.h"
13 #include <string.h>
14
15 //////////////////////////////////////////////////////////////////
16 //////////////////////////////////////////////////////////////////
17 //////////////////////////////////////////////////////////////////
18 //////////////////////////////////////////////////////////////////
19
20
21 //-----
22 //-----
23 //-----
24 //-----
25 //-----
26
27 template<class T> class nmmt
28 {
29 protected:
30     T* m_container;
31     // bool m_flag_new;
32 public:
33     int m_height,m_width,m_size,m_stride,m_border;
34     T* m_data;
35
36     nmmt(int Height, int Width,int Border=0):m_height(Height), m_width(Width)//,m_flag_new(true)
37     {
38         m_border=Border;
39         m_container=new T[(m_height+2*m_border)*m_width];
40         m_data=m_container+m_width*m_border;
41         //nmppsMalloc_64(&m_container,(m_height+2*m_border)*m_width);
42         //m_data=nmppsAddr_7m_container,m_border*m_width);
43         m_size=m_width*m_height;
44         m_stride=m_width;
45         ASSERTE(m_data!=NULL);
46     }
47
48 //////////////////////////////////////////////////////////////////
49 //////////////////////////////////////////////////////////////////
50 //////////////////////////////////////////////////////////////////
51 //////////////////////////////////////////////////////////////////
52 //////////////////////////////////////////////////////////////////
53 //////////////////////////////////////////////////////////////////
54 //////////////////////////////////////////////////////////////////
55 //////////////////////////////////////////////////////////////////
56 //////////////////////////////////////////////////////////////////
57
58 //////////////////////////////////////////////////////////////////
59 //////////////////////////////////////////////////////////////////
60 //////////////////////////////////////////////////////////////////
61 //////////////////////////////////////////////////////////////////
62 //////////////////////////////////////////////////////////////////
63 //////////////////////////////////////////////////////////////////
64 //////////////////////////////////////////////////////////////////
65 //////////////////////////////////////////////////////////////////
66 //////////////////////////////////////////////////////////////////
67     inline nmvec<T> operator [] (int y) const
68 {
69         ASSERTE(y>=-m_border);
70         ASSERTE(y<m_height+m_border);
71         return nmvec<T>(m_data+y*m_stride);
72     }
73 //////////////////////////////////////////////////////////////////
74
75     //nmmt(const nmmt<T>& mtr)
76     nmmt( nmmt<T>& mtr)
77     {
78         m_height=mtr.m_height;
79         m_width =mtr.m_width;
80         m_border=mtr.m_border;
81         m_size =mtr.m_size;
82         m_container=new T[(m_height+2*m_border)*m_width];
83         m_data=m_container+m_width*m_border;
84
85         //m_flag_new=true;
86         m_stride=m_width;
87         for(int y=0; y<m_height; y++)
88             memcpy(m_data+y*m_stride,mtr.m_data+y*mtr.m_stride,m_width*sizeof(T));
89
90     };
91     nmmt(T* Data,int Height,int Width,int Stride=0):m_height(Height), m_width(Width),
92     m_data(Data),m_stride(Stride),m_border(0) //m_flag_new(FALSE),
93     {
94         if (m_stride==0)
95             m_stride=m_width;
96         m_size =m_height*m_width;
97         m_container=0;
98     };

```

```

98
99  nmmtr(const T* Data,int Height,int Width,int Stride=0):m_height(Height), m_width(Width),
100 {m_data((T*)Data),m_stride(Stride),m_border(0) //m_flag_new(false),
101   if (m_stride==0)
102     m_stride=m_width;
103   m_size =m_height*m_width;
104   m_container=0;
105 };
106
107 ~nmmtr()
108 {
109   if (m_container)
110     delete m_container;
111 }
112
113 //nmmtr<T>& operator= (const nmmtr<T>& mtr)
114 nmmtr<T>& operator= ( const nmmtr<T>& mtr)
115 {
116   ASSERTE(mtr.m_height==m_height);
117   ASSERTE(mtr.m_width ==m_width);
118   for(int y=0; y<m_height; y++)
119     (*this)[y]=mtr[y];
120   return *this;
121 }
122
123 nmmtr<T>& operator*= (const nmint<T>& val)
124 {
125   T* pData=m_data;
126   for(int y=0; y< m_height; y++,pData+=m_stride)
127     for(int x=0; x<m_width; x++)
128       pData[x]*=val.m_value;
129   return *this;
130 }
131
132 template <class T2> nmmtr<T2> operator* (const nmint<T2>& val)
133 {
134   nmmtr<T2> mRes(m_height,m_width);
135   T* pResData=mRes.m_data;
136   T* pData =m_data;
137   for(int y=0; y< m_height; y++,pResData+=m_stride,pData+=m_stride)
138     for(int x=0; x<m_width; x++)
139       pResData[x]=pData[x]*val.m_value;
140   return mRes;
141 }
142
143 template <class T2> nmvec<T2> operator* (const nmvec<T2>& vec)
144 {
145   ASSERTE(m_width==vec.m_size);
146   nmvec<T2> vRes(m_height);
147   T* pData =m_data;
148   for(int y=0;y<m_height;y++,pData+=m_stride)
149   {
150     T2 nSum(0);
151     for(int x=0; x<m_width; x++)
152       nSum+=pData[x]*vec.m_data[x];
153     vRes[y]=nSum;
154   }
155   return vRes;
156 }
157
158 template <class T2> nmmtr<T2> operator* (const nmmtr<T2>& mtr)
159 {
160   ASSERTE(m_width==mtr.m_height);
161   nmmtr<T2> mRes(m_height,mtr.m_width);
162   for(int y=0; y<mRes.m_height; y++)
163     for(int x=0; x<mRes.m_width; x++)
164     {
165       T2 sum(0);
166       for(int i=0;i<m_width;i++)
167         sum+=m_data[m_stride*y+i]*mtr.m_data[mtr.m_stride*i+x];
168       mRes.m_data[y*mRes.m_stride+x]=sum;
169     }
170   return mRes;
171 }
172
173 nmvec<T>& operator+=(const nmint<T> &val)
174 {
175   T* pData =m_data;
176   for(int y=0; y< m_height; y++,pData+=m_stride)
177     for(int x=0; x<m_width; x++)
178       pData[x]+=val.m_value;
179   return (*this);
180 }
181
182 nmmtr<T>& operator+=(const nmmtr<T> &mtr)
183 {

```

```

184     ASERTE (mtr.m_size==m_size);
185
186     T* pParData =mtr.m_data;
187     T* pData =m_data;
188     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
189         for(int x=0; x<m_width; x++)
190             pData[x]+=pParData[x];
191
192     return (*this);
193 }
194
195 nmmtr<T> operator+ (const nmmtr<T>& mtr) const
196 {
197     ASERTE (mtr.m_size==m_size);
198     nmmtr<T> res(*this);
199     res+=mtr;
200     return res;
201 }
202
203 nmmtr<T>& operator-= (const nmmtr<T> &mtr)
204 {
205     ASERTE (mtr.m_size==m_size);
206     T* pParData =mtr.m_data;
207     T* pData =m_data;
208     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
209         for(int x=0; x<m_width; x++)
210             pData[x]-=pParData[x];
211     return (*this);
212 }
213
214 nmmtr<T> operator- (const nmmtr<T>& mtr) const
215 {
216     ASERTE (mtr.m_size==m_size);
217     nmmtr <T> res(*this);
218     res-=mtr;
219     return res;
220 }
221
222 nmmtr<T> operator- () const
223 {
224     nmmtr<T> mRes(m_height,m_width);
225     T* pResData =mRes.m_data;
226     T* pData =m_data;
227     for(int y=0; y<m_height; y++, pResData+=mRes.m_stride, pData+=m_stride)
228         for(int x=0; x<m_width; x++)
229             pResData[x]=-pData[x];
230     return mRes;
231 }
232
233 nmmtr<T>& operator/=(const T val)
234 {
235     //ASERTE(val.m_value!=0);
236     T* pData =m_data;
237     for(int y=0; y<m_height; y++, pData+=m_stride)
238         for(int x=0; x<m_width; x++)
239             pData[x]/=val;
240
241     return (*this);
242 }
243
244 nmmtr<T> operator/ (const nmint<T> val) const
245 {
246     nmmtr<T> mRes(*this);
247     mRes/=val.m_value;
248     return mRes;
249 }
250
251 nmmtr<T>& operator>=(const int shr)
252 {
253     ASERTE(shr>=0);
254     if (shr==0)
255         return (*this);
256     T* pData =m_data;
257     for(int y=0; y<m_height; y++, pData+=m_stride)
258         for(int x=0; x<m_width; x++)
259             pData[x]>=shr;
260     return (*this);
261 }
262
263 nmmtr<T> operator>> (const int shr) const
264 {
265     ASERTE(shr>=0);
266     nmmtr<T> mRes(*this);
267     mRes>>=shr;
268     return mRes;
269 }
270

```

```

271 nmmtr<T>& operator<=(const int shl)
272 {
273     ASERTE(shl>=0);
274     if (shl==0)
275         return(*this);
276     T* pData =m_data;
277     for(int y=0; y<m_height; y++, pData+=m_stride)
278         for(int x=0; x<m_width; x++)
279             pData[x]<=shl;
280     return (*this);
281 }
282
283 nmmtr<T> operator< (const int shl) const
284 {
285     ASERTE(shl>=0);
286     nmmtr<T> mRes(*this);
287     mRes<=shl;
288     return mRes;
289 }
290
291 nmmtr<T>& operator|= (const nmmtr<T> &mtr)
292 {
293     ASERTE (mtr.m_size==m_size);
294     T* pParData =mtr.m_data;
295     T* pData =m_data;
296     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
297         for(int x=0; x<m_width; x++)
298             pData[x]|=pParData[x];
299     return (*this);
300 }
301
302 nmmtr<T> operator| (const nmmtr<T>& mtr) const
303 {
304     ASERTE (mtr.m_size==m_size);
305     nmmtr <T> res(*this);
306     res|=mtr;
307     return res;
308 }
309
310 nmmtr<T>& operator&= (const nmmtr<T> &mtr)
311 {
312     ASERTE (mtr.m_size==m_size);
313     T* pParData =mtr.m_data;
314     T* pData =m_data;
315     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
316         for(int x=0; x<m_width; x++)
317             pData[x]&=pParData[x];
318     return (*this);
319 }
320
321 nmmtr<T> operator& (const nmmtr<T>& mtr) const
322 {
323     ASERTE (mtr.m_size==m_size);
324     nmmtr<T> res(*this);
325     res&=mtr;
326     return res;
327 }
328
329 nmmtr<T>& operator^= (const nmint<T> &val)
330 {
331     T* pData =m_data;
332     for(int y=0; y<m_height; y++,pData+=m_stride)
333         for(int x=0; x<m_width; x++)
334             pData[x]^=val.m_value;
335     return (*this);
336 }
337
338 nmmtr<T> operator^ (const nmint<T>& val) const
339 {
340     nmmtr<T> res(*this);
341     res&=val;
342     return res;
343 }
344
345 nmmtr<T> operator~ () const
346 {
347     nmmtr<T> res(m_height,m_width);
348     T* pResData =res.m_data;
349     T* pData =m_data;
350     for(int y=0; y<m_height; y++,pResData+=res.m_stride,pData+=m_stride)
351         for(int x=0; x<m_width; x++)
352             pResData[x]=~pData[x];
353     return res;
354 }
355
356
357

```

```

358     inline T* addr(int y, int x)
359     {
360         return m_data+y*m_stride+x;
361     }
362
363
364     inline nmvec<T> vec(int y)
365     {
366         return nmvec<T>(m_data+y*m_stride,m_width,m_border);
367     }
368
369 void fill(nmint<T> &nVal)
370 {
371     T* pData =m_data;
372     for(int y=0; y< m_height; y++, pData+=m_stride)
373         for(int x=0; x<m_width; x++)
374             pData[x]=nVal.m_value;
375 }
376
377     nmmt<T> transpose()
378 {
379     nmmt<T> Z(m_width,m_height);
380     for(int y=0;y<m_height;y++)
381         for(int x=0;x<m_width;x++)
382             Z[x][y]=(*this)[y][x];
383     return Z;
384 }
385
386     template<class T2> void set(nmmt<T2>& mSrcMtr) const
387 {
388     T* pSrcData=mSrcMtr.m_data;
389     T* pDstData=m_data;
390     for(int y=0; y< m_height; y++, pDstData+=m_stride, pSrcData+=mSrcMtr.m_stride)
391         for(int x=0; x<m_width; x++)
392             *pDstData[x]=pSrcData[x];
393 }
394
395     template<class T2> void set(mtr<T2> &Mtr)
396 {
397     T* pData =m_data;
398     for(int y=0; y< m_height; y++, pData+=m_stride)
399         for(int x=0; x<m_width; x++)
400             pData[x]=Mtr[y][x];
401 }
402
403     void set(const T val)
404 {
405     for(int y=0; y< m_height; y++)
406         vec(y).set(val);
407 }
408
409     void reset()
410 {
411     if (m_width==m_stride)
412     {
413         if (m_container)
414             memset(m_container,0,m_stride*(m_height+m_border*2)*sizeof(T));
415         else
416             memset(m_data,0,m_stride*m_height*sizeof(T));
417     }
418     else
419     {
420         T* p=m_data;
421         for(int y=0;y<m_width;y++,p+=m_stride)
422             memset(p,0,m_width*sizeof(T));
423     }
424 }
425
426 };
427
428 #ifndef __NM__
429 #else
430     typedef nmmt<char> nmmt8s;
431     typedef nmmt<short> nmmt16s;
432     typedef nmmt<unsigned char> nmmt8u;
433     typedef nmmt<unsigned short> nmmt16u;
434 #endif
435
436     typedef nmmt<int> nmmt32s;
437     typedef nmmt<long long> nmmt64s;
438     typedef nmmt<unsigned int> nmmt32u;
439     typedef nmmt<unsigned long long> nmmt64u;
440
441 #ifndef __NM__
442 /*
443     template <class T> inline ostream& operator<< (ostream& s, nmmt<T>& mtr)
444 {

```

```

445 s <"{\n";
446 for(int y=0;y<mtr.m_height-1;y++)
447 {
448 s < "\t{ ";
449 for(int x=0;x<mtr.m_width-1;x++)
450 {
451 s< (mtr[y][x]) < ", ";
452 }
453 s < (mtr[y][mtr.m_width-1]) < " },\n";
454 }
455
456 s < "\t{ ";
457 for(int x=0;x<mtr.m_width-1;x++)
458 {
459 s< (mtr[mtr.m_height-1][x]) < ", ";
460 }
461 s < (mtr[mtr.m_height-1][mtr.m_width-1]) < " }\n";
462 s < "};\n";
463 return s;
464 }
465
466
467
468 inline ostream& AsmArray (ostream& s, nmmtr64s& mtr)
469 {
470 s < " long[" < dec < mtr.m_width < "*" < mtr.m_height < "]=(\n";
471 for(int y=0;y<mtr.m_height-1;y++)
472 {
473 for(int x=0;x<mtr.m_width;x++)
474 {
475
476 int hi,lo;
477 hi=int(mtr[y][x].m_value>>32);
478 lo=int(mtr[y][x].m_value);
479 s < "0"
480 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < hi < " "
481 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < lo < "h1,";
482 }
483 s < "\n";
484 }
485
486 for(int x=0;x<mtr.m_width-1;x++)
487 {
488 int hi,lo;
489 hi=int(mtr[mtr.m_height-1][x].m_value>>32);
490 lo=int(mtr[mtr.m_height-1][x].m_value);
491 s < "0"
492 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < hi < " "
493 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < lo < "h1,";
494 }
495
496 {
497 int hi,lo;
498 hi=int(mtr[mtr.m_height-1][mtr.m_width-1].m_value>>32);
499 lo=int(mtr[mtr.m_height-1][mtr.m_width-1].m_value);
500 s < "0"
501 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < hi < " "
502 < hex < setw(8) < setfill('0') < setiosflags(ios::uppercase) < lo < "h1,";
503 }
504 s < ")\n;\n";
505
506 return s;
507 }
508
509 inline ostream& AsmArray (ostream& s, nmmtr8s& mtr)
510 {
511 ASSERTE(mtr.m_width%8==0);
512 nmmtr64s mClone((__int64*)mtr.m_data,mtr.m_height,mtr.m_width/8,mtr.m_stride/8);
513 AsmArray(s,mClone);
514 return s;
515 }
516 */
517 template<class T1, class T2> void DotMul(nmmtr<T1>& mSrcMtr1, nmmtr<T2>& mSrcMtr2, nmmtr<T2>& mDstMtr
518 )
519 {
520     for(int y=0; y< mSrcMtr1.m_height; y++)
521         for(int x=0; x<mSrcMtr1.m_width; x++)
522         {
523             nmint<T2> res=mSrcMtr1[y][x]*mSrcMtr2[y][x];
524             mDstMtr[y][x]=res;
525         }
526     }
527 template<class T1, class T2> void GetSum(nmmtr<T1>& mSrcMtr1, nmint<T2>& nResSum)
528 {
529     nResSum=0;
530     for(int y=0; y< mSrcMtr1.m_height; y++)

```

```

531     for(int x=0; x<mSrcMtr1.m_width; x++)
532         nResSum+=nmint<T2>(mSrcMtr1[y][x].m_value);
533     }
534
535 #endif
536
537 #endif
538
539
540
541 // template<class T2> void SetData(T2* Data)
542 // {
543 //     for(int i=0;i<m_size;i++)
544 //         m_data[i]=Data[i];
545 // }
546 //
547 // template<class T2> void GetData(T2* Data)
548 // {
549 //     for(int i=0;i<m_size;i++)
550 //         Data[i]=(T2)m_data[i].value;
551 // }
552
553 // inline nmvec<T> GetVec(int y)
554 // {
555 //     ASSERTE(y>=0);
556 //     ASSERTE(y<m_height);
557 //     return nmvec<T>(m_data+y*m_stride,m_width);
558 // }
559 //
560 // inline nmvec<T> GetVec(int y,int x,int len)
561 // {
562 //     ASSERTE(y>=0);
563 //     ASSERTE(y<m_height);
564 //     return nmvec<T>(m_data+y*m_stride+x,len);
565 // }
566 //
567 // inline nmmtr<T>& SetMtr(int y,int x,nmmtr<T>& mSrc)
568 // {
569 //     T* dst=m_data+y*m_width+x;
570 //     T* src=mSrc.m_data;
571 //     for(int i=0;i<mSrc.m_height;i++)
572 //     {
573 //         memcpy(dst,src,mSrc.m_stride*sizeof(T));
574 //         dst+=m_stride;
575 //         src+=mSrc.m_stride;
576 //     }
577 //     return *this;
578 // }
579 //
580 // inline nmmtr<T>& GetMtr(int y,int x,nmmtr<T>& mRes)
581 // {
582 //     T* src=m_data+y*m_stride+x;
583 //     T* dst=mRes.m_data;
584 //     for(int i=0;i<mRes.m_height;i++)
585 //     {
586 //         memcpy(dst,src,mRes.m_stride*sizeof(T));
587 //         dst+=mRes.m_stride;
588 //         src+=m_stride;
589 //     }
590 //     return mRes;
591 // }
592 //
593 // inline nmmtr<T> GetMtr(int y,int x,int height,int width)
594 // {
595 //     nmmtr<T> Res(height,width);
596 //     GetMtr(y,x,Res);
597 //     return Res;
598 // }
599
600 // template<class T2> nmmtr<T2>& DotMul(const nmmtr<T2>& mMtr,nmmtr<T2>& mRes) const
601 // {
602 //     T2* pMtrData =mMtr.m_data;
603 //     T2* pResData =mRes.m_data;
604 //     T* pData =m_data;
605 //     for(int y=0; y<m_height; y++, pResData+=mRes.m_stride, pMtrData+=mMtr.m_stride,pData +=m_stride)
606 //         for(int x=0; x<m_width; x++)
607 //             pResData[x]=pData[x]*pMtrData[x];
608 //     return mRes;
609 // }
610 //
611 // template<class T2> nmmtr<T2> DotMul(const nmmtr<T2>& Mtr) const
612 // {
613 //     nmmtr<T2> Res;
614 //     DotMul(Mtr,Res);
615 //     return Res;
616 // }
617 //

```

```

618 // template<class T2> void GetSum(nmint<T2>& nRes) const
619 // {
620 //     nRes=0;
621 //     T* pData = m_data;
622 //     for(int y=0; y< m_height; y++, pData+=m_stride)
623 //         for(int x=0; x<m_width; x++)
624 //             nRes+=pData[x];
625 //
626 //     return nRes;
627 // }
628
629 // template<class T2> nmmtr<T>& DotMul(nmmtr<T2>& mMtr1, nmmtr<T>& mMtr2)
630 // {
631 //     T2* pMtr1 = mMtr1.m_data;
632 //     T* pMtr2 = mMtr2.m_data;
633 //     T* pMtrRes=m_data;
634 //     for(int y=0; y< m_height; y++, pMtr1+=mMtr1.m_stride, pMtr2+=mMtr2.m_stride,pMtrRes+=m_stride)
635 //         for(int x=0; x<m_width; x++)
636 //             pMtrRes[x]=pMtr1[x]*pMtr2[x];
637 //     return (*this);
638 // }
639
640 // template<class T2> nmmtr<T2>& Convert(nmmtr<T2>& mRes) const
641 // {
642 //     T* pData = m_data;
643 //     T* pResData=mRes.m_data;
644 //     for(int y=0; y< m_height; y++, pData+=m_stride, pResData+=mRes.m_stride)
645 //         for(int x=0; x<m_width; x++)
646 //             pResData[x]=pData[x];
647 //     return mRes;
648 // }
649
650
651 //endif

```

7.93 tnmmtrpack.h

```

1
2 // Header file of template of class nmmtrpack          //          //
3 //-----//                                         //-----//
4 //-----//                                         //-----//
5 #ifndef _Tnmmtrpack_H_INCLUDED_
6 #define _Tnmmtrpack_H_INCLUDED_
7
8
9
10
11
12 #include "tnmvec.h"
13 #include <string.h>
14
15 //-----//
16 //-----//
17 //-----// Class of nmmtrpacks
18 //-----//
19
20 //-----//
21 //-----//
22 //-----//
23 //-----//
24 //-----//
25 //-----//
26
27 template<class T> class nmmtrpack
28 {
29 protected:
30     T* m_container;
31     bool m_flag_new;
32 public:
33     int m_height,m_width,m_size,m_stride,m_border;
34     T* m_data;
35
36     nmmtrpack(int Height, int Width,int Border=0):m_height(Height), m_width(Width),m_flag_new(true)
37     {
38         m_border=Border;
39         //m_container=new T[(m_height+2*m_border)*m_width];
40         //m_data=m_container+m_width*m_border;
41         nmppsMalloc_64s(&m_container,(m_height+2*m_border)*m_width);
42         m_data=nmppsAddr_(m_container,m_border*m_width);
43         m_size=m_width*m_height;
44         m_stride=m_width;
45         _ASSERT(m_data!=NULL);
46     }
47
48 //##ifndef NMTL_DISABLE_INDEX_CHECK
49 //    _INLINE__ nmint<T>* operator[](int y)
50 //    {
51
52
53
54
55
56
57
58 //##endif
59 //    _INLINE__ nmint<T>* operator[](int y)
60 //    {

```

```

61 //      _ASERTE(y>=-m_border);
62 //      _ASERTE(y<m_height+m_border);
63 //
64 //      return (nmint<T>)(m_data+y*m_stride);
65 //
66 //#else
67     -- INLINE __ nmvecpack<T> operator [](int y) const
68 {
69     _ASERTE(y>=-m_border);
70     _ASERTE(y<m_height+m_border);
71     return nmvecpack<T>(nmppsAddr_(m_data,y*m_stride),m_width);
72 }
73 //#endif
74
75     nmmtPack(const nmmtPack<T>& mtr)
76 {
77     m_height=mtr.m_height;
78     m_width=mtr.m_width;
79     m_border=mtr.m_border;
80     m_size=mtr.m_size;
81     //m_container=new T[(m_height+2*m_border)*m_width];
82     //m_data=m_container+m_width*m_border;
83     nmppsMalloc_64s(&m_container,(m_height+2*m_border)*m_width);
84     m_data=nmppsAddr_(m_container,m_border*m_width);
85
86     m_flag_new=true;
87     m_stride=m_width;
88     for(int y=0; y<m_height; y++)
89         //memcpy(m_data+y*m_stride,mtr.m_data+y*mtr.m_stride,m_width*sizeof(T));
90         //nmppsCopy(nmppsAddr_(mtr.m_data),y*mtr.m_stride)
91         (*this)[y]=mtr[y];
92
93 };
94     nmmtPack(T* Data,int Height,int Width,int Stride=0):m_height(Height), m_width(Width),
95     m_data(Data),m_flag_new(false),m_stride(Stride),m_border(0)
96 {
97     if (m_stride==0)
98         m_stride=m_width;
99     m_size=m_height*m_width;
100    m_container=0;
101 };
102     nmmtPack(const T* Data,int Height,int Width,int Stride=0):m_height(Height), m_width(Width),
103     m_data((T*)Data),m_flag_new(false),m_stride(Stride),m_border(0)
104 {
105     if (m_stride==0)
106         m_stride=m_width;
107     m_size=m_height*m_width;
108     m_container=0;
109 };
110     ~nmmtPack()
111 {
112     if (m_flag_new)
113         //delete m_container;
114         nmppsFree(m_container);
115 }
116
117     nmmtPack<T>& operator=(const nmmtPack<T>& mtr)
118 {
119     _ASERTE(mtr.m_height==m_height);
120     _ASERTE(mtr.m_width==m_width);
121     for(int y=0; y<m_height; y++)
122         (*this)[y]=mtr[y];
123     return *this;
124 }
125
126     nmmtPack<T>& operator*=(const nmint<T>& val)
127 {
128     T* pData=m_data;
129     for(int y=0; y<m_height; y++,pData+=m_stride)
130         for(int x=0; x<m_width; x++)
131             pData[x]*=val.m_value;
132     return *this;
133 }
134
135     template <class T2> nmmtPack<T2> operator*(const nmint<T2>& val)
136 {
137     nmmtPack<T2> mRes(m_height,m_width);
138     T* pResData=mRes.m_data;
139     T* pData =m_data;
140     for(int y=0; y<m_height; y++,pResData+=m_stride,pData+=m_stride)
141         for(int x=0; x<m_width; x++)
142             pResData[x]=pData[x]*val.m_value;
143     return mRes;
144 }
145

```

