

JUK1

Generated by Doxygen 1.9.1

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 AutoDifferentiation Namespace Reference	9
5.1.1 Detailed Description	10
5.1.2 Function Documentation	10
5.1.2.1 requires() [1/3]	10
5.1.2.2 requires() [2/3]	10
5.1.2.3 requires() [3/3]	10
5.1.2.4 toRet() [1/3]	10
5.1.2.5 toRet() [2/3]	11
5.1.2.6 toRet() [3/3]	11
5.1.2.7 while()	11
5.1.3 Variable Documentation	11
5.1.3.1 AddableResult	11
5.1.3.2 arg	11
5.1.3.3 deriv	12
5.1.3.4 derivEval	12
5.1.3.5 DivisibleResult	12
5.1.3.6 exponent	12
5.1.3.7 func	12
5.1.3.8 it1	13
5.1.3.9 it2	13
5.1.3.10 it3	13
5.1.3.11 MultipliableResult	13
5.1.3.12 SubtractableResult	13
5.1.3.13 toRet	14
5.1.3.14 var	14
5.2 ForceCausal Namespace Reference	14
5.2.1 Function Documentation	14
5.2.1.1 F()	15
5.2.1.2 f0()	15
5.2.1.3 f0derivative()	16
5.2.1.4 getTau()	16

5.2.1.5 K()	17
6 Class Documentation	19
6.1 BJTN< T > Struct Template Reference	19
6.1.1 Detailed Description	20
6.1.2 Member Function Documentation	20
6.1.2.1 addDCAnalysisStampTo()	21
6.1.2.2 addNonLinearStampTo()	21
6.1.2.3 addToElements()	22
6.1.2.4 updateStoredState()	23
6.1.3 Member Data Documentation	23
6.1.3.1 alpha_f	23
6.1.3.2 alpha_r	23
6.1.3.3 b	23
6.1.3.4 c	24
6.1.3.5 e	24
6.1.3.6 l_cs	24
6.1.3.7 l_es	24
6.1.3.8 V_bc_crit	24
6.1.3.9 V_be_crit	24
6.1.3.10 V_Tc	25
6.1.3.11 V_Te	25
6.2 BJTP< T > Struct Template Reference	25
6.2.1 Detailed Description	26
6.2.2 Member Function Documentation	26
6.2.2.1 addDCAnalysisStampTo()	27
6.2.2.2 addNonLinearStampTo()	27
6.2.2.3 addToElements()	28
6.2.2.4 updateStoredState()	29
6.2.3 Member Data Documentation	29
6.2.3.1 alpha_f	29
6.2.3.2 alpha_r	29
6.2.3.3 b	29
6.2.3.4 c	30
6.2.3.5 e	30
6.2.3.6 l_cs	30
6.2.3.7 l_es	30
6.2.3.8 V_bc_crit	30
6.2.3.9 V_be_crit	30
6.2.3.10 V_Tc	31
6.2.3.11 V_Te	31
6.3 BJTResults< T > Struct Template Reference	31

6.3.1 Detailed Description	31
6.3.2 Member Data Documentation	31
6.3.2.1 g_cc	31
6.3.2.2 g_ce	32
6.3.2.3 g_ec	32
6.3.2.4 g_ee	32
6.3.2.5 l_c	32
6.3.2.6 l_e	32
6.4 Capacitor< T > Struct Template Reference	33
6.4.1 Detailed Description	34
6.4.2 Member Function Documentation	34
6.4.2.1 addDCAnalysisStampTo()	34
6.4.2.2 addDynamicStampTo()	35
6.4.2.3 addToElements()	35
6.4.2.4 updateStoredState()	36
6.4.3 Member Data Documentation	36
6.4.3.1 lastCurrent	36
6.4.3.2 n1	37
6.4.3.3 n2	37
6.4.3.4 trapezoidalRule	37
6.4.3.5 value	37
6.5 ForceCausal::CausalData< T > Struct Template Reference	37
6.5.1 Detailed Description	37
6.5.2 Member Data Documentation	38
6.5.2.1 data	38
6.5.2.2 tau	38
6.5.2.3 Ts	38
6.6 CircuitElements< T > Struct Template Reference	39
6.6.1 Detailed Description	40
6.6.2 Constructor & Destructor Documentation	40
6.6.2.1 CircuitElements()	40
6.6.3 Member Function Documentation	41
6.6.3.1 generateCompleteStamp()	41
6.6.3.2 generateDCStamp()	42
6.6.3.3 generateDynamicStamp()	42
6.6.3.4 generateNonLinearStamp()	43
6.6.3.5 generateStaticStamp()	44
6.6.3.6 setNewStampSize()	45
6.6.3.7 updateDCStoredState()	45
6.6.3.8 updateTimeStep()	46
6.6.4 Member Data Documentation	46
6.6.4.1 dcStamp	46

6.6.4.2 dynamicElements	47
6.6.4.3 dynamicStamp	47
6.6.4.4 dynamicStampIsFresh	47
6.6.4.5 nodeComponentMap	47
6.6.4.6 nonLinearElements	48
6.6.4.7 nonLinearStamp	48
6.6.4.8 nonLinearStampIsFresh	48
6.6.4.9 staticElements	48
6.6.4.10 staticStamp	48
6.6.4.11 staticStampIsFresh	49
6.7 Component< T > Struct Template Reference	49
6.7.1 Detailed Description	51
6.7.2 Constructor & Destructor Documentation	51
6.7.2.1 ~Component()	51
6.7.3 Member Function Documentation	52
6.7.3.1 addDCAnalysisStampTo()	52
6.7.3.2 addDynamicStampTo()	52
6.7.3.3 addNonLinearStampTo()	53
6.7.3.4 addStaticStampTo()	53
6.7.3.5 addToElements()	53
6.7.3.6 setTimestep()	54
6.7.3.7 updateDCStoredState()	54
6.7.3.8 updateStoredState()	55
6.7.4 Member Data Documentation	55
6.7.4.1 designator	55
6.8 CurrentSource< T > Struct Template Reference	56
6.8.1 Detailed Description	57
6.8.2 Member Function Documentation	57
6.8.2.1 addDCAnalysisStampTo()	57
6.8.2.2 addStaticStampTo()	58
6.8.2.3 addToElements()	58
6.8.3 Member Data Documentation	59
6.8.3.1 n1	59
6.8.3.2 n2	59
6.8.3.3 value	60
6.9 Diode< T > Struct Template Reference	60
6.9.1 Detailed Description	61
6.9.2 Member Function Documentation	61
6.9.2.1 addDCAnalysisStampTo()	61
6.9.2.2 addNonLinearStampTo()	62
6.9.2.3 addToElements()	63
6.9.2.4 updateStoredState()	64

6.9.3 Member Data Documentation	64
6.9.3.1 eta	64
6.9.3.2 l_sat	64
6.9.3.3 n1	64
6.9.3.4 n2	65
6.9.3.5 V_crit	65
6.9.3.6 V_T	65
6.10 Inductor< T > Struct Template Reference	65
6.10.1 Detailed Description	66
6.10.2 Member Function Documentation	67
6.10.2.1 addDCAnalysisStampTo()	67
6.10.2.2 addDynamicStampTo()	67
6.10.2.3 addToElements()	68
6.10.2.4 updateDCStoredState()	68
6.10.2.5 updateStoredState()	69
6.10.3 Member Data Documentation	69
6.10.3.1 dcCurrentIndex	69
6.10.3.2 lastCurrent	70
6.10.3.3 n1	70
6.10.3.4 n2	70
6.10.3.5 trapezoidalRule	70
6.10.3.6 value	70
6.11 LUPair< T > Struct Template Reference	71
6.11.1 Detailed Description	71
6.11.2 Constructor & Destructor Documentation	72
6.11.2.1 LUPair() [1/2]	72
6.11.2.2 LUPair() [2/2]	72
6.11.3 Member Function Documentation	72
6.11.3.1 toString()	72
6.11.4 Member Data Documentation	72
6.11.4.1 l	72
6.11.4.2 M	73
6.11.4.3 p	73
6.11.4.4 u	73
6.12 Matrix< T > Struct Template Reference	73
6.12.1 Detailed Description	74
6.12.2 Constructor & Destructor Documentation	74
6.12.2.1 Matrix() [1/2]	75
6.12.2.2 Matrix() [2/2]	75
6.12.3 Member Function Documentation	75
6.12.3.1 add() [1/2]	75
6.12.3.2 add() [2/2]	76

6.12.3.3 fill()	76
6.12.3.4 leftDivide() [1/3]	76
6.12.3.5 leftDivide() [2/3]	77
6.12.3.6 leftDivide() [3/3]	77
6.12.3.7 luPair() [1/2]	77
6.12.3.8 luPair() [2/2]	78
6.12.3.9 multiply() [1/2]	78
6.12.3.10 multiply() [2/2]	78
6.12.3.11 operator()() [1/2]	78
6.12.3.12 operator()() [2/2]	78
6.12.3.13 rowAddition()	79
6.12.3.14 subtract() [1/2]	79
6.12.3.15 subtract() [2/2]	79
6.12.3.16 swapRows()	79
6.12.3.17 toString()	80
6.12.3.18 transpose()	80
6.12.4 Member Data Documentation	80
6.12.4.1 data	80
6.12.4.2 M	80
6.12.4.3 N	81
6.13 NLCapacitor< T > Struct Template Reference	81
6.13.1 Detailed Description	82
6.13.2 Member Function Documentation	82
6.13.2.1 addDCAnalysisStampTo()	83
6.13.2.2 addNonLinearStampTo()	83
6.13.2.3 addToElements()	83
6.13.2.4 updateDCStoredState()	84
6.13.2.5 updateStoredState()	85
6.13.3 Member Data Documentation	85
6.13.3.1 C_last	85
6.13.3.2 C_o	85
6.13.3.3 C_p	86
6.13.3.4 i_last	86
6.13.3.5 n1	86
6.13.3.6 n2	86
6.13.3.7 P_10	86
6.13.3.8 P_11	86
6.13.3.9 u_last	87
6.14 NLCurrentSource< T > Struct Template Reference	87
6.14.1 Detailed Description	88
6.14.2 Member Function Documentation	88
6.14.2.1 addDCAnalysisStampTo()	88

6.14.2.2 addNonLinearStampTo()	90
6.14.2.3 addToElements()	91
6.14.3 Member Data Documentation	91
6.14.3.1 n1	92
6.14.3.2 n2	92
6.14.3.3 r1_neg	92
6.14.3.4 r1_pos	92
6.14.3.5 r2_neg	92
6.14.3.6 r2_pos	92
6.14.3.7 value	93
6.15 NLNMOS< T > Struct Template Reference	93
6.15.1 Detailed Description	94
6.15.2 Member Function Documentation	95
6.15.2.1 addNonLinearStampTo()	95
6.15.2.2 addToElements()	95
6.15.2.3 updateStoredState()	96
6.15.3 Member Data Documentation	96
6.15.3.1 alpha_DS	96
6.15.3.2 beta_DS	97
6.15.3.3 C_GD_last	97
6.15.3.4 C_GDo	97
6.15.3.5 C_GDp	97
6.15.3.6 C_GS_last	97
6.15.3.7 C_GSo	97
6.15.3.8 C_GSp	98
6.15.3.9 d	98
6.15.3.10 g	98
6.15.3.11 i_gd_last	98
6.15.3.12 i_gs_last	98
6.15.3.13 P_D10	98
6.15.3.14 P_D11	99
6.15.3.15 P_S10	99
6.15.3.16 P_S11	99
6.15.3.17 s	99
6.15.3.18 u_gd_last	99
6.15.3.19 u_gs_last	100
6.16 Resistor< T > Struct Template Reference	100
6.16.1 Detailed Description	101
6.16.2 Member Function Documentation	101
6.16.2.1 addDCAnalysisStampTo()	101
6.16.2.2 addStaticStampTo()	102
6.16.2.3 addToElements()	103

6.16.3 Member Data Documentation	103
6.16.3.1 currentIndex	103
6.16.3.2 group1	104
6.16.3.3 n1	104
6.16.3.4 n2	104
6.16.3.5 value	104
6.17 SimulationEnvironment< VT > Class Template Reference	105
6.17.1 Detailed Description	106
6.17.2 Constructor & Destructor Documentation	106
6.17.2.1 SimulationEnvironment()	106
6.17.3 Member Function Documentation	107
6.17.3.1 dataDump()	107
6.17.3.2 parseGraph()	108
6.17.3.3 printGraph()	108
6.17.3.4 printMultipleOnGraph()	108
6.17.3.5 setDCOpPoint()	110
6.17.3.6 simulate()	111
6.17.4 Member Data Documentation	111
6.17.4.1 elements	111
6.17.4.2 finalTime	112
6.17.4.3 initialTime	112
6.17.4.4 luPair	112
6.17.4.5 netlistPath	112
6.17.4.6 nodesToGraph	112
6.17.4.7 numCurrents	113
6.17.4.8 numDCCurrents	113
6.17.4.9 numNodes	113
6.17.4.10 outputPath	113
6.17.4.11 performDCAnalysis	113
6.17.4.12 scratchSpace	114
6.17.4.13 solutionMat	114
6.17.4.14 steps	114
6.17.4.15 timestep	114
6.18 SinusoidalVoltageSource< T > Struct Template Reference	115
6.18.1 Detailed Description	116
6.18.2 Member Function Documentation	116
6.18.2.1 addDCAnalysisStampTo()	116
6.18.2.2 addDynamicStampTo()	117
6.18.2.3 addToElements()	118
6.18.2.4 updateStoredState()	118
6.18.3 Member Data Documentation	118
6.18.3.1 currentIndex	119

6.18.3.2 degrees	119
6.18.3.3 frequency	119
6.18.3.4 n1	119
6.18.3.5 n2	119
6.18.3.6 offset	119
6.18.3.7 phase	120
6.18.3.8 V	120
6.19 SParameterBlock< T > Struct Template Reference	120
6.19.1 Detailed Description	121
6.19.2 Member Function Documentation	122
6.19.2.1 addDCAnalysisStampTo()	122
6.19.2.2 addDynamicStampTo()	122
6.19.2.3 addStaticStampTo()	123
6.19.2.4 addToElements()	123
6.19.2.5 aWaveConvValue()	124
6.19.2.6 beta_p()	125
6.19.2.7 R_p()	125
6.19.2.8 readInTouchstoneFile()	125
6.19.2.9 updateStoredState()	126
6.19.2.10 V_p()	126
6.19.3 Member Data Documentation	127
6.19.3.1 fracMaxToKeep	127
6.19.3.2 port	127
6.19.3.3 s	127
6.19.3.4 touchstoneFilePath	128
6.19.3.5 z_ref	128
6.20 SParameterBlockVF< T > Struct Template Reference	128
6.20.1 Detailed Description	129
6.20.2 Member Function Documentation	130
6.20.2.1 addDCAnalysisStampTo()	130
6.20.2.2 addDynamicStampTo()	130
6.20.2.3 addStaticStampTo()	131
6.20.2.4 addToElements()	131
6.20.2.5 aware_p()	132
6.20.2.6 bwave_p()	132
6.20.2.7 history_p()	133
6.20.2.8 readInPRR()	133
6.20.2.9 setConstants()	134
6.20.2.10 setFirstOrder()	134
6.20.2.11 setSecondOrder()	135
6.20.2.12 setTimeStep()	135
6.20.2.13 updateStoredState()	136

6.20.2.14 V_p()	137
6.20.3 Member Data Documentation	137
6.20.3.1 firstOrder	137
6.20.3.2 numPorts	138
6.20.3.3 port	138
6.20.3.4 z_ref	138
6.21 SParameterPort< T > Struct Template Reference	138
6.21.1 Detailed Description	138
6.21.2 Member Data Documentation	139
6.21.2.1 beta	139
6.21.2.2 current	139
6.21.2.3 negative	139
6.21.2.4 positive	139
6.21.2.5 R	140
6.21.2.6 s0	140
6.22 SParameterPortVF< T > Struct Template Reference	140
6.22.1 Detailed Description	140
6.22.2 Member Data Documentation	141
6.22.2.1 alpha	141
6.22.2.2 beta	141
6.22.2.3 current	141
6.22.2.4 from	141
6.22.2.5 negative	142
6.22.2.6 positive	142
6.22.2.7 R	142
6.23 SParameterSequence< T > Struct Template Reference	142
6.23.1 Detailed Description	143
6.23.2 Member Function Documentation	143
6.23.2.1 data() [1/2]	143
6.23.2.2 data() [2/2]	144
6.23.2.3 length() [1/2]	144
6.23.2.4 length() [2/2]	144
6.23.2.5 offset() [1/2]	145
6.23.2.6 offset() [2/2]	145
6.23.2.7 time() [1/2]	145
6.23.2.8 time() [2/2]	146
6.23.3 Member Data Documentation	146
6.23.3.1 _data	146
6.23.3.2 _time	147
6.23.3.3 numPorts	147
6.23.3.4 sParamLengthOffset	147
6.24 SParamLengthOffset Struct Reference	147

6.24.1 Detailed Description	147
6.24.2 Member Data Documentation	147
6.24.2.1 length	148
6.24.2.2 offset	148
6.25 SParamVFDataFrom< T > Struct Template Reference	148
6.25.1 Detailed Description	149
6.25.2 Member Data Documentation	149
6.25.2.1 exp_alpha	149
6.25.2.2 lambda	149
6.25.2.3 lambda_p	149
6.25.2.4 mu	149
6.25.2.5 mu_p	150
6.25.2.6 nu	150
6.25.2.7 nu_p	150
6.25.2.8 numPoles	150
6.25.2.9 pole	150
6.25.2.10 remainder	151
6.25.2.11 residue	151
6.25.2.12 x	151
6.26 Stamp< T > Struct Template Reference	151
6.26.1 Detailed Description	152
6.26.2 Constructor & Destructor Documentation	152
6.26.2.1 Stamp()	152
6.26.3 Member Function Documentation	153
6.26.3.1 add()	153
6.26.3.2 addDCAnalysisStamp()	153
6.26.3.3 addDynamicStamp()	153
6.26.3.4 addNonLinearStamp()	154
6.26.3.5 addStaticStamp()	154
6.26.3.6 clear()	154
6.26.3.7 solve()	155
6.26.4 Member Data Documentation	155
6.26.4.1 G	155
6.26.4.2 s	155
6.26.4.3 sizeG_A	155
6.26.4.4 sizeG_D	156
6.27 StaticLUPair< T, M > Struct Template Reference	156
6.27.1 Detailed Description	156
6.27.2 Member Function Documentation	157
6.27.2.1 toString()	157
6.27.3 Member Data Documentation	157
6.27.3.1 I	157

6.27.3.2 p	157
6.27.3.3 u	158
6.28 StaticMatrix< T, M, N, ST > Struct Template Reference	158
6.28.1 Detailed Description	159
6.28.2 Constructor & Destructor Documentation	159
6.28.2.1 StaticMatrix() [1/2]	159
6.28.2.2 StaticMatrix() [2/2]	159
6.28.3 Member Function Documentation	160
6.28.3.1 add() [1/2]	160
6.28.3.2 add() [2/2]	160
6.28.3.3 fill()	161
6.28.3.4 leftDivide() [1/2]	161
6.28.3.5 leftDivide() [2/2]	161
6.28.3.6 luPair() [1/2]	162
6.28.3.7 luPair() [2/2]	162
6.28.3.8 multiply() [1/2]	163
6.28.3.9 multiply() [2/2]	163
6.28.3.10 operator[]() [1/2]	163
6.28.3.11 operator[]() [2/2]	163
6.28.3.12 rowAddition()	164
6.28.3.13 subtract() [1/2]	164
6.28.3.14 subtract() [2/2]	164
6.28.3.15 swapRows()	165
6.28.3.16 toString()	165
6.28.3.17 transpose()	165
6.28.4 Member Data Documentation	166
6.28.4.1 rows	166
6.28.4.2 sizeM	166
6.28.4.3 sizeN	166
6.29 StaticRow< T, N, ST > Struct Template Reference	166
6.29.1 Detailed Description	167
6.29.2 Constructor & Destructor Documentation	167
6.29.2.1 StaticRow() [1/2]	167
6.29.2.2 StaticRow() [2/2]	167
6.29.3 Member Function Documentation	168
6.29.3.1 dot()	168
6.29.3.2 fill()	168
6.29.3.3 operator[]() [1/2]	168
6.29.3.4 operator[]() [2/2]	168
6.29.4 Member Data Documentation	168
6.29.4.1 columns	169
6.29.4.2 sizeN	169

6.30 TimeSeriesVoltageSource< T > Struct Template Reference	169
6.30.1 Detailed Description	170
6.30.2 Member Function Documentation	170
6.30.2.1 addDCAnalysisStampTo()	171
6.30.2.2 addDynamicStampTo()	171
6.30.2.3 addToElements()	172
6.30.2.4 lerp()	173
6.30.2.5 readInTimeSeries()	173
6.30.2.6 updateStoredState()	173
6.30.3 Member Data Documentation	174
6.30.3.1 currentIndex	174
6.30.3.2 dataSeries	174
6.30.3.3 lastTimeSeriesIndex	174
6.30.3.4 n1	174
6.30.3.5 n2	175
6.30.3.6 timeSeries	175
6.31 TransistorTestResult< T > Struct Template Reference	175
6.31.1 Detailed Description	175
6.31.2 Member Data Documentation	175
6.31.2.1 diff1	175
6.31.2.2 diff2	176
6.31.2.3 var	176
6.32 VoltageSource< T > Struct Template Reference	176
6.32.1 Detailed Description	177
6.32.2 Member Function Documentation	177
6.32.2.1 addDCAnalysisStampTo()	177
6.32.2.2 addStaticStampTo()	178
6.32.2.3 addToElements()	179
6.32.3 Member Data Documentation	179
6.32.3.1 currentIndex	179
6.32.3.2 n1	180
6.32.3.3 n2	180
6.32.3.4 value	180
7 File Documentation	181
7.1 AutoDifferentiation.hpp File Reference	181
7.2 AutoDifferentiation.hpp	183
7.3 AutoDiffTest.cpp File Reference	188
7.3.1 Function Documentation	189
7.3.1.1 main()	189
7.3.1.2 testBasicOutput_NoCheck()	189
7.3.1.3 testBJTModel()	190

7.3.1.4 testBJTModelAutoDiff()	190
7.3.1.5 testBJTModelControl()	191
7.3.1.6 transistorTest()	191
7.3.1.7 transistorTestAutoDiff()	192
7.3.1.8 transistorTestControl()	192
7.4 AutoDiffTest.cpp	193
7.5 BJT.hpp File Reference	197
7.6 BJT.hpp	198
7.7 Capacitor.hpp File Reference	201
7.8 Capacitor.hpp	202
7.9 CircuitElements.hpp File Reference	204
7.9.1 Enumeration Type Documentation	205
7.9.1.1 ComponentType	205
7.9.1.2 SolutionStage	205
7.10 CircuitElements.hpp	206
7.11 CircuitSimulator.cpp File Reference	208
7.11.1 Function Documentation	209
7.11.1.1 main()	209
7.12 CircuitSimulator.cpp	209
7.13 Component.hpp File Reference	211
7.14 Component.hpp	211
7.15 CurrentSource.hpp File Reference	213
7.16 CurrentSource.hpp	214
7.17 dft.hpp File Reference	214
7.17.1 Function Documentation	216
7.17.1.1 _fftHelperRadix2()	216
7.17.1.2 dft()	216
7.17.1.3 fft()	217
7.17.1.4 idft()	218
7.17.1.5 ifft()	218
7.17.1.6 nthRootOfUnity() [1/2]	218
7.17.1.7 nthRootOfUnity() [2/2]	219
7.18 dft.hpp	219
7.19 Diode.hpp File Reference	220
7.20 Diode.hpp	221
7.21 DynamicMatrix.hpp File Reference	222
7.21.1 Function Documentation	223
7.21.1.1 operator<()	224
7.21.1.2 operator<=()	224
7.21.1.3 operator>()	224
7.21.1.4 operator>=()	224
7.21.2 Variable Documentation	224