```

146 template <class T2> nmvec<T2> operator* (const nmvec<T2>& vec)
147 {
148     _ASSERT(m_width==vec.m_size);
149     nmvec<T2> vRes(m_height);
150     T* pData = m_data;
151     for(int y=0;y<m_height;y++,pData+=m_stride)
152     {
153         T2 nSum(0);
154         for(int x=0; x<m_width; x++)
155             nSum+=pData[x]*vec.m_data[x];
156         vRes[y]=nSum;
157     }
158     return vRes;
159 }
160
161 template <class T2> nmmtrpack<T2> operator* (const nmmtrpack<T2>& mtr)
162 {
163     _ASSERT(m_width==mtr.m_height);
164     nmmtrpack<T2> mRes(m_height,mtr.m_width);
165     for(int y=0; y<mRes.m_height; y++)
166         for(int x=0; x<mRes.m_width; x++)
167         {
168             T2 sum(0);
169             for(int i=0;i<m_width;i++)
170                 sum+=m_data[m_stride*y+i]*mtr.m_data[mtr.m_stride*i+x];
171             mRes.m_data[y*mRes.m_stride+x]=sum;
172         }
173     return mRes;
174 }
175
176 nmvec<T>& operator+=(const nmint<T> &val)
177 {
178     T* pData = m_data;
179     for(int y=0; y<m_height; y++,pData+=m_stride)
180         for(int x=0; x<m_width; x++)
181             pData[x]+=(val.m_value);
182     return (*this);
183 }
184
185 nmmtrpack<T>& operator+=(const nmmtrpack<T> &mtr)
186 {
187     _ASSERT (mtr.m_size==m_size);
188
189     T* pParData =mtr.m_data;
190     T* pData =m_data;
191     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
192         for(int x=0; x<m_width; x++)
193             pData[x]+=(pParData[x]);
194
195     return (*this);
196 }
197
198 nmmtrpack<T> operator+ (const nmmtrpack<T>& mtr) const
199 {
200     _ASSERT (mtr.m_size==m_size);
201     nmmtrpack<T> res(*this);
202     res+=mtr;
203     return res;
204 }
205
206 nmmtrpack<T>& operator-= (const nmmtrpack<T> &mtr)
207 {
208     _ASSERT (mtr.m_size==m_size);
209     T* pParData =mtr.m_data;
210     T* pData =m_data;
211     for(int y=0; y<m_height; y++,pParData+=mtr.m_stride,pData+=m_stride)
212         for(int x=0; x<m_width; x++)
213             pData[x]-=(pParData[x]);
214     return (*this);
215 }
216
217 nmmtrpack<T> operator- (const nmmtrpack<T>& mtr) const
218 {
219     _ASSERT (mtr.m_size==m_size);
220     nmmtrpack<T> res(*this);
221     res-=mtr;
222     return res;
223 }
224
225 nmmtrpack<T> operator- () const
226 {
227     nmmtrpack<T> mRes(m_height,m_width);
228     T* pResData =mRes.m_data;
229     T* pData =m_data;
230     for(int y=0; y<m_height; y++, pResData+=mRes.m_stride, pData+=m_stride)
231         for(int x=0; x<m_width; x++)
232             pResData[x]-=(pData[x]);

```

```

233     return mRes;
234 }
235
236 nmmtrpack<T>& operator/=(const T val)
237 {
238     //__ASSERTE(val.m_value!=0);
239     T* pData =m_data;
240     for(int y=0; y<m_height; y++, pData+=m_stride)
241         for(int x=0; x<m_width; x++)
242             pData[x]/=val;
243
244     return (*this);
245 }
246
247 nmmtrpack<T> operator/ (const nmint<T> val) const
248 {
249     nmmtrpack<T> mRes(*this);
250     mRes/=val.m_value;
251     return mRes;
252 }
253
254 nmmtrpack<T>& operator»=(const int shr)
255 {
256     __ASSERTE(shr>=0);
257     if (shr==0)
258         return(*this);
259     T* pData =m_data;
260     for(int y=0; y<m_height; y++, pData+=m_stride)
261         for(int x=0; x<m_width; x++)
262             pData[x]»=shr;
263     return (*this);
264 }
265
266 nmmtrpack<T> operator»» (const int shr) const
267 {
268     __ASSERTE(shr>=0);
269     nmmtrpack<T> mRes(*this);
270     mRes»»=shr;
271     return mRes;
272 }
273
274 nmmtrpack<T>& operator«=(const int shl)
275 {
276     __ASSERTE(shl>=0);
277     if (shl==0)
278         return(*this);
279     T* pData =m_data;
280     for(int y=0; y<m_height; y++, pData+=m_stride)
281         for(int x=0; x<m_width; x++)
282             pData[x]«=shl;
283     return (*this);
284 }
285
286 nmmtrpack<T> operator«» (const int shl) const
287 {
288     __ASSERTE(shl>=0);
289     nmmtrpack<T> mRes(*this);
290     mRes«»=shl;
291     return mRes;
292 }
293
294 nmmtrpack<T>& operator|= (const nmmtrpack<T> &mtr)
295 {
296     __ASSERTE (mtr.m_size==m_size);
297     T* pParData =mtr.m_data;
298     T* pData =m_data;
299     for(int y=0; y<m_height; y++, pParData+=mtr.m_stride,pData+=m_stride)
300         for(int x=0; x<m_width; x++)
301             pData[x]|=pParData[x];
302     return (*this);
303 }
304
305 nmmtrpack<T> operator| (const nmmtrpack<T>& mtr) const
306 {
307     __ASSERTE (mtr.m_size==m_size);
308     nmmtrpack<T> res(*this);
309     res|=mtr;
310     return res;
311 }
312
313 nmmtrpack<T>& operator&= (const nmmtrpack<T> &mtr)
314 {
315     __ASSERTE (mtr.m_size==m_size);
316     T* pParData =mtr.m_data;
317     T* pData =m_data;
318     for(int y=0; y<m_height; y++, pParData+=mtr.m_stride,pData+=m_stride)
319         for(int x=0; x<m_width; x++)

```

```

320     pData[x]&=pParData[x];
321     return (*this);
322 }
323
324 nmmtrpack<T> operator& (const nmmtrpack<T>& mtr) const
325 {
326     _ASSERTE (mtr.m_size==m_size);
327     nmmtrpack<T> res(*this);
328     res&=mtr;
329     return res;
330 }
331
332 nmmtrpack<T>& operator^= (const nmint<T> &val)
333 {
334     T* pData =m_data;
335     for(int y=0; y< m_height; y++,pData+=m_stride)
336         for(int x=0; x<m_width; x++)
337             pData[x]^=val.m_value;
338     return (*this);
339 }
340
341 nmmtrpack<T> operator^ (const nmint<T>& val) const
342 {
343     nmmtrpack<T> res(*this);
344     res&=val;
345     return res;
346 }
347
348 nmmtrpack<T> operator^- () const
349 {
350     nmmtrpack<T> res(m_height,m_width);
351     T* pResData =res.m_data;
352     T* pData =m_data;
353     for(int y=0; y< m_height; y++,pResData+=res.m_stride,pData+=m_stride)
354         for(int x=0; x<m_width; x++)
355             pResData[x]=~pData[x];
356     return res;
357 }
358
359
360 // template<class T2> void SetData(T2* Data)
361 // {
362 //     for(int i=0;i<m_size;i++)
363 //         m_data[i]=Data[i];
364 // }
365 //
366 // template<class T2> void GetData(T2* Data)
367 // {
368 //     for(int i=0;i<m_size;i++)
369 //         Data[i]=(T2)m_data[i].value;
370 // }
371
372 // - INLINE -- nmvec<T> GetVec(int y)
373 // {
374 //     _ASSERTE(y>=0);
375 //     _ASSERTE(y<m_height);
376 //     return nmvec<T>(m_data+y*m_stride,m_width);
377 // }
378
379 // - INLINE -- nmvec<T> GetVec(int y,int x,int len)
380 // {
381 //     _ASSERTE(y>=0);
382 //     _ASSERTE(y<m_height);
383 //     return nmvec<T>(m_data+y*m_stride+x,len);
384 // }
385
386 // - INLINE -- nmmtrpack<T>& SetMtr(int y,int x,nmmtrpack<T>& mSrc)
387 // {
388     T* dst=m_data+y*m_width+x;
389     T* src=mSrc.m_data;
390     for(int i=0;i<mSrc.m_height;i++)
391     {
392         memcpy(dst,src,mSrc.m_stride*sizeof(T));
393         dst+=m_stride;
394         src+=mSrc.m_stride;
395     }
396     return *this;
397 }
398
399 // - INLINE -- nmmtrpack<T>& GetMtr(int y,int x,nmmtrpack<T>& mRes)
400 // {
401     T* src=m_data+y*m_stride+x;
402     T* dst=mRes.m_data;
403     for(int i=0;i<mRes.m_height;i++)
404     {
405         memcpy(dst,src,mRes.m_stride*sizeof(T));
406         dst+=mRes.m_stride;

```

```

407         src+=m_stride;
408     }
409     return mRes;
410 }
411
412 // -INLINE-- nmmtrpack<T> GetMtr(int y,int x,int height,int width)
413 {
414     nmmtrpack<T> Res(height,width);
415     GetMtr(y,x,Res);
416     return Res;
417 }
418
419 // -INLINE-- T* Addr(int y, int x)
420 {
421     return m_data+y*m_stride+x;
422 }
423
424 // template<class T2> nmmtrpack<T2>& DotMul(const nmmtrpack<T2>& mMtr,nmmtrpack<T2>& mRes) const
425 //
426 {
427     T2* pMtrData =mMtr.m_data;
428     T2* pResData =mRes.m_data;
429     T* pData =m_data;
430     for(int y=0; y<m_height; y++, pResData+=mRes.m_stride, pMtrData+=mMtr.m_stride,pData +=m_stride)
431         for(int x=0; x<m_width; x++)
432             pResData[x]=pData[x]*pMtrData[x];
433     return mRes;
434 }
435 // template<class T2> nmmtrpack<T2> DotMul(const nmmtrpack<T2>& Mtr) const
436 //
437 {
438     nmmtrpack<T2> Res;
439     DotMul(Mtr,Res);
440     return Res;
441 }
442 // template<class T2> void GetSum(nmint<T2>& nRes) const
443 //
444 {
445     nRes=0;
446     T* pData =m_data;
447     for(int y=0; y<m_height; y++, pData+=m_stride)
448         for(int x=0; x<m_width; x++)
449             nRes+=pData[x];
450     return nRes;
451 }
452
453 // template<class T2> nmmtrpack<T>& DotMul(nmmtrpack<T2>& mMtr1, nmmtrpack<T>& mMtr2)
454 //
455 {
456     T2* pMtr1 =mMtr1.m_data;
457     T2* pMtr2 =mMtr2.m_data;
458     T* pMtrRes=m_data;
459     for(int y=0; y<m_height; y++, pMtr1+=mMtr1.m_stride, pMtr2+=mMtr2.m_stride,pMtrRes+=m_stride)
460         for(int x=0; x<m_width; x++)
461             pMtrRes[x]=pMtr1[x]*pMtr2[x];
462     return (*this);
463 }
464 // template<class T2> nmmtrpack<T2>& Convert(nmmtrpack<T2>& mRes) const
465 //
466 {
467     T* pData =m_data;
468     T* pResData=mRes.m_data;
469     for(int y=0; y<m_height; y++, pData+=m_stride, pResData+=mRes.m_stride)
470         for(int x=0; x<m_width; x++)
471             pResData[x]=pData[x];
472     return mRes;
473 }
474 template<class T2> void Set(nmmtrpack<T2>& mSrcMtr) const
475 {
476     T* pSrcData=mSrcMtr.m_data;
477     T* pDstData=m_data;
478     for(int y=0; y<m_height; y++, pDstData+=m_stride, pSrcData+=mSrcMtr.m_stride)
479         for(int x=0; x<m_width; x++)
480             *pDstData[x]=pSrcData[x];
481 }
482
483
484 void InitConst(nmint<T> &nVal)
485 {
486     T* pData =m_data;
487     for(int y=0; y<m_height; y++, pData+=m_stride)
488         for(int x=0; x<m_width; x++)
489             pData[x]=nVal.m_value;;
490 }
491
492 nmmtrpack<T> transpose()
493 {

```

```

494     nmmtrpack<T> Z(m_width,m_height);
495     for(int y=0;y<m_height;y++)
496         for(int x=0;x<m_width;x++)
497             Z[x][y]=(*this)[y][x];
498     return Z;
499 }
500
501
502 // template<class T2> void SetMatrix(mtr<T2> &Mtr)
503 //
504 //     T* pData = m_data;
505 //     for(int y=0; y< m_height; y++, pData+=m_stride)
506 //         for(int x=0; x<m_width; x++)
507 //             pData[x]=Mtr[y][x];
508 //
509
510     void reset()
511 {
512     if (m_width==m_stride)
513     {
514         if (m_container)
515             memset(m_container,0,m_stride*(m_height+m_border*2)*sizeof(T));
516         else
517             memset(m_data,0,m_stride*m_height*sizeof(T));
518     }
519     else
520     {
521         T* p=m_data;
522         for(int y=0;y<m_width;y++,p+=m_stride)
523             memset(p,0,m_width*sizeof(T));
524     }
525 }
526
527 };
528
529 typedef nmmtrpack<nm1> nmmtr1;
530 typedef nmmtrpack<nm2s> nmmtr2s;
531 typedef nmmtrpack<nm2u> nmmtr2u;
532 typedef nmmtrpack<nm4s> nmmtr4s;
533 typedef nmmtrpack<nm4u> nmmtr4u;
534
535 #ifndef WIN32
536 typedef nmmtrpack<nm8s> nmmtr8s;
537 typedef nmmtrpack<nm8u> nmmtr8u;
538 typedef nmmtrpack<nm16s> nmmtr16s;
539 typedef nmmtrpack<nm16u> nmmtr16u;
540 #endif
541
542
543
544
545
546
547 # endif
548

```

7.94 tnmvec.h

```

1 //-
2 //-
3 // $Workfile::: tnmvec. $
4 //-
5 // Векторно-матричная библиотека
6 //-
7 // Copyright (c) RC Module Inc.
8 //-
9 // $Revision: 1.2 $    $Date: 2004/12/21 16:52:31 $
10 //-
19 //-
20 #ifndef _TNMVEC_H_INCLUDED_
21 #define _TNMVEC_H_INCLUDED_
22 #ifdef __cplusplus
23
24 #include <string.h>
25 #include "tnmint.h"
26 //#include "nmpl.h"
27
28
29 //***** ****
30 //***** ****
31 //***** ****
32 //***** ****
33 //***** ****
34 //***** ****
35 //***** ****
36 //***** ****
37 //***** ****
38 //***** ****
39 //***** ****
40 //***** ****
41 //***** ****
42 //***** ****
43 //***** ****
44 //***** ****
45 //***** ****
46 //***** ****
47 //***** ****
48 //***** ****
49
50
51
52 template<class T> class nmvec

```

```

53 {
54     protected:
55     T*     m_container;
56     int    m_border;
57 public:
58     int    size;
59     T*    m_data;
60
61     nmvec(void* Data, int Size, int Border=0):m_data((T*)Data),size(Size),m_container(0),m_border(Border)
62     {
63
64     };
65
66     nmvec(int Size, int Border=0):size(Size),m_border(Border)
67     {
68         m_container=new T[size+2*m_border];
69         m_data=m_container+m_border;
70         //tmalloc(m_container,size+2*m_border);
71 //        nmppsMalloc_64s(&m_container,size+2*m_border);
72 //        m_data=nmppsAddr_(m_container,m_border);
73     }
74
75     nmvec(const nmvec<T> &vec)
76     {
77         size=vec.size;
78         m_border=vec.m_border;
79         m_container=new T[size+2*m_border];
80         m_data=m_container+m_border;
81         memcpy(m_data,vec.m_data,size*sizeof(T));
82 //        nmppsMalloc_64s(&m_container,size+2*m_border);
83 //        m_data=nmppsAddr_(m_container,m_border);
84 //        nmppsCopy_(vec.m_data,m_data,size);
85     };
86
87
88     ~nmvec()
89     {
90         if (m_container)
91             //delete []m_container;
92 //            nmppsFree(m_container);
93         m_container=0;
94
95     }
96
97
98     inline nmvec<T>& operator= (const nmvec<T>& vec) // спец. добавил const - компр ругается
99     {
100         ASERTE(vec.size==size);
101         memcpy(m_data,vec.m_data,size*sizeof(T));
102 //        nmppsCopy_(vec.m_data,m_data,size);
103         return *this;
104     }
105
106     inline nmint<T>& operator[] (int idx)
107     {
108         ASERTE(idx>=-m_border);
109         ASERTE(idx<size+m_border);
110         nmint<T>* res=(nmint<T>*)(m_data+idx);
111         return *res;
112     }
113
114     template <class T2> nmvec<T2>& operator*= (const nmint<T2> val)
115     {
116         for(int idx=0;idx<size;idx++)
117             m_data[idx]*=val.m_value;
118         return (*this);
119     }
120
121
122     template <class T2> nmvec<T2> operator* (const nmint<T2>& val)const
123     {
124         nmvec<T2> res(size);
125         for(int idx=0;idx<size;idx++)
126             res.m_data[idx]=m_data[idx]*val.m_value;
127         return res;
128     }
129
130     template <class T2> nmint<T2> operator* (const nmvec<T2>& vec)const
131     {
132         ASERTE(size==vec.size);
133         nmint<T2> res;
134         for (int idx=0;idx<size;idx++)
135             res.m_value+=m_data[idx]*vec.m_data[idx];
136         return res;
137     }
138
139
140     nmvec<T>& operator+= (const nmint<T> &val)

```

```

142     {
143         for(int idx=0;idx<size;idx++)
144             m_data[idx]+=val.m_value;
145         return (*this);
146     }
147
148     nmvec<T>& operator+= (const nmvec<T> &vec)
149     {
150         ASERTE (vec.size==size);
151         for(int idx=0;idx<size;idx++)
152             m_data[idx]+=vec.m_data[idx];
153         return (*this);
154     }
155
156
157     nmvec<T> operator+ (const nmint<T>& val)const
158     {
159         nmvec<T> res(*this);
160         res+=val;
161         return res;
162     }
163
164     nmvec<T> operator+ (const nmvec<T>& vec)const
165     {
166         ASERTE (vec.size==size);
167         nmvec<T> res(*this);
168         res+=vec;
169         return res;
170     }
171
172
173     nmvec<T>& operator-= (const nmint<T> &val)
174     {
175         for(int idx=0;idx<size;idx++)
176             m_data[idx]-=val.m_value;
177         return (*this);
178     }
179
180
181     nmvec<T>& operator-= (const nmvec<T> &vec)
182     {
183         ASERTE (vec.size==size);
184         for(int idx=0;idx<size;idx++)
185             m_data[idx]-=vec.m_data[idx];
186         return (*this);
187     }
188
189
190     nmvec<T> operator- ()const
191     {
192         nmvec<T> res(*this);
193         for(int idx=0;idx<size;idx++)
194             m_data[idx]=-m_data[idx];
195         return res;
196     }
197
198
199     nmvec<T> operator- (const nmint<T>& val)const
200     {
201         nmvec<T> res(*this);
202         res-=val;
203         return res;
204     }
205
206
207     nmvec<T> operator- (const nmvec<T>& vec)const
208     {
209         ASERTE (vec.size==size);
210         nmvec<T> res(*this);
211         res-=vec;
212         return res;
213     }
214
215
216
217     nmvec<T>& operator/=(const nmint<T> val)
218     {
219         ASERTE(val.m_value!=0);
220         for(int idx=0;idx<size;idx++)
221             m_data[idx]/=val.m_value;
222         return (*this);
223     }
224
225
226     nmvec<T> operator/ (const T val)const
227     {
228         nmvec<T> res(*this);
229         res/=val;
230         return res;
231     }
232

```

```

235     nmvec<T>& operator»=(const int shr)
236     {
237         ASERTE(shr>=0);
238         if (shr==0)
239             return(*this);
240         for(int idx=0;idx<size;idx++)
241             m_data[idx]»=shr;
242         return (*this);
243     }
244
245     nmvec<T> operator» (const int shr) const
246     {
247         ASERTE(shr>=0);
248         nmvec<T> res(*this);
249         res»=shr;
250         return res;
251     }
252
253     nmvec<T>& operator«=(const int shl)
254     {
255         ASERTE(shl>=0);
256         if (shl==0)
257             return(*this);
258         for(int idx=0;idx<size;idx++)
259             m_data[idx]«=shl;
260         return (*this);
261     }
262
263     nmvec<T> operator« (const int shl) const
264     {
265         ASERTE(shl>=0);
266         nmvec<T> res(*this);
267         res«=shl;
268         return res;
269     }
270     nmvec<T>& operator|= (const nmint<T> &val)
271     {
272         for(int idx=0;idx<size;idx++)
273             m_data[idx]=val;
274         return (*this);
275     }
276
277     nmvec<T>& operator|= (const nmvec<T> &vec)
278     {
279         ASERTE (vec.size==size);
280         for(int idx=0;idx<size;idx++)
281             m_data[idx]=vec.m_data[idx];
282         return (*this);
283     }
284
285     nmvec<T> operator| (const nmint<T>& val) const
286     {
287         nmvec <T> res(*this);
288         res|=val;
289         return res;
290     }
291
292     nmvec<T> operator| (const nmvec<T>& vec) const
293     {
294         ASERTE (vec.size==size);
295         nmvec <T> res(*this);
296         res.=vec;
297         return res;
298     }
299
300     nmvec<T>& operator&= (const nmint<T> &val)
301     {
302         for(int idx=0;idx<size;idx++)
303             m_data[idx]&=val;
304         return (*this);
305     }
306
307     nmvec<T>& operator&= (const nmvec<T> &vec)
308     {
309         ASERTE (vec.size==size);
310         for(int idx=0;idx<size;idx++)
311             m_data[idx]&=vec.m_data[idx];
312         return (*this);
313     }
314
315     nmvec<T>& operator&= (const nmint<T>& val) const
316     {
317         nmvec <T> res(*this);
318         res&=val;
319         return res;
320     }
321
322     nmvec<T> operator&= (const nmvec<T>& vec) const
323

```

```

326 {
327     ASERTE (vec.size==size);
328     nmvec <T> res(*this);
329     res&=vec;
330     return res;
331 }
332
334 nmvec<T>& operator^= (const nmint<T> &val)
335 {
336     for(int idx=0;idx<size;idx++)
337         m_data[idx]^=val;
338     return (*this);
339 }
340
341 nmvec<T>& operator^= (const nmvec<T> &vec)
342 {
343     ASERTE (vec.size==size);
344     for(int idx=0;idx<size;idx++)
345         m_data[idx]^=vec.m_data[idx];
346     return (*this);
347 }
348
350 nmvec<T> operator^ (const nmint<T>& val)const
351 {
352     nmvec <T> res(*this);
353     res^=val;
354     return res;
355 }
356
357 nmvec<T> operator^ (const nmvec<T>& vec)const
358 {
359     nmvec <T> res(*this);
360     res^=vec;
361     return res;
362 }
363
364
365 nmvec<T>& operator^- ()const
366 {
367     nmvec<T> res(*this);
368     for(int idx=0;idx<size;idx++)
369         res.m_data[idx]=-m_data[idx];
370     return res;
371 }
372
373
374 bool operator== (const nmvec<T>& vec)const
375 {
376     ASERTE (vec.size==size);
377     for(int idx=0;idx<size;idx++)
378         if (m_data[idx]!=vec.m_data[idx])
379             return false;
380         return true;
381 }
382
383
384 bool operator!= (const nmvec<T>& vec)const
385 {
386     ASERTE (vec.size==size);
387     for(int idx=0;idx<size;idx++)
388         if (m_data[idx]==vec.m_data[idx])
389             return false;
390         return true;
391 }
392
393 // template<class T2> void setsetData(T2* Data)
394 // {
395 //     for(int i=0;i<size;i++)
396 //         m_data[i]=Data[i];
397 // }
398
399 // template<class T2> void GetData(T2* Data)
400 // {
401 //     for(int i=0;i<size;i++)
402 //         Data[i]=(T2)m_data[i].value;
403 // }
404 void set(const T& val)
405 {
406     nmppsSet_(m_data,val,size);
407 }
408
409 void reset()
410 {
411     memset(m_container,0,(size+2*m_border)*sizeof(T));
412 }
413
414 };

```

```

415
416 #ifdef __NM__
417 #else
418 typedef nmvec<char>          nmvec8s;
419 typedef nmvec<unsigned char>   nmvec8u;
420
421 typedef nmvec<short>          nmvec16s;
422 typedef nmvec<unsigned short>  nmvec16u;
423 #endif
424
425 typedef nmvec<int>           nmvec32s;
426 typedef nmvec<long long>      nmvec64s;
427
428 typedef nmvec<unsigned int>   nmvec32u;
429 typedef nmvec<unsigned long long> nmvec64u;
430
431
432 #endif
433 #endif

```

7.95 tnmvecpack.h

```

1 //-
2 //-
3 // $Workfile::: tnmvec. $
4 // Векторно-матричная библиотека
5 // Copyright (c) RC Module Inc.
6 //
7 // $Revision: 1.1 $    $Date: 2004/11/22 13:50:17 $
10 //
19 //-
20 #ifndef _TNMVECPACK_H_INCLUDED_
21 #define _TNMVECPACK_H_INCLUDED_
22
23 #include <string.h>
24 #include "tnmint.h"
25 #include "nmpl.h"
26
27
28 template <class T> class nmintpack
29 {
30 public:
31     T *m_container;
32     int m_disp;
33     nmintpack(T* base,int idx)
34     {
35         m_container=base; //((T*)((int*)(base)+int disp(idx)));
36         m_disp=idx; //bit disp(idx);
37     }
38
39     nmintpack<T>& operator= (const nmintpack<T>& val)
40     {
41         int n=val;
42         nmppsPut_((T*)m_container,m_disp,n);
43         return (*this);
44     }
45
46     nmintpack<T>& operator= (const int& val)
47     {
48         nmppsPut_((T*)m_container,m_disp,val);
49         return (*this);
50     }
51
52     operator int (void) const
53     {
54
55         int n;
56         n=nmppsGetVal_((T*)m_container,m_disp);
57         return n;//get();
58     }
59
60
61     __INLINE__ int intdisp(int indx);
62     __INLINE__ int bitdisp(int indx);
63 };
64
65 __INLINE__ int nmintpack<nm1>::intdisp(int indx)
66 {
67     return indx »5;
68 }
69

```

```

70 - INLINE -- int nmintpack<nm2s>::intdisp(int indx)
71 {
72     return indx >> 4;
73 }
74
75 - INLINE -- int nmintpack<nm4s>::intdisp(int indx)
76 {
77     return indx >> 3;
78 }
79
80 - INLINE -- int nmintpack<nm8s>::intdisp(int indx)
81 {
82     return indx >> 2;
83 }
84
85 - INLINE -- int nmintpack<nm16s>::intdisp(int indx)
86 {
87     return indx >> 1;
88 }
89
90 - INLINE -- int nmintpack<nm1>::bitdisp(int indx)
91 {
92     return indx & 0x1F;
93 }
94 - INLINE -- int nmintpack<nm2s>::bitdisp(int indx)
95 {
96     return indx & 0xF;
97 }
98 - INLINE -- int nmintpack<nm4s>::bitdisp(int indx)
99 {
100    return indx & 0x7;
101 }
102
103 - INLINE -- int nmintpack<nm8s>::bitdisp(int indx)
104 {
105     return indx & 3;
106 }
107
108 - INLINE -- int nmintpack<nm16s>::bitdisp(int indx)
109 {
110     return indx & 2;
111 }
112
113
114 //*****
115 //*****
116
117 template<class T> class nmvecpack
118 {
119     protected:
120     T* m_container;
121     int m_border;
122     public:
123     int m_size;
124     T* m_data;
125
126
127     nmvecpack(void* Data, int Size, int Border=0):m_data((T*)Data),m_size(Size),m_container(0),m_border(Border){};
128
129     nmvecpack(int Size, int Border=0):m_size(Size),m_border(Border)
130     {
131         //m_container=new T[m_size+2*m_border];
132         //m_data=m_container+m_border;
133         //tmalloc(m_container,m_size+2*m_border);
134         nmppsMalloc_64s((T**)&m_container,m_size+2*m_border);
135         nmintpack<T> first(m_container,m_border);
136         m_data=first.m_container;// ERROR here nmppsAddr_((T*)m_container,m_border);
137         reset();
138     }
139
140     nmvecpack(const nmvecpack<T> &vec)
141     {
142         m_size=vec.m_size;
143         m_border=vec.m_border;
144         //m_container=new T[m_size+2*m_border];
145         //m_data=m_container+m_border;
146         //memcp(m_data,vec.m_data,m_size*sizeof(T));
147         nmppsMalloc_64s(&m_container,m_size+2*m_border);
148         m_data=nmppsAddr_(m_container,m_border);
149         nmppsCopy_(vec.m_data,m_data,m_size);
150     };
151
152 }
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174

```