7.21.2.1 arithmetic	225
7.22 DynamicMatrix.hpp	225
7.23 ElementsRegexBuilder.cpp File Reference	228
7.23.1 Function Documentation	229
7.23.1.1 generateRegex()	229
7.24 ElementsRegexBuilder.hpp	230
7.25 ElementsRegexBuilder.h File Reference	232
7.25.1 Function Documentation	232
7.25.1.1 generateRegex()	232
7.26 ElementsRegexBuilder.h	234
7.27 fft.cpp File Reference	235
7.27.1 Function Documentation	235
7.27.1.1 main()	235
7.27.1.2 myFunctionToSample()	236
7.28 fft.hpp	236
7.29 ForceCausal.hpp File Reference	237
7.29.1 Function Documentation	239
7.29.1.1 forceCausal()	239
7.30 ForceCausal.hpp	240
7.31 Inductor.hpp File Reference	241
7.32 Inductor.hpp	242
7.33 NLCapacitor.hpp File Reference	244
7.34 NLCapacitor.hpp	245
7.35 NLCurrentSource.hpp File Reference	247
7.36 NLCurrentSource.hpp	248
7.37 NLMOS.hpp File Reference	250
7.38 NLMOS.hpp	251
7.39 Resistor.hpp File Reference	254
7.40 Resistor.hpp	255
7.41 Simulator.hpp File Reference	256
7.41.1 Enumeration Type Documentation	257
7.41.1.1 LineType	257
7.41.1.2 SourceType	257
7.42 Simulator.hpp	258
7.43 SinusoidalVoltageSource.hpp File Reference	263
7.44 SinusoidalVoltageSource.hpp	264
7.45 SParameterBlock.hpp File Reference	265
7.46 SParameterBlock.hpp	267
7.47 SParameterBlockVF.hpp File Reference	271
7.48 SParameterBlockVF.hpp	272
7.49 StaticMatrix.hpp File Reference	278
7.49.1 Typedef Documentation	279

7.49.1.1 storageType	280
7.49.2 Function Documentation	280
7.49.2.1 operator<()	280
7.49.2.2 operator<=()	280
7.49.2.3 operator>()	280
7.49.2.4 operator>=()	281
7.49.3 Variable Documentation	281
7.49.3.1 arithmetic	281
7.50 StaticMatrix.hpp	281
7.51 test.cpp File Reference	285
7.51.1 Function Documentation	286
7.51.1.1 main()	286
7.52 test.cpp	287
7.53 TimeSeriesVoltageSource.hpp File Reference	288
7.54 TimeSeriesVoltageSource.hpp	289
7.55 VoltageSource.hpp File Reference	291
7.56 VoltageSource.hpp	292
Index	295

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

AutoDifferentiation	Namespace to hold the messiness of my auto-differentiator	9
ForceCausal	14

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BJTResults< T >	31
ForceCausal::CausalData< T >	37
CircuitElements< T >	39
CircuitElements< VT >	39
Component< T >	49
BJTN< T >	19
BJTP< T >	25
Capacitor< T >	33
CurrentSource< T >	56
Diode< T >	60
Inductor< T >	65
NLCapacitor< T >	81
NLCurrentSource< T >	87
NLNMOS< T >	93
Resistor< T >	100
SParameterBlock< T >	120
SParameterBlockVF< T >	128
SinusoidalVoltageSource< T >	115
TimeSeriesVoltageSource< T >	169
VoltageSource< T >	176
LUPair< T >	71
LUPair< VT >	71
Matrix< T >	73
Matrix< VT >	73
SimulationEnvironment< VT >	105
SParameterPort< T >	138
SParameterPortVF< T >	140
SParameterSequence< T >	142
SParamLengthOffset	147
SParamVFDataFrom< T >	148
Stamp< T >	151
StaticLUPair< T, M >	156
StaticMatrix< T, M, N, ST >	158
StaticMatrix< T, M, M >	158
StaticRow< T, N, ST >	166
TransistorTestResult< T >	175

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

BJTN< T >	
A simple NPN BJT model	19
BJTP< T >	
A simple PNP BJT model	25
BJTResults< T >	
.	31
Capacitor< T >	
An ideal capacitor model	33
ForceCausal::CausalData< T >	
Helper struct to return the result of the forced causal IDFT	37
CircuitElements< T >	
Glorified container for the different types of components	39
Component< T >	
A template base class to define the fundamental things a component should define	49
CurrentSource< T >	
An ideal current source	56
Diode< T >	
An ebbers moll diode model	60
Inductor< T >	
An ideal inductor model	65
LUPair< T >	
A helper class to store the L U and pivot matrices. Mainly useful in solving systems of equations	71
Matrix< T >	
A matrix class with support for LU-decomposition, and left division	73
NLCapacitor< T >	
Non-linear capacitor model of the form $C = C_p + C_o * (1.0 + \tanh(P_{10} + P_{11} * u))$	81
NLCurrentSource< T >	
An non-linear current source for the COBRA transistor model	87
NLNMOS< T >	
Non-linear FET model	93
Resistor< T >	
An ideal resistor model	100
SimulationEnvironment< VT >	
The main class to hold all of the relevant simulation data	105
SinusoidalVoltageSource< T >	
A sinusoidal voltage source model	115

SParameterBlock< T >	
A DTIR based model of an s-parameter block	120
SParameterBlockVF< T >	
A vectorfitting based model of an s-parameter block	128
SParameterPort< T >	
Helper struct to store the information for the ports of an S-Parameter Block	138
SParameterPortVF< T >	
Helper struct to store the information for the ports of an S-Parameter Block	140
SParameterSequence< T >	
Helper struct to store the DTIR sequence for the bloc	142
SParamLengthOffset	147
SParamVFDDataFrom< T >	148
Stamp< T >	
A helper struct to store the preallocated stamps for MNA	151
StaticLUPair< T, M >	
A compile-time sized L U and pivot grouping	156
StaticMatrix< T, M, N, ST >	
A compile-time sized matrix	158
StaticRow< T, N, ST >	
A compile-time sized matrix row	166
TimeSeriesVoltageSource< T >	
A time series voltage source model	169
TransistorTestResult< T >	175
VoltageSource< T >	
An ideal voltageSource	176

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

AutoDifferentiation.hpp	181
AutoDiffTest.cpp	188
BJT.hpp	197
Capacitor.hpp	201
CircuitElements.hpp	204
CircuitSimulator.cpp	208
Component.hpp	211
CurrentSource.hpp	213
dft.hpp	214
Diode.hpp	220
DynamicMatrix.hpp	222
ElementsRegexBuilder.cpp	228
ElementsRegexBuilder.h	232
fft.cpp	235
ForceCausal.hpp	237
Inductor.hpp	241
NLCapacitor.hpp	244
NLCurrentSource.hpp	247
NLNMOS.hpp	250
Resistor.hpp	254
Simulator.hpp	256
SinusoidalVoltageSource.hpp	263
SParameterBlock.hpp	265
SParameterBlockVF.hpp	271
StaticMatrix.hpp	278
test.cpp	285
TimeSeriesVoltageSource.hpp	288
VoltageSource.hpp	291

Chapter 5

Namespace Documentation

5.1 AutoDifferentiation Namespace Reference

a namespace to hold the messiness of my auto-differentiator

Functions

- template<typename ValType , size_t NumVars>
 requires (std::is_arithmetic< ValType >::value &&NumVars >=0) struct DiffVar
- template<typename ValType , size_t NumVars, typename F1 , typename F2 >
 requires (std::is_arithmetic< ValType >::value &&NumVars >=0) const expr auto diffFunc(const DiffVar< ValType
 • **while** (**it1** !=toRet.diffVars.end() &&**it2** !=arg.diffVars.end())
 • DiffVar< ValType, NumVars > **toRet** (**derivEval**)
 • template<typename ValType , size_t NumVars, typename OT >
 requires (std::is_arithmetic< ValType >::value &&NumVars >=0) &&(!std
 • DiffVar< ValType, NumVars > **toRet** (**func**(arg.var, exponent.var))
 • DiffVar< ValType, NumVars > **toRet** (**funcResult**)

Variables

- template<typename RT , typename LT , typename OT >
 concept **MultipliableResult**
- template<typename RT , typename LT , typename OT >
 concept **AddableResult**
- template<typename RT , typename LT , typename OT >
 concept **SubtractableResult**
- template<typename RT , typename LT , typename OT >
 concept **DivisibleResult**
- NumVars & **arg**
- NumVars F1 **func**
- NumVars F1 F2 **deriv**
- auto **it1** = **toRet**.diffVars.begin()
- auto **it2** = **arg**.diffVars.begin()
- auto **derivEval** = **deriv**(arg.var)
- return **toRet**
- **toRet** var = **derivEval**
- NumVars DiffVar< ValType, NumVars > **exponent**
- auto **it3** = **exponent**.diffVars.begin()

5.1.1 Detailed Description

a namespace to hold the messiness of my auto-differentiator

In essence the idea of this header is to allow easy determination of the jacobian of a non-linear element without having to manually differentiate a function. Sometimes this can also lead to faster implementations than an analytical derivative. The main advantage though, is being able to express the equation once in a natural format.

5.1.2 Function Documentation

5.1.2.1 `requires()` [1/3]

```
template<typename ValType , size_t NumVars, typename OT >
AutoDifferentiation::requires (
    std::is_arithmetic< ValType >::value &&NumVars >= 0 ) &&
```

Definition at line 368 of file [AutoDifferentiation.hpp](#).

5.1.2.2 `requires()` [2/3]

```
template<typename ValType , size_t NumVars, typename F1 , typename F2 >
AutoDifferentiation::requires (
    std::is_arithmetic< ValType >::value &&NumVars >= 0 ) const
```

5.1.2.3 `requires()` [3/3]

```
template<typename ValType , size_t NumVars>
AutoDifferentiation::requires (
    std::is_arithmetic< ValType >::value &&NumVars >= 0 )
```

Definition at line 41 of file [AutoDifferentiation.hpp](#).

5.1.2.4 `toRet()` [1/3]

```
DiffVar<ValType, NumVars> AutoDifferentiation::toRet (
    derivEval )
```

5.1.2.5 `toRet()` [2/3]

```
DiffVar<ValType, NumVars> AutoDifferentiation::toRet (
    func(arg.var, exponent.var) )
```

5.1.2.6 `toRet()` [3/3]

```
DiffVar<ValType, NumVars> AutoDifferentiation::toRet (
    funcResult )
```

5.1.2.7 `while()`

```
AutoDifferentiation::while (
    it1 != toRet.diffVars.end() && it2 != arg.diffVars.end() )
```

Definition at line 294 of file [AutoDifferentiation.hpp](#).

5.1.3 Variable Documentation

5.1.3.1 `AddableResult`

```
template<typename RT, typename LT, typename OT>
concept AutoDifferentiation::AddableResult
```

Initial value:

```
= requires(LT lhs, OT rhs) {
    std::is_same<RT, decltype(lhs += rhs)>::value;
}
```

Definition at line 25 of file [AutoDifferentiation.hpp](#).

5.1.3.2 `arg`

```
NumVars & AutoDifferentiation::arg
```

Initial value:

```
{
    return diffFunc(
        arg, [](ValType v) { return std::sin(v); },
        [](ValType v) { return std::cos(v); })
```

Definition at line 287 of file [AutoDifferentiation.hpp](#).

5.1.3.3 deriv

```
auto AutoDifferentiation::deriv
```

Initial value:

```
{
    DiffVar<ValType, NumVars> toRet(func(arg.var))
```

Definition at line 288 of file [AutoDifferentiation.hpp](#).

5.1.3.4 derivEval

```
auto AutoDifferentiation::derivEval = deriv(arg.var)
```

Definition at line 293 of file [AutoDifferentiation.hpp](#).

5.1.3.5 DivisibleResult

```
template<typename RT, typename LT, typename OT>
concept AutoDifferentiation::DivisibleResult
```

Initial value:

```
= requires(LT lhs, OT rhs) {
    std::is_same<RT, decltype(lhs /= rhs)>::value;
}
```

Definition at line 35 of file [AutoDifferentiation.hpp](#).

5.1.3.6 exponent

```
NumVars DiffVar<ValType, NumVars> AutoDifferentiation::exponent
```

Initial value:

```
{
    auto func = [](ValType v, ValType exponent) { return std::pow(v, exponent); }
```

Definition at line 379 of file [AutoDifferentiation.hpp](#).

5.1.3.7 func

```
NumVars F1 AutoDifferentiation::func
```

Definition at line 287 of file [AutoDifferentiation.hpp](#).

5.1.3.8 it1

```
auto AutoDifferentiation::it1 = toRet.diffVars.begin()
```

Definition at line 291 of file [AutoDifferentiation.hpp](#).

5.1.3.9 it2

```
auto AutoDifferentiation::it2 = arg.diffVars.begin()
```

Definition at line 292 of file [AutoDifferentiation.hpp](#).

5.1.3.10 it3

```
auto AutoDifferentiation::it3 = exponent.diffVars.begin()
```

Definition at line 390 of file [AutoDifferentiation.hpp](#).

5.1.3.11 MultipliableResult

```
template<typename RT , typename LT , typename OT >
concept AutoDifferentiation::MultipliableResult
```

Initial value:

```
= requires(LT lhs, OT rhs) {
    std::is_same<RT, decltype(lhs *= rhs)>::value;
}
```

Definition at line 20 of file [AutoDifferentiation.hpp](#).

5.1.3.12 SubtractableResult

```
template<typename RT , typename LT , typename OT >
concept AutoDifferentiation::SubtractableResult
```

Initial value:

```
= requires(LT lhs, OT rhs) {
    std::is_same<RT, decltype(lhs -= rhs)>::value;
}
```

Definition at line 30 of file [AutoDifferentiation.hpp](#).

5.1.3.13 `toRet`

```
return AutoDifferentiation::toRet
```

Definition at line 299 of file [AutoDifferentiation.hpp](#).

5.1.3.14 `var`

```
toRet AutoDifferentiation::var = derivEval
```

Definition at line 363 of file [AutoDifferentiation.hpp](#).

5.2 ForceCausal Namespace Reference

Classes

- struct [CausalData](#)
a helper struct to return the result of the forced causal IDFT

Functions

- template<typename T >
`std::complex< T > F (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau, T k, size_t n)`
- template<typename T >
`T K (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau)`
- template<typename T >
`T f0 (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau)`
- template<typename T >
`T f0derivative (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau, T step)`
- template<typename T >
`T getTau (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tol=1e-7, size_t maxIter=30, T step=1e-8)`

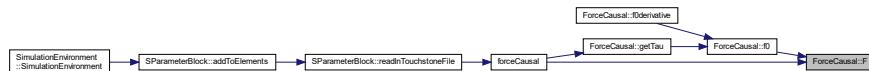
5.2.1 Function Documentation

5.2.1.1 F()

```
template<typename T >
std::complex<T> ForceCausal::F (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data,
    T tau,
    T k,
    size_t n )
```

Definition at line 12 of file [ForceCausal.hpp](#).

Here is the caller graph for this function:

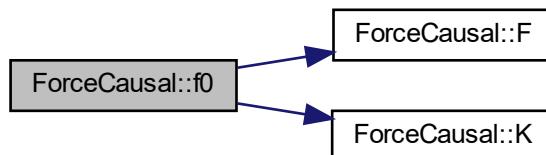


5.2.1.2 f0()

```
template<typename T >
T ForceCausal::f0 (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data,
    T tau )
```

Definition at line 28 of file [ForceCausal.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

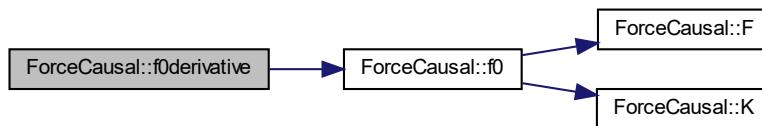


5.2.1.3 f0derivative()

```
template<typename T >
T ForceCausal::f0derivative (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data,
    T tau,
    T step )
```

Definition at line 42 of file [ForceCausal.hpp](#).

Here is the call graph for this function:

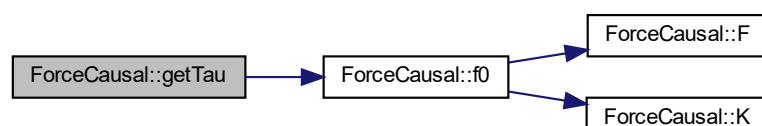


5.2.1.4 getTau()

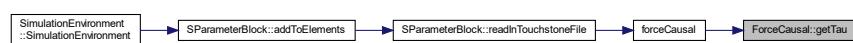
```
template<typename T >
T ForceCausal::getTau (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data,
    T tol = 1e-7,
    size_t maxIter = 30,
    T step = 1e-8 )
```

Definition at line 50 of file [ForceCausal.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.1.5 K()

```
template<typename T >
T ForceCausal::K (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data,
    T tau )
```

Definition at line 20 of file [ForceCausal.hpp](#).

Here is the caller graph for this function:



Chapter 6

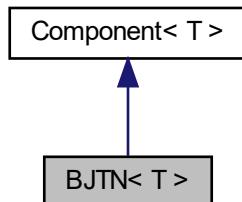
Class Documentation

6.1 BJT< T > Struct Template Reference

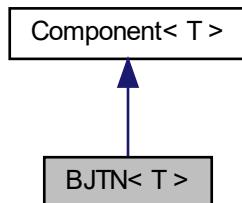
A simple NPN BJT model.

```
#include <BJT.hpp>
```

Inheritance diagram for BJT< T >:



Collaboration diagram for BJT< T >:



Public Member Functions

- void `addNonLinearStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep=0) const
adds this component's non-linear stamp to the target stamp.`
- void `updateStoredState` (const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep, `size_t` sizeG_A)
Updates any stored state based on the current solution index.
- void `addDCAAnalysisStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const
adds this component's DC stamp to the target stamp.`

Static Public Member Functions

- static void `addToElements` (const std::string &line, `CircuitElements< T >` &elements, `size_t` &numNodes, `size_t` &numCurrents, `size_t` &numDCCurrents)

Public Attributes

- `size_t` **c** = 0
- `size_t` **b** = 0
- `size_t` **e** = 0
- `T` **alpha_f** = 0.99
- `T` **alpha_r** = 0.02
- const `T` **I_es** = 2e-14
- const `T` **V_Te** = 26e-3
- const `T` **I_cs** = 99e-14
- const `T` **V_Tc** = 26e-3
- `T` **V_bc_crit** = **V_Tc** * std::log(**V_Tc** / (**I_cs** * std::sqrt(2)))
- `T` **V_be_crit** = **V_Te** * std::log(**V_Te** / (**I_es** * std::sqrt(2)))

6.1.1 Detailed Description

```
template<typename T>
struct BJTN< T >
```

A simple NPN BJT model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 10 of file [BJT.hpp](#).

6.1.2 Member Function Documentation

6.1.2.1 addDCAnalysisStampTo()

```
template<typename T >
void BJTN< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 113 of file [BJT.hpp](#).

Here is the call graph for this function:



6.1.2.2 addNonLinearStampTo()

```
template<typename T >
void BJTN< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 29 of file [BJT.hpp](#).

Here is the caller graph for this function:



6.1.2.3 addToElements()

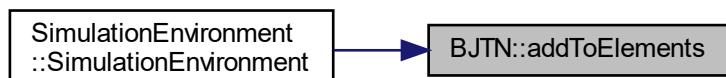
```
template<typename T >
static void BJTN< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 119 of file [BJT.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.4 updateStoredState()

```
template<typename T >
void BJTN< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 108 of file [BJT.hpp](#).

6.1.3 Member Data Documentation

6.1.3.1 alpha_f

```
template<typename T >
T BJTN< T >::alpha_f = 0.99
```

Definition at line 16 of file [BJT.hpp](#).

6.1.3.2 alpha_r

```
template<typename T >
T BJTN< T >::alpha_r = 0.02
```

Definition at line 17 of file [BJT.hpp](#).

6.1.3.3 b

```
template<typename T >
size_t BJTN< T >::b = 0
```

Definition at line 13 of file [BJT.hpp](#).

6.1.3.4 c

```
template<typename T >
size_t BJT< T >::c = 0
```

Definition at line 12 of file [BJT.hpp](#).

6.1.3.5 e

```
template<typename T >
size_t BJT< T >::e = 0
```

Definition at line 14 of file [BJT.hpp](#).

6.1.3.6 I_cs

```
template<typename T >
const T BJT< T >::I_cs = 99e-14
```

Definition at line 21 of file [BJT.hpp](#).

6.1.3.7 I_es

```
template<typename T >
const T BJT< T >::I_es = 2e-14
```

Definition at line 19 of file [BJT.hpp](#).

6.1.3.8 V_bc_crit

```
template<typename T >
T BJT< T >::V_bc_crit = V_Tc * std::log(V_Tc / (I_cs * std::sqrt(2)))
```

Definition at line 24 of file [BJT.hpp](#).

6.1.3.9 V_be_crit

```
template<typename T >
T BJT< T >::V_be_crit = V_Te * std::log(V_Te / (I_es * std::sqrt(2)))
```

Definition at line 25 of file [BJT.hpp](#).

6.1.3.10 V_Tc

```
template<typename T >
const T BJTN< T >::V\_Tc = 26e-3
```

Definition at line 22 of file [BJT.hpp](#).

6.1.3.11 V_Te

```
template<typename T >
const T BJTN< T >::V\_Te = 26e-3
```

Definition at line 20 of file [BJT.hpp](#).

The documentation for this struct was generated from the following file:

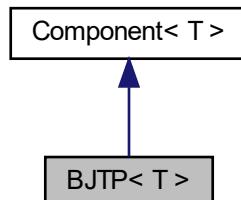
- [BJT.hpp](#)

6.2 BJTP< T > Struct Template Reference

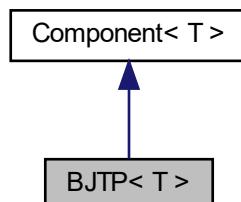
A simple PNP BJT model.

```
#include <BJT.hpp>
```

Inheritance diagram for BJTP< T >:



Collaboration diagram for BJTP< T >:



Public Member Functions

- void `addNonLinearStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep=0) const
adds this component's non-linear stamp to the target stamp.`
- void `updateStoredState` (const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep, `size_t` sizeG_A)
Updates any stored state based on the current solution index.
- void `addDCAAnalysisStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const
adds this component's DC stamp to the target stamp.`

Static Public Member Functions

- static void `addToElements` (const `std::string` &line, `CircuitElements< T >` &elements, `size_t` &numNodes, `size_t` &numCurrents, `size_t` &numDCCurrents)

Public Attributes

- `size_t` **c** = 0
- `size_t` **b** = 0
- `size_t` **e** = 0
- `T` **alpha_f** = 0.99
- `T` **alpha_r** = 0.02
- const `T` **I_es** = `2e-14`
- const `T` **V_Te** = `26e-3`
- const `T` **I_cs** = `99e-14`
- const `T` **V_Tc** = `26e-3`
- `T` **V_bc_crit** = `V_Tc * std::log(V_Tc / (I_cs * std::sqrt(2)))`
- `T` **V_be_crit** = `V_Te * std::log(V_Te / (I_es * std::sqrt(2)))`

6.2.1 Detailed Description

```
template<typename T>
struct BJTP< T >
```

A simple PNP BJT model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 151 of file [BJT.hpp](#).

6.2.2 Member Function Documentation

6.2.2.1 addDCAnalysisStampTo()

```
template<typename T >
void BJTP< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 260 of file [BJT.hpp](#).

Here is the call graph for this function:



6.2.2.2 addNonLinearStampTo()

```
template<typename T >
void BJTP< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 170 of file [BJT.hpp](#).

Here is the caller graph for this function:



6.2.2.3 addToElements()

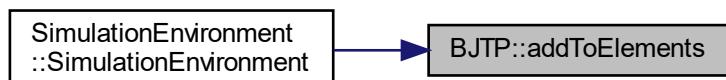
```
template<typename T >
static void BJTP< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 266 of file [BJT.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.2.4 updateStoredState()

```
template<typename T >
void BJTP< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 255 of file [BJT.hpp](#).

6.2.3 Member Data Documentation

6.2.3.1 alpha_f

```
template<typename T >
T BJTP< T >::alpha_f = 0.99
```

Definition at line 157 of file [BJT.hpp](#).

6.2.3.2 alpha_r

```
template<typename T >
T BJTP< T >::alpha_r = 0.02
```

Definition at line 158 of file [BJT.hpp](#).

6.2.3.3 b

```
template<typename T >
size_t BJTP< T >::b = 0
```

Definition at line 154 of file [BJT.hpp](#).

6.2.3.4 c

```
template<typename T >
size_t BJTP< T >::c = 0
```

Definition at line 153 of file [BJT.hpp](#).

6.2.3.5 e

```
template<typename T >
size_t BJTP< T >::e = 0
```

Definition at line 155 of file [BJT.hpp](#).

6.2.3.6 I_cs

```
template<typename T >
const T BJTP< T >::I_cs = 99e-14
```

Definition at line 162 of file [BJT.hpp](#).

6.2.3.7 I_es

```
template<typename T >
const T BJTP< T >::I_es = 2e-14
```

Definition at line 160 of file [BJT.hpp](#).

6.2.3.8 V_bc_crit

```
template<typename T >
T BJTP< T >::V_bc_crit = V_Tc * std::log(V_Tc / (I_cs * std::sqrt(2)))
```

Definition at line 165 of file [BJT.hpp](#).

6.2.3.9 V_be_crit

```
template<typename T >
T BJTP< T >::V_be_crit = V_Te * std::log(V_Te / (I_es * std::sqrt(2)))
```

Definition at line 166 of file [BJT.hpp](#).

6.2.3.10 V_Tc

```
template<typename T >
const T BJTP< T >::V\_Tc = 26e-3
```

Definition at line 163 of file [BJT.hpp](#).

6.2.3.11 V_Te

```
template<typename T >
const T BJTP< T >::V\_Te = 26e-3
```

Definition at line 161 of file [BJT.hpp](#).

The documentation for this struct was generated from the following file:

- [BJT.hpp](#)

6.3 **BJTResults< T >** Struct Template Reference

Public Attributes

- T [g_ee](#)
- T [g_ec](#)
- T [g_ce](#)
- T [g_cc](#)
- T [l_e](#)
- T [l_c](#)

6.3.1 Detailed Description

```
template<typename T>
struct BJTResults< T >
```

Definition at line 26 of file [AutoDiffTest.cpp](#).

6.3.2 Member Data Documentation

6.3.2.1 g_cc

```
template<typename T >
T BJTResults< T >::g\_cc
```

Definition at line 30 of file [AutoDiffTest.cpp](#).

6.3.2.2 g_ce

```
template<typename T >
T BJTResults< T >::g_ce
```

Definition at line [29](#) of file [AutoDiffTest.cpp](#).

6.3.2.3 g_ec

```
template<typename T >
T BJTResults< T >::g_ec
```

Definition at line [28](#) of file [AutoDiffTest.cpp](#).

6.3.2.4 g_ee

```
template<typename T >
T BJTResults< T >::g_ee
```

Definition at line [27](#) of file [AutoDiffTest.cpp](#).

6.3.2.5 I_c

```
template<typename T >
T BJTResults< T >::I_c
```

Definition at line [33](#) of file [AutoDiffTest.cpp](#).

6.3.2.6 I_e

```
template<typename T >
T BJTResults< T >::I_e
```

Definition at line [32](#) of file [AutoDiffTest.cpp](#).

The documentation for this struct was generated from the following file:

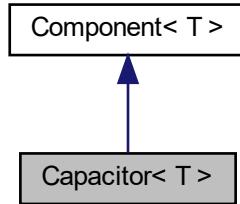
- [AutoDiffTest.cpp](#)

6.4 Capacitor< T > Struct Template Reference

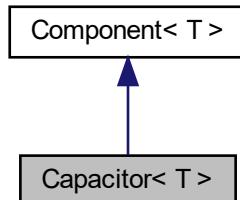
An ideal capacitor model.

```
#include <Capacitor.hpp>
```

Inheritance diagram for Capacitor< T >:



Collaboration diagram for Capacitor< T >:



Public Member Functions

- void [addDynamicStampTo \(Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep\) const](#)

Adds this component's dynamic stamp to the target stamp.
- void [updateStoredState \(const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A\)](#)

Updates any stored state based on the current solution index.
- void [addDCAnalysisStampTo \(Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents\) const](#)

adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements` (const std::string &line, `CircuitElements< T >` &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0
- T `lastCurrent` = 0
- bool `trapezoidalRule` = true

6.4.1 Detailed Description

```
template<typename T>
struct Capacitor< T >
```

An ideal capacitor model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 11 of file [Capacitor.hpp](#).

6.4.2 Member Function Documentation

6.4.2.1 addDCAnalysisStampTo()

```
template<typename T >
void Capacitor< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<code>destination</code>	The stamp to be added to.
<code>solutionMatrix</code>	A vector containing all past solutions to the circuit
<code>numCurrents</code>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 85 of file [Capacitor.hpp](#).

6.4.2.2 addDynamicStampTo()

```
template<typename T >
void Capacitor< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

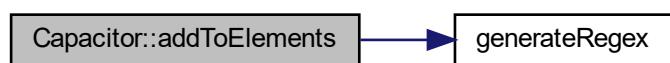
Definition at line 20 of file [Capacitor.hpp](#).

6.4.2.3 addToElements()

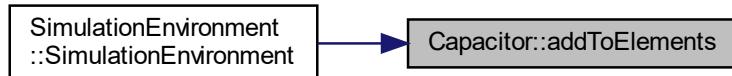
```
template<typename T >
static void Capacitor< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 97 of file [Capacitor.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.4.2.4 updateStoredState()

```

template<typename T >
void Capacitor< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
  
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component](#)< T >.

Definition at line 62 of file [Capacitor.hpp](#).

6.4.3 Member Data Documentation

6.4.3.1 lastCurrent

```

template<typename T >
T Capacitor< T >::lastCurrent = 0
  
```

Definition at line 17 of file [Capacitor.hpp](#).

6.4.3.2 n1

```
template<typename T >
size_t Capacitor< T >::n1 = 0
```

Definition at line 15 of file [Capacitor.hpp](#).

6.4.3.3 n2

```
template<typename T >
size_t Capacitor< T >::n2 = 0
```

Definition at line 16 of file [Capacitor.hpp](#).

6.4.3.4 trapezoidalRule

```
template<typename T >
bool Capacitor< T >::trapezoidalRule = true
```

Definition at line 19 of file [Capacitor.hpp](#).

6.4.3.5 value

```
template<typename T >
T Capacitor< T >::value = 0
```

Definition at line 13 of file [Capacitor.hpp](#).

The documentation for this struct was generated from the following file:

- [Capacitor.hpp](#)

6.5 ForceCausal::CausalData< T > Struct Template Reference

a helper struct to return the result of the forced causal IDFT

```
#include <ForceCausal.hpp>
```

Public Attributes

- T [tau](#)
- T [Ts](#)
- std::vector< T > [data](#)

6.5.1 Detailed Description

```
template<typename T>
struct ForceCausal::CausalData< T >
```

a helper struct to return the result of the forced causal IDFT

Template Parameters

<i>T</i>	
----------	--

Definition at line 68 of file [ForceCausal.hpp](#).

6.5.2 Member Data Documentation

6.5.2.1 data

```
template<typename T >
std::vector<T> ForceCausal::CausalData< T >::data
```

Definition at line 71 of file [ForceCausal.hpp](#).

6.5.2.2 tau

```
template<typename T >
T ForceCausal::CausalData< T >::tau
```

Definition at line 69 of file [ForceCausal.hpp](#).

6.5.2.3 Ts

```
template<typename T >
T ForceCausal::CausalData< T >::Ts
```

Definition at line 70 of file [ForceCausal.hpp](#).

The documentation for this struct was generated from the following file:

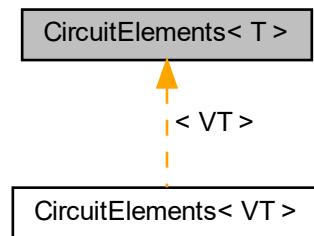
- [ForceCausal.hpp](#)

6.6 CircuitElements< T > Struct Template Reference

a glorified container for the different types of components.

```
#include <CircuitElements.hpp>
```

Inheritance diagram for CircuitElements< T >:



Public Member Functions

- **CircuitElements** (size_t numNodes=0, size_t numCurrents=0, size_t numDCCurrents=0)
Initialisation.
- void **setNewStampSize** (size_t numNodes, size_t numCurrents, size_t numDCCurrents=0)
Updates the size of all stamps.
- **Stamp< T > & generateStaticStamp ()**
Forces a clear of the static stamp, and generates a new one.
- **Stamp< T > & generateDynamicStamp** (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep)
Obtains the static stamp, then adds dynamic components to it.
- **Stamp< T > & generateNonLinearStamp** (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep)
Obtains the dynamic stamp, then adds dynamic components to it.
- **Stamp< T > & generateCompleteStamp** (SolutionStage stage, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep)
Generates the complete stamp up to a certain point.
- **Stamp< T > & generateDCStamp** (const Matrix< T > &solutionVector, size_t numCurrents)
Generates the complete DC stamp.
- void **updateTimeStep** (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep)
Updates the components at the end of each time step. Applies to dynamic and non-linear components.
- void **updateDCStoredState** (const Matrix< T > &solutionVector, size_t numCurrents)
Updates the components based on their DC value. Applies to dynamic and non-linear components.

Public Attributes

- `Stamp< T > staticStamp`
Preallocated stamp. Used for caching between loop iterations. Static stamps will only be generated once as a result.
- `Stamp< T > dynamicStamp`
Preallocated stamp. Used for caching between loop iterations. Dynamic stamps need to be updated on every timestep.
- `Stamp< T > nonLinearStamp`
Preallocated stamp. Used for caching between loop iterations. Non-Linear stamps must be updated on every newton-raphson iteration.
- `Stamp< T > dcStamp`
Preallocated stamp. Used for caching between loop iterations. DC stamps must be updated on every newton-raphson iteration.
- `std::vector< std::shared_ptr< Component< T > >> staticElements`
A container to store the static components.
- `std::vector< std::shared_ptr< Component< T > >> dynamicElements`
A container to store the dynamic components.
- `std::vector< std::shared_ptr< Component< T > >> nonLinearElements`
A container to store the Non-Linear components.
- `bool staticStampIsFresh = false`
A variable used to track if the cached stamp is current.
- `bool dynamicStampIsFresh = false`
A variable used to track if the cached stamp is current.
- `bool nonLinearStampIsFresh = false`
A variable used to track if the cached stamp is current.
- `std::multimap< size_t, std::shared_ptr< Component< T > >> nodeComponentMap`
A map to pair nodes with the components connected to them.

6.6.1 Detailed Description

```
template<typename T>
struct CircuitElements< T >
```

a glorified container for the different types of components.

Template Parameters

<code>T</code>	
----------------	--

Definition at line 46 of file [CircuitElements.hpp](#).

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CircuitElements()

```
template<typename T >
CircuitElements< T >::CircuitElements (
```

```
size_t numNodes = 0,
size_t numCurrents = 0,
size_t numDCCurrents = 0 ) [inline]
```

Initialisation.

Parameters

<i>numNodes</i>	The size of the stamps voltage dependants
<i>numCurrents</i>	The size of the stamps voltage dependants
<i>numDCCurrents</i>	The size of the stamps voltage dependants

Definition at line 88 of file [CircuitElements.hpp](#).

6.6.3 Member Function Documentation

6.6.3.1 generateCompleteStamp()

```
template<typename T >
Stamp<T>& CircuitElements< T >::generateCompleteStamp (
    SolutionStage stage,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) [inline]
```

Generates the complete stamp up to a certain point.

Parameters

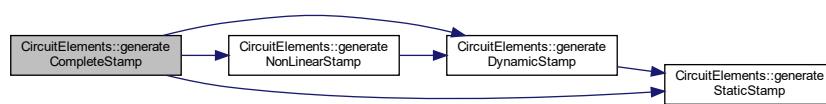
<i>solutionMatrix</i>	The solution matrix to use for the dynamic and non-linear stamp.
<i>currentSolutionIndex</i>	The current index we are at.
<i>timestep</i>	The time step being used.

Returns

The complete stamp.

Definition at line 196 of file [CircuitElements.hpp](#).

Here is the call graph for this function:



6.6.3.2 generateDCStamp()

```
template<typename T >
Stamp<T>& CircuitElements< T >::generateDCStamp (
    const Matrix< T > & solutionVector,
    size_t numCurrents ) [inline]
```

Generates the complete DC stamp.

Parameters

<i>solutionVector</i>	The solution vector to use for the dynamic and non-linear stamp.
<i>currentSolutionIndex</i>	The current index we are at.
<i>timestep</i>	The time step being used.

Returns

The complete stamp.

Definition at line 234 of file [CircuitElements.hpp](#).

Here is the caller graph for this function:



6.6.3.3 generateDynamicStamp()

```
template<typename T >
Stamp<T>& CircuitElements< T >::generateDynamicStamp (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) [inline]
```

Obtains the static stamp, then adds dynamic components to it.

Parameters

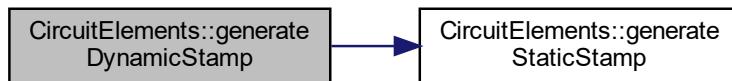
<i>solutionMatrix</i>	The solution matrix to use for the dynamic and non-linear stamp.
<i>currentSolutionIndex</i>	The current index we are at.
<i>timestep</i>	The time step being used.

Returns

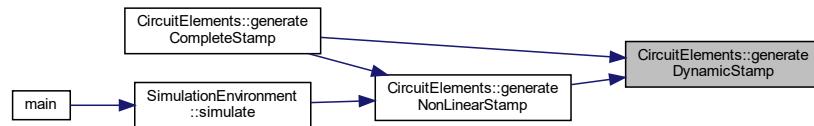
A reference to the cached stamp.

Definition at line 141 of file [CircuitElements.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.6.3.4 generateNonLinearStamp()**

```
template<typename T >
Stamp<T>& CircuitElements< T >::generateNonLinearStamp (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) [inline]
```

Obtains the dynamic stamp, then adds dynamic components to it.

Parameters

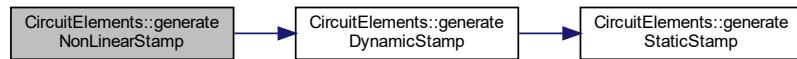
<i>solutionMatrix</i>	The solution matrix to use for the dynamic and non-linear stamp.
<i>currentSolutionIndex</i>	The current index we are at.
<i>timestep</i>	The time step being used.

Returns

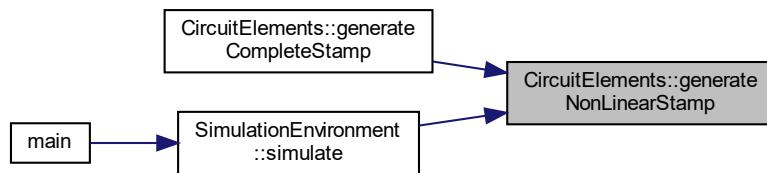
A reference to the cached stamp.

Definition at line 171 of file [CircuitElements.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3.5 generateStaticStamp()

```
template<typename T >
Stamp<T>& CircuitElements< T >::generateStaticStamp ( ) [inline]
```

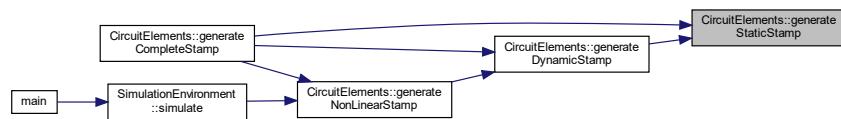
Forces a clear of the static stamp, and generates a new one.

Returns

A reference to the cached stamp.

Definition at line 114 of file [CircuitElements.hpp](#).

Here is the caller graph for this function:



6.6.3.6 setNewStampSize()

```
template<typename T >
void CircuitElements< T >::setNewStampSize (
    size_t numNodes,
    size_t numCurrents,
    size_t numDCCurrents = 0 ) [inline]
```

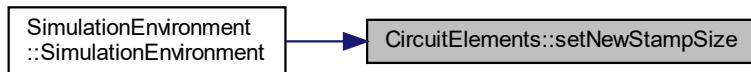
Updates the size of all stamps.

Parameters

<i>numNodes</i>	The size of the stamps voltage dependants
<i>numCurrents</i>	The size of the stamps voltage dependants
<i>numDCCurrents</i>	The size of the stamps voltage dependants

Definition at line 101 of file [CircuitElements.hpp](#).

Here is the caller graph for this function:



6.6.3.7 updateDCStoredState()

```
template<typename T >
void CircuitElements< T >::updateDCStoredState (
    const Matrix< T > & solutionVector,
    size_t numCurrents ) [inline]
```

Updates the components based on their DC value. Applies to dynamic and non-linear components.

Parameters

<i>solutionVector</i>	The solution vector to use for DC value
<i>numCurrents</i>	The number of currents used by the transient simulation.

Definition at line 278 of file [CircuitElements.hpp](#).

Here is the caller graph for this function:



6.6.3.8 updateTimeStep()

```

template<typename T >
void CircuitElements< T >::updateTimeStep (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) [inline]
  
```

Updates the components at the end of each time step. Applies to dynamic and non-linear components.

Parameters

<i>solutionMatrix</i>	The solution matrix to use for the dynamic and non-linear stamp.
<i>currentSolutionIndex</i>	The current index we are at.
<i>timestep</i>	The time step being used.

Definition at line 259 of file [CircuitElements.hpp](#).

Here is the caller graph for this function:



6.6.4 Member Data Documentation

6.6.4.1 dcStamp

```

template<typename T >
Stamp<T> CircuitElements< T >::dcStamp
  
```

Preallocated stamp. Used for caching between loop iterations. DC stamps must be updated on every newton-raphson iteration.

Definition at line 60 of file [CircuitElements.hpp](#).

6.6.4.2 dynamicElements

```
template<typename T >
std::vector<std::shared_ptr<Component<T> > > CircuitElements< T >::dynamicElements
```

A container to store the dynamic components.

Definition at line 69 of file [CircuitElements.hpp](#).

6.6.4.3 dynamicStamp

```
template<typename T >
Stamp<T> CircuitElements< T >::dynamicStamp
```

Preallocated stamp. Used for caching between loop iterations. Dynamic stamps need to be updated on every timestep.

Definition at line 52 of file [CircuitElements.hpp](#).

6.6.4.4 dynamicStampIsFresh

```
template<typename T >
bool CircuitElements< T >::dynamicStampIsFresh = false
```

A variable used to track if the cached stamp is current.

Definition at line 76 of file [CircuitElements.hpp](#).

6.6.4.5 nodeComponentMap

```
template<typename T >
std::multimap<size_t, std::shared_ptr<Component<T> > > CircuitElements< T >::nodeComponentMap
```

A map to pair nodes with the components connected to them.

Definition at line 81 of file [CircuitElements.hpp](#).

6.6.4.6 nonLinearElements

```
template<typename T >
std::vector<std::shared_ptr<Component<T> > > CircuitElements< T >::nonLinearElements
```

A container to store the Non-Linear components.

Definition at line 71 of file [CircuitElements.hpp](#).

6.6.4.7 nonLinearStamp

```
template<typename T >
Stamp<T> CircuitElements< T >::nonLinearStamp
```

Preallocated stamp. Used for caching between loop iterations. Non-Linear stamps must be updated on every newton-raphson iteration.

Definition at line 56 of file [CircuitElements.hpp](#).

6.6.4.8 nonLinearStampIsFresh

```
template<typename T >
bool CircuitElements< T >::nonLinearStampIsFresh = false
```

A variable used to track if the cached stamp is current.

Definition at line 78 of file [CircuitElements.hpp](#).

6.6.4.9 staticElements

```
template<typename T >
std::vector<std::shared_ptr<Component<T> > > CircuitElements< T >::staticElements
```

A container to store the static components.

Definition at line 67 of file [CircuitElements.hpp](#).

6.6.4.10 staticStamp

```
template<typename T >
Stamp<T> CircuitElements< T >::staticStamp
```

Preallocated stamp. Used for caching between loop iterations. Static stamps will only be generated once as a result.

Definition at line 49 of file [CircuitElements.hpp](#).

6.6.4.11 staticStampIsFresh

```
template<typename T >
bool CircuitElements< T >::staticStampIsFresh = false
```

A variable used to track if the cached stamp is current.

Definition at line 74 of file [CircuitElements.hpp](#).

The documentation for this struct was generated from the following file:

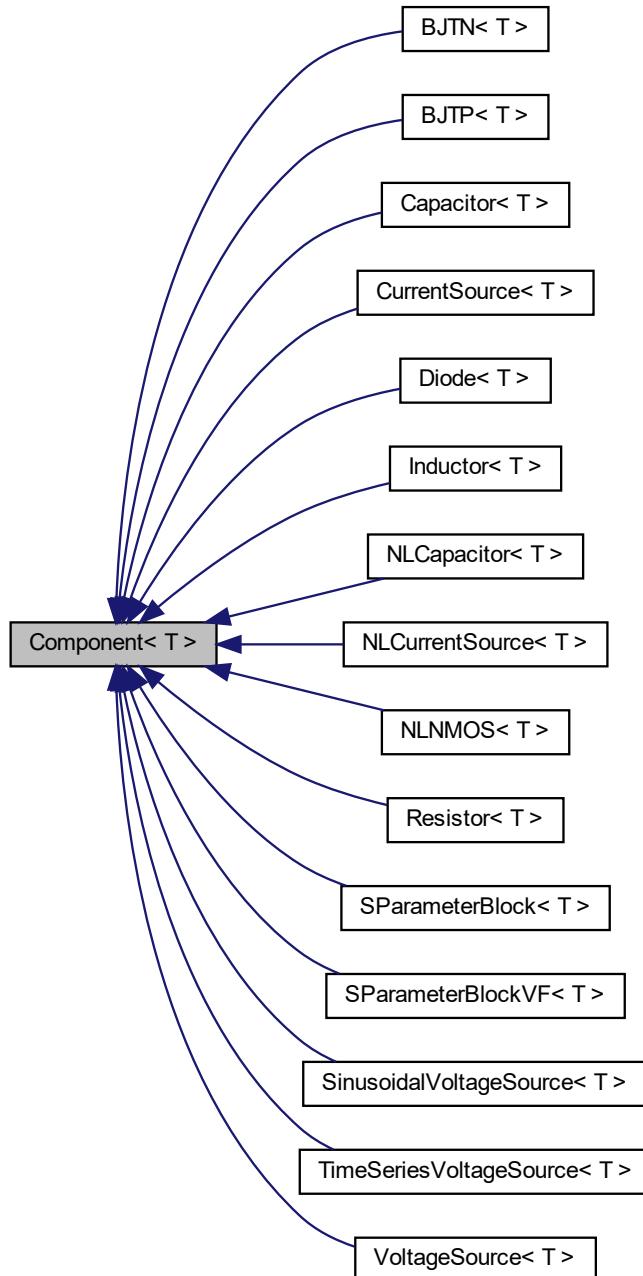
- [CircuitElements.hpp](#)

6.7 Component< T > Struct Template Reference

A template base class to define the fundamental things a component should define.

```
#include <Component.hpp>
```

Inheritance diagram for Component< T >:



Public Member Functions

- virtual void `addStaticStampTo (Stamp< T > &destination) const`
Adds this component's static stamp to the target stamp.
- virtual void `addDynamicStampTo (Stamp< T > &destination, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep) const`
Adds this component's dynamic stamp to the target stamp.

- virtual void `addNonLinearStampTo` (`Stamp< T >` &destination, const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep=0) const
adds this component's non-linear stamp to the target stamp.
- virtual void `updateStoredState` (const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep, `size_t` numCurrents)
Updates any stored state based on the current solution index.
- virtual void `addDCAAnalysisStampTo` (`Stamp< T >` &destination, const `Matrix< T >` &solutionVector, `size_t` numCurrents) const
adds this component's DC stamp to the target stamp.
- virtual void `updateDCStoredState` (const `Matrix< T >` &solutionVector, `size_t` sizeG_A, `size_t` numCurrents)
a function to update the stored state of a component based on a DC value
- virtual void `setTimestep` (`T` timestep)
initialises the component
- virtual `~Component` ()

Static Public Member Functions

- static void `addToElements` (const std::string &line, `CircuitElements< T >` &elements, `size_t` &numNodes, `size_t` &numCurrents, `size_t` &numDCCurrents)
Called as a helper to add the component to the elements class.