```

175     ~nmvecpack()
176     {
177         if (m_container)
178             //delete []m_container;
179         nmppsFree(m_container);
180         m_container=0;
181     }
182
183     nmvecpack<T>& operator= (const nmvecpack<T>& vec) // спец. добавил const - nmppr ругается
184     {
185         ASSERTE(vec.m_size==m_size);
186         //memcpy(m_data,vec.m_data,m_size*sizeof(T));
187         nmppsCopy_(vec.m_data,m_data,m_size);
188         return *this;
189     }
190
191     INLINE nmintpack<T> operator[] (int idx)
192     {
193         // ASSERTE(idx>=-m_border);
194         // ASSERTE(idx<m_size+m_border);
195         nmintpack<T> nm(m_data,idx);
196         return nm;
197     }
198
199     template <class T2> nmvecpack<T2>& operator*= (const nmint<T2> val)
200     {
201         for(int idx=0;idx<m_size;idx++)
202             m_data[idx]*=val.m_value;
203         return (*this);
204     }
205
206
207     template <class T2> nmvecpack<T2> operator* (const nmint<T2>& val) const
208     {
209         nmvecpack<T2> res(m_size);
210         for(int idx=0;idx<m_size;idx++)
211             res.m_data[idx]=m_data[idx]*val.m_value;
212         return res;
213     }
214
215     template <class T2> nmint<T2> operator* (const nmvecpack<T2>& vec) const
216     {
217         ASSERTE(m_size==vec.m_size);
218         nmint<T2> res;
219         for (int idx=0;idx<m_size;idx++)
220             res.m_value+=m_data[idx]*vec.m_data[idx];
221         return res;
222     }
223
224
225     nmvecpack<T>& operator+= (const nmint<T> &val)
226     {
227         for(int idx=0;idx<m_size;idx++)
228             m_data[idx]+=val.m_value;
229         return (*this);
230     }
231
232     nmvecpack<T>& operator+= (const nmvecpack<T> &vec)
233     {
234         ASSERTE (vec.m_size==m_size);
235         for(int idx=0;idx<m_size;idx++)
236             m_data[idx]+=vec.m_data[idx];
237         return (*this);
238     }
239
240
241     nmvecpack<T> operator+ (const nmint<T>& val) const
242     {
243         nmvecpack<T> res(*this);
244         res+=val;
245         return res;
246     }
247
248     nmvecpack<T> operator+ (const nmvecpack<T>& vec) const
249     {
250         ASSERTE (vec.m_size==m_size);
251         nmvecpack<T> res(*this);
252         res+=vec;
253         return res;
254     }
255
256     nmvecpack<T>& operator-= (const nmint<T> &val)
257     {
258         for(int idx=0;idx<m_size;idx++)
259             m_data[idx]-=val.m_value;
260         return (*this);
261     }
262
263
264
265

```

```

266     nmvecpack<T>& operator-= (const nmvecpack<T> &vec)
267     {
268         _ASSERTE (vec.m_size==m_size);
269         for(int idx=0;idx<m_size;idx++)
270             m_data[idx]-=vec.m_data[idx];
271         return (*this);
272     }
273
274     nmvecpack<T> operator- ()const
275     {
276         nmvecpack<T> res(*this);
277         for(int idx=0;idx<m_size;idx++)
278             m_data[idx]=-m_data[idx];
279         return res;
280     }
281
282     nmvecpack<T> operator- (const nmint<T>& val)const
283     {
284         nmvecpack<T> res(*this);
285         res-=val;
286         return res;
287     }
288
289     nmvecpack<T> operator- (const nmvecpack<T>& vec)const
290     {
291         _ASSERTE (vec.m_size==m_size);
292         nmvecpack<T> res(*this);
293         res-=vec;
294         return res;
295     }
296
297     nmvecpack<T>& operator/=(const nmint<T> val)
298     {
299         _ASSERTE(val.m_value!=0);
300         for(int idx=0;idx<m_size;idx++)
301             m_data[idx]/=val.m_value;
302         return (*this);
303     }
304
305     nmvecpack<T> operator/ (const T val)const
306     {
307         nmvecpack<T> res(*this);
308         res/=val;
309         return res;
310     }
311
312     nmvecpack<T>& operator>=(const int shr)
313     {
314         _ASSERTE(shr>=0);
315         if (shr==0)
316             return (*this);
317         for(int idx=0;idx<m_size;idx++)
318             m_data[idx]>=shr;
319         return (*this);
320     }
321
322     nmvecpack<T> operator> (const int shr)const
323     {
324         _ASSERTE(shr>=0);
325         nmvecpack<T> res(*this);
326         res>=shr;
327         return res;
328     }
329
330     nmvecpack<T>& operator<=(const int shl)
331     {
332         _ASSERTE(shl>=0);
333         if (shl==0)
334             return (*this);
335         for(int idx=0;idx<m_size;idx++)
336             m_data[idx]<=shl;
337         return (*this);
338     }
339
340     nmvecpack<T> operator<= (const int shl)const
341     {
342         _ASSERTE(shl>=0);
343         if (shl==0)
344             return (*this);
345         for(int idx=0;idx<m_size;idx++)
346             m_data[idx]<=shl;
347         return res;
348     }
349
350     nmvecpack<T>& operator|= (const nmint<T> &val)
351     {
352         _ASSERTE(shl>=0);
353         nmvecpack<T> res(*this);
354         res|=shl;
355         return res;
356     }
357 
```

```

358     for(int idx=0;idx<m_size;idx++)
359         m_data[idx]=val;
360     return (*this);
361 }
362
363 nmvecpack<T>& operator|= (const nmvecpack<T> &vec)
364 {
365     _ASSERTE (vec.m_size==m_size);
366     for(int idx=0;idx<m_size;idx++)
367         m_data[idx]=vec.m_data[idx];
368     return (*this);
369 }
370
371 nmvecpack<T> operator| (const nmint<T>& val)const
372 {
373     nmvecpack<T> res(*this);
374     res|=val;
375     return res;
376 }
377
378 nmvecpack<T> operator| (const nmvecpack<T>& vec)const
379 {
380     _ASSERTE (vec.m_size==m_size);
381     nmvecpack<T> res(*this);
382     res-=vec;
383     return res;
384 }
385
386
387 nmvecpack<T>& operator&= (const nmint<T> &val)
388 {
389     for(int idx=0;idx<m_size;idx++)
390         m_data[idx]&=val;
391     return (*this);
392 }
393
394
395 nmvecpack<T>& operator&= (const nmvecpack<T> &vec)
396 {
397     _ASSERTE (vec.m_size==m_size);
398     for(int idx=0;idx<m_size;idx++)
399         m_data[idx]&=vec.m_data[idx];
400     return (*this);
401 }
402
403 nmvecpack<T> operator&& (const nmint<T>& val)const
404 {
405     nmvecpack<T> res(*this);
406     res&=val;
407     return res;
408 }
409
410 nmvecpack<T> operator&& (const nmvecpack<T>& vec)const
411 {
412     _ASSERTE (vec.m_size==m_size);
413     nmvecpack<T> res(*this);
414     res&=vec;
415     return res;
416 }
417
418 nmvecpack<T>& operator^= (const nmint<T> &val)
419 {
420     for(int idx=0;idx<m_size;idx++)
421         m_data[idx]^=val;
422     return (*this);
423 }
424
425
426 nmvecpack<T>& operator^= (const nmvecpack<T> &vec)
427 {
428     _ASSERTE (vec.m_size==m_size);
429     for(int idx=0;idx<m_size;idx++)
430         m_data[idx]^=vec.m_data[idx];
431     return (*this);
432 }
433
434 nmvecpack<T> operator^& (const nmint<T>& val)const
435 {
436     nmvecpack<T> res(*this);
437     res^=val;
438     return res;
439 }
440
441
442 nmvecpack<T> operator^& (const nmvecpack<T>& vec)const
443 {
444     nmvecpack<T> res(*this);
445     res^=vec;
446     return res;
447 }
448
449

```

```

450     nmvecpack<T>& operator~ ()const
451 {
452     nmvecpack<T> res(*this);
453     for(int idx=0;idx<m_size;idx++)
454         res.m_data[idx]=m_data[idx];
455     return res;
456 }
457
458
459     bool operator== (const nmvecpack<T>& vec) const
460 {
461     _ASSERTE (vec.m_size==m_size);
462     for(int idx=0;idx<m_size;idx++)
463         if (m_data[idx]!=vec.m_data[idx])
464             return false;
465     return true;
466 }
467
468
469     bool operator!= (const nmvecpack<T>& vec) const
470 {
471     _ASSERTE (vec.m_size==m_size);
472     for(int idx=0;idx<m_size;idx++)
473         if (m_data[idx]==vec.m_data[idx])
474             return false;
475     return true;
476 }
477
478     template<class T2> void SetData(T2* Data)
479 {
480     for(int i=0;i<m_size;i++)
481         m_data[i]=Data[i];
482 }
483
484     template<class T2> void GetData(T2* Data)
485 {
486     for(int i=0;i<m_size;i++)
487         Data[i]=(T2)m_data[i].value;
488 }
489
490     void reset()
491 {
492     memset(m_container,0,(m_size+2*m_border)*sizeof(T));
493 }
494
495
496
497 };
498
499
500
501
502 typedef nmvecpack<nm1> nmvec1;
503 typedef nmvecpack<nm2s> nmvec2s;
504 typedef nmvecpack<nm2u> nmvec2u;
505 typedef nmvecpack<nm4s> nmvec4s;
506 typedef nmvecpack<nm4u> nmvec4u;
507
508 #ifndef WIN32
509 typedef nmvecpack<nm8s> nmvec8s;
510 typedef nmvecpack<nm8u> nmvec8u;
511 typedef nmvecpack<nm16s> nmvec16s;
512 typedef nmvecpack<nm16u> nmvec16u;
513 #endif
514
515
516 #endif

```

7.96 tvector.h

```

1 //-----
2 // $Workfile::: tVector. $
3 // Векторно-матричная библиотека
4 // Copyright (c) RC Module Inc.
5 // $Revision: 1.7 $   $Date: 2005/06/29 14:14:06 $
6 //-----
7 //-----#
8 //-----#
9 //-----#
10 //-----#
11 //-----#
12 //-----#
13 //-----#
14 //-----#
15 //-----#
16 //-----#
17 //-----#
18 //-----#
19 //-----#
20 #ifndef _TVECTOR_H_INCLUDED_
21 #define _TVECTOR_H_INCLUDED_
22

```

```

23 #ifndef ENABLE_ASERTE
24 #include <crtdbg.h>
25 #define ASERTE _ASERTE
26 #else
27 #define ASERTE(expression)
28 #endif
29
30
31 #include<string.h>
32 #include "nmtype.h"
33 #include <math.h>
34
35
36
37
38
39 template <class T> class mtr;
40
41
42 // #include "tMatrix.h"
43
44 //*****
45 //*****
46 //*****
47 //*****
48 //*****
49 //*****
50 //*****
51 //*****
52 //*****
53 //*****
54 //*****
55 //*****
56 //*****
57 //*****
58 //*****
59 //*****
60 //*****
61 //*****
62 //*****
63 //*****
64 //*****
65 //*****
66 //*****
67 //*****
68 //*****
69 //*****
70 //*****
71 //*****
72 //*****
73 //*****
74 //*****
75 //*****
76
77 //void* allocate()
78
79 template<class T> class vec
80 {
81 protected:
82     T* m_container;
83 public:
84     int m_border;
85     int size;
86     T *m_data;
87
88     explicit vec()
89     {
90         size=0;
91         m_border=0;
92         m_container=0;
93         m_data=0;
94     }
95
96     void resize(int Size, int Border=0)
97     {
98         if (m_container)
99             delete []m_container;
100        size=Size;
101        m_border=Border;
102        m_container=new T[size+2*m_border];
103        m_data=m_container+m_border;
104    }
105
106    explicit vec(T * Data, int Size, int Border=0):m_data(Data),size(Size),m_border(Border),m_container(0)
107    {
108
109    };
110
111    explicit vec(int Size, int Border=0)
112    {
113        size=Size;
114        m_border=Border;
115        m_container=new T[size+2*m_border];
116        m_data=m_container+m_border;
117    }
118
119    /*explicit*/ vec(const vec<T> &vect)
120    {
121        m_border=vect.m_border;
122        size =vect.size;
123        m_container=new T[size+2*m_border];
124        m_data=m_container+m_border;
125        for (int idx=0;idx<size;idx++)
126            m_data[idx]=vect.m_data[idx];
127    };
128 /*
129 template<class T2> explicit vec(const vec<T2> &vect)
130 {
131     m_border=vect.m_border;
132     size =vect.size;
133     m_container=new T[size+2*m_border];
134     m_data=m_container+m_border;
135     for (int idx=0;idx<size;idx++)
136         m_data[idx]=vect.m_data[idx];
137 };
138 */
139 */

```

```

140     ~vec()
141     {
142         if (m_container)
143             delete []m_container;
144     }
145
146     void reset()
147     {
148         ASSERT(m_container);
149         if (m_container)
150             memset(m_container,0,(size+2*m_border)*sizeof(T));
151         else
152             memset(m_data,0,size*sizeof(T));
153     }
154
155     vec<T>& InitRamp(T StartValue, T Increment)
156     {
157         m_data[0]=StartValue;
158         for(int i=1; i<size; i++){
159             m_data[i]=m_data[i-1]+Increment;
160         }
161         return *this;
162     }
163
164     int MaxPos(T& maxval)
165     {
166         maxval=m_data[0];
167         int maxpos=0;
168         for(int i=1; i<size; i++) {
169             if (maxval<m_data[i]){
170                 maxpos=i;
171                 maxval=m_data[i];
172             }
173         }
174         return maxpos;
175     }
176     int MinPos(T& minval)
177     {
178         minval=m_data[0];
179         int minpos=0;
180         for(int i=1; i<size; i++) {
181             if (minval>m_data[i]){
182                 minpos=i;
183                 minval=m_data[i];
184             }
185         }
186         return minpos;
187     }
188
189     double Mean()
190     {
191         T Res=m_data[0];
192         for(int i=1; i<size; i++) {
193             Res+=m_data[i];
194         }
195         return double(Res)/size;
196     }
197
198     T& CustomMax()
199     {
200         T max=m_data[0];
201         //int maxpos=0;
202         for(int i=1; i<size; i++) {
203             max = tCustomMax(max,m_data[i]);
204         }
205         return max;
206     }
207     vec<T>& operator= (const T& val)
208     {
209         for(int idx=0;idx<size;idx++)
210             m_data[idx]=val;
211         return *this;
212     }
213
214     vec<T>& operator= (const vec<T>& vect)
215     {
216         ASSERT(vect.size==size);
217         for(int idx=0;idx<size;idx++)
218             m_data[idx]=vect.m_data[idx];
219         return *this;
220     }
221
222     inline T* addr(int idx)
223     {
224         ASSERT(idx>=-m_border);
225         ASSERT(idx<size+m_border);
226         return m_data+idx;

```

```

227     }
228
229     inline T& operator [] (size_t idx)
230     {
231         ASSERTE(idx>=-m_border);
232         ASSERTE(idx<size+m_border);
233         T* res=(T*)(m_data+idx);
234         return *res;
235     }
236
237     vec<T>& operator*= (const T& val)
238     {
239         for(int idx=0;idx<size;idx++)
240             m_data[idx]*=val;
241         return (*this);
242     }
243
244     vec<T> operator* (const T& val) const
245     {
246         vec<T> res(size);
247         for(int idx=0;idx<size;idx++)
248             res.m_data[idx]=m_data[idx]*val;
249         return res;
250     }
251
252     T operator* (const vec<T>& vect) const
253     {
254         ASSERTE(size==vect.size);
255         T res=0;
256         for (int idx=0;idx<size;idx++)
257             res+=m_data[idx]*vect.m_data[idx];
258         return res;
259     }
260
261     vec<T> operator* (const mtr<T> matr) const
262     {
263         ASSERTE(size==matr.m_height);
264         vec<T> res(matr.m_width);
265         for(int i=0; i<matr.m_width; i++){
266             T sum=matr[0][i]*m_data[0];
267             for(int j=1; j<matr.m_height; j++){
268                 sum+=matr[j][i]*m_data[j];
269             }
270             res[i]=sum;
271         }
272         return res;
273     }
274     vec<T>& operator+= (const T &val)
275     {
276         for(int idx=0;idx<size;idx++)
277             m_data[idx]+=val;
278         return (*this);
279     }
280
281     vec<T>& operator+= (const vec<T> &vect)
282     {
283         ASSERTE (vect.size==size);
284         for(int idx=0;idx<size;idx++)
285             m_data[idx]+=vect.m_data[idx];
286         return (*this);
287     }
288
289     vec<T> operator+ (const T& val) const
290     {
291         vec<T> res(*this);
292         res+=val;
293         return res;
294     }
295
296     vec<T> operator+ (const vec<T>& vect) const
297     {
298         ASSERTE (vect.size==size);
299         vec<T> res(*this);
300         res+=vect;
301         return res;
302     }
303
304     vec<T>& operator-= (const T &val)
305     {
306         for(int idx=0;idx<size;idx++)
307             m_data[idx]-=val;
308         return (*this);
309     }
310
311     vec<T>& operator-= (const vec<T> &vect)
312     {
313         ASSERTE (vect.size==size);

```

```

314     for(int idx=0;idx<size;idx++)
315         m_data[idx]-=vect.m_data[idx];
316     return (*this);
317 }
318
319 vec<T> operator- (const vec<T>& vect) const
320 {
321     ASSERTE (vect.size==size);
322     vec<T> res(*this);
323     res-=vect;
324     return res;
325 }
326
327 vec<T> operator- () const
328 {
329     vec<T> res(*this);
330     for(int idx=0;idx<size;idx++)
331         res.m_data[idx]=-m_data[idx];
332     return res;
333 }
334
335 vec<T>& operator/=(const T val)
336 {
337     ASSERTE(val!=0);
338     for(int idx=0;idx<size;idx++)
339         m_data[idx]/=val;
340     return (*this);
341 }
342
343 vec<T> operator/ (const T val) const
344 {
345     vec<T> res(*this);
346     res/=val;
347     return res;
348 }
349
350 vec<T>& operator»=(const int shr)
351 {
352     ASSERTE(shr>=0);
353     if (shr==0) return (*this);
354     for(int idx=0;idx<size;idx++)
355         m_data[idx]»=shr;
356     return (*this);
357 }
358
359 vec<T> operator» (const int shr) const
360 {
361     ASSERTE(shr>=0);
362     vec<T> res(*this);
363     res»=shr;
364     return res;
365 }
366
367 vec<T>& operator«=(const int shl)
368 {
369     ASSERTE(shl>=0);
370     if (shl==0) return (*this);
371     for(int idx=0;idx<size;idx++)
372         m_data[idx]«=shl;
373     return (*this);
374 }
375
376 vec<T> operator« (const int shl) const
377 {
378     ASSERTE(shl>=0);
379     vec<T> res(*this);
380     res«=shl;
381     return res;
382 }
383
384 //-----
385 vec<T>& operator&=(const T& val)
386 {
387     for(int idx=0;idx<size;idx++)
388         m_data[idx]&=val;
389     return (*this);
390 }
391
392 vec<T>& operator&=(const vec<T>& vect)
393 {
394     for(int idx=0;idx<size;idx++)
395         m_data[idx]&=vect.m_data[idx];
396     return (*this);
397 }
398
399 vec<T> operator& (const T& val) const
400 {

```

```

401     vec<T> res(*this);
402     res&=val;
403     return res;
404 }
405
406     vec<T> operator& (const vec<T>& vect) const
407 {
408     vec<T> res(*this);
409     res&=vect;
410     return res;
411 }
412 //-----
413     vec<T>& operator|=(const T& val)
414 {
415     for(int idx=0;idx<size;idx++)
416         m_data[idx]|=val;
417     return (*this);
418 }
419
420     vec<T>& operator|=(const vec<T>& vect)
421 {
422     for(int idx=0;idx<size;idx++)
423         m_data[idx]|=vect.m_data[idx];
424     return (*this);
425 }
426
427     vec<T> operator| (const T& val) const
428 {
429     vec<T> res(*this);
430     res|=val;
431     return res;
432 }
433
434     vec<T> operator| (const vec<T>& vect) const
435 {
436     vec<T> res(*this);
437     res|=vect;
438     return res;
439 }
440 //-----
441     vec<T>& operator^=(const T& val)
442 {
443     for(int idx=0;idx<size;idx++)
444         m_data[idx]^=val;
445     return (*this);
446 }
447
448     vec<T>& operator^=(const vec<T>& vect)
449 {
450     for(int idx=0;idx<size;idx++)
451         m_data[idx]^=vect.m_data[idx];
452     return (*this);
453 }
454
455     vec<T> operator^ (const T& val) const
456 {
457     vec<T> res(*this);
458     res^=val;
459     return res;
460 }
461
462     vec<T> operator^ (const vec<T>& vect) const
463 {
464     vec<T> res(*this);
465     res^=vect;
466     return res;
467 }
468     int sum()
469 {
470     int summ=0;
471     for(int i=0;i<size;i++)
472         summ+=m_data[i];
473     return summ;
474 }
475
476     bool operator== (const vec<T> &vect)
477 {
478     ASSERTE (vect.size==size);
479     for(int idx=0;idx<size;idx++)
480         if (m_data[idx]!=vect.m_data[idx])
481             return false;
482     return true;
483 }
484
485     bool operator!= (const vec<T> &vect)
486 {
487     ASSERTE (vect.size==size);

```

```

488     for(int idx=0;idx<size;idx++)
489         if (m_data[idx]!=vect.m_data[idx])
490             return true;
491     return false;
492 }
493 //-----
494
495 /*
496 template<class T2> void SetData(T2* Data)
497 {
498     for(int i=0;i<size;i++)
499     m_data[i]=Data[i];
500 }
501
502 template<class T2> void GetData(T2* Data)
503 {
504     for(int i=0;i<size;i++)
505     Data[i]=(T2)m_data[i].value;
506 }
507 */
508 //void InitConst(T Value)
509 //{
510 //    for (int idx=0;idx<size;idx++)
511 //        m_data[idx]=Value;
512 //}
513 //}
514 };
515 //template <class T> void InitInc(vec<T>& A,T StartValue=0,T Increment=1)
516 //{
517 //    for (int idx=0;idx<A.size();idx++,StartValue+=Increment)
518 //        A[idx]=StartValue;
519 //}
520
521 template<class T1, class T2> void Convert(vec<T1>& vSrc, vec<T2>& vDst){
522     ASSERT(vSrc.size()==vDst.size());
523     for(int i=0; i<vSrc.size(); i++){
524         vDst[i]=(T2)vSrc[i];
525     }
526 }
527
528 //bool operator== (const vec<T>& vec) const
529 //bool operator!= (const vec<T>& vec) const;
530
531
532 //***** Math function ****
533 /*   Math function
534 //***** Math function ****
535 */
536 /*
537 template <class T> T Norm2(vec<T>& X)
538 {
539 T Buffer(0);
540 for(int idx=0;idx<X.size();idx++)
541 for(int col=0;col<X.width;col++)
542 Buffer+=X[idx]*X[idx];
543 return sqrt(Buffer);
544 }
545
546 template <class T> double vectorError(vec<T>& A,double *B)
547 {
548 double Buffer=0;
549 double diff;
550 int idx=0;
551 for(int idx=0;idx<A.size();idx++,idx++)
552 for(int col=0;col<A.width;col++)
553 {
554 diff=double_(A[idx])-B[idx];
555 Buffer+=diff*diff;
556 }
557 return sqrt(Buffer)/(A.width*A.size);
558 }
559 */
560 template <class T> vec<T> Sqr(vec<T>& A)
561 {
562     vec<T> Res(A);
563     for (int i=0;i<A.size();i++)
564         Res[i]=A[i]*A[i];
565     return Res;
566 }
567 template <class T> vec<T> Sqrt(vec<T>& A)
568 {
569     vec<T> Res(A);
570     for (int i=0;i<A.size();i++)
571         Res[i]=(T)sqrt((double)A[i]);
572     return Res;
573 }
574
575 template <class T> T Summ(vec<T>& A)

```

```

576 {
577     T Sum(0);
578     for (int i=0;i<A.size;i++)
579         Sum+=A[i];
580     return Sum;
581 }
582
583 template <class T> void Clear(vec<T>& A)
584 {
585     for (int idx=0;idx<A.size;idx++)
586         A[idx]=0;
587 }
588
589
590
591
592 #endif

```

7.97 Файл g:/nmpp/include/nmtype.h

Структуры данных

- struct nmreg
- struct int15in16x4
- struct s_int32x2
- struct int31in32x2
- struct int30in32x2
- struct v16nm4s
- struct v4nm8s
- struct s_v8nm8s
- struct s_v16nm8s
- struct s_v4nm16s
- struct s_v8nm16s
- struct s_v16nm16s
- struct s_v2nm32s
- struct s_v4nm32s
- struct s_v8nm32s
- struct s_v16nm32s
- struct s_v16nm4u
- struct s_v4nm8u
- struct s_v8nm8u
- struct s_v16nm8u
- struct s_v4nm16u
- struct s_v8nm16u
- struct s_v16nm16u
- struct s_v2nm32u
- struct s_v4nm32u
- struct s_v8nm32u
- struct s_v16nm32u
- struct nm16sc
- struct s_nm32sc
- struct s_v2nm32f
- struct s_v4nm32f
- struct s_v4nm64f
- struct s_nm32fc
- struct s_nm32fcr
- struct s_nm64sc
- struct Tmp2BuffSpec

Макросы

- #define MEM_LOCAL 0
- #define MEM_GLOBAL 1
- #define HEAP0 0
- #define HEAP1 1
- #define HEAP2 2
- #define HEAP3 3
- #define sizeof32(t) (sizeof(t)/4)
- #define VEC_NM1(X) unsigned data[(X)/32];
- #define VEC_NM2U(X) unsigned data[(X)/16];
- #define VEC_NM2S(X) int data[(X)/16];
- #define VEC_NM4U(X)
- #define VEC_NM4S(X) int data[(X)/8];
- #define VEC_NM8U(X) unsigned int data[(X)/4];
- #define VEC_NM8S(X)
- #define VEC_NM16U(X)
- #define VEC_NM16S(X)
- #define VEC_NM32U(X) nm32u data[(X)];
- #define VEC_NM32S(X)
- #define CAPACITY_nm64s 1
-
- #define CAPACITY_nm32s 2
- #define CAPACITY_nm16s 4
- #define CAPACITY_nm8s 8
- #define CAPACITY_nm4s 16
- #define CAPACITY_nm2s 32
- #define CAPACITY_nm1 64
- #define NM16S(vec, size) nm64s vec##__64s[size/4]; nm16s* vec=(nm16s*)vec##__64s
- #define NM32S(vec, size) nm64s vec##__64s[size/2]; nm32s* vec=(nm32s*)vec##__64s
- #define __INLINE__ static __inline
- #define NM16Sx4(x0, x1, x2, x3)
- #define NM32Sx2(x0, x1)
- #define NM32Sx4(x0, x1, x2, x3) NM32Sx2(x0,x1),NM32Sx2(x2,x3)

Определения типов

- typedef signed long long INT64
- typedef unsigned long long UINT64
- typedef int nm1
- typedef void nm2s
- typedef void nm4s
- typedef char nm8s
- typedef nm8s nm8s7b
- typedef short nm16s
- typedef nm16s nm16s15b
- typedef int nm32s
- typedef struct s_int32x2 int32x2
- typedef int nm32s31b
- typedef int nm32s30b
- typedef long long nm64s
- typedef nm64s nm64s63b
- typedef void nm2u
- typedef void nm4u

- `typedef nm4u nm4u3b`
- `typedef unsigned char nm8u`
- `typedef nm8u nm8u7b`
- `typedef unsigned short nm16u`
- `typedef nm16u nm16u15b`
- `typedef unsigned int nm32u`
- `typedef unsigned int nm32u31b`
- `typedef unsigned long long nm64u`
- `typedef int int1b`
- `typedef int int2b`
- `typedef int int3b`
- `typedef int int4b`
- `typedef int int7b`
- `typedef int int8b`
- `typedef int int15b`
- `typedef int int16b`
- `typedef int int30b`
- `typedef int int31b`
- `typedef int int32b`
- `typedef INT64 int63b`
- `typedef INT64 int64b`
- `typedef unsigned int uint1b`
- `typedef unsigned int uint2b`
- `typedef unsigned int uint3b`
- `typedef unsigned int uint4b`
- `typedef unsigned int uint7b`
- `typedef unsigned int uint8b`
- `typedef unsigned int uint15b`
- `typedef unsigned int uint16b`
- `typedef unsigned int uint31b`
- `typedef unsigned int uint32b`
- `typedef UINT64 uint63b`
- `typedef nm64u uint64b`
- `typedef struct s_v8nm8s v8nm8s`
- `typedef struct s_v16nm8s v16nm8s`
- `typedef struct s_v4nm16s v4nm16s`
- `typedef struct s_v8nm16s v8nm16s`
- `typedef struct s_v16nm16s v16nm16s`
- `typedef struct s_v2nm32s v2nm32s`
- `typedef struct s_v4nm32s v4nm32s`
- `typedef struct s_v8nm32s v8nm32s`
- `typedef struct s_v16nm32s v16nm32s`
- `typedef v16nm8s v16nm8s7b`
- `typedef struct s_v16nm4u v16nm4u`
- `typedef struct s_v4nm8u v4nm8u`
- `typedef struct s_v8nm8u v8nm8u`
- `typedef struct s_v16nm8u v16nm8u`
- `typedef struct s_v4nm16u v4nm16u`
- `typedef struct s_v8nm16u v8nm16u`
- `typedef struct s_v16nm16u v16nm16u`
- `typedef struct s_v2nm32u v2nm32u`
- `typedef struct s_v4nm32u v4nm32u`
- `typedef struct s_v8nm32u v8nm32u`
- `typedef struct s_v16nm32u v16nm32u`
- `typedef v16nm4u v16nm4b3u`

- `typedef struct s_nm32sc nm32sc`
- `typedef struct s_v2nm32f v2nm32f`
- `typedef struct s_v4nm32f v4nm32f`
- `typedef struct s_v4nm64f v4nm64f`
- `typedef struct s_nm32fc nm32fc`
- `typedef struct s_nm32fcr nm32fcr`
- `typedef float nm32f`
- `typedef double nm64f`
- `typedef struct s_nm64sc nm64sc`
- `typedef unsigned long long uint64`
- `typedef unsigned int uint32`
- `typedef nm64sc int64sc`
- `typedef uint64 fifo64`
- `typedef uint32 fseq32`
- `typedef uint64 seq64`
- `typedef uint64 fseq64`

7.97.1 Подробное описание

Файл, содержащий определения типов упакованных данных.

7.97.2 Макросы

7.97.2.1 NM16Sx4