Public Attributes

- std::string `designator` = ""
The designator as in the netlist for e.g.

6.7.1 Detailed Description

```
template<typename T>
struct Component< T >
```

A template base class to define the fundamental things a component should define.

Template Parameters

<code>T</code>	The value type
----------------	----------------

Definition at line 108 of file `Component.hpp`.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 `~Component()`

```
template<typename T >
virtual Component< T >::~Component ( ) [inline], [virtual]
```

Definition at line 194 of file `Component.hpp`.

6.7.3 Member Function Documentation

6.7.3.1 addDCAnalysisStampTo()

```
template<typename T >
virtual void Component< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented in [VoltageSource< T >](#), [TimeSeriesVoltageSource< T >](#), [SParameterBlockVF< T >](#), [SParameterBlock< T >](#), [SinusoidalVoltageSource< T >](#), [Resistor< T >](#), [NLCurrentSource< T >](#), [NLCapacitor< T >](#), [Inductor< T >](#), [Diode< T >](#), [CurrentSource< T >](#), [Capacitor< T >](#), [BJTP< T >](#), and [BJTN< T >](#).

Definition at line 159 of file [Component.hpp](#).

6.7.3.2 addDynamicStampTo()

```
template<typename T >
virtual void Component< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented in [TimeSeriesVoltageSource< T >](#), [SinusoidalVoltageSource< T >](#), [Inductor< T >](#), [Capacitor< T >](#), [SParameterBlockVF< T >](#), and [SParameterBlock< T >](#).

Definition at line 125 of file [Component.hpp](#).

6.7.3.3 addNonLinearStampTo()

```
template<typename T >
virtual void Component< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented in [NLNMOS< T >](#), [NLCurrentSource< T >](#), [NLCapacitor< T >](#), [Diode< T >](#), [BJTP< T >](#), and [BJTN< T >](#).

Definition at line 136 of file [Component.hpp](#).

6.7.3.4 addStaticStampTo()

```
template<typename T >
virtual void Component< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented in [VoltageSource< T >](#), [SPparameterBlockVF< T >](#), [SPparameterBlock< T >](#), [Resistor< T >](#), and [CurrentSource< T >](#).

Definition at line 115 of file [Component.hpp](#).

6.7.3.5 addToElements()

```
template<typename T >
static void Component< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
```

```
size_t & numNodes,
size_t & numCurrents,
size_t & numDCCurrents ) [inline], [static]
```

Called as a helper to add the component to the elements class.

Parameters

<i>line</i>	The line to be parsed.
<i>elements</i>	A reference to the elements object the component is being added to.
<i>numNodes</i>	A reference to the current max number of nodes.
<i>numCurrents</i>	A reference to the current max number of currents.
<i>numDCCurrents</i>	A reference to the current max number of DC currents.

Definition at line 189 of file [Component.hpp](#).

6.7.3.6 setTimestep()

```
template<typename T >
virtual void Component< T >::setTimestep (
    T timestep ) [inline], [virtual]
```

initialises the component

Parameters

<i>timestep</i>	The length of each time step
-----------------	------------------------------

Reimplemented in [SParameterBlockVF< T >](#).

Definition at line 177 of file [Component.hpp](#).

6.7.3.7 updateDCStoredState()

```
template<typename T >
virtual void Component< T >::updateDCStoredState (
    const Matrix< T > & solutionVector,
    size_t sizeG_A,
    size_t numCurrents ) [inline], [virtual]
```

a function to update the stored state of a component based on a DC value

Parameters

<i>solutionVector</i>	The DC vector
<i>sizeG_A</i>	the number of voltages
<i>numCurrents</i>	the number of transient currents

Reimplemented in [NLCapacitor< T >](#), and [Inductor< T >](#).

Definition at line 170 of file [Component.hpp](#).

6.7.3.8 updateStoredState()

```
template<typename T >
virtual void Component< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented in [TimeSeriesVoltageSource< T >](#), [SParameterBlockVF< T >](#), [SParameterBlock< T >](#), [SinusoidalVoltageSource< T >](#), [NLNMOS< T >](#), [NLCapacitor< T >](#), [Inductor< T >](#), [Diode< T >](#), [Capacitor< T >](#), [BJTP< T >](#), and [BJTN< T >](#).

Definition at line 148 of file [Component.hpp](#).

6.7.4 Member Data Documentation

6.7.4.1 designator

```
template<typename T >
std::string Component< T >::designator = ""
```

The designator as in the netlist for e.g.

Definition at line 110 of file [Component.hpp](#).

The documentation for this struct was generated from the following file:

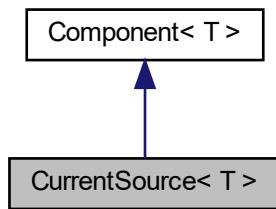
- [Component.hpp](#)

6.8 CurrentSource< T > Struct Template Reference

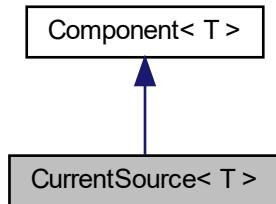
An ideal current source.

```
#include <CurrentSource.hpp>
```

Inheritance diagram for CurrentSource< T >:



Collaboration diagram for CurrentSource< T >:



Public Member Functions

- void `addStaticStampTo (Stamp< T > &stamp) const`
Adds this component's static stamp to the target stamp.
- void `addDCAnalysisStampTo (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const`
adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0

6.8.1 Detailed Description

```
template<typename T>
struct CurrentSource< T >
```

An ideal current source.

Template Parameters

<code>T</code>	The value type
----------------	----------------

Definition at line 10 of file [CurrentSource.hpp](#).

6.8.2 Member Function Documentation

6.8.2.1 addDCAnalysisStampTo()

```
template<typename T >
void CurrentSource< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

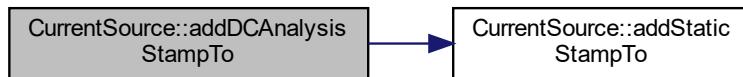
Parameters

<code>destination</code>	The stamp to be added to.
<code>solutionMatrix</code>	A vector containing all past solutions to the circuit
<code>numCurrents</code>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 31 of file [CurrentSource.hpp](#).

Here is the call graph for this function:



6.8.2.2 addStaticStampTo()

```
template<typename T >
void CurrentSource< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented from [Component< T >](#).

Definition at line 17 of file [CurrentSource.hpp](#).

Here is the caller graph for this function:



6.8.2.3 addToElements()

```
template<typename T >
static void CurrentSource< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
```

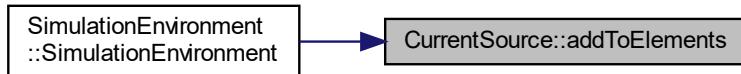
```
size_t & numCurrents,  
size_t & numDCCurrents ) [inline], [static]
```

Definition at line 37 of file [CurrentSource.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.3 Member Data Documentation

6.8.3.1 n1

```
template<typename T >  
size_t CurrentSource< T >::n1 = 0
```

Definition at line 14 of file [CurrentSource.hpp](#).

6.8.3.2 n2

```
template<typename T >  
size_t CurrentSource< T >::n2 = 0
```

Definition at line 15 of file [CurrentSource.hpp](#).

6.8.3.3 value

```
template<typename T >
T CurrentSource< T >::value = 0
```

Definition at line 12 of file [CurrentSource.hpp](#).

The documentation for this struct was generated from the following file:

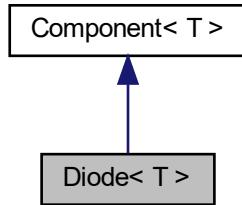
- [CurrentSource.hpp](#)

6.9 Diode< T > Struct Template Reference

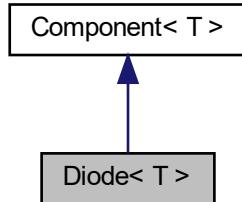
An ebbers moll diode model.

```
#include <Diode.hpp>
```

Inheritance diagram for Diode< T >:



Collaboration diagram for Diode< T >:



Public Member Functions

- void `addNonLinearStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep=0) const

$$\text{adds this component's non-linear stamp to the target stamp.}$$`
- void `updateStoredState` (const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep, `size_t` sizeG_A)

$$\text{Updates any stored state based on the current solution index.}$$
- void `addDCAnalysisStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const

$$\text{adds this component's DC stamp to the target stamp.}$$`

Static Public Member Functions

- static void `addToElements` (const `std::string` &line, `CircuitElements< T >` &elements, `size_t` &numNodes, `size_t` &numCurrents, `size_t` &numDCCurrents)

Public Attributes

- `size_t n1 = 0`
- `size_t n2 = 0`
- `const T I_sat = 2.52e-9`
- `const T V_T = 25.8563e-3`
- `const T eta = 2`
- `T V_crit = eta * V_T * std::log(eta * V_T / (I_sat * std::sqrt(2)))`

6.9.1 Detailed Description

```
template<typename T>
struct Diode< T >
```

An ebbers moll diode model.

Template Parameters

<code>T</code>	Value type
----------------	------------

Definition at line 11 of file [Diode.hpp](#).

6.9.2 Member Function Documentation

6.9.2.1 addDCAnalysisStampTo()

```
template<typename T >
void Diode< T >::addDCAnalysisStampTo (
```

```
Stamp< T > & destination,
const Matrix< T > & solutionVector,
size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 65 of file [Diode.hpp](#).

Here is the call graph for this function:



6.9.2.2 addNonLinearStampTo()

```
template<typename T >
void Diode< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 24 of file [Diode.hpp](#).

Here is the caller graph for this function:



6.9.2.3 addToElements()

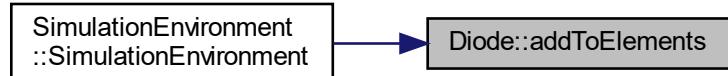
```
template<typename T >
static void Diode< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 71 of file [Diode.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.2.4 updateStoredState()

```
template<typename T >
void Diode< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 60 of file [Diode.hpp](#).

6.9.3 Member Data Documentation

6.9.3.1 eta

```
template<typename T >
const T Diode< T >::eta = 2
```

Definition at line 18 of file [Diode.hpp](#).

6.9.3.2 I_sat

```
template<typename T >
const T Diode< T >::I_sat = 2.52e-9
```

Definition at line 16 of file [Diode.hpp](#).

6.9.3.3 n1

```
template<typename T >
size_t Diode< T >::n1 = 0
```

Definition at line 13 of file [Diode.hpp](#).

6.9.3.4 n2

```
template<typename T >
size_t Diode< T >::n2 = 0
```

Definition at line 14 of file [Diode.hpp](#).

6.9.3.5 V_crit

```
template<typename T >
T Diode< T >::V_crit = eta * V_T * std::log(eta * V_T / (I_sat * std::sqrt(2)))
```

Definition at line 20 of file [Diode.hpp](#).

6.9.3.6 V_T

```
template<typename T >
const T Diode< T >::V_T = 25.8563e-3
```

Definition at line 17 of file [Diode.hpp](#).

The documentation for this struct was generated from the following file:

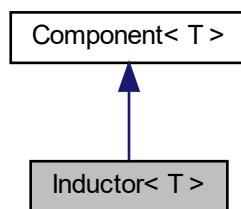
- [Diode.hpp](#)

6.10 Inductor< T > Struct Template Reference

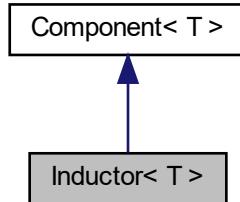
An ideal inductor model.

```
#include <Inductor.hpp>
```

Inheritance diagram for Inductor< T >:



Collaboration diagram for `Inductor< T >`:



Public Member Functions

- void `addDynamicStampTo (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep) const`
Adds this component's dynamic stamp to the target stamp.
- void `updateStoredState (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A)`
Updates any stored state based on the current solution index.
- void `updateDCStoredState (const Matrix< T > &solutionVector, size_t sizeG_A, size_t numCurrents)`
a function to update the stored state of a component based on a DC value
- void `addDCAssessmentStampTo (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const`
adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0
- T `lastCurrent` = 0
- size_t `dcCurrentIndex` = 0
- bool `trapezoidalRule` = true

6.10.1 Detailed Description

```
template<typename T>
struct Inductor< T >
```

An ideal inductor model.

Template Parameters

<i>T</i>	the value type
----------	----------------

Definition at line 11 of file [Inductor.hpp](#).

6.10.2 Member Function Documentation

6.10.2.1 addDCAnalysisStampTo()

```
template<typename T >
void Inductor< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 95 of file [Inductor.hpp](#).

6.10.2.2 addDynamicStampTo()

```
template<typename T >
void Inductor< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 22 of file [Inductor.hpp](#).

6.10.2.3 addToElements()

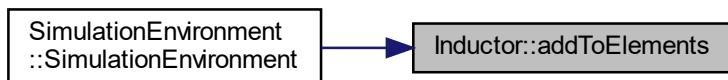
```
template<typename T >
static void Inductor< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 114 of file [Inductor.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.2.4 updateDCStoredState()

```
template<typename T >
void Inductor< T >::updateDCStoredState (
    const Matrix< T > & solutionVector,
    size_t sizeG_A,
    size_t numCurrents ) [inline], [virtual]
```

a function to update the stored state of a component based on a DC value

Parameters

<i>solutionVector</i>	The DC vector
<i>sizeG_A</i>	the number of voltages
<i>numCurrents</i>	the number of transient currents

Reimplemented from [Component< T >](#).

Definition at line 90 of file [Inductor.hpp](#).

6.10.2.5 updateStoredState()

```
template<typename T >
void Inductor< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 64 of file [Inductor.hpp](#).

6.10.3 Member Data Documentation**6.10.3.1 dcCurrentIndex**

```
template<typename T >
size_t Inductor< T >::dcCurrentIndex = 0
```

Definition at line 19 of file [Inductor.hpp](#).

6.10.3.2 lastCurrent

```
template<typename T >
T Inductor< T >::lastCurrent = 0
```

Definition at line 17 of file [Inductor.hpp](#).

6.10.3.3 n1

```
template<typename T >
size_t Inductor< T >::n1 = 0
```

Definition at line 15 of file [Inductor.hpp](#).

6.10.3.4 n2

```
template<typename T >
size_t Inductor< T >::n2 = 0
```

Definition at line 16 of file [Inductor.hpp](#).

6.10.3.5 trapezoidalRule

```
template<typename T >
bool Inductor< T >::trapezoidalRule = true
```

Definition at line 21 of file [Inductor.hpp](#).

6.10.3.6 value

```
template<typename T >
T Inductor< T >::value = 0
```

Definition at line 13 of file [Inductor.hpp](#).

The documentation for this struct was generated from the following file:

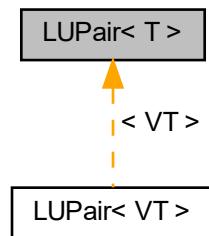
- [Inductor.hpp](#)

6.11 LUPair< T > Struct Template Reference

A helper class to store the L U and pivot matrices. Mainly useful in solving systems of equations.

```
#include <DynamicMatrix.hpp>
```

Inheritance diagram for LUPair< T >:



Public Member Functions

- [LUPair \(size_t _M\)](#)
- [LUPair \(const LUPair< T > &other\)](#)
- [std::string toString \(\)](#)

Public Attributes

- [Matrix< T > l](#)
- [Matrix< T > u](#)
- [std::vector< size_t > p](#)
- [size_t M](#)

6.11.1 Detailed Description

```
template<typename T>
struct LUPair< T >
```

A helper class to store the L U and pivot matrices. Mainly useful in solving systems of equations.

Template Parameters

<i>T</i>	the value type
----------	----------------

Definition at line [273](#) of file [DynamicMatrix.hpp](#).

6.11.2 Constructor & Destructor Documentation

6.11.2.1 LUPair() [1/2]

```
template<typename T >
LUPair< T >::LUPair (
    size_t _M )  [inline]
```

Definition at line 279 of file [DynamicMatrix.hpp](#).

6.11.2.2 LUPair() [2/2]

```
template<typename T >
LUPair< T >::LUPair (
    const LUPair< T > & other )  [inline]
```

Definition at line 282 of file [DynamicMatrix.hpp](#).

6.11.3 Member Function Documentation

6.11.3.1 toString()

```
template<typename T >
std::string LUPair< T >::toString ( )  [inline]
```

Definition at line 287 of file [DynamicMatrix.hpp](#).

6.11.4 Member Data Documentation

6.11.4.1 l

```
template<typename T >
Matrix<T> LUPair< T >::l
```

Definition at line 274 of file [DynamicMatrix.hpp](#).

6.11.4.2 M

```
template<typename T >  
size_t LUPair< T >::M
```

Definition at line 277 of file [DynamicMatrix.hpp](#).

6.11.4.3 p

```
template<typename T >  
std::vector<size_t> LUPair< T >::p
```

Definition at line 276 of file [DynamicMatrix.hpp](#).

6.11.4.4 u

```
template<typename T >
Matrix<T> LUPair< T >::u
```

Definition at line 275 of file [DynamicMatrix.hpp](#).

The documentation for this struct was generated from the following file:

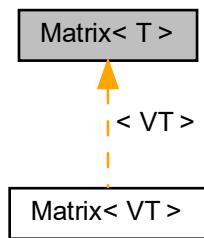
- DynamicMatrix.hpp

6.12 Matrix< T > Struct Template Reference

A matrix class with support for LU-decomposition, and left division.

```
#include <DynamicMatrix.hpp>
```

Inheritance diagram for Matrix< T >:



Public Member Functions

- `Matrix (size_t M, size_t N)`
- `Matrix (size_t M, size_t N, T initialValue)`
- `T & operator() (size_t m, size_t n)`
- `const T & operator() (size_t m, size_t n) const`
- `void fill (T fillVal)`
- `void rowAddition (size_t destinationRow, size_t sourceRow, T scalingFactor)`
- `void swapRows (size_t row1, size_t row2)`
- `Matrix< T > transpose ()`
- `Matrix< T > multiply (const Matrix< T > &rhs) const`
- `void multiply (const Matrix< T > &rhs, Matrix< T > &dest) const`
- `Matrix< T > add (const Matrix< T > &rhs) const`
- `void add (const Matrix< T > &rhs, Matrix< T > &dest) const`
- `Matrix< T > subtract (const Matrix< T > &rhs) const`
- `void subtract (const Matrix< T > &rhs, Matrix< T > &dest) const`
- `std::string toString () const`
- `LUPair< T > luPair () const`
- `void luPair (LUPair< T > &dest) const`
- `Matrix< T > leftDivide (const Matrix< T > &rhs) const`
- `void leftDivide (const Matrix< T > &rhs, const LUPair< T > &lu, Matrix< T > &scratchSpace, Matrix< T > &dest) const`
- template<typename Iterator>
`void leftDivide (const Matrix< T > &rhs, const LUPair< T > &lu, Matrix< T > &scratchSpace, Iterator destBegin, Iterator destEnd) const`

Public Attributes

- `std::vector< T > data`
- `size_t M`
- `size_t N`

6.12.1 Detailed Description

```
template<typename T>
struct Matrix< T >
```

A matrix class with support for LU-decomposition, and left division.

Template Parameters

<code>T</code>	
----------------	--

Definition at line 55 of file [DynamicMatrix.hpp](#).

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Matrix() [1/2]

```
template<typename T >
Matrix< T >::Matrix (
    size_t M,
    size_t N )  [inline]
```

Definition at line 60 of file [DynamicMatrix.hpp](#).

6.12.2.2 Matrix() [2/2]

```
template<typename T >
Matrix< T >::Matrix (
    size_t M,
    size_t N,
    T initialValue )  [inline]
```

Definition at line 64 of file [DynamicMatrix.hpp](#).

6.12.3 Member Function Documentation

6.12.3.1 add() [1/2]

```
template<typename T >
Matrix<T> Matrix< T >::add (
    const Matrix< T > & rhs ) const  [inline]
```

Definition at line 133 of file [DynamicMatrix.hpp](#).

Here is the caller graph for this function:



6.12.3.2 add() [2/2]

```
template<typename T >
void Matrix< T >::add (
    const Matrix< T > & rhs,
    Matrix< T > & dest ) const [inline]
```

Definition at line 140 of file [DynamicMatrix.hpp](#).

6.12.3.3 fill()

```
template<typename T >
void Matrix< T >::fill (
    T fillVal ) [inline]
```

Definition at line 76 of file [DynamicMatrix.hpp](#).

6.12.3.4 leftDivide() [1/3]

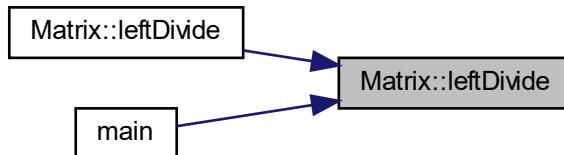
```
template<typename T >
Matrix<T> Matrix< T >::leftDivide (
    const Matrix< T > & rhs ) const [inline]
```

Definition at line 225 of file [DynamicMatrix.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.3.5 leftDivide() [2/3]

```
template<typename T >
template<typename Iterator >
void Matrix< T >::leftDivide (
    const Matrix< T > & rhs,
    const LUPair< T > & lu,
    Matrix< T > & scratchSpace,
    Iterator destBegin,
    Iterator destEnd ) const [inline]
```

Definition at line 240 of file [DynamicMatrix.hpp](#).

6.12.3.6 leftDivide() [3/3]

```
template<typename T >
void Matrix< T >::leftDivide (
    const Matrix< T > & rhs,
    const LUPair< T > & lu,
    Matrix< T > & scratchSpace,
    Matrix< T > & dest ) const [inline]
```

Definition at line 233 of file [DynamicMatrix.hpp](#).

Here is the call graph for this function:

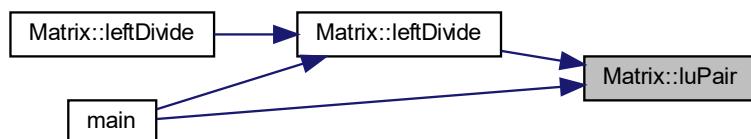


6.12.3.7 luPair() [1/2]

```
template<typename T >
LUPair<T> Matrix< T >::luPair ( ) const [inline]
```

Definition at line 177 of file [DynamicMatrix.hpp](#).

Here is the caller graph for this function:



6.12.3.8 luPair() [2/2]

```
template<typename T >
void Matrix< T >::luPair (
    LUPair< T > & dest ) const [inline]
```

Definition at line 185 of file [DynamicMatrix.hpp](#).

6.12.3.9 multiply() [1/2]

```
template<typename T >
Matrix<T> Matrix< T >::multiply (
    const Matrix< T > & rhs ) const [inline]
```

Definition at line 105 of file [DynamicMatrix.hpp](#).

6.12.3.10 multiply() [2/2]

```
template<typename T >
void Matrix< T >::multiply (
    const Matrix< T > & rhs,
    Matrix< T > & dest ) const [inline]
```

Definition at line 112 of file [DynamicMatrix.hpp](#).

6.12.3.11 operator()() [1/2]

```
template<typename T >
T& Matrix< T >::operator() ( 
    size_t m,
    size_t n ) [inline]
```

Definition at line 68 of file [DynamicMatrix.hpp](#).

6.12.3.12 operator()() [2/2]

```
template<typename T >
const T& Matrix< T >::operator() ( 
    size_t m,
    size_t n ) const [inline]
```

Definition at line 72 of file [DynamicMatrix.hpp](#).

6.12.3.13 rowAddition()

```
template<typename T >
void Matrix< T >::rowAddition (
    size_t destinationRow,
    size_t sourceRow,
    T scalingFactor ) [inline]
```

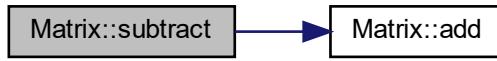
Definition at line 82 of file [DynamicMatrix.hpp](#).

6.12.3.14 subtract() [1/2]

```
template<typename T >
Matrix<T> Matrix< T >::subtract (
    const Matrix< T > & rhs ) const [inline]
```

Definition at line 148 of file [DynamicMatrix.hpp](#).

Here is the call graph for this function:



6.12.3.15 subtract() [2/2]

```
template<typename T >
void Matrix< T >::subtract (
    const Matrix< T > & rhs,
    Matrix< T > & dest ) const [inline]
```

Definition at line 155 of file [DynamicMatrix.hpp](#).

6.12.3.16 swapRows()

```
template<typename T >
void Matrix< T >::swapRows (
    size_t row1,
    size_t row2 ) [inline]
```

Definition at line 90 of file [DynamicMatrix.hpp](#).

6.12.3.17 `toString()`

```
template<typename T >
std::string Matrix< T >::toString ( ) const [inline]
```

Definition at line 163 of file [DynamicMatrix.hpp](#).

Here is the caller graph for this function:



6.12.3.18 `transpose()`

```
template<typename T >
Matrix<T> Matrix< T >::transpose ( ) [inline]
```

Definition at line 95 of file [DynamicMatrix.hpp](#).

6.12.4 Member Data Documentation

6.12.4.1 `data`

```
template<typename T >
std::vector<T> Matrix< T >::data
```

Definition at line 56 of file [DynamicMatrix.hpp](#).

6.12.4.2 `M`

```
template<typename T >
size_t Matrix< T >::M
```

Definition at line 57 of file [DynamicMatrix.hpp](#).

6.12.4.3 N

```
template<typename T>
size_t Matrix< T >::N
```

Definition at line 58 of file [DynamicMatrix.hpp](#).

The documentation for this struct was generated from the following file:

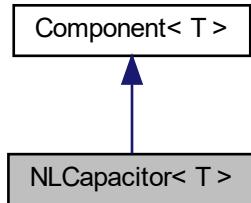
- [DynamicMatrix.hpp](#)

6.13 NLCapacitor< T > Struct Template Reference

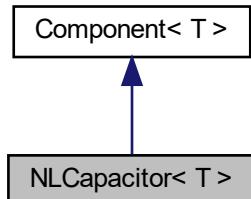
a non-linear capacitor model of the form $C = C_p + C_o * (1.0 + \tanh(P_{10} + P_{11} * u))$

```
#include <NLCapacitor.hpp>
```

Inheritance diagram for NLCapacitor< T >:



Collaboration diagram for NLCapacitor< T >:



Public Member Functions

- void `addNonLinearStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep=0) const
adds this component's non-linear stamp to the target stamp.`
- void `updateStoredState` (const `Matrix< T >` &solutionMatrix, const `size_t` currentSolutionIndex, `T` timestep, `size_t` sizeG_A)
Updates any stored state based on the current solution index.
- void `updateDCStoredState` (const `Matrix< T >` &solutionVector, `size_t` sizeG_A, `size_t` numCurrents)
a function to update the stored state of a component based on a DC value
- void `addDCAcknowledgmentStampTo` (`Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const
adds this component's DC stamp to the target stamp.`

Static Public Member Functions

- static void `addToElements` (const std::string &line, `CircuitElements< T >` &elements, `size_t` &numNodes, `size_t` &numCurrents, `size_t` &numDCCurrents)

Public Attributes

- `size_t n1 = 0`
- `size_t n2 = 0`
- `T C_p = 0`
- `T C_o = 0`
- `T P_10 = 0`
- `T P_11 = 0`
- `T u_last = 0`
- `T i_last = 0`
- `T C_last = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u_last))`

6.13.1 Detailed Description

```
template<typename T>
struct NLCapacitor< T >
```

a non-linear capacitor model of the form $C = C_p + C_o * (1.0 + \tanh(P_{10} + P_{11} * u))$

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 11 of file [NLCapacitor.hpp](#).

6.13.2 Member Function Documentation

6.13.2.1 addDCAnalysisStampTo()

```
template<typename T >
void NLCapacitor< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 123 of file [NLCapacitor.hpp](#).

6.13.2.2 addNonLinearStampTo()

```
template<typename T >
void NLCapacitor< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 27 of file [NLCapacitor.hpp](#).

6.13.2.3 addToElements()

```
template<typename T >
static void NLCapacitor< T >::addToElements (
```

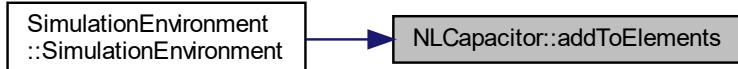
```
const std::string & line,
CircuitElements< T > & elements,
size_t & numNodes,
size_t & numCurrents,
size_t & numDCCurrents ) [inline], [static]
```

Definition at line 135 of file [NLCapacitor.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.2.4 updateDCStoredState()

```
template<typename T >
void NLCapacitor< T >::updateDCStoredState (
    const Matrix< T > & solutionVector,
    size_t sizeG_A,
    size_t numCurrents ) [inline], [virtual]
```

a function to update the stored state of a component based on a DC value

Parameters

<i>solutionVector</i>	The DC vector
<i>sizeG_A</i>	the number of voltages
<i>numCurrents</i>	the number of transient currents

Reimplemented from [Component< T >](#).

Definition at line 99 of file [NLCapacitor.hpp](#).

6.13.2.5 updateStoredState()

```
template<typename T >
void NLCapacitor< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 74 of file [NLCapacitor.hpp](#).

6.13.3 Member Data Documentation

6.13.3.1 C_last

```
template<typename T >
T NLCapacitor< T >::C_last = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u_last))
```

Definition at line 24 of file [NLCapacitor.hpp](#).

6.13.3.2 C_o

```
template<typename T >
T NLCapacitor< T >::C_o = 0
```

Definition at line 17 of file [NLCapacitor.hpp](#).

6.13.3.3 C_p

```
template<typename T >
T NLCapacitor< T >::C_p = 0
```

Definition at line 16 of file [NLCapacitor.hpp](#).

6.13.3.4 i_last

```
template<typename T >
size_t NLCapacitor< T >::i_last = 0
```

Definition at line 22 of file [NLCapacitor.hpp](#).

6.13.3.5 n1

```
template<typename T >
size_t NLCapacitor< T >::n1 = 0
```

Definition at line 13 of file [NLCapacitor.hpp](#).

6.13.3.6 n2

```
template<typename T >
size_t NLCapacitor< T >::n2 = 0
```

Definition at line 14 of file [NLCapacitor.hpp](#).

6.13.3.7 P_10

```
template<typename T >
T NLCapacitor< T >::P_10 = 0
```

Definition at line 18 of file [NLCapacitor.hpp](#).

6.13.3.8 P_11

```
template<typename T >
T NLCapacitor< T >::P_11 = 0
```

Definition at line 19 of file [NLCapacitor.hpp](#).

6.13.3.9 u_last

```
template<typename T>
T NLCapacitor< T >::u_last = 0
```

Definition at line 21 of file [NLCapacitor.hpp](#).

The documentation for this struct was generated from the following file:

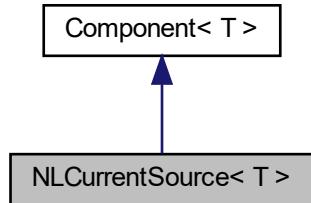
- [NLCapacitor.hpp](#)

6.14 NLCURRENTSOURCE< T > Struct Template Reference

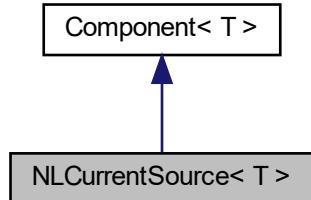
An non-linear current source for the COBRA transistor model.

```
#include <NLCurrentSource.hpp>
```

Inheritance diagram for NLCurrentSource< T >:



Collaboration diagram for NLCurrentSource< T >:



Public Member Functions

- void `addNonLinearStampTo (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t current←SolutionIndex, T timestep=0) const`
adds this component's non-linear stamp to the target stamp.
- void `addDCAnalysisStampTo (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const`
adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0
- size_t `r1_pos` = 0
- size_t `r1_neg` = 0
- size_t `r2_pos` = 0
- size_t `r2_neg` = 0

6.14.1 Detailed Description

```
template<typename T>
struct NLCurrentSource< T >
```

An non-linear current source for the COBRA transistor model.

Template Parameters

<code>T</code>	The value type
----------------	----------------

Definition at line 11 of file [NLCurrentSource.hpp](#).

6.14.2 Member Function Documentation

6.14.2.1 addDCAnalysisStampTo()

```
template<typename T >
void NLCurrentSource< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
```

```
const Matrix< T > & solutionVector,  
size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

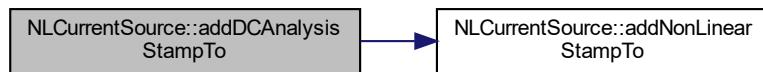
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 120 of file [NLCurrentSource.hpp](#).

Here is the call graph for this function:

**6.14.2.2 addNonLinearStampTo()**

```
template<typename T >
void NLCurrentSource< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

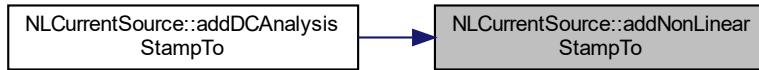
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 24 of file [NLCurrentSource.hpp](#).

Here is the caller graph for this function:



6.14.2.3 addToElements()

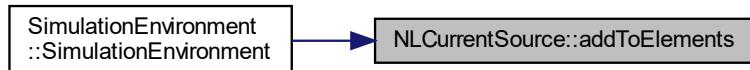
```
template<typename T >
static void NLCURRENTSOURCE< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 127 of file [NLCURRENTSOURCE.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.14.3 Member Data Documentation

6.14.3.1 n1

```
template<typename T >
size_t NLCurrentSource< T >::n1 = 0
```

Definition at line 15 of file [NLCurrentSource.hpp](#).

6.14.3.2 n2

```
template<typename T >
size_t NLCurrentSource< T >::n2 = 0
```

Definition at line 16 of file [NLCurrentSource.hpp](#).

6.14.3.3 r1_neg

```
template<typename T >
size_t NLCurrentSource< T >::r1_neg = 0
```

Definition at line 19 of file [NLCurrentSource.hpp](#).

6.14.3.4 r1_pos

```
template<typename T >
size_t NLCurrentSource< T >::r1_pos = 0
```

Definition at line 18 of file [NLCurrentSource.hpp](#).

6.14.3.5 r2_neg

```
template<typename T >
size_t NLCurrentSource< T >::r2_neg = 0
```

Definition at line 21 of file [NLCurrentSource.hpp](#).

6.14.3.6 r2_pos

```
template<typename T >
size_t NLCurrentSource< T >::r2_pos = 0
```

Definition at line 20 of file [NLCurrentSource.hpp](#).

6.14.3.7 value

```
template<typename T >
T NLCurrentSource< T >::value = 0
```

Definition at line 13 of file [NLCurrentSource.hpp](#).

The documentation for this struct was generated from the following file:

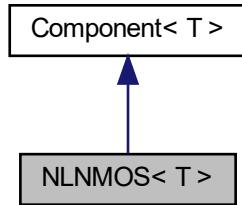
- [NLCurrentSource.hpp](#)

6.15 NLNMOS< T > Struct Template Reference

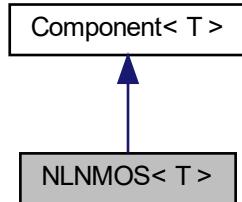
a non-linear FET model

```
#include <NLNMOS.hpp>
```

Inheritance diagram for NLNMOS< T >:



Collaboration diagram for NLNMOS< T >:



Public Member Functions

- void `addNonLinearStampTo (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep=0) const`
adds this component's non-linear stamp to the target stamp.
- void `updateStoredState (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A)`
Updates any stored state based on the current solution index.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- size_t `d` = 0
- size_t `g` = 0
- size_t `s` = 0
- const T `C_GSp` = 0.01
- const T `C_GSo` = 0.5
- const T `P_S10` = 0
- const T `P_S11` = 0.5
- const T `C_GDp` = 0.5
- const T `C_GDo` = 1
- const T `P_D10` = -1
- const T `P_D11` = 0.4
- const T `beta_DS` = 1.3
- const T `alpha_DS` = 0.42
- T `u_gd_last` = 0
- T `u_gs_last` = 0
- T `i_gd_last` = 0
- T `i_gs_last` = 0
- T `C_GD_last` = `C_GDp + C_GDo * (1.0 + std::tanh(P_D10 + P_D11 * u_gd_last))`
- T `C_GS_last` = `C_GSp + C_GSo * (1.0 + std::tanh(P_S10 + P_S11 * u_gs_last))`

6.15.1 Detailed Description

```
template<typename T>
struct NLNMOS< T >
```

a non-linear FET model

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 10 of file [NLNMOS.hpp](#).

6.15.2 Member Function Documentation

6.15.2.1 addNonLinearStampTo()

```
template<typename T >
void NLNMOS< T >::addNonLinearStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) const [inline], [virtual]
```

adds this component's non-linear stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line [40](#) of file [NLNMOS.hpp](#).

6.15.2.2 addToElements()

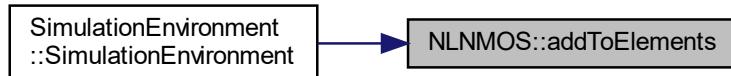
```
template<typename T >
static void NLNMOS< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line [181](#) of file [NLNMOS.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.15.2.3 updateStoredState()

```

template<typename T >
void NLNMOS< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
  
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 142 of file [NLNMOS.hpp](#).

6.15.3 Member Data Documentation

6.15.3.1 alpha_DS

```

template<typename T >
const T NLNMOS< T >::alpha_DS = 0.42
  
```

Definition at line 27 of file [NLNMOS.hpp](#).

6.15.3.2 beta_DS

```
template<typename T >
const T NLNMOS< T >::beta_DS = 1.3
```

Definition at line 26 of file [NLNMOS.hpp](#).

6.15.3.3 C_GD_last

```
template<typename T >
T NLNMOS< T >::C_GD_last = C_GDp + C_GDo * (1.0 + std::tanh(P_D10 + P_D11 * u_gd_last))
```

Definition at line 36 of file [NLNMOS.hpp](#).

6.15.3.4 C_GDo

```
template<typename T >
const T NLNMOS< T >::C_GDo = 1
```

Definition at line 22 of file [NLNMOS.hpp](#).

6.15.3.5 C_GDp

```
template<typename T >
const T NLNMOS< T >::C_GDp = 0.5
```

Definition at line 21 of file [NLNMOS.hpp](#).

6.15.3.6 C_GS_last

```
template<typename T >
T NLNMOS< T >::C_GS_last = C_GSp + C_GSo * (1.0 + std::tanh(P_S10 + P_S11 * u_gs_last))
```

Definition at line 37 of file [NLNMOS.hpp](#).

6.15.3.7 C_GSo

```
template<typename T >
const T NLNMOS< T >::C_GSo = 0.5
```

Definition at line 18 of file [NLNMOS.hpp](#).

6.15.3.8 C_GSp

```
template<typename T >
const T NLNMOS< T >::C_GSp = 0.01
```

Definition at line 17 of file [NLNMOS.hpp](#).

6.15.3.9 d

```
template<typename T >
size_t NLNMOS< T >::d = 0
```

Definition at line 12 of file [NLNMOS.hpp](#).

6.15.3.10 g

```
template<typename T >
size_t NLNMOS< T >::g = 0
```

Definition at line 13 of file [NLNMOS.hpp](#).

6.15.3.11 i_gd_last

```
template<typename T >
T NLNMOS< T >::i_gd_last = 0
```

Definition at line 33 of file [NLNMOS.hpp](#).

6.15.3.12 i_gs_last

```
template<typename T >
T NLNMOS< T >::i_gs_last = 0
```

Definition at line 34 of file [NLNMOS.hpp](#).

6.15.3.13 P_D10

```
template<typename T >
const T NLNMOS< T >::P_D10 = -1
```

Definition at line 23 of file [NLNMOS.hpp](#).

6.15.3.14 P_D11

```
template<typename T >
const T NLNMOS< T >::P_D11 = 0.4
```

Definition at line 24 of file [NLNMOS.hpp](#).

6.15.3.15 P_S10

```
template<typename T >
const T NLNMOS< T >::P_S10 = 0
```

Definition at line 19 of file [NLNMOS.hpp](#).

6.15.3.16 P_S11

```
template<typename T >
const T NLNMOS< T >::P_S11 = 0.5
```

Definition at line 20 of file [NLNMOS.hpp](#).

6.15.3.17 s

```
template<typename T >
size_t NLNMOS< T >::s = 0
```

Definition at line 14 of file [NLNMOS.hpp](#).

6.15.3.18 u_gd_last

```
template<typename T >
T NLNMOS< T >::u_gd_last = 0
```

Definition at line 29 of file [NLNMOS.hpp](#).

6.15.3.19 u_gs_last

```
template<typename T >  
T NLNMOS< T >::u_gs_last = 0
```

Definition at line 30 of file [NLNMOS.hpp](#).

The documentation for this struct was generated from the following file:

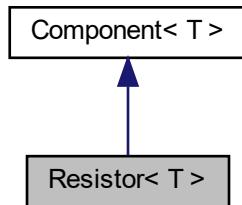
- [NLNMOS.hpp](#)

6.16 Resistor< T > Struct Template Reference

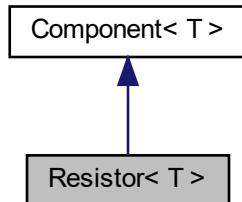
An ideal resistor model.

```
#include <Resistor.hpp>
```

Inheritance diagram for Resistor< T >:



Collaboration diagram for Resistor< T >:



Public Member Functions

- void `addStaticStampTo (Stamp< T > &stamp) const`
Adds this component's static stamp to the target stamp.
- void `addDCAnalysisStampTo (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const`
adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0
- size_t `currentIndex` = 0
- bool `group1` = true

6.16.1 Detailed Description

```
template<typename T>
struct Resistor< T >
```

An ideal resistor model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 10 of file [Resistor.hpp](#).

6.16.2 Member Function Documentation

6.16.2.1 addDCAnalysisStampTo()

```
template<typename T >
void Resistor< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 54 of file [Resistor.hpp](#).

Here is the call graph for this function:

**6.16.2.2 addStaticStampTo()**

```
template<typename T >
void Resistor< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented from [Component< T >](#).

Definition at line 20 of file [Resistor.hpp](#).

Here is the caller graph for this function:



6.16.2.3 addToElements()

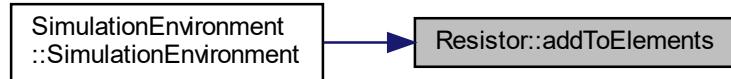
```
template<typename T>
static void Resistor<T>::addToElements(
    const std::string & line,
    CircuitElements<T> & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents) [inline], [static]
```

Definition at line 60 of file [Resistor.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.16.3 Member Data Documentation

6.16.3.1 currentIndex

```
template<typename T>
size_t Resistor<T>::currentIndex = 0
```

Definition at line 16 of file [Resistor.hpp](#).

6.16.3.2 group1

```
template<typename T >
bool Resistor< T >::group1 = true
```

Definition at line 18 of file [Resistor.hpp](#).

6.16.3.3 n1

```
template<typename T >
size_t Resistor< T >::n1 = 0
```

Definition at line 14 of file [Resistor.hpp](#).

6.16.3.4 n2

```
template<typename T >
size_t Resistor< T >::n2 = 0
```

Definition at line 15 of file [Resistor.hpp](#).

6.16.3.5 value

```
template<typename T >
T Resistor< T >::value = 0
```

Definition at line 12 of file [Resistor.hpp](#).

The documentation for this struct was generated from the following file:

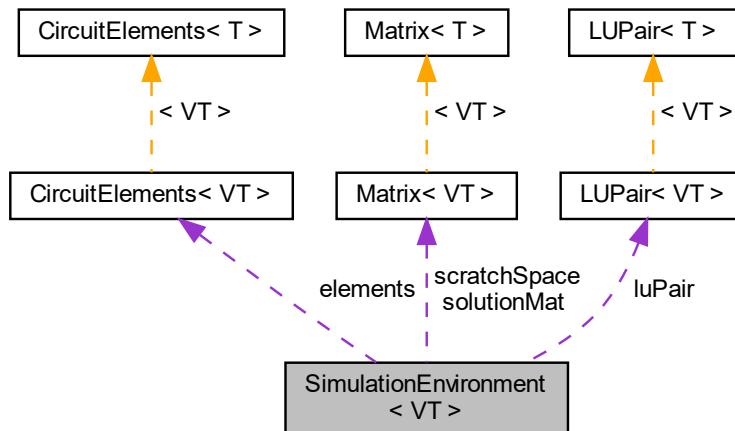
- [Resistor.hpp](#)

6.17 SimulationEnvironment< VT > Class Template Reference

The main class to hold all of the relevant simulation data.

```
#include <Simulator.hpp>
```

Collaboration diagram for SimulationEnvironment< VT >:



Public Member Functions

- **SimulationEnvironment** (std::string *netlistPath*)

Setup the simulation environment from a netlist.
- void **setDCOpPoint** ()

a function to determine and set the DC operating point
- void **simulate** ()

Where a lot of the magic starts. This is what runs the simulation.
- void **printGraph** (size_t *node*)

Outputs a single node's (or current's) time series to a graph, saving as both eps and png.
- void **printMultipleOnGraph** (std::vector< size_t > *nodeVec*, std::string *suffix=""*)

Similar to printGraph, but instead can plot multiple series on the same graph.
- void **dataDump** ()

Simple function to dump the output of the simulator in a matlab table readable format.

Private Member Functions

- void **parseGraph** (std::string *line*)

Helper function to pull indices from graph netlist directive.

Private Attributes

- std::string `outputFilePath` = "datadump.txt"
- std::string `netlistPath` = ""
- double `initialTime`
- double `timestep`
- double `finalTime`
- size_t `steps`
- size_t `numNodes` = 1
- size_t `numCurrents` = 0
- size_t `numDCCurrents` = 0
- bool `performDCAnalysis` = true
- `CircuitElements< VT > elements`
A collection of all the circuit elements.
- `LUPair< VT > luPair`
Preallocated space to prevent repeated allocations and deallocations.
- `Matrix< VT > scratchSpace`
Preallocated space to prevent repeated allocations and deallocations. using during the leftDivide to solve the MNA system.
- std::vector< std::vector< size_t > > `nodesToGraph`
Keeps track of the nodes to be graphed after simulation.
- `Matrix< VT > solutionMat`
Preallocated space to store the results in.

6.17.1 Detailed Description

```
template<typename VT>
class SimulationEnvironment< VT >
```

The main class to hold all of the relevant simulation data.

Template Parameters

<code>VT</code>	The type used for values. e.g. double, float, etc
-----------------	---

Definition at line 46 of file [Simulator.hpp](#).

6.17.2 Constructor & Destructor Documentation

6.17.2.1 `SimulationEnvironment()`

```
template<typename VT >
SimulationEnvironment< VT >::SimulationEnvironment (
    std::string netlistPath ) [inline]
```

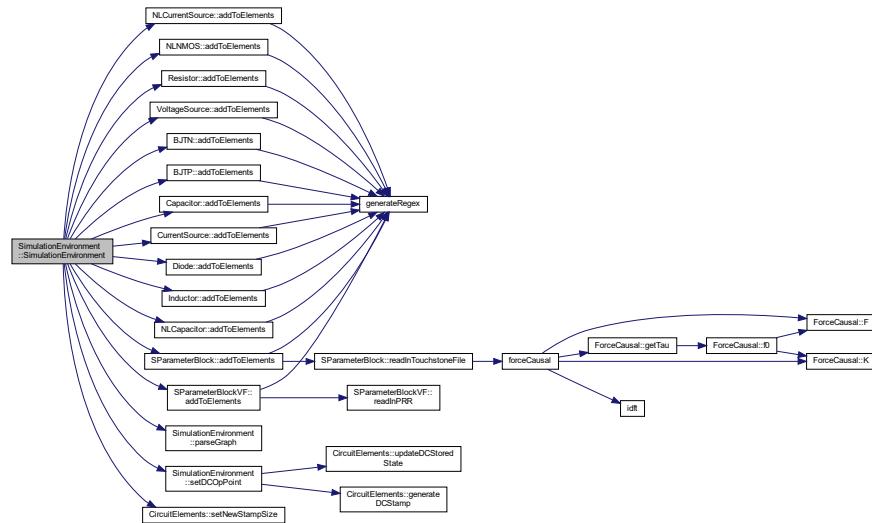
Setup the simulation environment from a netlist.

Parameters

<code>netlistPath</code>	Path to the netlist that is going to be used in the simulation
--------------------------	--

Definition at line 52 of file [Simulator.hpp](#).

Here is the call graph for this function:



6.17.3 Member Function Documentation

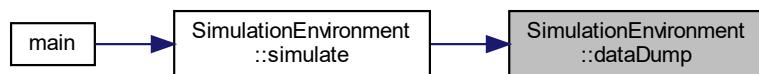
6.17.3.1 dataDump()