```
#define NM16Sx4(
    x0,
    x1,
    x2,
    x3 )
```

Макроопределение:

```
((long long)(x3&0xFFFF)<<48)| \
((long long)(x2&0xFFFF)<<32)| \
((long long)(x1&0xFFFF)<<16)| \
((long long)(x0&0xFFFF))
```

7.97.2.2 NM32Sx2

```
#define NM32Sx2(
    x0,
    x1 )
```

Макроопределение:

```
((long long)(unsigned(x1))<<32)| \
((long long)(unsigned(x0)))
```

7.97.2.3 VEC_NM16S

```
#define VEC_NM16S(
    X )
```

Макроопределение:

```
int data[(X)/2]; \
void set(int i, int val){ ((unsigned short*)data)[i]=val;} \
short operator[](int index){ return ((unsigned short*)data)[index];}
```

7.97.2.4 VEC_NM16U

```
#define VEC_NM16U(
    X )
```

Макроопределение:

```
unsigned int data[(X)/2]; \
void set(int i, int val){ ((unsigned short*)data)[i]=val;} \
unsigned short operator[](int index){ return ((unsigned short*)data)[index];}
```

7.97.2.5 VEC_NM32S

```
#define VEC_NM32S(
    X )
```

Макроопределение:

```
int data[(X)]; \
void set(int i, int val){ ((int*)data)[i]=val;} \
int operator[](int index){ return ((int*)data)[index];}
```

7.97.2.6 VEC_NM4U

```
#define VEC_NM4U(
    X )
```

Макроопределение:

```
unsigned data[(X)/8]; \
uint4b get(int nIndex){return nmget((nm4u*)data,nIndex); }\
uint4b operator[](int index){ return nmget((nm4u*)data,index); }
```

7.97.2.7 VEC_NM8S

```
#define VEC_NM8S(
    X )
```

Макроопределение:

```
int data[(X)/4]; \
void set(int i, int val){ ((char*)data)[i]=val;} \
char operator[](int index){ return ((char*)data)[index];}
```

7.98 nmtype.h

См. документацию.

```

1
2
3 //-----
4 // $Workfile:: nmtype.h      $
5 // Векторно-матричная библиотека
6 //
7 // Copyright (c) RC Module Inc.
8 //
9 // $Revision: 1.1 $   $Date: 2004/11/22 13:50:02 $
10 //
11 //-----
```

```

14
20
21 #ifndef _NMTYPE_H_INCLUDED_
22 #define _NMTYPE_H_INCLUDED_
```

```

23
24 #define MEM_LOCAL 0
25 #define MEM_GLOBAL 1
26 #define HEAP0 0
27 #define HEAP1 1
28 #define HEAP2 2
29 #define HEAP3 3
30
31 #ifdef __GNUC__
32 #define __int64 long long
33#endif
34 // if new compiler
35 #ifdef __NM__
36     //typedef long __int64;
37#endif
38     typedef signed long long INT64;
39     typedef unsigned long long UINT64;
40
41 #ifdef __NM
42 #define sizeof32(t) sizeof(t)
43 // #define malloc32(size_int32) malloc(size_int32)
44#else
45 #define sizeof32(t) (sizeof(t)/4)
46 // #define malloc32(size_int32) malloc(size_int32*4)
47#endif
48
49
50
51 //-----
```

```

57 struct nmreg { int nVal; };
58 //-----
59
60 //*****
```

```

61
84 //-----
```

```

97 #ifdef __NM
98     typedef void nm1;
99#else
100    typedef int nm1;
101#endif
102 //-----
```

```

116 typedef void nm2s;
117 //-----
```

```

129 typedef void nm4s;
130 //-----
```

```

142 /*
143 struct int8x8{
144 #ifdef __NM__
145     unsigned long items;
146#else
147     signed char item[8];
148#endif
149 };//*
150
151 // #ifdef __NM__
152 //     typedef void nm8s;
153 // #else
154 //     typedef signed char nm8s;
155 // #endif
156
157 #ifdef __NM__
158 #ifdef __cplusplus
159     typedef struct {
160         virtual void func() = 0;
161     } nm8s;
```

```
162 #else
163     typedef void nm8s;
164 #endif
165
166 #else
167     typedef char nm8s;
168 #endif
169
170
171
172 //typedef signed char nm8s;
173 //typedef void nm8s;
174 //typedef struct s_nm8s{
175 //#ifdef __NM__
176 //    unsigned long vec;
177 //#else
178 //    char num[8];
179 //#endif
180 //} nm8s;
181 //
182
183
184 //-----
196 //typedef signed char nm8s7b;
197     typedef nm8s nm8s7b;
198 /*struct int7in8x8{
199 #ifdef __NM__
200     unsigned long items;
201 #else
202     char item[8];
203 #endif
204 };*/
205 //-----
217 /*
218 struct int16x4{
219 #ifdef __NM__
220     unsigned long items;
221 #else
222     signed short item[4];
223 #endif
224 };
225 */
226
227 //#ifdef __NM__
228 //typedef void nm16s;
229 //#else
230 //typedef signed short nm16s;
231 //#endif
232
233 #ifdef __NM__
234 #ifdef __cplusplus
235 typedef struct {
236     virtual void func() = 0;
237 } nm16s;
238 #else
239 typedef void nm16s;
240 #endif
241
242 #else
243 typedef short nm16s;
244 #endif
245
246
247
248
249
250 //-----
265 typedef nm16s nm16s15b;
266
267
268
269 struct int15in16x4{
270 #ifdef __NM__
271     unsigned long items;
272 #else
273     signed short item[4];
274 #endif
275 };
276
277
278
279
280 //-----
292 typedef int nm32s;
293 typedef struct s_int32x2{
294 //public:
295     int hi;
```

```

296     int lo;
297     //unsigned long long data;
298 //  int32x2(){
299 //    data=0;
300 //  }
301 //  int32x2(int item0, int item1){
302 //    set(0,item0);
303 //    set(1,item1);
304 //  }
305 //  int32x2& operator = (unsigned long long value){
306 //    data = value;
307 //    return *this;
308 //  }
309 //  int &lo(){
310 //    return *((int*)&data);
311 //  }
312 //  int &hi(){
313 //    return *((int*)&data)+1;
314 //  }
315 //  void set(int indx, int value){
316 //    ((int*)&data)[indx]=value;
317 //  }
318 //  int get(int indx){
319 //    return ((int*)&data)[indx];
320 //  }
321 }int32x2;
322 //-----
334 typedef int nm32s31b;
335 struct int31in32x2{
336 #ifdef __NM__
337   unsigned long items;
338 #else
339   int item[2];
340 #endif
341 };
342 //-----
354 typedef int nm32s30b;
355 struct int30in32x2{
356 #ifdef __NM__
357   unsigned long items;
358 #else
359   int item[2];
360 #endif
361 };
362 //-----
374 //typedef __int64 nm64s;
375 typedef long long nm64s;
376
377
378 //-----
390 typedef nm64s nm64s63b;
391
404 //=====
405 //-----
417 typedef void nm4u;
430 typedef nm4u nm4u3b;
431
432 //-----
433
445 #ifdef __NM__
446   typedef void nm8u;
447 #else
448   typedef unsigned char nm8u;
449 #endif
450 //-----
451
463 typedef nm8u nm8u7b;
464 //-----
478 #ifdef __NM__
479   typedef void nm16u;
480 #else
481   typedef unsigned short nm16u;
482 #endif
483 //-----
495 typedef nm16u nm16u15b;
496 //-----
509 typedef unsigned int nm32u;
510
511 //-----
523 typedef unsigned int nm32u31b;
524 //-----
536 typedef unsigned long long nm64u;
537
538 //=====
539

```

```

540
541
553 //=====
553 typedef int int1b;
554 //-----
566 typedef int int2b;
567 //-
579 typedef int int3b;
580 //-
592 typedef int int4b;
593 //-
605 typedef int int7b;
606 //-
618 typedef int int8b;
619 //-
631 typedef int int15b;
632 //-
644 typedef int int16b;
645 //-
657 typedef int int30b;
658 //-
670 typedef int int31b;
671 //-
683 typedef int int32b;
684 //-
696 typedef INT64 int63b;
697 //-
709 typedef INT64 int64b;
710
722 //=====
722 typedef unsigned int uint1b;
723 //-
735 typedef unsigned int uint2b;
736 //-
748 typedef unsigned int uint3b;
749 //-
761 typedef unsigned int uint4b;
762 //-
774 typedef unsigned int uint7b;
775 //-
787 typedef unsigned int uint8b;
788 //-
800 typedef unsigned int uint15b;
801 //-
813 typedef unsigned int uint16b;
814 //-
826 typedef unsigned int uint31b;
827 //-
839 typedef unsigned int uint32b;
840 //-
852 typedef UINT64 uint63b;
853 //-
865 typedef nm64u uint64b;
866
867
868
869
870
871 //=====
872 //===== nmget =====
873 /*
874 int2b nmget_2s(struct nm2s* pVec, int nIndex)
875 {
876 nm32u nBase=((nm32u)nIndex)/16;
877 nm32u nDisp=((nm32u)nIndex)%16;
878 nm32s nVal=((nm32s*)pVec)[nBase];
879 nDisp<=1;
880 nVal<=(32-2-nDisp);
881 nVal>=30;
882 return nVal;
883 }
884
885 - INLINE -- int4b nmget_4s(nm4s* pVec, int nIndex)
886 {
887 nm32u nBase=((nm32u)nIndex)/8;
888 nm32u nDisp=((nm32u)nIndex)%8;
889 nm32s nVal=((nm32s*)pVec)[nBase];
890 nDisp<=2;
891 nVal<=(32-4-nDisp);
892 nVal>=28;
893 return nVal;
894 }
895
896 - INLINE -- int8b nmget_8s(nm8s* pVec, int nIndex)
897 {
898 return (char*)pVec[nIndex];

```

```

899 }
900
901 INLINE __ int16b nmget_16s(nm16s* pVec, int nIndex)
902 {  

903     return (hsort)pVec[nIndex];  

904 }
905
906 ----- uint -----
907 INLINE __ uint1b nmget_1u(nm1* pVec, int nIndex)
908 {  

909     nm32u nBase=((nm32u)nIndex)/32;  

910     nm32u nDisp=((nm32u)nIndex)%32;  

911     nm32u nVal=((nm32u*)pVec)[nBase];  

912     nVal>>nDisp;  

913     nVal&=1;  

914     return nVal;  

915 }
916 INLINE __ uint2b nmget_2u(nm2u* pVec, int nIndex)
917 {  

918     nm32u nBase=((nm32u)nIndex)/16;  

919     nm32u nDisp=((nm32u)nIndex)%16;  

920     nm32u nVal=((nm32u*)pVec)[nBase];  

921     nVal>>nDisp*2;  

922     nVal&=3;  

923     return nVal;  

924 }
925
926
927 INLINE __ uint4b nmget_4u(nm4u* pVec, int nIndex)
928 {  

929     nm32u nBase=((nm32u)nIndex)/8;  

930     nm32u nDisp=((nm32u)nIndex)%8;  

931     nm32u nVal=((nm32u*)pVec)[nBase];  

932     nVal>>nDisp*4;  

933     nVal&=15;  

934     return nVal;  

935 }
936
937 INLINE __ uint8b nmget_8u(nm8u* pVec, int nIndex)
938 {  

939     return pVec[nIndex];  

940 }
941
942 INLINE __ uint16b nmget_16u(nm16u* pVec, int nIndex)
943 {  

944     return pVec[nIndex];  

945 }
946 */
947
948 //=====
949 #define VEC_NM1(X) unsigned data[(X)/32];
950 #define VEC_NM2U(X) unsigned data[(X)/16];
951 #define VEC_NM2S(X) int data[(X)/16];
952 #define VEC_NM4U(X) \
953     unsigned data[(X)/8]; \
954     uint4b get(int nIndex){return nmget((nm4u*)data,nIndex);}\
955     uint4b operator[] (int index){ return nmget((nm4u*)data,index);}\
956
957
958 #define VEC_NM4S(X) int data[(X)/8];
959 #define VEC_NM8U(X) unsigned int data[(X)/4];
960 #define VEC_NM8S(X) \
961     int data[(X)/4]; \
962     void set(int i, int val){ ((char*)data)[i]=val;} \
963     char operator[] (int index){ return ((char*)data)[index];}
964
965 #define VEC_NM16U(X) \
966     unsigned int data[(X)/2]; \
967     void set(int i, int val){ ((unsigned short*)data)[i]=val;} \
968     unsigned short operator[] (int index){ return ((unsigned short*)data)[index];}
969
970 #define VEC_NM16S(X) \
971     int data[(X)/2]; \
972     void set(int i, int val){ ((unsigned short*)data)[i]=val;} \
973     short operator[] (int index){ return ((unsigned short*)data)[index];}
974
975 #define VEC_NM32U(X) nm32u data[(X)];
976 #define VEC_NM32S(X) \
977     int data[(X)]; \
978     void set(int i, int val){ ((int*)data)[i]=val;} \
979     int operator[] (int index){ return ((int*)data)[index];}
980
981
982
983
984 //-----

```

```

990 //struct v16nm4s {VEC_NM4S(16)};
991 struct v16nm4s {
992     unsigned long long vec[1];
993 };
994
995 //-----
1002 //struct v4{VEC_NM8S(4)};
1003 struct v4nm8s {
1004     unsigned long long vec[1];
1005 };
1006 //-----
1013 //struct v8nm8s {VEC_NM8S(8)};
1014 typedef struct s_v8nm8s {
1015     unsigned long long vec[1];
1016 }v8nm8s ;
1017 //-----
1024 //struct v16nm8s {VEC_NM8S(16)};
1025 typedef struct s_v16nm8s {
1026     unsigned long long vec[2];
1027 } v16nm8s;
1028 //-----
1035 //struct v4nm16s {VEC_NM16S(4)};
1036 typedef struct s_v4nm16s {
1037     unsigned long long vec[1];
1038 } v4nm16s;
1039
1040
1041 //-----
1048 //struct v8nm16s {VEC_NM16S(8)};
1049 typedef struct s_v8nm16s {
1050     unsigned long long vec[2];
1051 }v8nm16s ;
1052 //-----
1059 //struct v16nm16s {VEC_NM16S(16)};
1060 typedef struct s_v16nm16s {
1061     unsigned long long vec[4];
1062 } v16nm16s ;
1063
1064 //-----
1071 //struct v2nm32s {VEC_NM32S(2)};
1072 typedef struct s_v2nm32s {
1073     unsigned long long vec[1];
1074 }v2nm32s ;
1075 //-----
1082 //struct v4nm32s {VEC_NM32S(4)};
1083 typedef struct s_v4nm32s {
1084     unsigned long long vec[2];
1085 } v4nm32s ;
1086 //-----
1093 //struct v8nm32s {VEC_NM32S(8) };
1094 typedef struct s_v8nm32s {
1095     unsigned long long vec[4];
1096 } v8nm32s ;
1097 //-----
1104 //struct v16nm32s {VEC_NM32S(16) };
1105 typedef struct s_v16nm32s {
1106     unsigned long long vec[8];
1107 } v16nm32s ;
1108 //-----
1120 typedef v16nm8s v16nm8s7b;
1121
1122 //=====
1123
1124 //-----
1130 //struct v16nm4u {VEC_NM4U(16) };
1131 typedef struct s_v16nm4u {
1132     unsigned long long vec[1];
1133 } v16nm4u;
1134 //-----
1141 //struct v4nm8u {VEC_NM8U(4)      };
1142 typedef struct s_v4nm8u {
1143     unsigned long long vec[1];
1144 } v4nm8u;
1145 //-----
1152 //struct v8nm8u {VEC_NM8U(8)      };
1153 typedef struct s_v8nm8u {
1154     unsigned long long vec[1];
1155 }v8nm8u;
1156 //-----
1163 //struct v16nm8u {VEC_NM8U(16) };
1164 typedef struct s_v16nm8u {
1165     unsigned long long vec[2];
1166 } v16nm8u;
1167 //-----
1174 //struct v4nm16u {VEC_NM16U(4) };
1175 typedef struct s_v4nm16u {

```

```

1176     unsigned long long vec[1];
1177 } v4nm16u;
1178 //-----
1185 //struct v8nm16u {VEC_NM16U(8) };
1186 typedef struct s_v8nm16u {
1187     unsigned long long vec[2];
1188 } v8nm16u;
1189 //-----
1196 //struct v16nm16u {VEC_NM16U(16) };
1197 typedef struct s_v16nm16u {
1198     unsigned long long vec[4];
1199 } v16nm16u;
1200 //-----
1207 //struct v2nm32u {VEC_NM32U(2) };
1208 typedef struct s_v2nm32u {
1209     unsigned long long vec[1];
1210 } v2nm32u;
1211 //-----
1218 //struct v4nm32u {VEC_NM32U(4) };
1219 typedef struct s_v4nm32u {
1220     unsigned long long vec[2];
1221 } v4nm32u;
1222 //-----
1229 //struct v8nm32u {VEC_NM32U(8) };
1230 typedef struct s_v8nm32u {
1231     unsigned long long vec[4];
1232 } v8nm32u;
1233 //-----
1240 //struct v16nm32u {VEC_NM32U(16) };
1241 typedef struct s_v16nm32u {
1242     unsigned long long vec[8];
1243 } v16nm32u;
1244 //-----
1245
1258 typedef v16nm4u v16nm4b3u;
1259
1260 //=====
1276 struct nm16sc
1277 {
1278
1279     signed short r;
1280     signed short c;
1281
1282     //nm16sc() : r(0), c(0) {}
1283     //nm16sc(nm16s b) : r(b), c(0) {}
1284 };
1285 //-----
1300 typedef struct s_nm32sc
1301 {
1302
1303     nm32s re;
1304     nm32s im;
1305
1306     //nm32sc() : re(0), im(0) {}
1307     //nm32sc(nm32s b) : re(b), im(0) {}
1308 }nm32sc;
1309
1310 typedef struct s_v2nm32f
1311 {
1312     float v0;
1313     float v1;
1314 } v2nm32f;
1315
1316 typedef struct s_v4nm32f
1317 {
1318     float vec[4];
1319 } v4nm32f;
1320
1321 typedef struct s_v4nm64f
1322 {
1323     double vec[4];
1324 } v4nm64f;
1325
1326
1327 typedef struct s_nm32fc
1328 {
1329     float re;
1330     float im;
1331 }nm32fc;
1332
1333
1334 typedef struct s_nm32fcr
1335 {
1336
1337     float im;
1338     float re;

```

```

1339
1340     //|nm32sc() : re(0), im(0) {}
1341     //|nm32sc(nm32s b) : re(b), im(0) {}
1342 }nm32fcr;
1343
1344 typedef float nm32f;
1345 typedef double nm64f;
1346
1347 //-----
1348
1349 typedef struct s_nm64sc
1350 {
1351     long long re;
1352     long long im;
1353
1354     //| nm64sc() : re(0), im(0) {}
1355     //| nm64sc(nm64s b) : re(b), im(0) {}
1356
1357     //| int Print();
1358
1359 } nm64sc;
1360
1361 #ifndef uint64
1362 typedef unsigned long long uint64;
1363 #endif
1364
1365 #ifndef uint32
1366 typedef unsigned int uint32;
1367 #endif
1368
1369 typedef nm64sc int64sc;
1370 typedef uint64 fifo64;
1371 typedef uint32 fseq32;
1372
1373
1374 typedef uint64 seq64;
1375 typedef uint64 fseq64;
1376
1377 typedef struct {
1378     void* buffer0;
1379     void* buffer1;
1380     fseq64 route;
1381     int mode;
1382     int status;
1383 } Tmp2BuffSpec;
1384
1385 //_
1386 //|_--INLINE int NM_CAPACITY(nm1*) {return 64;}
1387 //|_--INLINE int NM_CAPACITY(nm2*) {return 32;}
1388 //|_--INLINE int NM_CAPACITY(nm4*) {return 16;}
1389 //|_--INLINE int NM_CAPACITY(nm8*) {return 8;}
1390 //|_--INLINE int NM_CAPACITY(nm16*) {return 4;}
1391 //|_--INLINE int NM_CAPACITY(nm32*) {return 2;}
1392 //|_--INLINE int NM_CAPACITY(nm64*) {return 1;}
1393
1394 //|_*|
1395 //|_*|
1396 //|_*|
1397 //|_*|
1398 //|_*|
1399 //|_*|
1400 //|_*|
1401 //|_*|
1402 //|_*|
1403 //|_*|
1404 //|_*|
1405 //|_*|
1406 //|_*|
1407 //|_*|
1408 //|_*|
1409 //|_*|
1410 //|_*|
1411 //|_*|
1412 //|_*|
1413 //|_*|
1414 //|_*|
1415 //|_*|
1416 //|_*|
1417 //|_*|
1418 //|_*|
1419 //|_*|
1420 //|_*|
1421 //|_*|
1422 //|_*|
1423 //|_*|
1424 //|_*|
1425 //|_*|
1426 //|_*|
1427 //|_*|
1428 #define CAPACITY_nm64s 1
1429 #define CAPACITY_nm32s 2
1430 #define CAPACITY_nm16s 4
1431 #define CAPACITY_nm8s 8
1432 #define CAPACITY_nm4s 16
1433 #define CAPACITY_nm2s 32
1434 #define CAPACITY_nm1 64
1435
1436 //|#ifdef __NM__
1437 #define NM16S(vec,size) nm64s vec##__64s[size/4]; nm16s* vec=(nm16s*)vec##__64s
1438 //|#else
1439 //|#define NM16S(vec,size) nm16s vec[size];
1440 //|#endif
1441
1442 //|#ifdef __NM__
1443 #define NM32S(vec,size) nm64s vec##__64s[size/2]; nm32s* vec=(nm32s*)vec##__64s
1444 //|#else
1445 //|#define NM32S(vec,size) nm32s vec[size];
1446 //|#endif
1447
1448 /**
1449 #ifdef __NM__

```

```

1450 #define MTR16S(mtr,ydim,xdim) static nm64s mtr##__64s[ydim][xdim/4]; nm16s* mtrvec=(nm16s*)vec##__64s
1451 #else
1452 #define MTR16S(mtr,ydim,xdim) nm16s vec[size];
1453 #endif
1454 */
1455 //
1456 //
1457 #ifndef __cplusplus
1458 #define __INLINE__ static inline // because 1>d:\git\nmpp\include\nmchar.h(17) : error C2574 :
1459 //'#define __INLINE__ inline
1460 #else
1461 #ifdef __NM__
1462 #define __INLINE__ static inline
1463 #else
1464 #define __INLINE__ static __inline
1465 #endif
1466 #endif
1467
1468 #define NM16Sx4(x0,x1,x2,x3) ((long long)(x3&0xFFFF)<<48)| \
1469 ((long long)(x2&0xFFFF)<<32)| \
1470 ((long long)(x1&0xFFFF)<<16)| \
1471 ((long long)(x0&0xFFFF))
1472 #define NM32Sx2(x0,x1) ((long long)(unsigned(x1))<<32)| \
1473 ((long long)(unsigned(x0)))
1474 #define NM32Sx4(x0,x1,x2,x3) NM32Sx2(x0,x1),NM32Sx2(x2,x3)
1475
1476 #endif
1477
1478 // _NMTYPE_H_INCLUDED_

```

7.99 vCore.h

```

1
2
90 //-----
91
92
93
94 //-----
139
140 void vec_0_sub_data(nmreg nb1, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
142
143
144 //-----
187
188 void vec_activate_data(nmreg f1cr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
190
191
192 //-----
237
238 void vec_activate_data_add_0(nmreg f1cr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
240
241
242 //-----
288
289 void vec_activate_data_xor_data(nmreg f1cr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
291
292
293 //-----
346
347 void vec_activate_data_add_ram(nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg
     gr6);
349
350
351 //-----
418
419 void vec_Add_VV_shift(nmreg nb1, nmreg sb, nmreg woper, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4,
     nmreg gr5, nmreg ar6, nmreg gr6);
421
422
423 //-----
461
462 void vec_afifo(nmreg ar0, nmreg gr5, nmreg ar6, nmreg gr6);
464
465
466 //-----
507
508 void vec_data(nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
510
511 //-----
544
545 void vec_CopyEvenToEven_32f(nmreg ar0, nmreg ar6, nmreg gr5);
547

```