```
template<typename VT >
void SimulationEnvironment< VT >::dataDump ( ) [inline]
```

Simple function to dump the output of the simulator in a matlab table readable format.

Definition at line 392 of file [Simulator.hpp](#).

Here is the caller graph for this function:



6.17.3.2 parseGraph()

```
template<typename VT >
void SimulationEnvironment< VT >::parseGraph (
    std::string line ) [inline], [private]
```

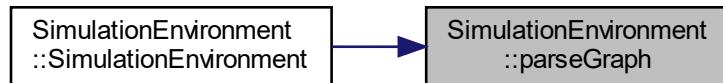
Helper function to pull indices from graph netlist directive.

Parameters

<i>line</i>	The line to parse
-------------	-------------------

Definition at line 466 of file [Simulator.hpp](#).

Here is the caller graph for this function:



6.17.3.3 printGraph()

```
template<typename VT >
void SimulationEnvironment< VT >::printGraph (
    size_t node ) [inline]
```

Outputs a single node's (or current's) time series to a graph, saving as both eps and png.

Parameters

<i>node</i>	The index of the node to plot
-------------	-------------------------------

Definition at line 333 of file [Simulator.hpp](#).

6.17.3.4 printMultipleOnGraph()

```
template<typename VT >
void SimulationEnvironment< VT >::printMultipleOnGraph (
```

```
std::vector< size_t > nodeVec,  
std::string suffix = "" ) [inline]
```

Similar to printGraph, but instead can plot multiple series on the same graph.

Parameters

<i>nodeVec</i>	A vector of node indices
<i>suffix</i>	What's appended to "Graph" to make the name of the output file.

Definition at line 362 of file [Simulator.hpp](#).

Here is the caller graph for this function:



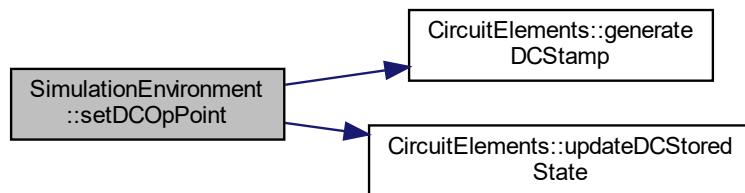
6.17.3.5 setDCOpPoint()

```
template<typename VT >
void SimulationEnvironment< VT >::setDCOpPoint ( ) [inline]
```

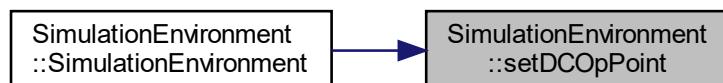
a function to determine and set the DC operating point

Definition at line 230 of file [Simulator.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.17.3.6 simulate()

```
template<typename VT >
void SimulationEnvironment< VT >::simulate ( ) [inline]
```

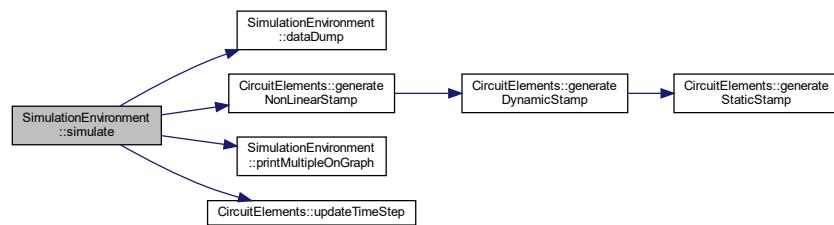
Where a lot of the magic starts. This is what runs the simulation.

This function is a simple newton-raphson solver, it still has room for improvement, such as early termination when the loop converges.

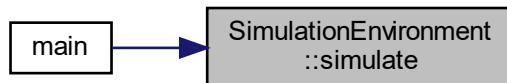
After the simulation has run to completion, the raw data is dumped, and any graphs that were due to be generated are created.

Definition at line 265 of file [Simulator.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.17.4 Member Data Documentation

6.17.4.1 elements

```
template<typename VT >
CircuitElements<VT> SimulationEnvironment< VT >::elements [private]
```

A collection of all the circuit elements.

Definition at line 493 of file [Simulator.hpp](#).

6.17.4.2 finalTime

```
template<typename VT >
double SimulationEnvironment< VT >::finalTime [private]
```

Definition at line 485 of file [Simulator.hpp](#).

6.17.4.3 initialTime

```
template<typename VT >
double SimulationEnvironment< VT >::initialTime [private]
```

Definition at line 483 of file [Simulator.hpp](#).

6.17.4.4 luPair

```
template<typename VT >
LUPair<VT> SimulationEnvironment< VT >::luPair [private]
```

Preallocated space to prevent repeated allocations and deallocations.

Definition at line 496 of file [Simulator.hpp](#).

6.17.4.5 netlistPath

```
template<typename VT >
std::string SimulationEnvironment< VT >::netlistPath = "" [private]
```

Definition at line 481 of file [Simulator.hpp](#).

6.17.4.6 nodesToGraph

```
template<typename VT >
std::vector<std::vector<size_t>> SimulationEnvironment< VT >::nodesToGraph [private]
```

Keeps track of the nodes to be graphed after simulation.

Definition at line 502 of file [Simulator.hpp](#).

6.17.4.7 numCurrents

```
template<typename VT >
size_t SimulationEnvironment< VT >::numCurrents = 0 [private]
```

Definition at line 489 of file [Simulator.hpp](#).

6.17.4.8 numDCCurrents

```
template<typename VT >
size_t SimulationEnvironment< VT >::numDCCurrents = 0 [private]
```

Definition at line 490 of file [Simulator.hpp](#).

6.17.4.9 numNodes

```
template<typename VT >
size_t SimulationEnvironment< VT >::numNodes = 1 [private]
```

Definition at line 488 of file [Simulator.hpp](#).

6.17.4.10 outputPath

```
template<typename VT >
std::string SimulationEnvironment< VT >::outputPath = "datadump.txt" [private]
```

Definition at line 480 of file [Simulator.hpp](#).

6.17.4.11 performDCAnalysis

```
template<typename VT >
bool SimulationEnvironment< VT >::performDCAnalysis = true [private]
```

Definition at line 491 of file [Simulator.hpp](#).

6.17.4.12 scratchSpace

```
template<typename VT >
Matrix<VT> SimulationEnvironment< VT >::scratchSpace [private]
```

Preallocated space to prevent repeated allocations and deallocations. using during the leftDivide to solve the MNA system.

Definition at line 499 of file [Simulator.hpp](#).

6.17.4.13 solutionMat

```
template<typename VT >
Matrix<VT> SimulationEnvironment< VT >::solutionMat [private]
```

Preallocated space to store the results in.

Definition at line 505 of file [Simulator.hpp](#).

6.17.4.14 steps

```
template<typename VT >
size_t SimulationEnvironment< VT >::steps [private]
```

Definition at line 486 of file [Simulator.hpp](#).

6.17.4.15 timestep

```
template<typename VT >
double SimulationEnvironment< VT >::timestep [private]
```

Definition at line 484 of file [Simulator.hpp](#).

The documentation for this class was generated from the following file:

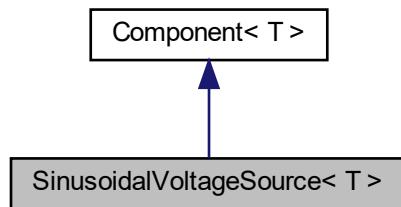
- [Simulator.hpp](#)

6.18 SinusoidalVoltageSource< T > Struct Template Reference

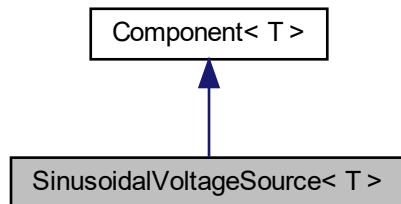
A sinusoidal voltage source model.

```
#include <SinusoidalVoltageSource.hpp>
```

Inheritance diagram for SinusoidalVoltageSource< T >:



Collaboration diagram for SinusoidalVoltageSource< T >:



Public Member Functions

- void [addDynamicStampTo \(Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep\) const](#)

Adds this component's dynamic stamp to the target stamp.
- void [updateStoredState \(const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A\)](#)

Updates any stored state based on the current solution index.
- void [addDCAnalysisStampTo \(Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents\) const](#)

adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements` (const std::string &line, `CircuitElements< T >` &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)

Public Attributes

- size_t `n1` = 0
- size_t `n2` = 0
- size_t `currentIndex` = 0
- T `V` = 1
- T `phase` = 0
- T `frequency` = 1
- T `offset` = 0
- bool `degrees` = true

6.18.1 Detailed Description

```
template<typename T>
struct SinusoidalVoltageSource< T >
```

A sinusoidal voltage source model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 11 of file `SinusoidalVoltageSource.hpp`.

6.18.2 Member Function Documentation

6.18.2.1 addDCAnalysisStampTo()

```
template<typename T >
void SinusoidalVoltageSource< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

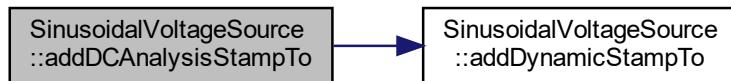
Parameters

<code>destination</code>	The stamp to be added to.
<code>solutionMatrix</code>	A vector containing all past solutions to the circuit
<code>numCurrents</code>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 56 of file [SinusoidalVoltageSource.hpp](#).

Here is the call graph for this function:



6.18.2.2 addDynamicStampTo()

```
template<typename T >
void SinusoidalVoltageSource< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

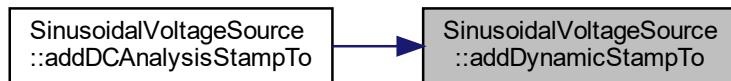
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 22 of file [SinusoidalVoltageSource.hpp](#).

Here is the caller graph for this function:

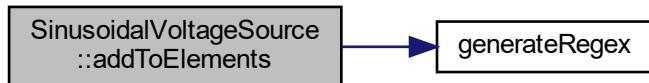


6.18.2.3 addToElements()

```
template<typename T >
static void SinusoidalVoltageSource< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 62 of file [SinusoidalVoltageSource.hpp](#).

Here is the call graph for this function:



6.18.2.4 updateStoredState()

```
template<typename T >
void SinusoidalVoltageSource< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 51 of file [SinusoidalVoltageSource.hpp](#).

6.18.3 Member Data Documentation

6.18.3.1 currentIndex

```
template<typename T >
size_t SinusoidalVoltageSource< T >::currentIndex = 0
```

Definition at line 14 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.2 degrees

```
template<typename T >
bool SinusoidalVoltageSource< T >::degrees = true
```

Definition at line 20 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.3 frequency

```
template<typename T >
T SinusoidalVoltageSource< T >::frequency = 1
```

Definition at line 18 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.4 n1

```
template<typename T >
size_t SinusoidalVoltageSource< T >::n1 = 0
```

Definition at line 12 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.5 n2

```
template<typename T >
size_t SinusoidalVoltageSource< T >::n2 = 0
```

Definition at line 13 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.6 offset

```
template<typename T >
T SinusoidalVoltageSource< T >::offset = 0
```

Definition at line 19 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.7 phase

```
template<typename T >
T SinusoidalVoltageSource< T >::phase = 0
```

Definition at line 17 of file [SinusoidalVoltageSource.hpp](#).

6.18.3.8 V

```
template<typename T >
T SinusoidalVoltageSource< T >::V = 1
```

Definition at line 16 of file [SinusoidalVoltageSource.hpp](#).

The documentation for this struct was generated from the following file:

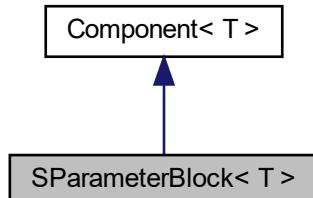
- [SinusoidalVoltageSource.hpp](#)

6.19 SParameterBlock< T > Struct Template Reference

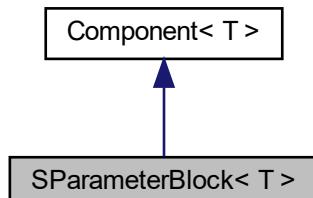
A DTIR based model of an s-parameter block.

```
#include <SParameterBlock.hpp>
```

Inheritance diagram for SParameterBlock< T >:



Collaboration diagram for SParameterBlock< T >:



Public Member Functions

- T [aWaveConvValue](#) (size_t portIndex, const Matrix< T > &solutionMatrix, const size_t n, T sTimePoint, const T simulationTimestep, size_t sizeG_A) const

performs a linear interpolation and returns the a wave value for use in the convolution.
- T [V_p](#) (size_t p, const Matrix< T > &solutionMatrix, const size_t n, T simulationTimestep, size_t sizeG_A) const

Determines the equivalent port voltage source by convolving the (historic) a wave values with the DTIR.
- T [R_p](#) (size_t p) const
- T [beta_p](#) (size_t p) const
- void [addStaticStampTo](#) (Stamp< T > &stamp) const

Adds this component's static stamp to the target stamp.
- void [addDynamicStampTo](#) (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T simulationTimestep) const

Adds this component's dynamic stamp to the target stamp.
- void [addDCAAnalysisStampTo](#) (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const

adds this component's DC stamp to the target stamp.
- void [updateStoredState](#) (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A)

Updates any stored state based on the current solution index.
- void [readInTouchstoneFile](#) ()

reads in the s-parameter data from a touchstone file. Currently it ignores the units of frequency, and only works with real-imag formatting

Static Public Member Functions

- static void [addToElements](#) (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)

Public Attributes

- std::string [touchstoneFilePath](#) = ""
- std::vector< [SParameterPort](#)< T > > [port](#)
- [SParameterSequence](#)< T > [s](#)
- T [z_ref](#) = 0
- T [fracMaxToKeep](#) = 0

6.19.1 Detailed Description

```
template<typename T>
struct SParameterBlock< T >
```

A DTIR based model of an s-parameter block.

Template Parameters

T	
---	--

Definition at line 87 of file [SParameterBlock.hpp](#).

6.19.2 Member Function Documentation

6.19.2.1 addDCAnalysisStampTo()

```
template<typename T >
void SParameterBlock< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 240 of file [SParameterBlock.hpp](#).

6.19.2.2 addDynamicStampTo()

```
template<typename T >
void SParameterBlock< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 228 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



6.19.2.3 addStaticStampTo()

```
template<typename T >
void SParameterBlock< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented from [Component< T >](#).

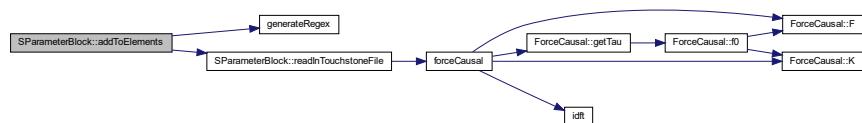
Definition at line 194 of file [SParameterBlock.hpp](#).

6.19.2.4 addToElements()

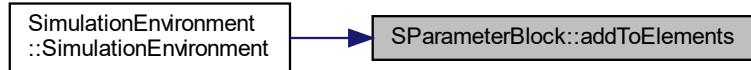
```
template<typename T >
static void SParameterBlock< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 387 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.19.2.5 aWaveConvValue()

```

template<typename T >
T SParameterBlock< T >::aWaveConvValue (
    size_t portIndex,
    const Matrix< T > & solutionMatrix,
    const size_t n,
    T sTimePoint,
    const T simulationTimestep,
    size_t sizeG_A ) const [inline]
  
```

performs a linear interpolation and returns the a wave value for use in the convolution.

Parameters

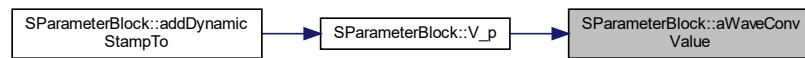
<i>portIndex</i>	The port being processed
<i>solutionMatrix</i>	The working solution vector
<i>n</i>	The current time step
<i>sTimePoint</i>	The time of DTIR
<i>simulationTimestep</i>	The timestep of the simulation
<i>sizeG_A</i>	The size of the voltage dependant portion of the stamp

Returns

the value for use in the convolution

Definition at line 106 of file [SParameterBlock.hpp](#).

Here is the caller graph for this function:



6.19.2.6 beta_p()

```
template<typename T>
T SParameterBlock< T >::beta_p (
    size_t p ) const [inline]
```

Definition at line 190 of file [SParameterBlock.hpp](#).

6.19.2.7 R_p()

```
template<typename T>
T SParameterBlock< T >::R_p (
    size_t p ) const [inline]
```

Definition at line 186 of file [SParameterBlock.hpp](#).

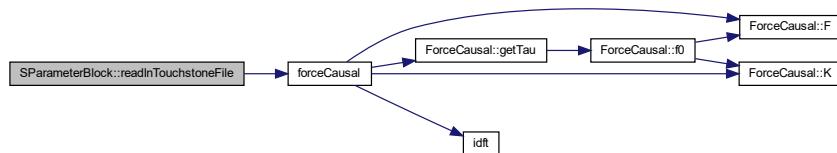
6.19.2.8 readInTouchstoneFile()

```
template<typename T>
void SParameterBlock< T >::readInTouchstoneFile ( ) [inline]
```

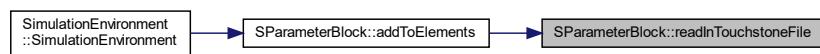
reads in the s-parameter data from a touchstone file. Currently it ignores the units of frequency, and only works with real-imag formatting

Definition at line 308 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.19.2.9 updateStoredState()

```
template<typename T >
void SParameterBlock< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 299 of file [SParameterBlock.hpp](#).

6.19.2.10 V_p()

```
template<typename T >
T SParameterBlock< T >::V_p (
    size_t p,
    const Matrix< T > & solutionMatrix,
    const size_t n,
    T simulationTimestep,
    size_t sizeG_A ) const [inline]
```

Determines the equivalent port voltage source by convolving the (historic) a wave values with the DTIR.

Parameters

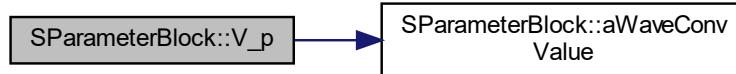
<i>p</i>	The port index
<i>solutionMatrix</i>	The solution vector of the circuit
<i>n</i>	The current timestep
<i>simulationTimestep</i>	the simulations timestep
<i>sizeG_A</i>	the size of the voltage dependant part of the stamp

Returns

Equivalent port voltage source voltage

Definition at line 166 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.19.3 Member Data Documentation

6.19.3.1 fracMaxToKeep

```
template<typename T >
T SParameterBlock< T >::fracMaxToKeep = 0
```

Definition at line 93 of file [SParameterBlock.hpp](#).

6.19.3.2 port

```
template<typename T >
std::vector<SParameterPort<T>> SParameterBlock< T >::port
```

Definition at line 89 of file [SParameterBlock.hpp](#).

6.19.3.3 s

```
template<typename T >
SParameterSequence<T> SParameterBlock< T >::s
```

Definition at line 90 of file [SParameterBlock.hpp](#).

6.19.3.4 touchstoneFilePath

```
template<typename T >
std::string SParameterBlock< T >::touchstoneFilePath = ""
```

Definition at line 88 of file [SParameterBlock.hpp](#).

6.19.3.5 z_ref

```
template<typename T >
T SParameterBlock< T >::z_ref = 0
```

Definition at line 92 of file [SParameterBlock.hpp](#).

The documentation for this struct was generated from the following file:

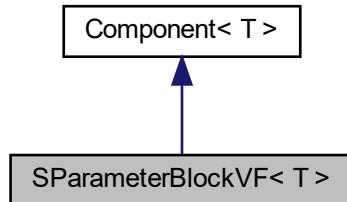
- [SParameterBlock.hpp](#)

6.20 SParameterBlockVF< T > Struct Template Reference

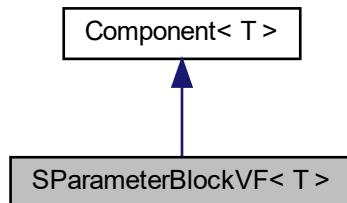
A vectorfitting based model of an s-parameter block.

```
#include <SParameterBlockVF.hpp>
```

Inheritance diagram for SParameterBlockVF< T >:



Collaboration diagram for SParameterBlockVF< T >:



Public Member Functions

- std::complex< T > **history_p** (const size_t p, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, const size_t sizeG_A) const
- T **V_p** (const size_t p, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, const size_t sizeG_A) const
- T **awave_p** (const size_t p, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, const size_t sizeG_A) const
- T **bwave_p** (const size_t p, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, const size_t sizeG_A) const
- void **addStaticStampTo** (Stamp< T > &stamp) const

Adds this component's static stamp to the target stamp.
- void **addDynamicStampTo** (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T simulationTimestep) const

Adds this component's dynamic stamp to the target stamp.
- void **updateStoredState** (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A)

Updates any stored state based on the current solution index.
- void **setConstants** (T timestep)
- void **setFirstOrder** (T timestep)
- void **setSecondOrder** (T timestep)
- void **setTimestep** (T timestep)

Initialises the component
- void **readInPRR** (std::string filePath, size_t numPorts)
- void **addDCAnalysisStampTo** (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const

adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void **addToElements** (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)

Public Attributes

- std::vector< SParameterPortVF< T > > **port**
- size_t **numPorts** = 0
- bool **firstOrder** = true
- T **z_ref** = 0

6.20.1 Detailed Description

```
template<typename T>
struct SParameterBlockVF< T >
```

A vectorfitting based model of an s-parameter block.

Template Parameters

T	
---	--

Definition at line 78 of file [SParameterBlockVF.hpp](#).

6.20.2 Member Function Documentation

6.20.2.1 addDCAnalysisStampTo()

```
template<typename T >
void SParameterBlockVF< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 445 of file [SParameterBlockVF.hpp](#).

6.20.2.2 addDynamicStampTo()

```
template<typename T >
void SParameterBlockVF< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

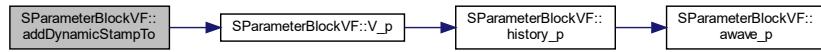
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

Reimplemented from [Component< T >](#).

Definition at line 192 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



6.20.2.3 addStaticStampTo()

```
template<typename T >
void SParameterBlockVF< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented from [Component< T >](#).

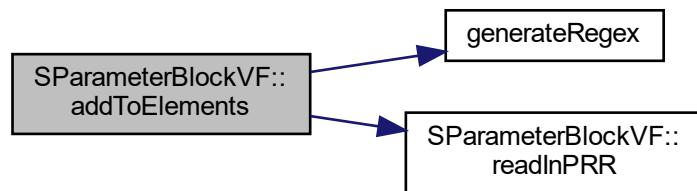
Definition at line 153 of file [SParameterBlockVF.hpp](#).

6.20.2.4 addToElements()

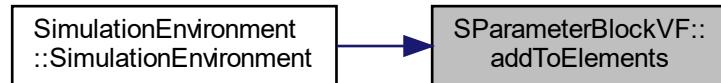
```
template<typename T >
static void SParameterBlockVF< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 502 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



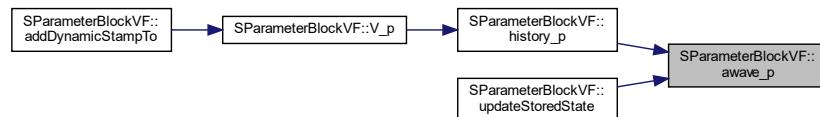
6.20.2.5 awave_p()

```

template<typename T >
T SParameterBlockVF< T >::awave_p (
    const size_t p,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    const size_t sizeG_A ) const [inline]
  
```

Definition at line 113 of file [SParameterBlockVF.hpp](#).

Here is the caller graph for this function:



6.20.2.6 bwave_p()

```

template<typename T >
T SParameterBlockVF< T >::bwave_p (
    const size_t p,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    const size_t sizeG_A ) const [inline]
  
```

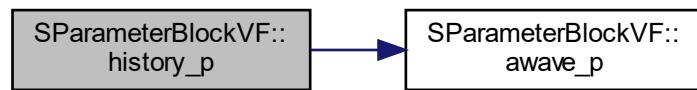
Definition at line 133 of file [SParameterBlockVF.hpp](#).

6.20.2.7 history_p()

```
template<typename T >
std::complex<T> SParameterBlockVF< T >::history_p (
    const size_t p,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    const size_t sizeG_A ) const [inline]
```

Definition at line 85 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

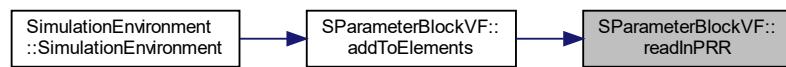


6.20.2.8 readInPRR()

```
template<typename T >
void SParameterBlockVF< T >::readInPRR (
    std::string filePath,
    size_t numPorts ) [inline]
```

Definition at line 385 of file [SParameterBlockVF.hpp](#).

Here is the caller graph for this function:

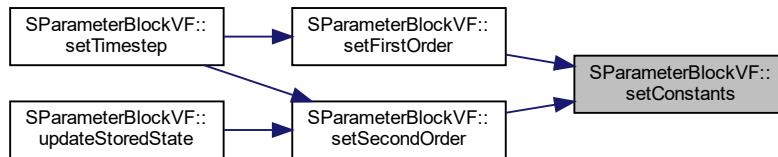


6.20.2.9 setConstants()

```
template<typename T >
void SParameterBlockVF< T >::setConstants (
    T timestep ) [inline]
```

Definition at line 235 of file [SParameterBlockVF.hpp](#).

Here is the caller graph for this function:

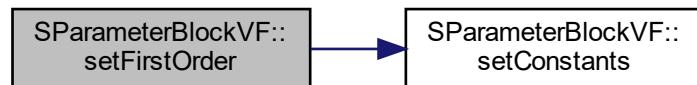


6.20.2.10 setFirstOrder()

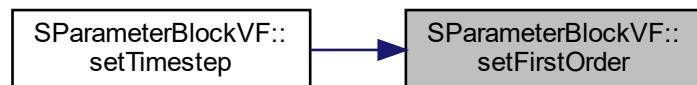
```
template<typename T >
void SParameterBlockVF< T >::setFirstOrder (
    T timestep ) [inline]
```

Definition at line 256 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

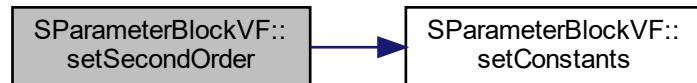


6.20.2.11 setSecondOrder()

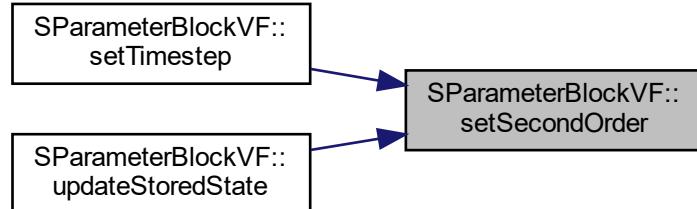
```
template<typename T >
void SParameterBlockVF< T >::setSecondOrder (
    T timestep ) [inline]
```

Definition at line 285 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.20.2.12 setTimestep()

```
template<typename T >
void SParameterBlockVF< T >::setTimestep (
    T timestep ) [inline], [virtual]
```

initialises the component

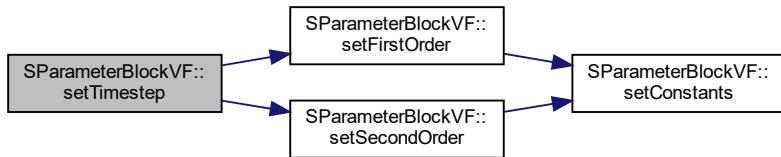
Parameters

<i>timestep</i>	The length of each time step
-----------------	------------------------------

Reimplemented from [Component< T >](#).

Definition at line 319 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



6.20.2.13 updateStoredState()

```
template<typename T >
void SParameterBlockVF< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

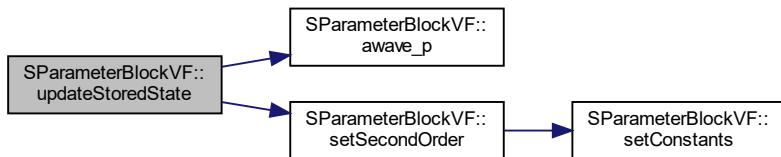
Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 204 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



6.20.2.14 V_p()

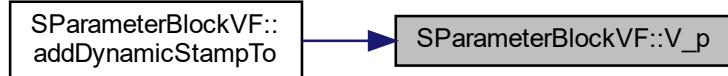
```
template<typename T >
T SParameterBlockVF< T >::V_p (
    const size_t p,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    const size_t sizeG_A ) const [inline]
```

Definition at line 106 of file [SParameterBlockVF.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.20.3 Member Data Documentation

6.20.3.1 firstOrder

```
template<typename T >
bool SParameterBlockVF< T >::firstOrder = true
```

Definition at line 81 of file [SParameterBlockVF.hpp](#).

6.20.3.2 numPorts

```
template<typename T >
size_t SParameterBlockVF< T >::numPorts = 0
```

Definition at line 80 of file [SParameterBlockVF.hpp](#).

6.20.3.3 port

```
template<typename T >
std::vector<SParameterPortVF<T> > SParameterBlockVF< T >::port
```

Definition at line 79 of file [SParameterBlockVF.hpp](#).

6.20.3.4 z_ref

```
template<typename T >
T SParameterBlockVF< T >::z_ref = 0
```

Definition at line 83 of file [SParameterBlockVF.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlockVF.hpp](#)

6.21 SParameterPort< T > Struct Template Reference

a helper struct to store the information for the ports of an S-Parameter Block

```
#include <SParameterBlock.hpp>
```

Public Attributes

- size_t **positive** = 0
The positive index.
- size_t **negative** = 0
The negative index.
- size_t **current** = 0
The current index.
- T **R** = 0
The equivalent resistance.
- T **beta** = 0
a constant factor
- std::vector< T > **s0**
The start of each [SParameterSequence](#).

6.21.1 Detailed Description

```
template<typename T>
struct SParameterPort< T >
```

a helper struct to store the information for the ports of an S-Parameter Block

Template Parameters

<i>T</i>	The value type
----------	----------------

Definition at line 20 of file [SParameterBlock.hpp](#).

6.21.2 Member Data Documentation

6.21.2.1 beta

```
template<typename T >
T SParameterPort< T >::beta = 0
```

a constant factor

Definition at line 30 of file [SParameterBlock.hpp](#).

6.21.2.2 current

```
template<typename T >
size_t SParameterPort< T >::current = 0
```

The current index.

Definition at line 26 of file [SParameterBlock.hpp](#).

6.21.2.3 negative

```
template<typename T >
size_t SParameterPort< T >::negative = 0
```

The negative index.

Definition at line 24 of file [SParameterBlock.hpp](#).

6.21.2.4 positive

```
template<typename T >
size_t SParameterPort< T >::positive = 0
```

The positive index.

Definition at line 22 of file [SParameterBlock.hpp](#).

6.21.2.5 R

```
template<typename T >
T SParameterPort< T >::R = 0
```

The equivalent resistance.

Definition at line 28 of file [SParameterBlock.hpp](#).

6.21.2.6 s0

```
template<typename T >
std::vector<T> SParameterPort< T >::s0
```

The start of each [SParameterSequence](#).

Definition at line 32 of file [SParameterBlock.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlock.hpp](#)

6.22 SParameterPortVF< T > Struct Template Reference

a helper struct to store the information for the ports of an S-Parameter Block

```
#include <SParameterBlockVF.hpp>
```

Public Attributes

- size_t **positive** = 0
The positive index.
- size_t **negative** = 0
The negative index.
- size_t **current** = 0
The current index.
- std::complex< T > **beta** = 0
a constant factor equal to $1 / (1 - \lambda)$
- std::vector< std::complex< T > > **alpha**
The equivalent scalar for controlled sources. Index is the second port.
- std::complex< T > **R** = 0
The equivalent resistance.
- std::vector< [SParamVFDDataFrom](#)< T > > **from**

6.22.1 Detailed Description

```
template<typename T>
struct SParameterPortVF< T >
```

a helper struct to store the information for the ports of an S-Parameter Block

Template Parameters

<i>T</i>	The value type
----------	----------------

Definition at line 51 of file [SParameterBlockVF.hpp](#).

6.22.2 Member Data Documentation

6.22.2.1 alpha

```
template<typename T >
std::vector<std::complex<T> > SParameterPortVF< T >::alpha
```

The equivalent scalar for controlled sources. Index is the second port.

Definition at line 65 of file [SParameterBlockVF.hpp](#).

6.22.2.2 beta

```
template<typename T >
std::complex<T> SParameterPortVF< T >::beta = 0
```

a constant factor equal to $1 / (1 - \text{lambda})$

Definition at line 61 of file [SParameterBlockVF.hpp](#).

6.22.2.3 current

```
template<typename T >
size_t SParameterPortVF< T >::current = 0
```

The current index.

Definition at line 57 of file [SParameterBlockVF.hpp](#).

6.22.2.4 from

```
template<typename T >
std::vector<SParamVFDataFrom<T> > SParameterPortVF< T >::from
```

Definition at line 70 of file [SParameterBlockVF.hpp](#).

6.22.2.5 negative

```
template<typename T >
size_t SParameterPortVF< T >::negative = 0
```

The negative index.

Definition at line 55 of file [SParameterBlockVF.hpp](#).

6.22.2.6 positive

```
template<typename T >
size_t SParameterPortVF< T >::positive = 0
```

The positive index.

Definition at line 53 of file [SParameterBlockVF.hpp](#).

6.22.2.7 R

```
template<typename T >
std::complex<T> SParameterPortVF< T >::R = 0
```

The equivalent resistance.

Definition at line 68 of file [SParameterBlockVF.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlockVF.hpp](#)

6.23 SParameterSequence< T > Struct Template Reference

a helper struct to store the DTIR sequence for the bloc

```
#include <SParameterBlock.hpp>
```

Public Member Functions

- size_t & [length](#) (size_t a, size_t b)
- const size_t [length](#) (size_t a, size_t b) const
- size_t & [offset](#) (size_t a, size_t b)
- const size_t [offset](#) (size_t a, size_t b) const
- T & [data](#) (size_t a, size_t b, size_t n)
- const T & [data](#) (size_t a, size_t b, size_t n) const
- T & [time](#) (size_t a, size_t b, size_t n)
- const T & [time](#) (size_t a, size_t b, size_t n) const

Public Attributes

- std::vector< T > `_data`
- std::vector< T > `_time`
- std::vector< SParamLengthOffset > `sParamLengthOffset`
- size_t `numPorts` = 0

6.23.1 Detailed Description

```
template<typename T>
struct SParameterSequence< T >
```

a helper struct to store the DTIR sequence for the bloc

Template Parameters

<code>T</code>	The value type
----------------	----------------

Definition at line 44 of file [SParameterBlock.hpp](#).

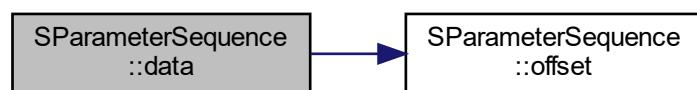
6.23.2 Member Function Documentation

6.23.2.1 data() [1/2]

```
template<typename T >
T& SParameterSequence< T >::data (
    size_t a,
    size_t b,
    size_t n )  [inline]
```

Definition at line 66 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:

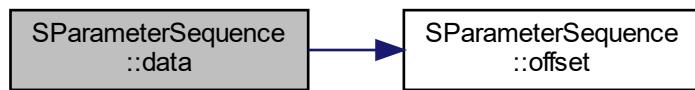


6.23.2.2 `data()` [2/2]

```
template<typename T >
const T& SParameterSequence< T >::data (
    size_t a,
    size_t b,
    size_t n ) const [inline]
```

Definition at line 70 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



6.23.2.3 `length()` [1/2]

```
template<typename T >
size_t& SParameterSequence< T >::length (
    size_t a,
    size_t b ) [inline]
```

Definition at line 50 of file [SParameterBlock.hpp](#).

6.23.2.4 `length()` [2/2]

```
template<typename T >
const size_t SParameterSequence< T >::length (
    size_t a,
    size_t b ) const [inline]
```

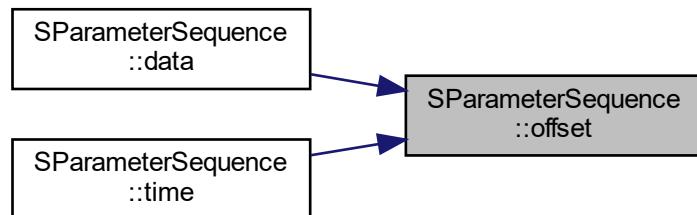
Definition at line 54 of file [SParameterBlock.hpp](#).

6.23.2.5 offset() [1/2]

```
template<typename T>
size_t& SParameterSequence<T>::offset(
    size_t a,
    size_t b) [inline]
```

Definition at line 58 of file [SParameterBlock.hpp](#).

Here is the caller graph for this function:



6.23.2.6 offset() [2/2]

```
template<typename T>
const size_t SParameterSequence<T>::offset(
    size_t a,
    size_t b) const [inline]
```

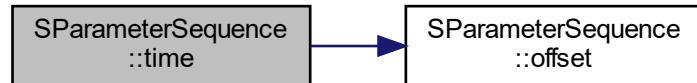
Definition at line 62 of file [SParameterBlock.hpp](#).

6.23.2.7 time() [1/2]

```
template<typename T>
T& SParameterSequence<T>::time(
    size_t a,
    size_t b,
    size_t n) [inline]
```

Definition at line 74 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:

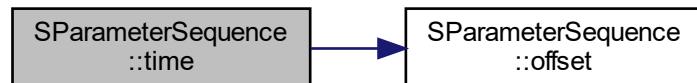


6.23.2.8 time() [2/2]

```
template<typename T >
const T& SParameterSequence< T >::time (
    size_t a,
    size_t b,
    size_t n ) const [inline]
```

Definition at line 78 of file [SParameterBlock.hpp](#).

Here is the call graph for this function:



6.23.3 Member Data Documentation

6.23.3.1 _data

```
template<typename T >
std::vector<T> SParameterSequence< T >::_data
```

Definition at line 45 of file [SParameterBlock.hpp](#).

6.23.3.2 _time

```
template<typename T >
std::vector<T> SParameterSequence< T >::_time
```

Definition at line 46 of file [SParameterBlock.hpp](#).

6.23.3.3 numPorts

```
template<typename T >
size_t SParameterSequence< T >::numPorts = 0
```

Definition at line 48 of file [SParameterBlock.hpp](#).

6.23.3.4 sParamLengthOffset

```
template<typename T >
std::vector<SParamLengthOffset> SParameterSequence< T >::sParamLengthOffset
```

Definition at line 47 of file [SParameterBlock.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlock.hpp](#)

6.24 SParamLengthOffset Struct Reference

```
#include <SParameterBlock.hpp>
```

Public Attributes

- size_t [length](#)
- size_t [offset](#)

6.24.1 Detailed Description

Definition at line 35 of file [SParameterBlock.hpp](#).

6.24.2 Member Data Documentation

6.24.2.1 length

```
size_t SParamLengthOffset::length
```

Definition at line 36 of file [SParameterBlock.hpp](#).

6.24.2.2 offset

```
size_t SParamLengthOffset::offset
```

Definition at line 37 of file [SParameterBlock.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlock.hpp](#)

6.25 SParamVFDataFrom< T > Struct Template Reference

```
#include <SParameterBlockVF.hpp>
```

Public Attributes

- size_t **numPoles** = 0
- std::vector< std::complex< T > > **pole**
- std::vector< std::complex< T > > **residue**
- std::complex< T > **remainder** = 0
- std::vector< std::complex< T > > **lambda_p**
the per-pole contribution of the current aware
- std::vector< std::complex< T > > **mu_p**
the per-pole contribution of the previous aware
- std::vector< std::complex< T > > **nu_p**
the per-pole contribution of the 2nd previous aware
- std::vector< std::complex< T > > **exp_alpha**
- std::complex< T > **lambda** = 0
the contribution of the current aware
- std::complex< T > **mu** = 0
the contribution of the previous aware
- std::complex< T > **nu** = 0
the contribution of the 2nd previous aware
- std::vector< std::complex< T > > **x**
The previous x values.

6.25.1 Detailed Description

```
template<typename T>
struct SParamVFDataFrom< T >
```

Definition at line 19 of file [SParameterBlockVF.hpp](#).

6.25.2 Member Data Documentation

6.25.2.1 exp_alpha

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::exp_alpha
```

Definition at line 33 of file [SParameterBlockVF.hpp](#).

6.25.2.2 lambda

```
template<typename T >
std::complex<T> SParamVFDataFrom< T >::lambda = 0
```

the contribution of the current aware

Definition at line 36 of file [SParameterBlockVF.hpp](#).

6.25.2.3 lambda_p

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::lambda_p
```

the per-pole contribution of the current aware

Definition at line 27 of file [SParameterBlockVF.hpp](#).

6.25.2.4 mu

```
template<typename T >
std::complex<T> SParamVFDataFrom< T >::mu = 0
```

the contribution of the previous aware

Definition at line 38 of file [SParameterBlockVF.hpp](#).

6.25.2.5 mu_p

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::mu_p
```

the per-pole contribution of the previous aware

Definition at line 29 of file [SParameterBlockVF.hpp](#).

6.25.2.6 nu

```
template<typename T >
std::complex<T> SParamVFDataFrom< T >::nu = 0
```

the contribution of the 2nd previous aware

Definition at line 40 of file [SParameterBlockVF.hpp](#).

6.25.2.7 nu_p

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::nu_p
```

the per-pole contribution of the 2nd previous aware

Definition at line 31 of file [SParameterBlockVF.hpp](#).

6.25.2.8 numPoles

```
template<typename T >
size_t SParamVFDataFrom< T >::numPoles = 0
```

Definition at line 20 of file [SParameterBlockVF.hpp](#).

6.25.2.9 pole

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::pole
```

Definition at line 22 of file [SParameterBlockVF.hpp](#).

6.25.2.10 remainder

```
template<typename T >
std::complex<T> SParamVFDataFrom< T >::remainder = 0
```

Definition at line 24 of file [SParameterBlockVF.hpp](#).

6.25.2.11 residue

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::residue
```

Definition at line 23 of file [SParameterBlockVF.hpp](#).

6.25.2.12 x

```
template<typename T >
std::vector<std::complex<T> > SParamVFDataFrom< T >::x
```

The previous x values.

Definition at line 43 of file [SParameterBlockVF.hpp](#).

The documentation for this struct was generated from the following file:

- [SParameterBlockVF.hpp](#)

6.26 Stamp< T > Struct Template Reference

A helper struct to store the preallocated stamps for MNA.

```
#include <Component.hpp>
```

Public Member Functions

- **Stamp** (size_t _sizeG_A, size_t _sizeG_D)
Sets the initial size of the stamp pair.
- void **clear** ()
Clears the stamps to 0s.
- void **add** (const **Stamp**< T > &rhs)
Combines two stamps together into the current stamp. This is not done via the operator, as it has side-effects.
- void **addStaticStamp** (const std::shared_ptr< **Component**< T > > &rhs)
A helper function to add a static component to the stamp.
- void **addDynamicStamp** (const std::shared_ptr< **Component**< T > > &rhs, const **Matrix**< T > &solution←
Matrix, const size_t currentSolutionIndex, T timestep)
A helper function to add a dynamic component to the stamp.
- void **addNonLinearStamp** (const std::shared_ptr< **Component**< T > > &rhs, const **Matrix**< T > &solution←
Matrix, const size_t currentSolutionIndex, T timestep=0)
A helper function to add a non-linear component to the stamp.
- void **addDCAAnalysisStamp** (const std::shared_ptr< **Component**< T > > &rhs, const **Matrix**< T > &solutionMatrix,
const size_t numCurrents)
A helper function to add a DC component to the stamp.
- **Matrix**< T > **solve** ()
An alias for left dividing G by s.

Public Attributes

- size_t `sizeG_A`
- size_t `sizeG_D`
- `Matrix< T > G`
- `Matrix< T > s`

6.26.1 Detailed Description

```
template<typename T>
struct Stamp< T >
```

A helper struct to store the preallocated stamps for MNA.

Template Parameters

<code>T</code>	The value type
----------------	----------------

The matrix has the following structure:

```
///      | G_A | G_B |
/// G = -----|-----|
///          | G_C | G_D |
```

Definition at line 23 of file [Component.hpp](#).

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Stamp()

```
template<typename T >
Stamp< T >::Stamp (
    size_t _sizeG_A,
    size_t _sizeG_D ) [inline]
```

Sets the initial size of the stamp pair.

Parameters

<code>_sizeG_A</code>	Size of the voltage dependence portion of the stamps (Group I)
<code>_sizeG_D</code>	Size of the current dependence portion of the stamps (Group II)

Definition at line 36 of file [Component.hpp](#).

6.26.3 Member Function Documentation

6.26.3.1 add()

```
template<typename T >
void Stamp< T >::add (
    const Stamp< T > & rhs ) [inline]
```

Combines two stamps together into the current stamp. This is not done via the operator, as it has side-effects.

Parameters

<i>rhs</i>	The stamps to add to the current stamp
------------	--

Definition at line 52 of file [Component.hpp](#).

6.26.3.2 addDCAnalysisStamp()

```
template<typename T >
void Stamp< T >::addDCAnalysisStamp (
    const std::shared_ptr< Component< T > > & rhs,
    const Matrix< T > & solutionMatrix,
    const size_t numCurrents ) [inline]
```

A helper function to add a DC component to the stamp.

Parameters

<i>rhs</i>	Component to be added
------------	---------------------------------------

Definition at line 88 of file [Component.hpp](#).

6.26.3.3 addDynamicStamp()

```
template<typename T >
void Stamp< T >::addDynamicStamp (
    const std::shared_ptr< Component< T > > & rhs,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) [inline]
```

A helper function to add a dynamic component to the stamp.

Parameters

<i>rhs</i>	Component to be added
------------	---------------------------------------

Definition at line 67 of file [Component.hpp](#).

6.26.3.4 addNonLinearStamp()

```
template<typename T >
void Stamp< T >::addNonLinearStamp (
    const std::shared_ptr< Component< T > > & rhs,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep = 0 ) [inline]
```

A helper function to add a non-linear component to the stamp.

Parameters

<i>rhs</i>	Component to be added
------------	---------------------------------------

Definition at line 77 of file [Component.hpp](#).

6.26.3.5 addStaticStamp()

```
template<typename T >
void Stamp< T >::addStaticStamp (
    const std::shared_ptr< Component< T > > & rhs ) [inline]
```

A helper function to add a static component to the stamp.

Parameters

<i>rhs</i>	Component to be added
------------	---------------------------------------

Definition at line 60 of file [Component.hpp](#).

6.26.3.6 clear()

```
template<typename T >
void Stamp< T >::clear ( ) [inline]
```

Clears the stamps to 0s.

Definition at line 43 of file [Component.hpp](#).

6.26.3.7 solve()

```
template<typename T >
Matrix<T> Stamp< T >::solve ( ) [inline]
```

An alias for left dividing G by s.

Returns

a solution vector the same dimension as s.

Definition at line 96 of file [Component.hpp](#).

6.26.4 Member Data Documentation

6.26.4.1 G

```
template<typename T >
Matrix<T> Stamp< T >::G
```

Definition at line 27 of file [Component.hpp](#).

6.26.4.2 s

```
template<typename T >
Matrix<T> Stamp< T >::s
```

Definition at line 28 of file [Component.hpp](#).

6.26.4.3 sizeG_A

```
template<typename T >
size_t Stamp< T >::sizeG_A
```

Definition at line 24 of file [Component.hpp](#).

6.26.4.4 sizeG_D

```
template<typename T >
size_t Stamp< T >::sizeG_D
```

Definition at line 25 of file [Component.hpp](#).

The documentation for this struct was generated from the following file:

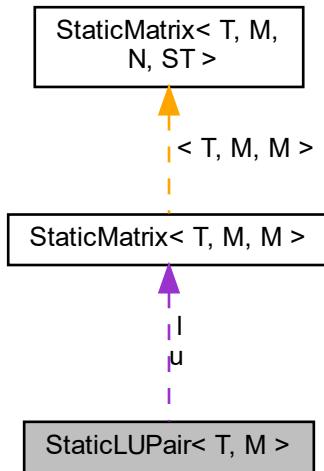
- [Component.hpp](#)

6.27 StaticLUPair< T, M > Struct Template Reference

A compile-time sized L U and pivot grouping.