```

548 //-----
549 //-----
583
584 void vec_CopyOddToEven_32f(nmreg ar0, nmreg ar6, nmreg gr5);
586
587
588 //-----
633
634 void vec_data_add_afifo(nmreg nb1, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6);
636
637
638 //-----
687
688 void vec_data_add_ram(nmreg nb1, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg gr6);
690
691
692 //-----
737
738 void vec_data_and_ram(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg gr6);
740
741
742 //-----
787
788 void vec_data_or_ram(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg gr6);
790
791
792 //-----
841
842 void vec_data_sub_ram(nmreg nb1, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg gr6);
844
845
846 //-----
891
892 void vec_data_xor_ram(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5, nmreg ar6, nmreg gr6);
894
895
896 //-----
897
909 void vec_FilterCoreRow2(nmreg ar0, nmreg ar4, nmreg ar6, nmreg gr1, nmreg gr4, nmreg gr6);
911
912
913 //-----
923
924 void vec_FilterCoreRow4(nmreg ar0, nmreg ar4, nmreg ar6, nmreg gr1, nmreg gr4, nmreg gr6);
926
927
928 //-----
938
939 void vec_FilterCoreRow8(nmreg ar0, nmreg ar4, nmreg ar6, nmreg gr1, nmreg gr4, nmreg gr6);
941
942
943 //-----
992
993 void vec_And(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg gr5, nmreg ar6, nmreg gr6);
995
996
997 //-----
1054
1055 void vec_Mask(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar2, nmreg gr2, nmreg gr5, nmreg ar6, nmreg gr6);
1057
1058
1059 //-----
1109
1110 void vec_Or(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg gr5, nmreg ar6, nmreg gr6);
1112
1113
1114 //-----
1164
1165 void vec_Xor(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg gr5, nmreg ar6, nmreg gr6);
1167
1168
1169 //-----
1225
1226 void vec_Abs(nmreg nb1, nmreg sb, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
1228
1229
1230 //-----
1284
1285 void vec_Add(nmreg nb1, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg gr5, nmreg ar6, nmreg gr6);
1287
1288
1289 //-----
1356
1357 void vec_ClipExt(nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg ar2, nmreg ar3, nmreg gr5, nmreg ar6,
           nmreg gr6);
1359
1360

```

```

1361 //-----
1434
1435 void vec_ClipMul2D2W8_AddVr(nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg
    gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1437
1438
1439 //-----
1440
1441
14534 void vec_ClipMulNDNW2_AddVr(nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg
    gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1536
1537
1538 //-----
1539
1540
1632 void vec_ClipMulNDNW4_AddVr(nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg
    gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1634
1635 // \restr
1636 // \ru При выходе из функции изменяется содержимое регистров: ar0,ar1,gr1,ar2,gr2,ar3,gr3,ar4,ar6,gr7.
1637 // \en On exiting the function the contents of registers changes: ar0,ar1,gr1,ar2,gr2,ar3,gr3,ar4,ar6,gr7.
1638
1639
1640 //-----
1641
1642
1732 void vec_ClipMulNDNW8_AddVr(nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg
    gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1734
1735
1736 //-----
1788
1789 void vec_IncNeg(nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
1791
1792
1793 //-----
1862
1863 void vec_Mul2D2W1_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4,
    nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1865
1866
1867 //-----
1936
1937 void vec_Mul2D2W2_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4,
    nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
1939
1940
1941 //-----
2010
2011 void vec_Mul2D2W4_AddVr(nmreg nb1, nmreg sb, nmreg f1cr, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1,
    nmreg ar4, nmreg gr5, nmreg ar6, nmreg gr6);
2013
2014
2015 //-----
2084
2085 void vec_Mul2D2W8_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4,
    nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
2087
2088
2089 //-----
2166
2167 void vec_Mul3D3W2_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar2,
    nmreg gr2, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
2169
2170
2171
2172 //-----
2249
2250 void vec_Mul3D3W8_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar2,
    nmreg gr2, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
2252
2253
2254
2255 //-----
2340
2341 void vec_Mul4D4W2_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar2,
    nmreg gr2, nmreg ar3, nmreg gr3, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
2343
2344
2345 //-----
2408
2409 void vec_MulVN_AddVN(nmreg nb1, nmreg sb, nmreg f1cr, nmreg woper, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1,
    nmreg gr5, nmreg ar6, nmreg gr6);
2411
2412
2413 //-----

```

```

2464 void vec_Sub(nmreg nb1, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg gr5, nmreg ar6, nmreg gr6);
2467
2468
2469 //-----
2534
2535 void vec_SubAbs(nmreg nb1, nmreg sb, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4, nmreg gr5,
2536   nmreg ar6, nmreg gr6);
2537
2538
2539 //*****
2540
2610 void vec_SubVN_Abs(nmreg nb1, nmreg sb, nmreg f1cr, nmreg woper, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr5,
2611   nmreg ar6, nmreg gr6);
2612
2613
2614 //-----
2678
2679 void vec_Swap(nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar4, nmreg gr4, nmreg gr5, nmreg ar6, nmreg gr6);
2681
2682
2683 //-----
2755
2756 void vec_MUL_2V4toW8_shift(nmreg nb1, nmreg sb, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar3, nmreg gr4,
2757   nmreg ar5, nmreg gr5, nmreg ar6, nmreg gr6);
2758
2759
2760 //-----
2832
2833 void vec_MUL_2V8toW16_shift(nmreg nb1, nmreg sb, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar3, nmreg
2834   gr4, nmreg ar5, nmreg gr5, nmreg ar6, nmreg gr6);
2835
2836
2837 //-----
2877
2878 void vec_not_data(nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
2880
2881
2882 //-----
2923
2924 void vec_ram(nmreg ar0, nmreg gr5, nmreg ar6, nmreg gr6);
2926
2927
2928 //-----
2973
2974 void vec_ram_sub_data(nmreg nb1, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
2976
2977
2978 //-----
3036
3037 void vec_vsum_activate_data_0(nmreg nb1, nmreg sb, nmreg f1cr, nmreg woper, nmreg ar0, nmreg gr0, nmreg ar1, nmreg
3038   gr1, nmreg gr5, nmreg ar6, nmreg gr6);
3039
3040
3041 //-----
3089
3090 void vec_vsum_data_0(nmreg nb1, nmreg sb, nmreg woper, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg gr6);
3092
3093
3094 //-----
3147
3148 void vec_vsum_data_afifo(nmreg nb1, nmreg sb, nmreg woper, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6);
3150
3151
3152 //-----
3208
3209 void vec_vsum_data_vr(nmreg nb1, nmreg sb, nmreg woper, nmreg vr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6,
3210   nmreg gr6);
3211
3212
3213 //-----
3261
3262 void vec_vsum_shift_data_0(nmreg nb1, nmreg sb, nmreg woper, nmreg ar0, nmreg gr0, nmreg gr5, nmreg ar6, nmreg
3263   gr6);
3264
3265
3266 //-----
3318
3319 void vec_vsum_shift_data_vr(nmreg nb1, nmreg sb, nmreg woper, nmreg vr, nmreg ar0, nmreg gr0, nmreg gr5, nmreg
3320   ar6, nmreg gr6);
3321
3322 //-----
3373
3374 void vec_vsum_shift_data_afifo(nmreg nb1, nmreg sb, nmreg f1cr, nmreg woper, nmreg ar0, nmreg gr0, nmreg gr5, nmreg
3375   ar6);
3376
3377 //-----

```

```

3378
3379
3380
3436 void vec_CompareMinV(nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar3, nmreg gr3, nmreg
3437     gr5, nmreg ar6, nmreg gr6);
3438 //-----
3494 void vec_CompareMaxV(nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar1, nmreg gr1, nmreg ar3, nmreg gr3,
3495     nmreg gr5, nmreg ar6, nmreg gr6);
3497
3498
3499 //-----
3500
3534 void vec_DupValueInVector8(nmreg ar1, nmreg gr1);
3536
3537
3538 //-----
3539
3572 void vec_DupValueInVector16(nmreg ar1, nmreg gr1);
3574
3575
3576 //-----
3577
3601 void vec_BuildDiagWeights8(nmreg ar1, nmreg gr1);
3603
3604
3605 //-----
3606
3607
3630 void vec_BuildDiagWeights16(nmreg ar1, nmreg gr1);
3632
3633 //-----
3634
3635
3659 void vec_MaxVal_v8nm8s(nmreg ar0, nmreg gr7);
3661
3662 //-----
3684
3685 void vec_MaxVal_v4nm16s(nmreg ar0, nmreg gr7);
3687
3688 //-----
3689
3690
3691
3736 void vec_MaxVal( nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar4, nmreg gr5, nmreg ar6);
3738
3739 //-----
3740
3741
3765 void vec_MinVal_v8nm8s(nmreg ar0, nmreg gr7);
3767
3768 //-----
3790
3791 void vec_MinVal_v4nm16s(nmreg ar0, nmreg gr7);
3793
3794 //-----
3795
3796
3797
3840 void vec_MinVal( nmreg nb1, nmreg f1cr, nmreg ar0, nmreg gr0, nmreg ar4, nmreg gr5, nmreg ar6);
3842
3843
3844 //-----
3845
3846
3911 void vec_AccMul1D1W32_AddVr(nmreg nb1, nmreg sb, nmreg vr, nmreg ar0, nmreg gr0, nmreg ar4, nmreg gr4, nmreg
3912     gr5, nmreg ar6, nmreg gr6);
3913
3914
3915
3916

```

7.100 ownmalloc.h

7.101 ringbuffer_.h

```

1 #ifndef RING_INCLUDED
2 #define RING_INCLUDED
3 #include <string.h>

```

```

4 #include <time.h>
5
6 #ifndef __NM
7 #undef __CLOCKS_PER_SEC
8 #define CLOCKS_PER_SEC 320000000
9 #endif
10
11
12 #define MIN(a,b) ((a) > (b) ? (b) : (a))
13
14 #define RING_BUFFER DECLARED 0xDEC1A8ED
15 #define RING_BUFFER ALLOCATED 0xA10CA7ED
16 #define RING_BUFFER MEM_ERROR 0xDEADF00D
17 #define RING_BUFFER_ERROR 0xBEDABEDA
18 #define RING_BUFFER_OK 0x0
19
20 #define EXIT_ON_TIMEOUT 1
21 #define EXIT_ON_COMPLETED 0
22 #define EXIT_ON_FULL_BUFFER 2
23 #define EXIT_ON_EMPTY_BUFFER 2
24
25 #ifdef __NM
26 #define bytesizeof(T) (sizeof(T)«2)
27 #else
28 #define bytesizeof(T) sizeof(T)
29 #endif
30
31 typedef void *(*memcpy_ptr)(void *, void const *, size_t);
32 typedef void *(*t_bytcpy)(void *to, int toIndex, void const *from, int fromIndex, size_t size);
33 template <class T> class C_RingBuffer{
34     public:
35         T* data;
36         size_t size;
37         size_t head;
38         size_t tail;
39         size_t* head_addr;
40         size_t* tail_addr;
41         size_t id;
42
43         t_bytcpy push_memcpy;
44         t_bytcpy pop_memcpy;
45
46         // ----- dma -----
47         memcpy_ptr dma_init;
48         size_t(*dma_check)();
49         T* dma_ptr;
50         size_t dma_left;
51         size_t dma_size;
52
53         unsigned timeout; // msec
54         unsigned time2sleep; // msec
55         int pad[16-3-sizeof(memcpy_ptr)/sizeof(int)];
56
57         void Sleep(clock_t msec){
58             #ifdef __NM
59                 clock_t dt=msec*(CLOCKS_PER_SEC/1000);
60                 clock_t t0=clock();
61                 while(clock()-t0<dt){
62                     _asm{
63                         rep 32 with vfalse;
64                         rep 32 with afifo and afifo;
65                         rep 32 with afifo and afifo;
66                         rep 32 with afifo and afifo;
67                         rep 32 with afifo and afifo;
68                         rep 32 with afifo and afifo;
69                         rep 32 with afifo and afifo;
70                         rep 32 with afifo and afifo;
71                         rep 32 with afifo and afifo;
72                         rep 32 with afifo and afifo;
73                         rep 32 [ar7] = afifo;
74                     }
75                 }
76
77             #else
78                 ::Sleep(msec);
79             #endif
80         }
81         C_RingBuffer(){
82             //sizeof_size_t=sizeof(size_t);
83             //sizeof_T=sizeof(T);
84             //Init(0,0,(memcpy_ptr)memcpy,(memcpy_ptr)memcpy);
85             Init(0,0,0,0);
86         }
87         C_RingBuffer(T* buffer,size_t count, t_bytcpy pushMemcpy, t_bytcpy popMemcpy ){
88             Init(buffer,count,pushMemcpy,popMemcpy);
89         }
90     }

```

```

91 //void Init(T* buffer,size_t count, memcpy_ptr pushmemcpy, memcpy_ptr popmemcpy ){
92 bool Init(T* buffer,size_t count, t_bytcpy pushmemcpy, t_bytcpy popmemcpy ){
93     if (buffer==0) return false;
94     if (count&(count-1)!=0) return false ;
95     data=buffer;
96     size=count;
97     head=0;
98     tail=0;
99     head_addr=&head;
100    tail_addr=&tail;
101    id=0x600DB00F;
102    push_memcpy=(t_bytcpy)pushmemcpy;
103    pop_memcpy=(t_bytcpy)popmemcpy;
104    dma_init=0;
105    dma_check=0;
106    dma_ptr =0;
107    dma_left=0;
108    dma_size=0;
109    time2sleep=1; // msec
110    timeout=100; //
111    return true;
112 }
113 }
114
115    __INLINE__ bool IsFull(){
116        return (head-tail==size);
117    }
118    __INLINE__ bool IsEmpty(){
119        return (head==tail);
120    }
121    __INLINE__ size_t PushAvail(){
122        return (size-(head-tail));
123    }
124    __INLINE__ size_t PopAvail(){
125        return (head-tail);
126    }
127
128 T* Head(){
129     if (PushAvail() ==0)
130         return 0;
131     size_t headPointer=head&(size-1);
132     T* pElement=data+headPointer;
133     return pElement;
134 }
135 T* Tail(){
136     if (PopAvail() ==0)
137         return 0;
138     size_t tailPointer=tail&(size-1);
139     T* pElement=data+tailPointer;
140     return pElement;
141 }
142
143
144 bool MoveHead(int numElements){
145     size_t h=head+numElements;
146     if (h>tail+size)
147         return RING_BUFFER_ERROR;
148     head=h;
149     return RING_BUFFER_OK;
150 }
151
152 bool MoveTail(int numElements){
153     size_t t=tail+numElements;
154     if (t>head)
155         return RING_BUFFER_ERROR;
156     tail=t;
157     return RING_BUFFER_OK;
158 }
159
160
161 size_t Push(T* pSrcElements, size_t numElements, int ExitMode= EXIT_ON_COMPLETED)
162 {
163     size_t initHead=head;
164     size_t count;
165     size_t diff;
166     clock_t t0=clock();
167     size_t posSrc=0;
168     while (numElements){
169         size_t Tail=tail;
170         size_t posHead=head&(size-1);
171         size_t posTail=Tail&(size-1);
172         if (posTail<posHead || head==Tail)
173             diff=size-posHead;
174         else if (posHead<posTail)
175             diff= posTail-posHead;
176         else if (ExitMode&EXIT_ON_FULL_BUFFER) // buffer is full
177             return (head-initHead);

```

```

178     else if (ExitMode&&EXIT_ON_TIMEOUT){
179         if ((clock()-t0)>timeout*(CLOCKS_PER_SEC/1000))
180             return (head-initHead);
181         Sleep(time2sleep);
182         continue;
183     } else { // EXIT_ON_COMPLETED
184         Sleep(time2sleep);
185         continue;
186     }
187
188     count=MIN(numElements,diff);
189     if (push_memcpy(data,posHead*bytesizeof(T),pSrcElements, posSrc*bytesizeof(T),count*bytesizeof(T))){
190         head+=count;
191         posSrc+=count;
192         numElements -=count;
193     }
194 }
195 return (head-initHead);
196
197 size_t Push(T pSrcElement)
198 {
199     return Push(&pSrcElement,1,EXIT_ON_COMPLETED);
200 }
202
203 void PushRequest(T* pSrcElements, size_t numElements)
204 {
205     size_t posHead;
206     size_t posTail;
207     size_t diff=0;
208     while(diff==0){
209         posHead=head&(size-1);
210         posTail=tail&(size-1);
211         if (posTail<posHead || head==tail)
212             diff=size-posHead;
213         else if (posHead<posTail)
214             diff=posTail-posHead;
215         else { // buffer is full
216             Sleep(time2sleep);
217             continue;
218         }
219     }
220     dma_ptr =pSrcElements;
221     dma_left =numElements;
222     dma_size =MIN(dma_left,diff);
223     //dma_init(data,posHead,pSrcElements,0,dma_size*sizeof(T));
224 }
225
226
227 bool isPushCompleted()
228 {
229     if (dma_left==0)
230         return true;
231     if (dma_check())
232         return false;
233
234     head +=dma_size;
235     dma_ptr +=dma_size;
236     dma_left -=dma_size;
237     if (dma_left){
238         PushRequest(dma_ptr,dma_left); //?
239         return false;
240     }
241     return true;
242 }
243 }
244
245
246
247
248
249
250 size_t Pop(T* pDstElements, size_t numElements, int ExitMode= EXIT_ON_COMPLETED)
251 {
252     size_t initTail=tail;
253     size_t count;
254     size_t diff;
255     clock_t t0=clock();
256     size_t posDst=0;
257     while (numElements){
258         size_t Head=head;
259         size_t posHead=Head&(size-1);
260         size_t posTail=tail&(size-1);
261         if (posTail<posHead )
262             diff= posHead-posTail;
263         else if (posHead<=posTail && Head!=tail)
264             diff=size-posTail;
265         else if (ExitMode&&EXIT_ON_EMPTY_BUFFER) // buffer is empty

```

```

266         return (*tail->initTail);
267     else if ((ExitMode&EXIT_ON_TIMEOUT) &
268             ((clock()-t0)>timeout))
269         return (*tail->initTail);
270     Sleep(time2sleep);
271     continue;
272 } else {
273     //Sleep(time2sleep);
274     continue;
275 }
276
277 count=MIN(numElements,diff);
278 pop_memcpy(pDstElements,pDst*bytesizeof(T),data,posTail*bytesizeof(T),count*bytesizeof(T));
279 tail+=count;
280 posDst+=count;
281 numElements -=count;
282 }
283 return (*tail->initTail);
284 }
285
286
287
288
289
290
291
292 };
293
294
295 #endif

```

7.102 ringremote.h

```

1 #ifndef RINGREMOTE_INCLUDED
2 #define RINGREMOTE_INCLUDED
3 #include <string.h>
4 #include <time.h>
5
6 #define MIN(a,b) ((a) > (b) ? (b) : (a))
7
8 #define RING_BUFFER_DECLARED 0xDEC1A8ED
9 #define RING_BUFFER_ALLOCATED 0xA10CA7ED
10 #define RING_BUFFER_MEM_ERROR 0xDEADF00D
11 #define RING_BUFFER_ERROR 0xBEDABEDA
12 #define RING_BUFFER_OK 0x0
13
14 #define EXIT_ON_TIMEOUT 1
15 #define EXIT_ON_COMPLETED 0
16 #define EXIT_ON_FULL_BUFFER 2
17 #define EXIT_ON_EMPTY_BUFFER 2
18
19 //void Sleep(int msec);
20
21 typedef void *(*t_memcpy)(void *to, int toIndex, const void *from, int fromIndex, size_t size);
22 typedef void *(*t_bytcpy)(void *to, int toIndex, void const *from, int fromIndex, size_t size);
23
24 #ifdef __NM_
25 #define bytesizeof(T) (sizeof(T)«2)
26 #else
27 #define bytesizeof(T) sizeof(T)
28 #endif
29
30 #if defined(_WIN32) && !defined(__GNUC__)
31 #include "crtdbg.h"
32 #else
33 #define _ASSERT(val)
34 #endif
35
36
37
38
39 template <class T> class C_RingBufferRemote{
40 private:
41
42 public:
43     //size_t sizeof_size_t;
44     //size_t sizeof_T;
45     size_t data_addr;
46     size_t head_addr;
47     size_t tail_addr;
48
49     size_t size;
50     size_t head;
51     size_t tail;

```

```

52     size_t id;
53     bool isConnected;
54
55
56     t_bytcpy push_memcpy;
57     t_bytcpy pop_memcpy;
58
59 // ----- dma -----
60     t_memcpy dma_init;
61     size_t(*dma_check)();
62     T* dma_ptr;
63     size_t dma_left;
64     size_t dma_size;
65 //-----
66     unsigned timeout;
67     unsigned time2sleep;
68     int pad[16-3* sizeof(t_bytcpy)/sizeof(int)];
69
70
71
72     C_RingBufferRemote(){
73         isConnected=false;
74     }
75
76     C_RingBufferRemote(size_t ringbuffer_addr, t_bytcpy push_memcpy, t_bytcpy pop_memcpy){
77         Init( ringbuffer_addr, push_memcpy, pop_memcpy);
78     }
79
80     size_t GetHead(){
81         pop_memcpy(&head,0,(const void*)head_addr,0,bytesizeof(size_t));
82         return head;
83     }
84     size_t GetTail(){
85         pop_memcpy(&tail,0,(const void*)tail_addr,0,bytesizeof(size_t));
86         return tail;
87     }
88     void SetHead(){
89         push_memcpy((void*)head_addr,0,&head,0, bytesizeof(size_t));
90     }
91     void SetTail(){
92         push_memcpy((void*)tail_addr,0,&tail,0,bytesizeof(size_t));
93     }
94
95     bool Init(size_t ringbuffer_addr, t_bytcpy push_memcpy, t_bytcpy pop_memcpy){
96         //sizeof_size_t=0;
97         //sizeof_T=0;
98         data_addr=0xCCCCCCCC;
99         head_addr=0xCCCCCCCC;
100        tail_addr=0xCCCCCCCC;
101        size=0xCCCCCCCC;
102        head=0xCCCCCCCC;
103        tail=0xCCCCCCCC;
104        isConnected=false;
105
106        dma_init=0;
107        dma_check=0;
108        dma_ptr =0;
109        dma_left=0;
110        dma_size=0;
111        time2sleep=10;
112        timeout=100;
113
114
115        push_memcpy=push_memcpy;
116        pop_memcpy =pop_memcpy;
117
118        pop_memcpy(&data_addr,0,(void*)ringbuffer_addr,+0*bytesizeof(size_t),bytesizeof(size_t));
119        pop_memcpy(&size, 0,(void*)ringbuffer_addr,+1*bytesizeof(size_t),bytesizeof(size_t));
120        pop_memcpy(&head_addr,0,(void*)ringbuffer_addr,+4*bytesizeof(size_t),bytesizeof(size_t));
121        pop_memcpy(&tail_addr,0,(void*)ringbuffer_addr,+5*bytesizeof(size_t),bytesizeof(size_t));
122        pop_memcpy(&id, 0,(void*)ringbuffer_addr,+6*bytesizeof(size_t),bytesizeof(size_t));
123
124        if ((size&(size-1))!=0) // check that size is power of two
125            return false;
126
127
128        if (id!=0x600DB00F)
129            return false;
130
131        id = 0xB00F600D;
132        push_memcpy((void*)ringbuffer_addr,+6*bytesizeof(size_t),&id,0,bytesizeof(size_t));
133        id =0;
134        pop_memcpy(&id, 0,(void*)ringbuffer_addr,+6*bytesizeof(size_t),bytesizeof(size_t));
135        if (id!=0xB00F600D) // no access to remote rinbuffer.id
136            return false;
137
138

```

```

139     GetHead();
140     GetTail();
141     if (head==0xCCCCCCCC || tail==0xCCCCCCCC)
142         return false;
143
144
145     isConnected = true;
146     return true;
147 }
148
149 --INLINE_ bool IsFull(){
150     GetHead();
151     GetTail();
152     return (head-tail==size);
153 }
154 --INLINE_ bool IsEmpty(){
155     GetHead();
156     GetTail();
157     return (head==tail);
158 }
159 --INLINE_ size_t GetWriteAvail(){
160     GetHead();
161     GetTail();
162     return (size-(head-tail));
163 }
164 --INLINE_ size_t GetReadAvail(){
165     GetHead();
166     GetTail();
167     return (head-tail);
168 }
169
170 /*
171 T* Head(){
172     if (GetWriteAvail()==0)
173         return 0;
174     size_t headPointer=head&(size-1);
175     T* pElement=(T*)(data_addr+headPointer*bytesizeof(T));
176     return pElement;
177 }
178 T* Tail(){
179     if (GetReadAvail()==0)
180         return 0;
181     size_t tailPointer=tail&(size-1);
182     T* pElement=(T*)(data_addr+tailPointer*sizeof_T);
183     return pElement;
184 }*/
185
186     bool Push(int numElements){
187         GetHead();
188         GetTail();
189         size_t h=head+numElements;
190         if (h>tail+size)
191             return RING_BUFFER_ERROR;
192         head=h;
193         return RING_BUFFER_OK;
194     }
195
196     bool Pop(int numElements){
197         GetHead();
198         GetTail();
199         size_t t=tail+numElements;
200         if (t>head)
201             return RING_BUFFER_ERROR;
202         tail=t;
203         return RING_BUFFER_OK;
204     }
205
206
207     size_t Push(T* pSrcElements, size_t numElements, int ExitMode= EXIT_ON_COMPLETED)
208 {
209     GetHead();
210     size_t initHead=head;
211     size_t count;
212     size_t diff;
213     clock_t t0=clock();
214     size_t posSrc=0;
215     while (numElements){
216         GetHead();
217         GetTail();
218         _ASSERTE(head>=tail);
219         size_t posHead=head&(size-1);
220         size_t posTail=tail&(size-1);
221         if (posTail<posHead || head==tail)
222             diff=size-posHead;
223         else if (posHead<posTail)
224             diff= posTail-posHead;
225         else if (ExitMode&EXIT_ON_FULL_BUFFER) // buffer is full

```

```

226     return (head.initHead);
227     else if (ExitMode&EXIT_ON_TIMEOUT){
228         if ((clock()-t0)>timeout*(CLOCKS_PER_SEC/1000))
229             return (head.initHead);
230         Sleep(time2sleep);
231         continue;
232     } else {
233         Sleep(time2sleep);
234         continue;
235     }
236
237     count=MIN(numElements,diff);
238     if
239         (push_memcp((void*)data_addr, posHead*bytesizeof(T), pSrcElements, posSrc*bytesizeof(T), count*bytesizeof(T))){  

240             head+=count;
241             posSrc+=count;
242             numElements -=count;
243             SetHead();
244             GetHead();
245         }
246     }
247     return (head.initHead);
248 }
249
250
251 /*
252 void PushRequest(T* pSrcElements, size_t numElements)
253 {
254     GetHead();
255     GetTail();
256     size_t posHead;
257     size_t posTail;
258     size_t diff=0;
259     while(diff==0){
260         posHead=head&(size-1);
261         posTail=tail&(size-1);
262         if (posTail<posHead || head==tail)
263             diff=size-posHead;
264         else if (posHead<posTail)
265             diff=posTail-posHead;
266         else { // buffer is full
267             Sleep(time2sleep);
268             continue;
269         }
270     }
271     dma_ptr =pSrcElements;
272     dma_left =numElements;
273     dma_size =MIN(dma_left,diff);
274     dma_init(data_addr+posHead*sizeof_T,pSrcElements,dma_size*sizeof_T);
275 }
276
277 bool isPushCompleted()
278 {
279     SetHead();
280     if (dma_left==0)
281         return true;
282     if (dma_check())
283         return false;
284
285     head +=dma_size;
286     dma_ptr +=dma_size;
287     dma_left -=dma_size;
288     SetHead();
289     if (dma_left){
290         PushRequest(dma_ptr,dma_left);
291         return false;
292     }
293     return true;
294 }
295 */
296
297 */
298
299
300
301
302     size_t Pop(T* pDstElements, size_t numElements, int ExitMode= EXIT_ON_COMPLETED)
303 {
304     GetTail();
305     size_t initTail=tail;
306     size_t count;
307     size_t diff;
308     clock_t t0=clock();
309     size_t posDst=0;
310     while (numElements){
311         //GetTail();

```

```

312     GetHead();
313     size_t posHead=head&(size-1);
314     size_t posTail=tail&(size-1);
315     if (posTail<posHead)
316         diff=posHead-posTail;
317     else if (posHead<=posTail && head!=tail)
318         diff=size-posTail;
319     else if ((ExitMode&EXIT_ON_EMPTY_BUFFER) // buffer is empty
320             || (ExitMode&EXIT_ON_TIMEOUT){
321         if ((clock()-t0)>timeout)
322             return (tail-initTail);
323         Sleep(time2sleep);
324         continue;
325     } else {
326         Sleep(time2sleep);
327         continue;
328     }
329
330     count=MIN(numElements,diff);
331     pop_memcpy(pDstElements,posDst*bytesizeof(T),(const
332 void*)(data_addr),posTail*bytesizeof(T),count*bytesizeof(T));
333     tail+=count;
334     posDst+=count;
335     numElements -=count;
336     SetTail();
337 }
338     return (tail-initTail);
339 }
340
341 size_t View(T* pDstElements, size_t numElements, int ExitMode= EXIT_ON_COMPLETED)
342 {
343     GetTail();
344     size_t initTail=tail;
345     size_t count;
346     size_t diff;
347     clock_t t0=clock();
348     size_t posDst=0;
349     while (numElements){
350         //GetTail();
351         GetHead();
352         size_t posHead=head&(size-1);
353         size_t posTail=tail&(size-1);
354         if (posTail<posHead)
355             diff=posHead-posTail;
356         else if (posHead<=posTail && head!=tail)
357             diff=size-posTail;
358         else if ((ExitMode&EXIT_ON_EMPTY_BUFFER){ // buffer is empty
359             size_t ret=tail-initTail;
360             tail=initTail;
361             return ret;
362         }
363         else if ((ExitMode&EXIT_ON_TIMEOUT){
364             if ((clock()-t0)>timeout){
365                 size_t ret=tail-initTail;
366                 tail=initTail;
367                 return ret;
368             }
369             Sleep(time2sleep);
370             continue;
371         } else {
372             Sleep(time2sleep);
373             continue;
374         }
375
376         count=MIN(numElements,diff);
377         pop_memcpy(pDstElements,posDst*bytesizeof(T),(const
378 void*)data_addr,posTail*bytesizeof(T),count*bytesizeof(T));
379         tail+=count;
380         posDst+=count;
381         numElements -=count;
382         //SetTail();
383     }
384     size_t ret=tail-initTail;
385     tail=initTail;
386     return ret;
387 }
388
389
390 };
391
392
393 #endif

```

7.103 rpc-host.h

```

1 #ifdef RPC
2 #include <aura/aura.h>
3 #include <ion/ion.h>
4 #include <memory.h>
5
6
7 extern struct aura_node *n;
8 /*
9 #define RPC_HOST_P(func) \
10 int ret; \
11 struct aura_buffer *iobuf_src = aura_buffer_request(n, size); \
12 memcpy(iobuf_src->data,src,size); \
13 struct aura_buffer *retbuf; \
14 ret = aura_call(n, func, &retbuf, iobuf_src, iobuf_dst, size); \
15 if (ret != 0) \
16 BUG(n, "Call func failed!"); \
17 memcpy(dst,iobuf_dst->data,size); \
18 aura_buffer_release( iobuf_src); \
19 aura_buffer_release( retbuf); \
20 slog(3, SLOG_INFO, "ARM: Call func ok");
21
22
23 #define RPC_HOST_PI(func,ptr) \
24 int ret; \
25 struct aura_buffer *iobuf_src = aura_buffer_request(n, size); \
26 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size); \
27 memcpy(iobuf_src->data,src,size); \
28 struct aura_buffer *retbuf; \
29 ret = aura_call(n, func, &retbuf, iobuf_src, iobuf_dst, size); \
30 if (ret != 0) \
31 BUG(n, "Call func failed!"); \
32 memcpy(dst,iobuf_dst->data,size*k); \
33 aura_buffer_release( iobuf_dst); \
34 aura_buffer_release( iobuf_src); \
35 aura_buffer_release( retbuf); \
36 slog(3, SLOG_INFO, "ARM: Call " #func " -ok");
37 */
38 #define RPC_HOST_I(func, val) \
39 int ret; \
40 struct aura_buffer *retbuf; \
41 ret = aura_call(n, func, &retbuf, val); \
42 if (ret != 0) \
43 BUG(n, "Call" #func "failed!"); \
44 aura_buffer_release( retbuf); \
45 slog(3, SLOG_INFO, "ARM: Call func ok");
46
47 #define RPC_HOST_PPI(func,src,dst,size,k) \
48 int ret; \
49 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
50 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k); \
51 memcpy(iobuf_src->data,src,size*k); \
52 struct aura_buffer *retbuf; \
53 ret = aura_call(n, func, &retbuf, iobuf_src, iobuf_dst, size); \
54 if (ret != 0) \
55 printf ("bug = %d\r\n",ret); \
56 BUG(n, "Call " #func " failed!"); } \
57 memcpy(dst,iobuf_dst->data,size*k); \
58 aura_buffer_release( iobuf_dst); \
59 aura_buffer_release( iobuf_src); \
60 aura_buffer_release( retbuf); \
61 slog(3, SLOG_INFO, "ARM: Call " #func " -ok");
62
63 #define RPC_HOST_PPI_R64_k0k1(func,src0,src1,size,ret,k0,k1) \
64 int ret; \
65 struct aura_buffer *iobuf_src0 = aura_buffer_request(n, size*k0); \
66 struct aura_buffer *iobuf_src1 = aura_buffer_request(n, size*k1); \
67 memcpy(iobuf_src0->data,src0,size*k0); \
68 memcpy(iobuf_src1->data,src1,size*k1); \
69 struct aura_buffer *retbuf; \
70 ret = aura_call(n, func, &retbuf, iobuf_src0,iobuf_src1, size); \
71 if (ret != 0) \
72 printf ("bug = %d\r\n",ret); \
73 BUG(n, "Call " #func " failed!"); } \
74 *res = aura_buffer_get_u32(retbuf); \
75 aura_buffer_release( iobuf_src0); \
76 aura_buffer_release( iobuf_src1); \
77 aura_buffer_release( retbuf); \
78 slog(3, SLOG_INFO, "ARM: Call " #func " -ok");
79
80
81
82 #define RPC_HOST_PPPI(func,src0,src1,dst,size,k) \
83 int ret; \
84 struct aura_buffer *iobuf_src0 = aura_buffer_request(n, size*k); \
85 struct aura_buffer *iobuf_src1 = aura_buffer_request(n, size*k); \

```

```

86 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k); \
87 memcpy(iobuf_src0->data,src0,size*k); \
88 memcpy(iobuf_src1->data,src1,size*k); \
89 struct aura_buffer *retbuf; \
90 ret = aura_call(n, func, &retbuf, iobuf_src0, iobuf_src1, iobuf_dst, size); \
91 if (ret != 0) \
92 BUG(n, "Call "#func " failed!"); \
93     memcpy(dst,iobuf_dst->data,size*k); \
94     aura_buffer_release( iobuf_dst); \
95     aura_buffer_release( iobuf_src1); \
96     aura_buffer_release( iobuf_src0); \
97     aura_buffer_release( retbuf); \
98     slog(3, SLOG_INFO, "ARM: Call "#func " -ok");
99
100
101 #define RPC_HOST_PIPI(func,src,val,dst,size,k) \
102 int ret; \
103 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
104 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k); \
105 memcpy(iobuf_src->data,src,size*k); \
106 struct aura_buffer *retbuf; \
107 ret = aura_call(n, func, &retbuf, iobuf_src, val, iobuf_dst, size); \
108 if (ret != 0) { \
109     printf ("bug = %d\r\n",ret); \
110     BUG(n, "Call "#func " failed!"); \
111     memcpy(dst,iobuf_dst->data,size*k); \
112     aura_buffer_release( iobuf_dst); \
113     aura_buffer_release( iobuf_src); \
114     aura_buffer_release( retbuf); \
115     slog(3, SLOG_INFO, "ARM: Call "#func " -ok");
116
117 #define RPC_HOST_PIPI2(func,src,val,dst,size,k1,k2) \
118 int ret; \
119 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k1); \
120 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k2); \
121 memcpy(iobuf_src->data,src,size*k1); \
122 struct aura_buffer *retbuf; \
123 ret = aura_call(n, func, &retbuf, iobuf_src, val, iobuf_dst, size); \
124 if (ret != 0) { \
125     printf ("bug = %d\r\n",ret); \
126     BUG(n, "Call "#func " failed!"); \
127     memcpy(dst,iobuf_dst->data,size*k2); \
128     aura_buffer_release( iobuf_dst); \
129     aura_buffer_release( iobuf_src); \
130     aura_buffer_release( retbuf); \
131     slog(3, SLOG_INFO, "ARM: Call "#func " -ok");
132
133 #define RPC_HOST_PLPI(func,src,val,dst,size,k1,k2) \
134 int ret; \
135 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k1); \
136 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k2); \
137 memcpy(iobuf_src->data,src,size*k1); \
138 struct aura_buffer *retbuf; \
139 ret = aura_call(n, func, &retbuf, iobuf_src, val, iobuf_dst, size); \
140 if (ret != 0) { \
141     printf ("bug = %d\r\n",ret); \
142     BUG(n, "Call "#func " failed!"); \
143     memcpy(dst,iobuf_dst->data,size*k2); \
144     aura_buffer_release( iobuf_dst); \
145     aura_buffer_release( iobuf_src); \
146     aura_buffer_release( retbuf); \
147     slog(3, SLOG_INFO, "ARM: Call "#func " -ok");
148
149 #define RPC_HOST_PPLI(func,src,val,dst,size,k) \
150 int ret; \
151 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
152 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k); \
153 memcpy(iobuf_src->data,src,size*k); \
154 struct aura_buffer *retbuf; \
155 ret = aura_call(n, func, &retbuf, iobuf_src, iobuf_dst, val, size); \
156 if (ret != 0) { \
157     printf ("bug = %d\r\n",ret); \
158     BUG(n, "Call "#func " failed!"); \
159     memcpy(dst,iobuf_dst->data,size*k); \
160     aura_buffer_release( iobuf_dst); \
161     aura_buffer_release( iobuf_src); \
162     aura_buffer_release( retbuf); \
163     slog(3, SLOG_INFO, "ARM: Call "#func " -ok");
164
165
166 #define RPC_HOST_PIR(func,src,size,res,k) \
167 int ret; \
168 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
169 memcpy(iobuf_src->data,src,size*k); \
170 struct aura_buffer *retbuf; \
171 ret = aura_call(n, func, &retbuf, iobuf_src, size); \
172 if (ret != 0) \

```

```

173 BUG(n, "Call \"#func\" failed!"); \
174     *res = aura_buffer_get_u32(retbuf); \
175     aura_buffer_release( iobuf_src); \
176     aura_buffer_release( retbuf); \
177     slog(3, SLOG_INFO, "ARM: Call \"#func\" -ok");
178
179 #define RPC_HOST_PPI_I(func,src,dst,handle,ret,k) \
180 int ret; \
181 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
182 struct aura_buffer *iobuf_dst = aura_buffer_request(n, size*k); \
183 memcpy(iobuf_src->data,src,size*k); \
184 memcpy(iobuf_dst->data,dst,size*k); \
185 struct aura_buffer *retbuf; \
186 ret = aura_call(n, func, &retbuf, iobuf_src, iobuf_dst, handle); \
187 if (ret != 0) \
188     BUG(n, "Call \"#func\" failed!"); \
189     *res = aura_buffer_get_u32(retbuf); \
190     aura_buffer_release( iobuf_src); \
191     aura_buffer_release( iobuf_dst); \
192     aura_buffer_release( retbuf); \
193     slog(3, SLOG_INFO, "ARM: Call \"#func\" -ok");
194
195
196 #define RPC_HOST_PIR64(func,src,size,res,k) \
197 int ret; \
198 struct aura_buffer *iobuf_src = aura_buffer_request(n, size*k); \
199 memcpy(iobuf_src->data,src,size*k); \
200 struct aura_buffer *retbuf; \
201 ret = aura_call(n, func, &retbuf, iobuf_src, size); \
202 if (ret != 0) \
203     BUG(n, "Call \"#func\" failed!"); \
204     *res = aura_buffer_get_u64(retbuf); \
205     aura_buffer_release( iobuf_src); \
206     aura_buffer_release( retbuf); \
207     slog(3, SLOG_INFO, "ARM: Call \"#func\" -ok");
208
209 //memcpy(iobuf_src1->data,pSrcMtr,nHeight*nWidth*k1);
210
211 //memcpy(iobuf_src1->data,pSrcMtr1,nHeight1*nWidth1*k1); \
212 //memcpy(iobuf_src2->data,pSrcMtr2,nWidth1*nWidth2*k2 );
213 // memcpyp(pDstMtr, iobuf_dst->data, nHeight1*nWidth2*k2);
214
215
216
217
218 #define RPC_HOST_PPIR64(func,srcVec0,srcVec1, size, dst,k1,k2) \
219 int ret; \
220 struct aura_buffer *iobuf_src1 = aura_buffer_request(n, size*k1); \
221 struct aura_buffer *iobuf_src2 = aura_buffer_request(n, size*k2 ); \
222 memcpy(iobuf_src1->data,srcVec0,size*k1); \
223 memcpy(iobuf_src2->data,srcVec1,size*k2); \
224 struct aura_buffer *retbuf; \
225 ret = aura_call(n, func, &retbuf, iobuf_src1, iobuf_src2, size); \
226 if (ret != 0){ \
227     slog(3, SLOG_ERROR, "bug = %d", ret); \
228     BUG(n, "Call \"#func\" failed!"); } \
229     *dst = aura_buffer_get_u64(retbuf); \
230     aura_buffer_release( iobuf_src1); \
231     aura_buffer_release( iobuf_src2); \
232     aura_buffer_release( retbuf); \
233     slog(3, SLOG_INFO, "ARM: Call \"#func\" -ok");
234
235
236 #endif

```

7.104 rpc-nmc-func.h

```

1 #include "rpc/rpc-nmc.h"
2
3
4 //-----
5 #ifdef RPC_nmppsAbs_64s
6 NMC_RPC_PPI(nmppsAbs_64s);
7 #endif
8
9 #ifdef RPC_nmppsAbs_32s
10 NMC_RPC_PPI(nmppsAbs_32s);
11 #endif
12
13 #ifdef RPC_nmppsAbs_16s
14 NMC_RPC_PPI(nmppsAbs_16s);
15 #endif
16

```

```

17 #ifdef RPC_nmppsAbs_8s
18 NMC_RPC_PPPI(nmppsAbs_8s);
19 #endif
20 //-----
21 #ifdef RPC_nmppsAbs1_8s
22 NMC_RPC_PPPI(nmppsAbs1_8s);
23 #endif
24
25 #ifdef RPC_nmppsAbs1_16s
26 NMC_RPC_PPPI(nmppsAbs1_16s);
27 #endif
28
29 #ifdef RPC_nmppsAbs1_32s
30 NMC_RPC_PPPI(nmppsAbs1_32s);
31 #endif
32
33 #ifdef RPC_nmppsAbs1_64s
34 NMC_RPC_PPPI(nmppsAbs1_64s);
35 #endif
36
37 //-----
38 #ifdef RPC_nmppsAdd_8s
39 NMC_RPC_PPPI(nmppsAdd_8s);
40 #endif
41
42 #ifdef RPC_nmppsAdd_16s
43 NMC_RPC_PPPI(nmppsAdd_16s);
44 #endif
45
46 #ifdef RPC_nmppsAdd_32s
47 NMC_RPC_PPPI(nmppsAdd_32s);
48 #endif
49
50 #ifdef RPC_nmppsAdd_64s
51 NMC_RPC_PPPI(nmppsAdd_64s);
52 #endif
53 //-----
54 #ifdef RPC_nmppsSub_8s
55 NMC_RPC_PPPI(nmppsSub_8s);
56 #endif
57
58 #ifdef RPC_nmppsSub_16s
59 NMC_RPC_PPPI(nmppsSub_16s);
60 #endif
61
62 #ifdef RPC_nmppsSub_32s
63 NMC_RPC_PPPI(nmppsSub_32s);
64 #endif
65
66 #ifdef RPC_nmppsSub_64s
67 NMC_RPC_PPPI(nmppsSub_64s);
68 #endif
69 //-----
70 #ifdef RPC_nmppsAddC_8s
71 NMC_RPC_PIPPI(nmppsAddC_8s);
72 #endif
73
74 #ifdef RPC_nmppsAddC_16s
75 NMC_RPC_PIPPI(nmppsAddC_16s);
76 #endif
77
78 #ifdef RPC_nmppsAddC_32s
79 NMC_RPC_PIPPI(nmppsAddC_32s);
80 #endif
81
82 #ifdef RPC_nmppsAddC_64s
83 NMC_RPC_PLPI(nmppsAddC_64s);
84 #endif
85
86 //-----
87 #ifdef RPC_nmppsSubC_8s
88 NMC_RPC_PIPPI(nmppsSubC_8s);
89 #endif
90
91 #ifdef RPC_nmppsSubC_16s
92 NMC_RPC_PIPPI(nmppsSubC_16s);
93 #endif
94
95 #ifdef RPC_nmppsSubC_32s
96 NMC_RPC_PIPPI(nmppsSubC_32s);
97 #endif
98
99 #ifdef RPC_nmppsSubC_64s
100 NMC_RPC_PLPI(nmppsSubC_64s);
101 #endif
102
103 //-----

```

```

104 #ifdef RPC_nmpssSubCRev_8s
105 NMC_RPC_PIPI(nmpssSubCRev_8s);
106 #endif
107
108 #ifdef RPC_nmpssSubCRev_16s
109 NMC_RPC_PIPI(nmpssSubCRev_16s);
110 #endif
111
112 #ifdef RPC_nmpssSubCRev_32s
113 NMC_RPC_PIPI(nmpssSubCRev_32s);
114 #endif
115
116 #ifdef RPC_nmpssSubCRev_64s
117 NMC_RPC_PLPI(nmpssSubCRev_64s);
118 #endif
119
120 //-----
121 #ifdef RPC_nmpssRShiftC_8s
122 NMC_RPC_PIPI(nmpssRShiftC_8s);
123 #endif
124
125 #ifdef RPC_nmpssRShiftC_16s
126 NMC_RPC_PIPI(nmpssRShiftC_16s);
127 #endif
128
129 #ifdef RPC_nmpssRShiftC_32s
130 NMC_RPC_PIPI(nmpssRShiftC_32s);
131 #endif
132
133 #ifdef RPC_nmpssRShiftC_64s
134 NMC_RPC_PIPI(nmpssRShiftC_64s);
135 #endif
136
137 //-----
138 #ifdef RPC_nmpssMin_8s7b
139 NMC_RPC_PIR(nmpssMin_8s7b);
140 #endif
141
142 #ifdef RPC_nmpssMin_16s15b
143 NMC_RPC_PIR(nmpssMin_16s15b);
144 #endif
145
146 #ifdef RPC_nmpssMin_32s31b
147 NMC_RPC_PIR(nmpssMin_32s31b);
148 #endif
149
150 #ifdef RPC_nmpssMin_64s63b
151 NMC_RPC_PIR64(nmpssMin_64s63b);
152 #endif
153
154 //-----
155 #ifdef RPC_nmpssMax_8s7b
156 NMC_RPC_PIR(nmpssMax_8s7b);
157 #endif
158
159 #ifdef RPC_nmpssMax_16s15b
160 NMC_RPC_PIR(nmpssMax_16s15b);
161 #endif
162
163 #ifdef RPC_nmpssMax_32s31b
164 NMC_RPC_PIR(nmpssMax_32s31b);
165 #endif
166
167 #ifdef RPC_nmpssMax_64s63b
168 NMC_RPC_PIR64(nmpssMax_64s63b);
169 #endif
170
171 //-----
172 #ifdef RPC_nmpssAndC_8u
173 NMC_RPC_PIPI(nmpssAndC_8u);
174 #endif
175
176 #ifdef RPC_nmpssAndC_16u
177 NMC_RPC_PIPI(nmpssAndC_16u);
178 #endif
179
180 #ifdef RPC_nmpssAndC_32u
181 NMC_RPC_PIPI(nmpssAndC_32u);
182 #endif
183
184 #ifdef RPC_nmpssAndC_64u
185 NMC_RPC_PLPI(nmpssAndC_64u);
186 #endif
187 //-----
188 #ifdef RPC_nmpssOrC_8u
189 NMC_RPC_PIPI(nmpssOrC_8u);
190 #endif

```

```
191 #ifdef RPC_nmpssOrC_16u
192 NMC_RPC_PIPI(nmpssOrC_16u);
193 #endif
194 #endif
195
196 #ifdef RPC_nmpssOrC_32u
197 NMC_RPC_PLPI(nmpssOrC_32u);
198 #endif
199 //-----
200 #ifdef RPC_nmpssOrC_64u
201 NMC_RPC_PLPI(nmpssOrC_64u);
202 #endif
203 //-----
204 #ifdef RPC_nmpssXorC_8u
205 NMC_RPC_PIPI(nmpssXorC_8u);
206 #endif
207
208 #ifdef RPC_nmpssXorC_16u
209 NMC_RPC_PIPI(nmpssXorC_16u);
210 #endif
211
212 #ifdef RPC_nmpssXorC_32u
213 NMC_RPC_PLPI(nmpssXorC_32u);
214 #endif
215
216 #ifdef RPC_nmpssXorC_64u
217 NMC_RPC_PLPI(nmpssXorC_64u);
218 #endif
219 //-----
220 #ifdef RPC_nmpssNot_64u
221 NMC_RPC_PPI(nmpssNot_64u);
222 #endif
223
224 #ifdef RPC_nmpssNot_32u
225 NMC_RPC_PPI(nmpssNot_32u);
226 #endif
227
228 #ifdef RPC_nmpssNot_16u
229 NMC_RPC_PPI(nmpssNot_16u);
230 #endif
231
232 #ifdef RPC_nmpssNot_8u
233 NMC_RPC_PPI(nmpssNot_8u);
234 #endif
235 //-----
236 #ifdef RPC_nmpssAnd_64u
237 NMC_RPC_PPPI(nmpssAnd_64u);
238 #endif
239
240 #ifdef RPC_nmpssAnd_32u
241 NMC_RPC_PPPI(nmpssAnd_32u);
242 #endif
243
244 #ifdef RPC_nmpssAnd_16u
245 NMC_RPC_PPPI(nmpssAnd_16u);
246 #endif
247
248 #ifdef RPC_nmpssAnd_8u
249 NMC_RPC_PPPI(nmpssAnd_8u);
250 #endif
251 //-----
252 #ifdef RPC_nmpssOr_64u
253 NMC_RPC_PPPI(nmpssOr_64u);
254 #endif
255
256 #ifdef RPC_nmpssOr_32u
257 NMC_RPC_PPPI(nmpssOr_32u);
258 #endif
259
260 #ifdef RPC_nmpssOr_16u
261 NMC_RPC_PPPI(nmpssOr_16u);
262 #endif
263
264 #ifdef RPC_nmpssOr_8u
265 NMC_RPC_PPPI(nmpssOr_8u);
266 #endif
267 //-----
268 #ifdef RPC_nmpssXor_64u
269 NMC_RPC_PPPI(nmpssXor_64u);
270 #endif
271
272 #ifdef RPC_nmpssXor_32u
273 NMC_RPC_PPPI(nmpssXor_32u);
274 #endif
275
276 #ifdef RPC_nmpssXor_16u
277 NMC_RPC_PPPI(nmpssXor_16u);
```

```

278 #endif
279
280 #ifdef RPC_nmpssXor_8u
281 NMC_RPC_PPI(nmpssXor_8u);
282 #endif
283 //-----
284 #ifdef RPC_nmpssRShiftC_8u
285 NMC_RPC_PPI(nmpssRShiftC_8u);
286 #endif
287
288 #ifdef RPC_nmpssRShiftC_16u
289 NMC_RPC_PPI(nmpssRShiftC_16u);
290 #endif
291
292 #ifdef RPC_nmpssRShiftC_32u
293 NMC_RPC_PPI(nmpssRShiftC_32u);
294 #endif
295
296 #ifdef RPC_nmpssRShiftC_64u
297 NMC_RPC_PPI(nmpssRShiftC_64u);
298 #endif
299
300 //-----
301 #ifdef RPC_nmpssSum_8s
302 NMC_RPC_PIR(nmpssSum_8s);
303 #endif
304
305 #ifdef RPC_nmpssSum_16s
306 NMC_RPC_PIR64(nmpssSum_16s);
307 #endif
308
309 #ifdef RPC_nmpssSum_32s
310 NMC_RPC_PIR64(nmpssSum_32s);
311 #endif
312
313 #ifdef RPC_nmpssSum_64s
314 NMC_RPC_PIR64(nmpssSum_64s);
315 #endif
316 //-----
317 #ifdef RPC_nmpssCmpNe0_64s
318 NMC_RPC_PPI(nmpssCmpNe0_64s);
319 #endif
320
321 #ifdef RPC_nmpssCmpNe0_32s
322 NMC_RPC_PPI(nmpssCmpNe0_32s);
323 #endif
324
325 #ifdef RPC_nmpssCmpNe0_16s
326 NMC_RPC_PPI(nmpssCmpNe0_16s);
327 #endif
328
329 #ifdef RPC_nmpssCmpNe0_8s
330 NMC_RPC_PPI(nmpssCmpNe0_8s);
331 #endif
332 //-----
333 #ifdef RPC_nmpssCmpLt0_64s
334 NMC_RPC_PPI(nmpssCmpLt0_64s);
335 #endif
336
337 #ifdef RPC_nmpssCmpLt0_32s
338 NMC_RPC_PPI(nmpssCmpLt0_32s);
339 #endif
340
341 #ifdef RPC_nmpssCmpLt0_16s
342 NMC_RPC_PPI(nmpssCmpLt0_16s);
343 #endif
344
345 #ifdef RPC_nmpssCmpLt0_8s
346 NMC_RPC_PPI(nmpssCmpLt0_8s);
347 #endif
348
349 //-----
350 #ifdef RPC_nmpssCmpNeC_64s
351 NMC_RPC_PLPI(nmpssCmpNeC_64s);
352 #endif
353
354 #ifdef RPC_nmpssCmpNeC_32s
355 NMC_RPC_PPI(nmpssCmpNeC_32s);
356 #endif
357
358 #ifdef RPC_nmpssCmpNeC_16s
359 NMC_RPC_PPI(nmpssCmpNeC_16s);
360 #endif
361
362 #ifdef RPC_nmpssCmpNeC_8s
363 NMC_RPC_PPI(nmpssCmpNeC_8s);
364 #endif

```

```

365 //-----
366 // #ifdef RPC_nmppsCmpNeC_64s8u
367 // NMC_RPC_PLPI(nmppsCmpNeC_64s8u);
368 // #endif
369 //
370 // #ifdef RPC_nmppsCmpNeC_32s8u
371 // NMC_RPC_PIPI(nmppsCmpNeC_32s8u);
372 // #endif
373 //
374 // #ifdef RPC_nmppsCmpNeC_16s8u
375 // NMC_RPC_PIPI(nmppsCmpNeC_16s8u);
376 // #endif
377 //
378 // #ifdef RPC_nmppsCmpNeC_8s8u
379 // NMC_RPC_PIPI(nmppsCmpNeC_8s8u);
380 // #endif
381
382 //-----
383 // #ifdef RPC_nmppsCmpNe_64s8um
384 // NMC_RPC_PPPI(nmppsCmpNe_64s8um);
385 // #endif
386 //
387 // #ifdef RPC_nmppsCmpNe_32s8um
388 // NMC_RPC_PPPI(nmppsCmpNe_32s8um);
389 // #endif
390 //
391 // #ifdef RPC_nmppsCmpNe_16s8um
392 // NMC_RPC_PPPI(nmppsCmpNe_16s8um);
393 // #endif
394 //
395 // #ifdef RPC_nmppsCmpNe_8s8um
396 // NMC_RPC_PPPI(nmppsCmpNe_8s8um);
397 // #endif
398
399
400
401 //-----
402 #ifdef RPC_nmppsCmpLtC_64s
403 NMC_RPC_PLPI(nmppsCmpLtC_64s);
404 #endif
405
406 #ifdef RPC_nmppsCmpLtC_32s
407 NMC_RPC_PIPI(nmppsCmpLtC_32s);
408 #endif
409
410 #ifdef RPC_nmppsCmpLtC_16s
411 NMC_RPC_PIPI(nmppsCmpLtC_16s);
412 #endif
413
414 #ifdef RPC_nmppsCmpLtC_8s
415 NMC_RPC_PIPI(nmppsCmpLtC_8s);
416 #endif
417
418 //-----
419 // #ifdef RPC_nmppsCmpLtC_64s8u
420 // NMC_RPC_PLPI(nmppsCmpLtC_64s8u); // Not supported by simulink
421 // #endif
422 //
423 // #ifdef RPC_nmppsCmpLtC_32s8um
424 // NMC_RPC_PIPI(nmppsCmpLtC_32s8um);
425 // #endif
426 //
427 // #ifdef RPC_nmppsCmpLtC_16s8um
428 // NMC_RPC_PIPI(nmppsCmpLtC_16s8um);
429 // #endif
430 //
431 // #ifdef RPC_nmppsCmpLtC_8s8um
432 // NMC_RPC_PIPI(nmppsCmpLtC_8s8um);
433 // #endif
434
435 //-----
436 // #ifdef RPC_nmppsCmpLt_64s8um
437 // NMC_RPC_PPPI(nmppsCmpLt_64s8um);
438 // #endif
439 //
440 // #ifdef RPC_nmppsCmpLt_32s8um
441 // NMC_RPC_PPPI(nmppsCmpLt_32s8um);
442 // #endif
443 //
444 // #ifdef RPC_nmppsCmpLt_16s8um
445 // NMC_RPC_PPPI(nmppsCmpLt_16s8um);
446 // #endif
447 //
448 // #ifdef RPC_nmppsCmpLt_8s8um
449 // NMC_RPC_PPPI(nmppsCmpLt_8s8um);
450 // #endif
451

```

```

452
453
454 //-----
455
456 #if defined(RPC_nmppmMul_mm_8s8s) || defined(RPC_nmppmMul_mm_colmajor_8s8s)
457 NMC_RPC_PIIPPI(nmppmMul_mm_8s8s);
458 #endif
459
460 #if defined(RPC_nmppmMul_mm_8s16s) || defined(RPC_nmppmMul_mm_colmajor_8s16s)
461 NMC_RPC_PIIPPI(nmppmMul_mm_8s16s);
462 #endif
463
464 #if defined(RPC_nmppmMul_mm_8s32s) || defined(RPC_nmppmMul_mm_colmajor_8s32s)
465 NMC_RPC_PIIPPI(nmppmMul_mm_8s32s);
466 #endif
467
468 #if defined(RPC_nmppmMul_mm_8s64s) || defined(RPC_nmppmMul_mm_colmajor_8s64s)
469 NMC_RPC_PIIPPI(nmppmMul_mm_8s64s);
470 #endif
471
472 #if defined(RPC_nmppmMul_mm_16s16s) || defined(RPC_nmppmMul_mm_colmajor_16s16s)
473 NMC_RPC_PIIPPI(nmppmMul_mm_16s16s);
474 #endif
475
476 #if defined(RPC_nmppmMul_mm_16s32s) || defined(RPC_nmppmMul_mm_colmajor_16s32s)
477 NMC_RPC_PIIPPI(nmppmMul_mm_16s32s);
478 #endif
479
480 #if defined(RPC_nmppmMul_mm_16s64s) || defined(RPC_nmppmMul_mm_colmajor_16s64s)
481 NMC_RPC_PIIPPI(nmppmMul_mm_16s64s);
482 #endif
483
484 #if defined(RPC_nmppmMul_mm_32s32s) || defined(RPC_nmppmMul_mm_colmajor_32s32s)
485 NMC_RPC_PIIPPI(nmppmMul_mm_32s32s);
486 #endif
487
488 #if defined(RPC_nmppmMul_mm_32s64s) || defined(RPC_nmppmMul_mm_colmajor_32s64s)
489 NMC_RPC_PIIPPI(nmppmMul_mm_32s64s);
490 #endif
491
492 //-----
493 //ifdef RPC_nmppsDotProd_8s8sm
494 //NMC_RPC_PPIR64(nmppsDotProd_8s8sm);
495 //endif
496 //
497 //ifdef RPC_nmppsDotProd_8s16sm
498 //NMC_RPC_PPIR64(nmppsDotProd_8s16sm);
499 //endif
500 //
501 //ifdef RPC_nmppsDotProd_8s32sm
502 //NMC_RPC_PPIR64(nmppsDotProd_8s32sm);
503 //endif
504 //
505 //ifdef RPC_nmppsDotProd_8s64s
506 //NMC_RPC_PPIR64(nmppsDotProd_8s64s);
507 //endif
508 //
509 //ifdef RPC_nmppsDotProd_16s16sm
510 //NMC_RPC_PPIR64(nmppsDotProd_16s16sm);
511 //endif
512 //
513 //ifdef RPC_nmppsDotProd_16s32sm
514 //NMC_RPC_PPIR64(nmppsDotProd_16s32sm);
515 //endif
516 //
517 //ifdef RPC_nmppsDotProd_16s64s
518 //NMC_RPC_PPIR64(nmppsDotProd_16s64s);
519 //endif
520 //
521 //ifdef RPC_nmppsDotProd_32s32sm
522 //NMC_RPC_PPIR64(nmppsDotProd_32s32sm);
523 //endif
524 //
525 //ifdef RPC_nmppsDotProd_32s64s
526 //NMC_RPC_PPIR64(nmppsDotProd_32s64s);
527 //endif
528 //
529 //ifdef RPC_nmppsDotProd_64s64s
530 //NMC_RPC_PPIR64(nmppsDotProd_64s64s);
531 //endif
532
533 //-----
534 #if defined(RPC_nmppmMul_mv_8s64s) || defined(RPC_nmppmMul_mv_colmajor_8s64s)
535 NMC_RPC_PPPPI(nmppmMul_mv_8s64s);
536 #endif
537
538 #if defined(RPC_nmppmMul_mv_16s64s) || defined(RPC_nmppmMul_mv_colmajor_16s64s)

```

```

539 NMC_RPC_PPPI(nmppmMul_mv_16s64s);
540 #endif
541
542 #if defined(RPC_nmppmMul_mv_32s64s) || defined(RPC_nmppmMul_mv_colmajor_32s64s)
543 NMC_RPC_PPPI(nmppmMul_mv_32s64s);
544 #endif
545 //-----
546 #ifdef RPC_nmppsMulC_8s
547 NMC_RPC_PIPI(nmppsMulC_8s      );
548 #endif
549 #ifdef RPC_nmppsMulC_8s16s
550 NMC_RPC_PIPI(nmppsMulC_8s16s  );
551 #endif
552 #ifdef RPC_nmppsMulC_8s32s
553 NMC_RPC_PIPI(nmppsMulC_8s32s  );
554 #endif
555 #ifdef RPC_nmppsMulC_8s64s
556 NMC_RPC_PLPI(nmppsMulC_8s64s  );
557 #endif
558 #ifdef RPC_nmppsMulC_16s
559 NMC_RPC_PIPI(nmppsMulC_16s    );
560 #endif
561 #ifdef RPC_nmppsMulC_16s32s
562 NMC_RPC_PIPI(nmppsMulC_16s32s );
563 #endif
564 #ifdef RPC_nmppsMulC_16s64s
565 NMC_RPC_PLPI(nmppsMulC_16s64s );
566 #endif
567 #ifdef RPC_nmppsMulC_32s
568 NMC_RPC_PIPI(nmppsMulC_32s    );
569 #endif
570 #ifdef RPC_nmppsMulC_32s64s
571 NMC_RPC_PLPI(nmppsMulC_32s64s );
572 #endif
573 #ifdef RPC_nmppsMulC_64s
574 NMC_RPC_PLPI(nmppsMulC_64s    );
575 #endif
576

```

7.105 rpc-nmc.h

```

1 #include "easynmc\aura.h"
2
3
4 //typedef void (func_p_t)(void* );
5 //typedef void*(func_i_p_t)(int);
6 typedef void (func_ppi_t)(void*,void*,int);
7 typedef int (func_ppi_i_t)(void*,void*,int);
8 typedef int (func_ppr_i_t)(void*,void*,int* );
9 typedef void (func_pppi_t)(void*,void*,void*,int);
10 typedef void (func_pipi_t)(void*,int,void*,int);
11 typedef void (func_plpi_t)(void*,long,void*,int);
12 typedef void (func_pip_t)(void*,int,void* );
13 typedef void (func_ppp_t)(void*,void*,void* );
14 typedef int (func_ppp_i_t)(void*,void*,void* );
15 typedef void (func_pppi_i_t)(void*,void*,void*,int,int);
16 typedef void (func_pippi_t)(void*,int,int,void*,void*,int);
17 typedef void (func_ppip_t)(void*,void*,int,void* );
18
19 #include <time.h>
20
21 //define NMC_RPC_I_P(func) \
22 void rpc_ ## func(void *in, void *out) \
23 //{
24 // unsigned i = aura_get_u32(); \
25 // func_i_t *unifunc=(func_i_t*)func; \
26 // void* p=unifunc(i); \
27 // aura_put_u32((int)p); \
28 //}
29 //
30 //define NMC_RPC_I(func) \
31 void rpc_ ## func(void *in, void *out) \
32 //{
33 // unsigned i = aura_get_u32(); \
34 // func_i_p_t *unifunc=(func_i_p_t*)func; \
35 // unifunc(i); \
36 //}
37
38
39 #define NMC_RPC_PPI(func) \
40 void rpc_ ## func(void *in, void *out) \
41 { \
42 clock_t t0,t1,t2; \

```

```

43 t0=clock(); \
44 aura_buffer buf_src = aura_get_buf(); \
45 aura_buffer buf_dst = aura_get_buf(); \
46 int *src = aura_buffer_to_ptr(buf_src); \
47 int *dst = aura_buffer_to_ptr(buf_dst); \
48 unsigned size = aura_get_u32(); \
49 func_ppi_t *unifunc=(func_ppi_t*)func; \
50 unifunc(src,dst,size); \
51 }
52
53 #define NMC_RPC_PPI(func) \
54 void rpc_ ## func(void *in, void *out) \
55 { \
56 aura_buffer buf_src = aura_get_buf(); \
57 aura_buffer buf_dst = aura_get_buf(); \
58 int *src = aura_buffer_to_ptr(buf_src); \
59 int *dst = aura_buffer_to_ptr(buf_dst); \
60 unsigned size = aura_get_u32(); \
61 func_ppi_i_t *unifunc=(func_ppi_i_t*)func; \
62 int ret=unifunc(src,dst,size); \
63 aura_put_u32(ret); \
64 }
65
66
67
68 #define NMC_RPC_PPPI(func) \
69 void rpc_ ## func(void *in, void *out) \
70 { \
71 aura_buffer buf_src0 = aura_get_buf(); \
72 aura_buffer buf_src1 = aura_get_buf(); \
73 aura_buffer buf_dst = aura_get_buf(); \
74 int *src0 = aura_buffer_to_ptr(buf_src0); \
75 int *src1 = aura_buffer_to_ptr(buf_src1); \
76 int *dst = aura_buffer_to_ptr(buf_dst); \
77 unsigned size = aura_get_u32(); \
78 func_pppi_t *unifunc=(func_pppi_t*)func; \
79 unifunc(src0,src1,dst,size); \
80 }
81
82 #define NMC_RPC_PIPI(func) \
83 void rpc_ ## func(void *in, void *out) \
84 { \
85 aura_buffer buf_src = aura_get_buf(); \
86 unsigned val = aura_get_u32(); \
87 aura_buffer buf_dst = aura_get_buf(); \
88 int *src = aura_buffer_to_ptr(buf_src); \
89 int *dst = aura_buffer_to_ptr(buf_dst); \
90 unsigned size = aura_get_u32(); \
91 func_pipi_t *unifunc=(func_pipi_t*)func; \
92 unifunc(src,val,dst,size); \
93 }
94
95
96
97 #define NMC_RPC_PLPI(func) \
98 void rpc_ ## func(void *in, void *out) \
99 { \
100 aura_buffer buf_src = aura_get_buf(); \
101 unsigned long long val = aura_get_u64(); \
102 aura_buffer buf_dst = aura_get_buf(); \
103 int *src = aura_buffer_to_ptr(buf_src); \
104 int *dst = aura_buffer_to_ptr(buf_dst); \
105 unsigned size = aura_get_u32(); \
106 func_plpi_t *unifunc=(func_plpi_t*)func; \
107 unifunc(src,val,dst,size); \
108 }
109
110 #define NMC_RPC_PPPLI(func) \
111 void rpc_ ## func(void *in, void *out) \
112 { \
113 aura_buffer buf_src = aura_get_buf(); \
114 aura_buffer buf_dst = aura_get_buf(); \
115 unsigned long long val = aura_get_u64(); \
116 int *src = aura_buffer_to_ptr(buf_src); \
117 int *dst = aura_buffer_to_ptr(buf_dst); \
118 unsigned size = aura_get_u32(); \
119 func_pipi_t *unifunc=(func_pipi_t*)func; \
120 }
121
122 #define NMC_RPC_PPP(func) \
123 void rpc_ ## func(void *in, void *out) \
124 { \
125 aura_buffer buf_src0 = aura_get_buf(); \
126 aura_buffer buf_src1 = aura_get_buf(); \
127 aura_buffer buf_dst = aura_get_buf(); \
128 int *src0 = aura_buffer_to_ptr(buf_src0); \
129 int *src1 = aura_buffer_to_ptr(buf_src1); \

```

```

130 int *dst = aura_buffer_to_ptr(buf_dst); \
131 unsigned size = aura_get_u32(); \
132 func_ppp_t *unifunc=(func_ppp_t*)func; \
133 unifunc(src0,src1,dst); \
134 }
135
136 #define NMC_RPC_PPP_I(func) \
137 void rpc_ ## func(void *in, void *out) \
138 { \
139 aura_buffer buf_src0 = aura_get_buf(); \
140 aura_buffer buf_src1 = aura_get_buf(); \
141 aura_buffer buf_dst = aura_get_buf(); \
142 int *src0 = aura_buffer_to_ptr(buf_src0); \
143 int *src1 = aura_buffer_to_ptr(buf_src1); \
144 int *dst = aura_buffer_to_ptr(buf_dst); \
145 unsigned size = aura_get_u32(); \
146 func_ppp_t *unifunc=(func_ppp_t*)func; \
147 int ret = unifunc(src0,src1,dst); \
148 aura_put_u32(ret); \
149 }
150
151
152 #define NMC_RPC_PIR(func) \
153 void rpc_ ## func(void *in, void *out) \
154 { \
155 aura_buffer buf_src0 = aura_get_buf(); \
156 int *src0 = aura_buffer_to_ptr(buf_src0); \
157 unsigned size = aura_get_u32(); \
158 func_pip_t *unifunc=(func_pip_t*)func; \
159 int ret; \
160 unifunc(src0,size,&ret); \
161 aura_put_u32(ret); \
162 }
163
164 #define NMC_RPC_PPR_I(func) \
165 void rpc_ ## func(void *in, void *out) \
166 { \
167 int handle=123; \
168 aura_buffer buf_src = aura_get_buf(); \
169 aura_buffer buf_dst = aura_get_buf(); \
170 int *src = aura_buffer_to_ptr(buf_src); \
171 int *dst = aura_buffer_to_ptr(buf_dst); \
172 func_ppr_i_t *unifunc=(func_ppr_i_t*)func; \
173 int ret = unifunc(src,dst,&handle); \
174 aura_put_u32(handle); \
175 aura_put_u32(ret); \
176 }
177
178
179
180 #define NMC_RPC_PIR64(func) \
181 void rpc_ ## func(void *in, void *out) \
182 { \
183 aura_buffer buf_src0 = aura_get_buf(); \
184 int *src0 = aura_buffer_to_ptr(buf_src0); \
185 unsigned size = aura_get_u32(); \
186 func_pip_t *unifunc=(func_pip_t*)func; \
187 long ret; \
188 unifunc(src0,size,&ret); \
189 aura_put_u64(ret); \
190 }
191
192 // #define NMC_RPC_PPIR64(func) \
193 // void rpc_ ## func(void *in, void *out) \
194 // { \
195 // aura_buffer buf_src0 = aura_get_buf(); \
196 // aura_buffer buf_src1 = aura_get_buf(); \
197 // int *src0 = aura_buffer_to_ptr(buf_src0); \
198 // int *src1 = aura_buffer_to_ptr(buf_src1); \
199 // unsigned size = aura_get_u32(); \
200 // func_ppip_t *unifunc=(func_ppip_t*)func; \
201 // long ret; \
202 // unifunc(src0,src1,size,&ret); \
203 // aura_put_u64(ret); \
204 // }
205 //
206
207 #define NMC_RPC_PPPII(func) \
208 void rpc_ ## func(void *in, void *out) \
209 { \
210 aura_buffer buf_src0 = aura_get_buf(); \
211 aura_buffer buf_src1 = aura_get_buf(); \
212 aura_buffer buf_dst = aura_get_buf(); \
213 int *src0 = aura_buffer_to_ptr(buf_src0); \
214 int *src1 = aura_buffer_to_ptr(buf_src1); \
215 int *dst = aura_buffer_to_ptr(buf_dst); \
216 unsigned height = aura_get_u32(); \

```

```

217 unsigned width = aura_get_u32(); \
218 func_pppi_t *unifunc=(func_pppi_t*)func; \
219 unifunc(src0,src1,dst,height,width); \
220 }
221
222 #define NMC_RPC_PIIPPI(func) \
223 void rpc_ ## func(void *in, void *out) \
224 { \
225     aura_buffer buf_src0 = aura_get_buf(); \
226     unsigned height = aura_get_u32(); \
227     unsigned width0 = aura_get_u32(); \
228     aura_buffer buf_src1 = aura_get_buf(); \
229     aura_buffer buf_dst = aura_get_buf(); \
230     unsigned width1 = aura_get_u32(); \
231     int *src0 = aura_buffer_to_ptr(buf_src0); \
232     int *src1 = aura_buffer_to_ptr(buf_src1); \
233     int *dst = aura_buffer_to_ptr(buf_dst); \
234     func_pppi_t *unifunc=(func_pppi_t*)func; \
235     unifunc(src0,height,width0,src1,dst,width1); \
236 }

```

7.106 test.h

```

1
2
3 template <int K> void* nmppsMalloc(int size);
4 template<> void* nmppsMalloc<8>(int size) { return nmppsMalloc_8s(size); }
5 template<> void* nmppsMalloc<16>(int size){ return nmppsMalloc_16s(size); }
6 template<> void* nmppsMalloc<32>(int size){ return nmppsMalloc_32s(size); }
7 template<> void* nmppsMalloc<64>(int size){ return nmppsMalloc_64s(size); }
8
9 #define XNM_TYPE(x) nm##x##s
10 #define NM_TYPE(x) XNM_TYPE(x)
11
12 #define xnmpsMalloc(bits,size) nmppsMalloc ## bits ## s(size)
13 #define nmppsMalloc(bits,size) xnmpsMalloc(bits,size)
14
15 #define xnmpsSet(bits,buf,val,size) nmppsSet ## bits ## s(buf,val,size)
16 #define nmppsSet(bits,buf,val,size) xnmpsSet(bits,buf,val,size)
17
18 #define xnmpsCrcAcc(bits,src,size,crc) nmppsCrcAcc ## bits ## s(src,size, crc)
19 #define nmppsCrcAcc(bits,src,size,crc) xnmpsCrcAcc(bits,src,size,crc)
20
21 #define xnmpsRandUniform(bits,buf,size) nmppsRandUniform ## bits ## s(buf,size)
22 #define nmppsRandUniform(bits,buf,size) xnmpsRandUniform(bits,buf,size)
23
24
25 inline int SIZE32(int bits, int size) {
26     switch (bits){
27         case(64):
28             return size << 1;
29         case(32):
30             return size ;
31         case(16):
32             return size >>1;
33         case(8):
34             return size >>2;
35         case(4):
36             return size >>3;
37     }
38     return 0;
39 }
40
41 inline int DIM(int bits) {
42     switch (bits) {
43         case(64):
44             return 1;
45         case(32):
46             return 2;
47         case(16):
48             return 4;
49         case(8):
50             return 8;
51         case(4):
52             return 16;
53         case(2):
54             return 32;
55         case(1):
56             return 64;
57     }
58     return 0;
59 }
60
61 #include "stdio.h"

```

```
62 inline void dump(int bits,void* buf, int height, int width){
63     unsigned *p=(unsigned*)buf;
64     switch(bits){
65         case(64):
66             for(int y=0,i=0; y<height; y++){
67                 for(int x=0; x<width; x++){
68                     unsigned lo=nmppsGet_32u(p,i++);
69                     unsigned hi=nmppsGet_32u(p,i++);
70                     printf("%08X%08X", hi,lo);
71                 }
72                 printf("\n");
73             }
74             break;
75
76         case(32):
77             for(int y=0,i=0; y<height; y++){
78                 for(int x=0; x<width; x++){
79                     unsigned val=nmppsGet_32u(p,i++);
80                     printf("%08X ", val);
81                 }
82                 printf("\n");
83             }
84             break;
85
86         case(16):
87             for(int y=0, i=0; y<height; y++){
88                 for(int x=0; x<width; x++,i++){
89                     unsigned val=nmppsGet_16u((nm16u*)buf,i);
90                     printf("%04X ", val);
91                 }
92                 printf("\n");
93             }
94             break;
95         case(8):
96             for(int y=0, i=0; y<height; y++){
97                 for(int x=0; x<width; x++,i++){
98                     unsigned val=nmppsGet_8u((nm8u*)buf,i);
99                     printf("%02X ", val);
100                }
101                printf("\n");
102            }
103            break;
104        }
105 }
```

Предметный указатель

, 287

Allocate
 C_PlessyCornerDetector, 351
 C_PlessyCornerDetector_16s, 353
 C_PlessyCornerDetector_32f, 354

AllocateMaxAvail
 C_Heap, 348

BLASS-LEVEL1, 41
 nmlblas_dasum, 42
 nmlblas_daxpy, 43
 nmlblas_dcopy, 43
 nmlblas_ddot, 44
 nmlblas_dnrm2, 44
 nmlblas_drot, 45
 nmlblas_drotg, 46
 nmlblas_drotm, 46
 nmlblas_dscal, 46
 nmlblas_dsdot, 47
 nmlblas_dswap, 48
 nmlblas_dznrm2, 48
 nmlblas_idamax, 49
 nmlblas_isamax, 49
 nmlblas_sasum, 50
 nmlblas_saxpy, 50
 nmlblas_scnrm2, 51
 nmlblas_scopy, 51
 nmlblas_sdot, 52
 nmlblas_sdsdot, 52
 nmlblas_snrm2, 53
 nmlblas_srot, 53
 nmlblas_srotm, 54
 nmlblas_sscal, 54
 nmlblas_sswap, 54

BLASS-LEVEL2, 55
 nmlblas_dgemv, 55
 nmlblas_sgmv, 57
 nmlblas_sger, 58

BLASS-LEVEL3, 58

C_2DSubPixelMinPosition, 343
 Find, 343
 Release, 343

C_2DTrigSubPixelMinPosition, 344
 Find, 344
 Release, 344

C_Allocator32, 345

C_BoxImg< T >, 345

C_BoxVec< T >, 346

C_Heap, 346
 AllocateMaxAvail, 348

C_Img< T >, 348

C_MultiHeap, 349

C_PlessyCornerDetector, 350
 Allocate, 351
 DeAllocate, 351
 FindCorners, 351
 Release, 351
 SetThreshold, 351

C_PlessyCornerDetector_16s, 352
 Allocate, 353
 CountPlessy, 353
 DeAllocate, 353
 SetThreshold, 353

C_PlessyCornerDetector_32f, 353
 Allocate, 354
 CountDer, 354
 CountPlessy, 355
 DeAllocate, 355
 SetThreshold, 355

C_RingBuffer< T >, 356
 PushRequest, 356

C_RingBufferRemote< T >, 357

CountDer
 C_PlessyCornerDetector_32f, 354

CountPlessy
 C_PlessyCornerDetector_16s, 353
 C_PlessyCornerDetector_32f, 355

Cplx_float, 358

DeAllocate
 C_PlessyCornerDetector, 351
 C_PlessyCornerDetector_16s, 353
 C_PlessyCornerDetector_32f, 355

DFT-8, 22
 nmppsDFT8Fwd_32fc, 22

ds_struct, 358

EnterHardMode, 359

FFT-1024, 28, 107
 FFT_Fwd1024, 107

FFT-128, 25

FFT-16, 23

FFT-2048, 28, 111
 FFT_Fwd2048, 111

FFT-256, 26, 99
 FFT_Fwd256, 100

FFT-32, 24

FFT-4096, 29, 114
 FFT_Fwd4096, 114
 FFT-512, 27, 103
 FFT_Fwd512, 103
 FFT-64, 25
 FFT-8192, 116
 FFT_Fwd8192, 117
 FFT-Common, 38
 FFT_Fwd1024
 FFT-1024, 107
 FFT_Fwd2048
 FFT-2048, 111
 FFT_Fwd256
 FFT-256, 100
 FFT_Fwd4096
 FFT-4096, 114
 FFT_Fwd512
 FFT-512, 103
 FFT_Fwd8192
 FFT-8192, 117
 FFT_Inv1024
 IFFT-1024, 109
 FFT_Inv2048
 IFFT-2048, 112
 FFT_Inv256
 IFFT-256, 101
 FFT_Inv256Set6bit
 IFFT-256, 103
 FFT_Inv4096
 IFFT-4096, 115
 FFT_Inv512
 IFFT-512, 105
 FFT_Inv8192
 IFFT-8192, 118
 FFTFwdInitAlloc, 21
 FFTInvInitAlloc, 21
 Find
 C_2DSubPixelMinPosition, 343
 C_2DTrigSubPixelMinPosition, 344
 FindCorners
 C_PlessyCornerDetector, 351
 Fix-point 32, 73
 Fix-point 64, 72
 Floodfill, 76
 FloodFill8, 76
 FloodFill8
 Floodfill, 76
 g:/nmpp/include/crtdbg2.h, 397
 g:/nmpp/include/fft.h, 397
 g:/nmpp/include/fft_32fcr.h, 401
 g:/nmpp/include/fftexp.h, 402
 g:/nmpp/include/macros_fpu.h, 404
 g:/nmpp/include/malloc32.h, 405
 g:/nmpp/include/metric.h, 408
 g:/nmpp/include/minrep.h, 408
 g:/nmpp/include/multiheap.h, 408
 g:/nmpp/include/nmblas.h, 418, 422
 g:/nmpp/include/nmblas/nmblas_sgemm.h, 425, 427
 g:/nmpp/include/nmchar.h, 429
 g:/nmpp/include/nmdef.h, 435
 g:/nmpp/include/nmplc.h, 439
 g:/nmpp/include/nmplc/cArithmetic.h, 436
 g:/nmpp/include/nmplc/cfixpnt32.h, 436
 g:/nmpp/include/nmplc/cfixpnt64.h, 437
 g:/nmpp/include/nmplc/cInit.h, 438
 g:/nmpp/include/nmplc/cInteger.h, 439
 g:/nmpp/include/nmplc/nmplc.h, 439
 g:/nmpp/include/nmpli.h, 455
 g:/nmpp/include/nmpli/filter.h, 439
 g:/nmpp/include/nmpli/iArithmetics.h, 440
 g:/nmpp/include/nmpli/iCellTexture.h, 440
 g:/nmpp/include/nmpli/iConvert.h, 441
 g:/nmpp/include/nmpli/iDef.h, 444
 g:/nmpp/include/nmpli/iDeinterlace.h, 445
 g:/nmpp/include/nmpli/iFilter.h, 445
 g:/nmpp/include/nmpli/iFiltration.h, 447
 g:/nmpp/include/nmpli/iFloodFill.h, 448
 g:/nmpp/include/nmpli/iPlessy.h, 449
 g:/nmpp/include/nmpli/iPlessyDetector.h, 449
 g:/nmpp/include/nmpli/iPrint.h, 450
 g:/nmpp/include/nmpli/iReordering.h, 451
 g:/nmpp/include/nmpli/iResample.h, 451
 g:/nmpp/include/nmpli/iSelect.h, 452
 g:/nmpp/include/nmpli/isubpixel2d.h, 452
 g:/nmpp/include/nmpli/isubpixel2dimpl.h, 453
 g:/nmpp/include/nmpli/iSupport.h, 453
 g:/nmpp/include/nmpli/nmpli.h, 455
 g:/nmpp/include/nmpli/warpimg.h, 456
 g:/nmpp/include/nmplm.h, 464
 g:/nmpp/include/nmplm/mInit.h, 457
 g:/nmpp/include/nmplm/mInverse.h, 458
 g:/nmpp/include/nmplm/mMatrixVector.h, 459
 g:/nmpp/include/nmplm/mMatrixVectorDev.h, 460
 g:/nmpp/include/nmplm/mStat.h, 461
 g:/nmpp/include/nmplm/mSupport.h, 462
 g:/nmpp/include/nmplm/nmplm.h, 464
 g:/nmpp/include/nmpls.h, 473
 g:/nmpp/include/nmpls/fft.h, 397
 g:/nmpp/include/nmpls/fft_old.h, 464
 g:/nmpp/include/nmpls/ffttext.h, 469
 g:/nmpp/include/nmpls/fftref.h, 473
 g:/nmpp/include/nmpls/nmpls.h, 473
 g:/nmpp/include/nmpls/sConvolution.h, 474
 g:/nmpp/include/nmpls/sCorrelation.h, 474
 g:/nmpp/include/nmpls/sFiltration.h, 474
 g:/nmpp/include/nmpls/sfir.h, 475
 g:/nmpp/include/nmpls/sResample.h, 476
 g:/nmpp/include/nmplv.h, 477
 g:/nmpp/include/nmplv/nmplv.h, 477
 g:/nmpp/include/nmplv/nmtl.h, 478
 g:/nmpp/include/nmplv/sElementary.h, 478
 g:/nmpp/include/nmplv/vArithmetics.h, 478
 g:/nmpp/include/nmplv/vArithmeticsDev.h, 482

g:/nmpp/include/nmplv/vBitwise.h, 482
g:/nmpp/include/nmplv/vInit.h, 484
g:/nmpp/include/nmplv/vSelect.h, 486
g:/nmpp/include/nmplv/vStat.h, 490
g:/nmpp/include/nmplv/vSupport.h, 491
g:/nmpp/include/nmplv/vSupport_.h, 493
g:/nmpp/include/nmplv/vTransform.h, 495
g:/nmpp/include/nmpp-cpp.h, 496
g:/nmpp/include/nmpp.h, 496
g:/nmpp/include/nmshort.h, 497
g:/nmpp/include/nmtl.h, 478
g:/nmpp/include/nmtl/nmtl.h, 478
g:/nmpp/include/nmtl/nmtlio.h, 501
g:/nmpp/include/nmtl/tcmplx.h, 505
g:/nmpp/include/nmtl/tcmplx_spec.h, 508
g:/nmpp/include/nmtl/tfixpoint.h, 509
g:/nmpp/include/nmtl/tfixpointmath.h, 513
g:/nmpp/include/nmtl/tmatrix.h, 514
g:/nmpp/include/nmtl/tmatrix_spec.h, 523
g:/nmpp/include/nmtl/tnmcube.h, 524
g:/nmpp/include/nmtl/tnmcvec.h, 526
g:/nmpp/include/nmtl/tnmint.h, 529
g:/nmpp/include/nmtl/tnmmtr.h, 533
g:/nmpp/include/nmtl/tnmmtrpack.h, 540
g:/nmpp/include/nmtl/tnmvec.h, 546
g:/nmpp/include/nmtl/tnmvecpack.h, 551
g:/nmpp/include/nmtl/tvector.h, 556
g:/nmpp/include/nmtype.h, 563, 568
g:/nmpp/include/nmvcore/vCore.h, 576
g:/nmpp/include/ownmalloc.h, 580
g:/nmpp/include/ringbuffer_.h, 580
g:/nmpp/include/ringremote.h, 584
g:/nmpp/include/rpc/rpc-host.h, 589
g:/nmpp/include/rpc/rpc-nmc-func.h, 591
g:/nmpp/include/rpc/rpc-nmc.h, 598
g:/nmpp/include/test.h, 601
g:/nmpp/include/tfixpoint.h, 513
GetVec
 контроль переполнения, 279
I_2DSubPixelMinPosition, 359
I_PlessyCornerDetector, 359
IDFT-8, 30
IFFT-1024, 35, 109
 FFT_Inv1024, 109
 nmppsFFT1024Inv_32fc, 35
IFFT-128, 33
IFFT-16, 31
IFFT-2048, 36, 112
 FFT_Inv2048, 112
IFFT-256, 34, 101
 FFT_Inv256, 101
 FFT_Inv256Set6bit, 103
IFFT-32, 31
IFFT-4096, 37, 115
 FFT_Inv4096, 115
IFFT-512, 34, 105
 FFT_Inv512, 105
IFFT-64, 32
IFFT-8192, 118
 FFT_Inv8192, 118
IFFT-Common, 38
im
 s_nm64sc, 377
implement, 278
int15b
 Типы скалярных данных, 145
int15in16x4, 360
int16b
 Типы скалярных данных, 146
int1b
 Типы скалярных данных, 146
int2b
 Типы скалярных данных, 146
int30b
 Типы скалярных данных, 146
int30in32x2, 360
int31b
 Типы скалярных данных, 146
int31in32x2, 360
int32b
 Типы скалярных данных, 147
int3b
 Типы скалярных данных, 147
int4b
 Типы скалярных данных, 147
int63b
 Типы скалярных данных, 147
int64b
 Типы скалярных данных, 147
int7b
 Типы скалярных данных, 148
int8b
 Типы скалярных данных, 148
Integer operations, 72
mtr< T >, 360
MTR_Addr, 95
MTR_Copy, 87
MTR_Copyau, 85
MTR_fpResolve_Gauss, 88
MTR_fpResolve_PivotGauss, 89
MTR_Free, 95
MTR_GetVal, 97
MTR_Malloc, 95
MTR_ProdUnitV, 94
MTR_SetVal, 96
nm1
 Типы векторных данных, 134
nm16s
 Типы векторных данных, 135
nm16s15b
 Типы векторных данных, 135
nm16sc, 362
NM16Sx4
 nmtype.h, 566
nm16u

- Типы векторных данных, 135
- nm16u15b
 - Типы векторных данных, 135
- nm2s
 - Типы векторных данных, 136
- nm2u
 - Типы векторных данных, 136
- nm32s
 - Типы векторных данных, 136
- nm32s30b
 - Типы векторных данных, 136
- nm32s31b
 - Типы векторных данных, 137
- NM32Sx2
 - nmtype.h, 566
- nm32u
 - Типы векторных данных, 137
- nm32u31b
 - Типы векторных данных, 137
- nm4s
 - Типы векторных данных, 137
- nm4u
 - Типы векторных данных, 138
- nm4u3b
 - Типы векторных данных, 138
- nm64s
 - Типы векторных данных, 138
- nm64s63b
 - Типы векторных данных, 138
- nm64u
 - Типы векторных данных, 139
- nm8s
 - Типы векторных данных, 139
- nm8s7b
 - Типы векторных данных, 139
- nm8u
 - Типы векторных данных, 139
- nm8u7b
 - Типы векторных данных, 140
- nmblas, 278
- nmblas_dasum
 - BLASS-LEVEL1, 42
- nmblas_daxpy
 - BLASS-LEVEL1, 43
- nmblas_dcopy
 - BLASS-LEVEL1, 43
- nmblas_ddot
 - BLASS-LEVEL1, 44
- nmblas_dgemv
 - BLASS-LEVEL2, 55
- nmblas_dnrm2
 - BLASS-LEVEL1, 44
- nmblas_drot
 - BLASS-LEVEL1, 45
- nmblas_drotg
 - BLASS-LEVEL1, 46
- nmblas_drotm
 - BLASS-LEVEL1, 46
- nmblas_dscal
 - BLASS-LEVEL1, 46
- nmblas_dsdot
 - BLASS-LEVEL1, 47
- nmblas_dswap
 - BLASS-LEVEL1, 48
- nmblas_dznrm2
 - BLASS-LEVEL1, 48
- nmblas_idamax
 - BLASS-LEVEL1, 49
- nmblas_isamax
 - BLASS-LEVEL1, 49
- nmblas_sasum
 - BLASS-LEVEL1, 50
- nmblas_saxpy
 - BLASS-LEVEL1, 50
- nmblas_scnrm2
 - BLASS-LEVEL1, 51
- nmblas_scopy
 - BLASS-LEVEL1, 51
- nmblas_sdot
 - BLASS-LEVEL1, 52
- nmblas_sdsdot
 - BLASS-LEVEL1, 52
- nmblas_sgenv
 - BLASS-LEVEL2, 57
- nmblas_sger
 - BLASS-LEVEL2, 58
- nmblas_snrm2
 - BLASS-LEVEL1, 53
- nmblas_srot
 - BLASS-LEVEL1, 53
- nmblas_srotm
 - BLASS-LEVEL1, 54
- nmblas_sscal
 - BLASS-LEVEL1, 54
- nmblas_sswap
 - BLASS-LEVEL1, 54
- nmchar, 362
- nmchar1D< N >, 363
- nmchar2D< Y, X >, 363
- nmintpack< T >, 363
- nmmtr< T >, 364
- nmmtrpack< T >, 365
- nmppcDivC, 59
- nmppcDoubleToFix32, 61
- nmppcDoubleToFix64, 68
- nmppcFix32toDouble, 62
- nmppcFix64Exp01, 70
- nmppcFix64toDouble, 68
- nmppcFixArcTan32, 61
- nmppcFixArcTan64, 70
- nmppcFixDiv64, 69
- nmppcFixDivMod32, 66
- nmppcFixExp32, 60
- nmppcFixInv32, 64
- nmppcFixMul32, 63
- nmppcFixSinCos32, 60

nmppcFixSinCos64, 69
nmppcFixSqrt32, 62
nmppcFixSqrt64, 67
nmppcMedian3_32u, 123
nmppcProdC, 59
nmppcSqrt, 71
nmppcTblFixArcCos32, 65
nmppcTblFixArcSin32, 64
nmppcTblFixCos32, 65
nmppcTblFixSin32, 66
nmppiDeinterlaceBlend
 Функции деинтерлейсинга, 74
nmppiDeinterlaceSplit
 Функции деинтерлейсинга, 75
nmppiMergeFromBlocks, 83
nmppiRelease, 83
nmppiSplitIntoBlocks, 82
nmppmCopyua, 84
nmppMerge, 271
nmppmLUDecomp, 89
nmppmLUResolve, 90
nmppmMul_mm, 90
nmppmMul_mv_, 91
nmppmMul_mv_8xH, 92
nmppmMul_mv__AddC, 93
nmppsAbs, 162
nmppsAbs1, 163
nmppsAbsDiff, 176
nmppsAbsDiff1, 178
nmppsAbsDiff_f, 177
nmppsAdd, 168, 169
nmppsAdd_AddC, 170
nmppsAddC, 165, 166
nmppsAddC_p, 166
nmppsAddr_, 263
nmppsAnd, 200
nmppsAnd4V_, 201
nmppsAndC, 199
nmppsAndNotV_, 202
nmppsClipCC_, 247
nmppsClipConvert_AddC_, 250
nmppsClipPowC_, 246
nmppsClipRShiftConvert_AddC_, 249
nmppsCmpEq0, 253
nmppsCmpEq0(Reduced), 242
 nmppsCmpEq0_16u15b, 243
 nmppsCmpEq0_32u31b, 243
 nmppsCmpEq0_8u7b, 244
nmppsCmpEq0_16u15b
 nmppsCmpEq0(Reduced), 243
nmppsCmpEq0_32u31b
 nmppsCmpEq0(Reduced), 243
nmppsCmpEq0_8u7b
 nmppsCmpEq0(Reduced), 244
nmppsCmpEqC, 251
nmppsCmpEqV_, 258
nmppsCmpGt0, 244
nmppsCmpGtC, 256
nmppsCmpGteC, 258
nmppsCmpLt, 260
nmppsCmpLt0, 240
nmppsCmpLtC, 242, 255
nmppsCmpLteC, 241
nmppsCmpNe, 259
nmppsCmpNe0, 252
nmppsCmpNeC, 254
nmppsCmpNeC_flag, 254
nmppsCmpNeV_, 261
nmppsConjMul, 185
nmppsConvert, 216, 218
 nmppsConvert_1s2s, 217
 nmppsConvert_1u2u, 218
nmppsConvert_1s2s
 nmppsConvert, 217
nmppsConvert_1u2u
 nmppsConvert, 218
nmppsConvert_rounding, 219
nmppsConvertRisc, 220
nmppsCopy, 222
nmppsCopy_, 221
nmppsCopyEvenToEven_32f, 223
nmppsCopyEvenToOdd_32f, 223
nmppsCopyOddToEven_32f, 223
nmppsCopyOddToOdd_32f, 222
nmppsCopyRisc, 224
nmppsCopyua_, 224
nmppsCreateResample, 129
nmppsDecimate, 273
nmppsDFT8Fwd_32fc
 DFT-8, 22
nmppsDisplaceBits, 211
nmppsDivC, 193
nmppsDotProd, 196
nmppsDotProd_sm, 195
nmppsFFT1024Inv_32fc
 IFFT-1024, 35
nmppsFFTFree_32fc
 Быстрое преобразование Фурье, 121
NmppsFFTSpec, 367
NmppsFFTSpec_32fc, 367
nmppsFIR_Xs, 124
nmppsFIRFree, 127
nmppsFIRGetSize_Xs, 127
nmppsFIRInit_Xs, 125
nmppsFIRInitAlloc_Xs, 126
nmppsFirstNonZeroIdx, 235
nmppsFirstZeroIdx, 234
NmppsFrame_16s, 368
NmppsFrame_16u, 368
NmppsFrame_32s, 368
NmppsFrame_32u, 368
NmppsFrame_64s, 369
NmppsFrame_64u, 369
NmppsFrame_8s, 369
NmppsFrame_8u, 369
nmppsFree, 40

nmppsGetVal_, 265
 nmppsGetVal_(return), 267
 nmppsLastNonZeroIndx, 236
 nmppsLastZeroIndx, 235
 nmppsMalloc, 39
 NmppsMallocSpec, 370
 nmppsMaskV_, 208
 nmppsMax_, 226
 nmppsMax_16s15b, 227
 nmppsMax_32s31b, 227
 nmppsMax_8s7b, 227
 nmppsMax_16s15b
 nmppsMax_, 227
 nmppsMax_32s31b
 nmppsMax_, 227
 nmppsMax_8s7b
 nmppsMax_, 227
 nmppsMaxEvery_, 238
 nmppsMaxIdx_, 230
 nmppsMerge, 220
 nmppsMin, 228
 nmppsMin_16s15b, 229
 nmppsMin_32s31b, 229
 nmppsMin_8s7b, 229
 nmppsMin_16s15b
 nmppsMin, 229
 nmppsMin_32s31b
 nmppsMin, 229
 nmppsMin_8s7b
 nmppsMin, 229
 nmppsMinCmpLtV_, 239
 nmppsMinEvery_, 237
 nmppsMinIdx_, 231
 nmppsMinIdxVN_, 232
 nmppsMinMaxEvery_, 245
 nmppsMul, 184
 nmppsMul_Add, 181
 nmppsMul_AddC, 187, 188
 nmppsMul_ConjMul_Add, 182
 nmppsMul_Mul_Add, 181
 nmppsMul_Mul_Sub, 186
 nmppsMulC, 179, 180
 nmppsMulC_Add, 187
 nmppsMulC_AddC, 183, 189
 nmppsMulC_AddC_2x32s, 190
 nmppsMulC_AddC_2x32s
 nmppsMulC_AddC, 190
 nmppsMulC_AddV_AddC, 184, 191
 nmppsNeg, 164
 nmppsNot_, 198
 nmppsOr, 203
 nmppsOr3V_, 204
 nmppsOr4V_, 205
 nmppsOrC, 202
 nmppSplit, 270
 nmppSplit_32fc, 271
 nmppSplitTmp, 269
 nmppsRamp_, 215
 nmppsRand, 71
 nmppsRandUniform, 213, 214
 nmppsRandUniform_64s, 214
 nmppsRandUniform_, 215
 nmppsRandUniform_64s
 nmppsRandUniform, 214
 nmppsRemap_, 268
 nmppsResample_perf, 131
 nmppsResampleDown2, 128
 nmppsResampleUp3Down2, 128
 nmppsRShiftC, 208
 nmppsRShiftC_, 209
 nmppsRShiftC_AddC_, 210
 nmppsRShiftC_MulC_AddC, 191
 nmppsSet-инициализация, 212
 nmppsSetResample, 130
 nmppsSetVal_, 264
 nmppsSub, 175, 176
 nmppsSubC, 170, 171
 nmppsSubCRev, 173, 174
 nmppsSum, 194
 nmppsSum_1, 195
 nmppsSumN, 192
 nmppsSwap_, 225
 nmppsThreshold_Lt_Gt_f, 248
 NmppsTmpSpec, 370
 nmppsWeightedSum, 197
 nmppsXCorr_32s, 122
 nmppsXor, 207
 nmppsXorC, 206
 nmreg, 370
 nmshort, 371
 nmshort2D< Y, X >, 371
 nmtype.h
 NM16Sx4, 566
 NM32Sx2, 566
 VEC_NM16S, 566
 VEC_NM16U, 567
 VEC_NM32S, 567
 VEC_NM4U, 567
 VEC_NM8S, 567
 nmvecpack< T >, 372
 operator<<
 контроль переполнения, 279
 PushRequest
 C_RingBuffer< T >, 356
 re
 s_nm64sc, 377
 Release
 C_2DSubPixelMinPosition, 343
 C_2DTrigSubPixelMinPosition, 344
 C_PlessyCornerDetector, 351
 RGB24_nm8u, 373
 RGB32_nm10s, 374
 RGB32_nm10u, 374
 RGB32_nm8s, 374

RGB32_nm8u, 374
RoundShift, 375
RPoint, 375

S_BufferInfo, 375
s_int32x2, 376
s_nm32fc, 376
s_nm32fcr, 377
s_nm32sc, 377
s_nm64sc, 377
 im, 377
 re, 377
S_nmppiFilterKernel, 378
S_nmppiFilterKernel_32s32s, 378
s_v16nm16s, 378
s_v16nm16u, 379
s_v16nm32s, 379
s_v16nm32u, 379
s_v16nm4u, 380
s_v16nm8s, 380
s_v16nm8u, 381
s_v2nm32f, 381
s_v2nm32s, 381
s_v2nm32u, 382
s_v4nm16s, 382
s_v4nm16u, 382
s_v4nm32f, 383
s_v4nm32s, 383
s_v4nm32u, 383
s_v4nm64f, 384
s_v4nm8u, 384
s_v8nm16s, 385
s_v8nm16u, 385
s_v8nm32s, 385
s_v8nm32u, 386
s_v8nm8s, 386
s_v8nm8u, 387
SetThreshold
 C_PlessyCornerDetector, 351
 C_PlessyCornerDetector_16s, 353
 C_PlessyCornerDetector_32f, 355
SpecTmp1, 387
spot_struct, 387

tagSegmentInfo, 388
tcube< T >, 388
tfixpoint< T, point >, 389
Tmp2BuffSpec, 390
tmtr< T >, 390

uint15b
 Типы скалярных данных, 148
uint16b
 Типы скалярных данных, 148
uint16ptr, 391
uint1b
 Типы скалярных данных, 148
uint2b
 Типы скалярных данных, 149

uint31b
 Типы скалярных данных, 149
uint32b
 Типы скалярных данных, 149
uint3b
 Типы скалярных данных, 149
uint4b
 Типы скалярных данных, 149
uint63b
 Типы скалярных данных, 150
uint64b
 Типы скалярных данных, 150
uint7b
 Типы скалярных данных, 150
uint8b
 Типы скалярных данных, 150
uint8ptr, 391

v16nm16s
 Типы векторных данных, 140
v16nm16u
 Типы векторных данных, 140
v16nm32s
 Типы векторных данных, 140
v16nm32u
 Типы векторных данных, 140
v16nm4b3u
 Типы векторных данных, 141
v16nm4s, 392
v16nm4u
 Типы векторных данных, 141
v16nm8s
 Типы векторных данных, 141
v16nm8s7b
 Типы векторных данных, 141
v16nm8u
 Типы векторных данных, 141
v2nm32s
 Типы векторных данных, 142
v2nm32u
 Типы векторных данных, 142
v4nm16s
 Типы векторных данных, 142
v4nm16u
 Типы векторных данных, 142
v4nm32s
 Типы векторных данных, 142
v4nm32u
 Типы векторных данных, 143
v4nm8s, 392
v4nm8u
 Типы векторных данных, 143
v8nm16s
 Типы векторных данных, 143
v8nm16u
 Типы векторных данных, 143
v8nm32s
 Типы векторных данных, 143
v8nm32u

Типы векторных данных, 144
 v8nm8s
 Типы векторных данных, 144
 v8nm8u
 Типы векторных данных, 144
`vec< T >`, 393
`Vec_0_sub_data`, 283
`Vec_Abs`, 299
`Vec_AccMul1D1W32_AddVr`, 340
`Vec_activate_data`, 284
`Vec_activate_data_add_0`, 285
`Vec_activate_data_add_ram`, 287
`Vec_activate_data_xor_data`, 286
`Vec_Add`, 300
`Vec_afifo`, 288
`Vec_And`, 297
`Vec_BuildDiagWeights16`, 334
`Vec_BuildDiagWeights8`, 334
`Vec_ClipExt`, 301
`Vec_ClipMul2D2W8_AddVr`, 302
`Vec_ClipMulNDNW2_AddVr`, 303
`Vec_ClipMulNDNW4_AddVr`, 304
`Vec_ClipMulNDNW8_AddVr`, 305
`Vec_ClipRShiftConvert_AddC`, 262
`Vec_CompareMaxV`, 331
`Vec_CompareMinV`, 331
`Vec_copyEvenToEven_32f`, 290
`Vec_CopyOddToEven_32f`, 291
`Vec_data`, 289
`Vec_data_add_afifo`, 291
`Vec_data_add_ram`, 292
`Vec_data_and_ram`, 293
`Vec_data_or_ram`, 293
`Vec_data_sub_ram`, 294
`Vec_data_xor_ram`, 295
`Vec_DupValueInVector16`, 333
`Vec_DupValueInVector8`, 332
`Vec_FilterCoreRow2`, 296
`Vec_FilterCoreRow4`, 296
`Vec_FilterCoreRow8`, 296
`Vec_IncNeg`, 306
`vec_Mask`
 Элементарные функции, 281
`Vec_MaxVal`, 336
`Vec_MaxVal_v4nm16s`, 335
`Vec_MaxVal_v8nm8s`, 335
`Vec_MinVal`, 338
`Vec_MinVal_v4nm16s`, 338
`Vec_MinVal_v8nm8s`, 337
`Vec_Mul2D2W1_AddVr`, 307
`Vec_Mul2D2W2_AddVr`, 308
`Vec_Mul2D2W4_AddVr`, 309
`Vec_Mul2D2W8_AddVr`, 310
`Vec_Mul3D3W2_AddVr`, 311
`Vec_Mul3D3W8_AddVr`, 312
`Vec_Mul4D4W2_AddVr`, 314
`Vec_MUL_2V4toW8_shift`, 320
`Vec_MUL_2V8toW16_shift`, 321
`Vec_MulVN_AddVN`, 315
`VEC_NM16S`
 nmtype.h, 566
`VEC_NM16U`
 nmtype.h, 567
`VEC_NM32S`
 nmtype.h, 567
`VEC_NM4U`
 nmtype.h, 567
`VEC_NM8S`
 nmtype.h, 567
`Vec_not_data`, 322
`Vec_Or`, 297
`VEC_QSort`, 268
`Vec_ram`, 323
`Vec_ram_sub_data`, 324
`Vec_Sub`, 316
`Vec_SubAbs`, 317
`Vec_SubVN_Abs`, 318
`Vec_Swap`, 319
`Vec_vsum_activate_data_0`, 325
`Vec_vsum_data_0`, 326
`Vec_vsum_data_afifo`, 326
`Vec_vsum_data_vr`, 327
`Vec_vsum_shift_data_0`, 328
`Vec_vsum_shift_data_afifo`, 330
`Vec_vsum_shift_data_vr`, 329
`Vec_Xor`, 298

`Wi_4096_fixed`, 395

 Арифметические действия, 84
 Арифметические операции, 74, 152, 153
 Базовые регистровые функции библиотеки, 278
 Блочное переупорядочивание, 81
 Быстрое преобразование Фурье, 120
 nmppsFFTFree_32fc, 121
 Векторно-матричные операции, 99
 Векторные функции, 274, 275
 Возведение в степень, 161
 Вычисление квадратного корня, 161
 Изменение размеров, 84, 119
 Инициализация, 72
 Инициализация и копирование, 84, 151, 152
 КИХ-фильтрация, 75, 124
 Логические и бинарные операции, 154
 Масочная фильтрация, 84, 119
 Матричные функции, 275
 Натуральный логарифм, 160
 Обращение матрицы, 99
 Операции выборки, 84
 Операции сравнения, 155, 157
 Переупорядочивание и сортировка, 157
 Переупорядочивание изображений, 81
 Свертка, 119
 Скалярные функции, 276
 Статистические функции, 157
 Типы векторных данных, 132
 nm1, 134

- nm16s, 135
- nm16s15b, 135
- nm16u, 135
- nm16u15b, 135
- nm2s, 136
- nm2u, 136
- nm32s, 136
- nm32s30b, 136
- nm32s31b, 137
- nm32u, 137
- nm32u31b, 137
- nm4s, 137
- nm4u, 138
- nm4u3b, 138
- nm64s, 138
- nm64s63b, 138
- nm64u, 139
- nm8s, 139
- nm8s7b, 139
- nm8u, 139
- nm8u7b, 140
- v16nm16s, 140
- v16nm16u, 140
- v16nm32s, 140
- v16nm32u, 140
- v16nm4b3u, 141
- v16nm4u, 141
- v16nm8s, 141
- v16nm8s7b, 141
- v16nm8u, 141
- v2nm32s, 142
- v2nm32u, 142
- v4nm16s, 142
- v4nm16u, 142
- v4nm32s, 142
- v4nm32u, 143
- v4nm8u, 143
- v8nm16s, 143
- v8nm16u, 143
- v8nm32s, 143
- v8nm32u, 144
- v8nm8s, 144
- v8nm8u, 144
- Типы данных, 274
- Типы скалярных данных, 145
 - int15b, 145
 - int16b, 146
 - int1b, 146
 - int2b, 146
 - int30b, 146
 - int31b, 146
 - int32b, 147
 - int3b, 147
 - int4b, 147
 - int63b, 147
 - int64b, 147
 - int7b, 148
 - int8b, 148
- uint15b, 148
- uint16b, 148
- uint1b, 148
- uint2b, 149
- uint31b, 149
- uint32b, 149
- uint3b, 149
- uint4b, 149
- uint63b, 150
- uint64b, 150
- uint7b, 150
- uint8b, 150
- Тригонометрические функции, 158
- Функции деинтерлейсинга, 74
 - nmppiDeinterlaceBlend, 74
 - nmppiDeinterlaceSplit, 75
- Функции обработки изображений, 276
- Функции обработки сигналов, 275
- Функции поддержки, 84, 98, 151
- Функции с плавающей точкой, 274
- Функция деления векторов, 159
- Целевые функции, 282
- Целочисленные функции, 274
- Экспонента, 159
- Элементарные математические функции, 158
- Элементарные функции, 280, 282
 - vec_Mask, 281
- контроль переполнения, 278
 - GetVec, 279
 - operator<<, 279
- функции взвешенного суммирования, 282