```
#include <StaticMatrix.hpp>
```

Collaboration diagram for StaticLUPair< T, M >:



Public Member Functions

- std::string [toString \(\)](#)

Public Attributes

- [StaticMatrix< T, M, M > l](#)
- [StaticMatrix< T, M, M > u](#)
- std::array< size_t, M > p

6.27.1 Detailed Description

```
template<typename T, size_t M>
struct StaticLUPair< T, M >
```

A compile-time sized L U and pivot grouping.

Template Parameters

<i>T</i>	The value type
<i>M</i>	The L and U sizes

Definition at line 342 of file [StaticMatrix.hpp](#).

6.27.2 Member Function Documentation

6.27.2.1 `toString()`

```
template<typename T , size_t M>
std::string StaticLUPair< T, M >::toString ( ) [inline]
```

Definition at line 347 of file [StaticMatrix.hpp](#).

Here is the call graph for this function:



6.27.3 Member Data Documentation

6.27.3.1 `l`

```
template<typename T , size_t M>
StaticMatrix<T, M, M> StaticLUPair< T, M >::l
```

Definition at line 343 of file [StaticMatrix.hpp](#).

6.27.3.2 `p`

```
template<typename T , size_t M>
std::array<size_t, M> StaticLUPair< T, M >::p
```

Definition at line 345 of file [StaticMatrix.hpp](#).

6.27.3.3 u

```
template<typename T , size_t M>
StaticMatrix<T, M, M> StaticLUPair< T, M >::u
```

Definition at line 344 of file [StaticMatrix.hpp](#).

The documentation for this struct was generated from the following file:

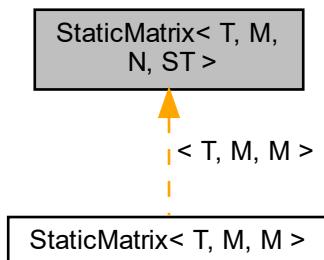
- [StaticMatrix.hpp](#)

6.28 StaticMatrix< T, M, N, ST > Struct Template Reference

A compile-time sized matrix.

```
#include <StaticMatrix.hpp>
```

Inheritance diagram for StaticMatrix< T, M, N, ST >:



Public Member Functions

- [StaticMatrix \(\)](#)
- [StaticMatrix \(T initialValue\)](#)
- void [fill \(T fillVal\)](#)
- [StaticRow< T, N > & operator\[\] \(size_t index\)](#)
- [StaticRow< T, N > operator\[\] \(size_t index\) const](#)
- void [rowAddition \(size_t destinationRow, size_t sourceRow, T scalingFactor\)](#)
- void [swapRows \(size_t row1, size_t row2\)](#)
- [StaticMatrix< T, N, M > transpose \(\)](#)
- template<size_t N2>
 - [StaticMatrix< T, M, N2 > multiply \(const StaticMatrix< T, N, N2 > &rhs\) const](#)
- template<size_t N2>
 - void [multiply \(const StaticMatrix< T, N, N2 > &rhs, StaticMatrix< T, M, N2 > &dest\) const](#)
- [StaticMatrix< T, M, N > add \(const StaticMatrix< T, M, N > &rhs\) const](#)
- void [add \(const StaticMatrix< T, M, N > &rhs, StaticMatrix< T, M, N > &dest\) const](#)
- [StaticMatrix< T, M, N > subtract \(const StaticMatrix< T, N, N > &rhs\) const](#)
- void [subtract \(const StaticMatrix< T, N, N > &rhs, StaticMatrix< T, M, N > &dest\) const](#)
- std::string [toString \(\) const](#)
- [StaticLUPair< T, M > luPair \(\) const](#)
- void [luPair \(StaticLUPair< T, M > &dest\) const](#)
- [StaticMatrix< T, M, 1 > leftDivide \(const StaticMatrix< T, M, 1 > &rhs\) const](#)
- void [leftDivide \(const StaticMatrix< T, M, 1 > &rhs, const StaticLUPair< T, M > &lu, StaticMatrix< T, M, 1 > &scratchSpace, StaticMatrix< T, M, 1 > &dest\) const](#)

Public Attributes

- ST [rows](#)

Static Public Attributes

- static constexpr size_t [sizeM](#) = M
- static constexpr size_t [sizeN](#) = N

6.28.1 Detailed Description

```
template<typename T, size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
struct StaticMatrix< T, M, N, ST >
```

A compile-time sized matrix.

Template Parameters

<i>T</i>	The value type
<i>M</i>	The matrix column length
<i>N</i>	The row length
<i>ST</i>	the type of storage used (vector vs array)

Definition at line 108 of file [StaticMatrix.hpp](#).

6.28.2 Constructor & Destructor Documentation

6.28.2.1 StaticMatrix() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix< T, M, N, ST >::StaticMatrix ( ) [inline]
```

Definition at line 111 of file [StaticMatrix.hpp](#).

6.28.2.2 StaticMatrix() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix< T, M, N, ST >::StaticMatrix (
    T initialValue ) [inline]
```

Definition at line 117 of file [StaticMatrix.hpp](#).

Here is the call graph for this function:



6.28.3 Member Function Documentation

6.28.3.1 add() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix<T, M, N> StaticMatrix< T, M, N, ST >::add (
    const StaticMatrix< T, M, N > & rhs ) const [inline]
```

Definition at line 210 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:



6.28.3.2 add() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::add (
    const StaticMatrix< T, M, N > & rhs,
    StaticMatrix< T, M, N > & dest ) const [inline]
```

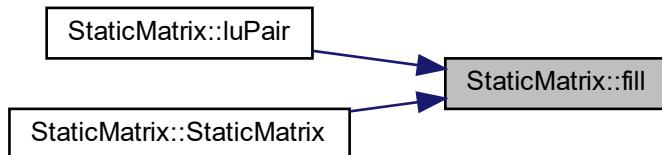
Definition at line 216 of file [StaticMatrix.hpp](#).

6.28.3.3 fill()

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::fill (
    T fillVal ) [inline]
```

Definition at line 145 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:



6.28.3.4 leftDivide() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix<T, M, 1> StaticMatrix< T, M, N, ST >::leftDivide (
    const StaticMatrix< T, M, 1 > & rhs ) const [inline]
```

Definition at line 302 of file [StaticMatrix.hpp](#).

Here is the call graph for this function:



6.28.3.5 leftDivide() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::leftDivide (
    const StaticMatrix< T, M, 1 > & rhs,
    const StaticLUPair< T, M > & lu,
    StaticMatrix< T, M, 1 > & scratchSpace,
    StaticMatrix< T, M, 1 > & dest ) const [inline]
```

Definition at line 310 of file [StaticMatrix.hpp](#).

6.28.3.6 luPair() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticLUPair<T, M> StaticMatrix< T, M, N, ST >::luPair ( ) const [inline]
```

Definition at line 252 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:

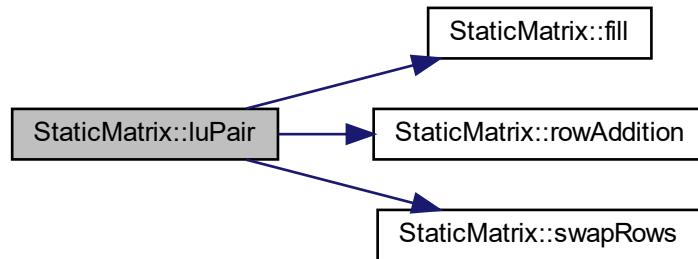


6.28.3.7 luPair() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::luPair (
    StaticLUPair< T, M > & dest ) const [inline]
```

Definition at line 260 of file [StaticMatrix.hpp](#).

Here is the call graph for this function:



6.28.3.8 multiply() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
template<size_t N2>
StaticMatrix<T, M, N2> StaticMatrix< T, M, N, ST >::multiply (
    const StaticMatrix< T, N, N2 > & rhs ) const [inline]
```

Definition at line 182 of file [StaticMatrix.hpp](#).

6.28.3.9 multiply() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
template<size_t N2>
void StaticMatrix< T, M, N, ST >::multiply (
    const StaticMatrix< T, N, N2 > & rhs,
    StaticMatrix< T, M, N2 > & dest ) const [inline]
```

Definition at line 189 of file [StaticMatrix.hpp](#).

6.28.3.10 operator[]() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticRow<T, N>& StaticMatrix< T, M, N, ST >::operator[] (
    size_t index ) [inline]
```

Definition at line 151 of file [StaticMatrix.hpp](#).

6.28.3.11 operator[]() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticRow<T, N> StaticMatrix< T, M, N, ST >::operator[] (
    size_t index ) const [inline]
```

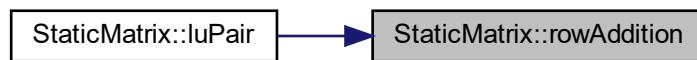
Definition at line 155 of file [StaticMatrix.hpp](#).

6.28.3.12 rowAddition()

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::rowAddition (
    size_t destinationRow,
    size_t sourceRow,
    T scalingFactor ) [inline]
```

Definition at line 159 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:



6.28.3.13 subtract() [1/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix<T, M, N> StaticMatrix< T, M, N, ST >::subtract (
    const StaticMatrix< T, N, N > & rhs ) const [inline]
```

Definition at line 224 of file [StaticMatrix.hpp](#).

Here is the call graph for this function:



6.28.3.14 subtract() [2/2]

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::subtract (
    const StaticMatrix< T, N, N > & rhs,
    StaticMatrix< T, M, N > & dest ) const [inline]
```

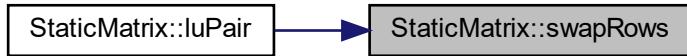
Definition at line 231 of file [StaticMatrix.hpp](#).

6.28.3.15 swapRows()

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
void StaticMatrix< T, M, N, ST >::swapRows (
    size_t row1,
    size_t row2 ) [inline]
```

Definition at line 167 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:

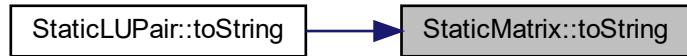


6.28.3.16 toString()

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
std::string StaticMatrix< T, M, N, ST >::toString () const [inline]
```

Definition at line 239 of file [StaticMatrix.hpp](#).

Here is the caller graph for this function:



6.28.3.17 transpose()

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
StaticMatrix<T, N, M> StaticMatrix< T, M, N, ST >::transpose () [inline]
```

Definition at line 171 of file [StaticMatrix.hpp](#).

6.28.4 Member Data Documentation

6.28.4.1 rows

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
ST StaticMatrix< T, M, N, ST >::rows
```

Definition at line 109 of file [StaticMatrix.hpp](#).

6.28.4.2 sizeM

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
constexpr size_t StaticMatrix< T, M, N, ST >::sizeM = M [static], [constexpr]
```

Definition at line 142 of file [StaticMatrix.hpp](#).

6.28.4.3 sizeN

```
template<typename T , size_t M, size_t N = M, typename ST = storageType<StaticRow<T, N>, M>>
constexpr size_t StaticMatrix< T, M, N, ST >::sizeN = N [static], [constexpr]
```

Definition at line 143 of file [StaticMatrix.hpp](#).

The documentation for this struct was generated from the following file:

- [StaticMatrix.hpp](#)

6.29 StaticRow< T, N, ST > Struct Template Reference

A compile-time sized matrix row.

```
#include <StaticMatrix.hpp>
```

Public Member Functions

- [StaticRow \(\)](#)
- [StaticRow \(T initialValue\)](#)
- void [fill \(T value\)](#)
- T & [operator\[\] \(size_t index\)](#)
- const T & [operator\[\] \(size_t index\) const](#)
- T [dot \(StaticRow< T, N > other\)](#)

Public Attributes

- ST [columns](#)

Static Public Attributes

- static constexpr size_t [sizeN](#) = N

6.29.1 Detailed Description

```
template<typename T, size_t N, typename ST = storageType<T, N>>
struct StaticRow< T, N, ST >
```

A compile-time sized matrix row.

Template Parameters

<i>T</i>	The value type
<i>N</i>	The row length
<i>ST</i>	the type of storage used (vector vs array)

Definition at line [58](#) of file [StaticMatrix.hpp](#).

6.29.2 Constructor & Destructor Documentation

6.29.2.1 StaticRow() [1/2]

```
template<typename T , size_t N, typename ST = storageType<T, N>>
StaticRow< T, N, ST >::StaticRow ( ) [inline]
```

Definition at line [63](#) of file [StaticMatrix.hpp](#).

6.29.2.2 StaticRow() [2/2]

```
template<typename T , size_t N, typename ST = storageType<T, N>>
StaticRow< T, N, ST >::StaticRow (
    T initialValue ) [inline]
```

Definition at line [69](#) of file [StaticMatrix.hpp](#).

6.29.3 Member Function Documentation

6.29.3.1 dot()

```
template<typename T , size_t N, typename ST = storageType<T, N>>
T StaticRow< T, N, ST >::dot (
    StaticRow< T, N > other ) [inline]
```

Definition at line 88 of file [StaticMatrix.hpp](#).

6.29.3.2 fill()

```
template<typename T , size_t N, typename ST = storageType<T, N>>
void StaticRow< T, N, ST >::fill (
    T value ) [inline]
```

Definition at line 76 of file [StaticMatrix.hpp](#).

6.29.3.3 operator[]() [1/2]

```
template<typename T , size_t N, typename ST = storageType<T, N>>
T& StaticRow< T, N, ST >::operator[] (
    size_t index ) [inline]
```

Definition at line 80 of file [StaticMatrix.hpp](#).

6.29.3.4 operator[]() [2/2]

```
template<typename T , size_t N, typename ST = storageType<T, N>>
const T& StaticRow< T, N, ST >::operator[] (
    size_t index ) const [inline]
```

Definition at line 84 of file [StaticMatrix.hpp](#).

6.29.4 Member Data Documentation

6.29.4.1 columns

```
template<typename T , size_t N, typename ST = storageType<T, N>>
ST StaticRow< T, N, ST >::columns
```

Definition at line 59 of file [StaticMatrix.hpp](#).

6.29.4.2 sizeN

```
template<typename T , size_t N, typename ST = storageType<T, N>>
constexpr size_t StaticRow< T, N, ST >::sizeN = N [static], [constexpr]
```

Definition at line 61 of file [StaticMatrix.hpp](#).

The documentation for this struct was generated from the following file:

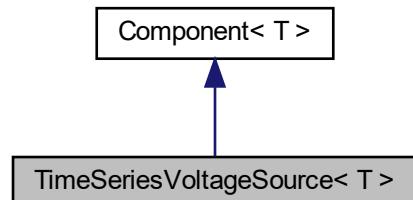
- [StaticMatrix.hpp](#)

6.30 TimeSeriesVoltageSource< T > Struct Template Reference

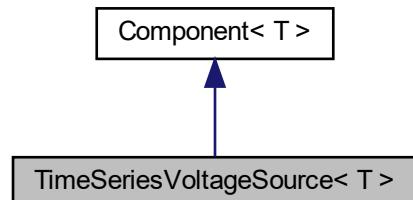
A time series voltage source model.

```
#include <TimeSeriesVoltageSource.hpp>
```

Inheritance diagram for TimeSeriesVoltageSource< T >:



Collaboration diagram for TimeSeriesVoltageSource< T >:



Public Member Functions

- T `lerp` (size_t lowIndex, T timeVal) const
- void `addDynamicStampTo` (Stamp< T > &stamp, const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep) const

Adds this component's dynamic stamp to the target stamp.
- void `updateStoredState` (const Matrix< T > &solutionMatrix, const size_t currentSolutionIndex, T timestep, size_t sizeG_A)

Updates any stored state based on the current solution index.
- void `addDCAAnalysisStampTo` (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const

adds this component's DC stamp to the target stamp.
- void `readInTimeSeries` (T timescale, const std::string &seriesPath)

Static Public Member Functions

- static void `addToElements` (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)

Public Attributes

- size_t `n1` = 0
- size_t `n2` = 0
- size_t `currentIndex` = 0
- size_t `lastTimeSeriesIndex` = 0
- std::vector< T > `timeSeries`
- std::vector< T > `dataSeries`

6.30.1 Detailed Description

```
template<typename T>
struct TimeSeriesVoltageSource< T >
```

A time series voltage source model.

Template Parameters

<code>T</code>	the value type
----------------	----------------

Definition at line 11 of file [TimeSeriesVoltageSource.hpp](#).

6.30.2 Member Function Documentation

6.30.2.1 addDCAnalysisStampTo()

```
template<typename T >
void TimeSeriesVoltageSource< T >::addDCAnalysisStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionVector,
    size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

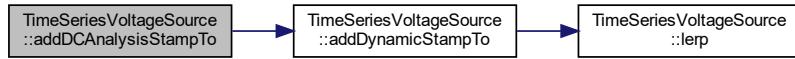
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 67 of file [TimeSeriesVoltageSource.hpp](#).

Here is the call graph for this function:



6.30.2.2 addDynamicStampTo()

```
template<typename T >
void TimeSeriesVoltageSource< T >::addDynamicStampTo (
    Stamp< T > & destination,
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep ) const [inline], [virtual]
```

Adds this component's dynamic stamp to the target stamp.

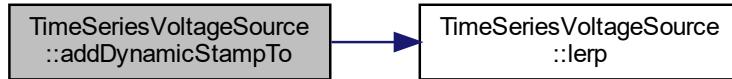
Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step

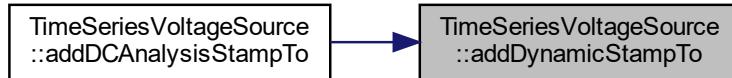
Reimplemented from [Component< T >](#).

Definition at line 29 of file [TimeSeriesVoltageSource.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



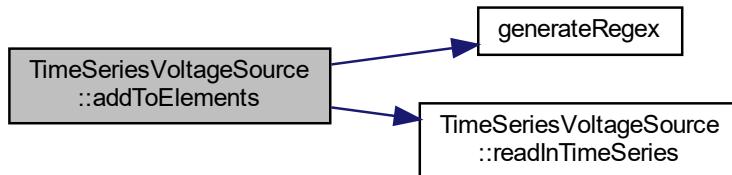
6.30.2.3 addToElements()

```

template<typename T >
static void TimeSeriesVoltageSource< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
  
```

Definition at line 73 of file [TimeSeriesVoltageSource.hpp](#).

Here is the call graph for this function:

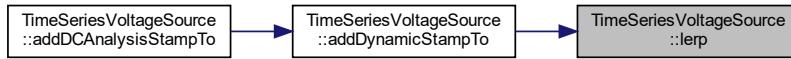


6.30.2.4 lerp()

```
template<typename T >
T TimeSeriesVoltageSource< T >::lerp (
    size_t lowIndex,
    T timeVal ) const [inline]
```

Definition at line 20 of file [TimeSeriesVoltageSource.hpp](#).

Here is the caller graph for this function:

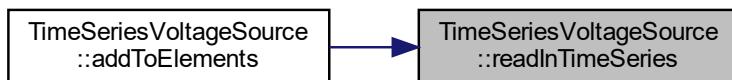


6.30.2.5 readInTimeSeries()

```
template<typename T >
void TimeSeriesVoltageSource< T >::readInTimeSeries (
    T timescale,
    const std::string & seriesPath ) [inline]
```

Definition at line 117 of file [TimeSeriesVoltageSource.hpp](#).

Here is the caller graph for this function:



6.30.2.6 updateStoredState()

```
template<typename T >
void TimeSeriesVoltageSource< T >::updateStoredState (
    const Matrix< T > & solutionMatrix,
    const size_t currentSolutionIndex,
    T timestep,
    size_t numCurrents ) [inline], [virtual]
```

Updates any stored state based on the current solution index.

Parameters

<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>currentSolutionIndex</i>	The current timeStep index
<i>timestep</i>	The length of each time step
<i>sizeG_A</i>	the size of the A portion of G, marks the end of the equiv currents

Reimplemented from [Component< T >](#).

Definition at line 55 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3 Member Data Documentation

6.30.3.1 currentIndex

```
template<typename T >
size_t TimeSeriesVoltageSource< T >::currentIndex = 0
```

Definition at line 14 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3.2 dataSeries

```
template<typename T >
std::vector<T> TimeSeriesVoltageSource< T >::dataSeries
```

Definition at line 18 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3.3 lastTimeSeriesIndex

```
template<typename T >
size_t TimeSeriesVoltageSource< T >::lastTimeSeriesIndex = 0
```

Definition at line 15 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3.4 n1

```
template<typename T >
size_t TimeSeriesVoltageSource< T >::n1 = 0
```

Definition at line 12 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3.5 n2

```
template<typename T >
size_t TimeSeriesVoltageSource< T >::n2 = 0
```

Definition at line 13 of file [TimeSeriesVoltageSource.hpp](#).

6.30.3.6 timeSeries

```
template<typename T >
std::vector<T> TimeSeriesVoltageSource< T >::timeSeries
```

Definition at line 17 of file [TimeSeriesVoltageSource.hpp](#).

The documentation for this struct was generated from the following file:

- [TimeSeriesVoltageSource.hpp](#)

6.31 TransistorTestResult< T > Struct Template Reference

Public Attributes

- T [var](#)
- T [diff1](#)
- T [diff2](#)

6.31.1 Detailed Description

```
template<typename T>
struct TransistorTestResult< T >
```

Definition at line 203 of file [AutoDiffTest.cpp](#).

6.31.2 Member Data Documentation

6.31.2.1 diff1

```
template<typename T >
T TransistorTestResult< T >::diff1
```

Definition at line 205 of file [AutoDiffTest.cpp](#).

6.31.2.2 diff2

```
template<typename T >
T TransistorTestResult< T >::diff2
```

Definition at line 206 of file [AutoDiffTest.cpp](#).

6.31.2.3 var

```
template<typename T >
T TransistorTestResult< T >::var
```

Definition at line 204 of file [AutoDiffTest.cpp](#).

The documentation for this struct was generated from the following file:

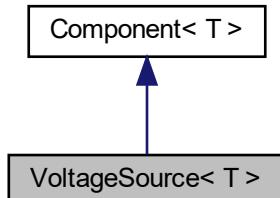
- [AutoDiffTest.cpp](#)

6.32 VoltageSource< T > Struct Template Reference

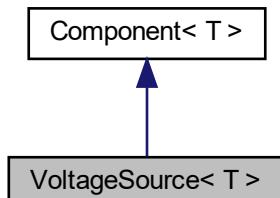
An ideal voltageSource.

```
#include <VoltageSource.hpp>
```

Inheritance diagram for VoltageSource< T >:



Collaboration diagram for VoltageSource< T >:



Public Member Functions

- void `addStaticStampTo (Stamp< T > &stamp) const`
Adds this component's static stamp to the target stamp.
- void `addDCAnalysisStampTo (Stamp< T > &stamp, const Matrix< T > &solutionVector, size_t numCurrents) const`
adds this component's DC stamp to the target stamp.

Static Public Member Functions

- static void `addToElements (const std::string &line, CircuitElements< T > &elements, size_t &numNodes, size_t &numCurrents, size_t &numDCCurrents)`

Public Attributes

- T `value` = 0
- size_t `n1` = 0
- size_t `n2` = 0
- size_t `currentIndex` = 0

6.32.1 Detailed Description

```
template<typename T>
struct VoltageSource< T >
```

An ideal voltageSource.

Template Parameters

<code>T</code>	The value type
----------------	----------------

Definition at line 10 of file [VoltageSource.hpp](#).

6.32.2 Member Function Documentation

6.32.2.1 addDCAnalysisStampTo()

```
template<typename T >
void VoltageSource< T >::addDCAnalysisStampTo (
Stamp< T > & destination,
const Matrix< T > & solutionVector,
size_t numCurrents ) const [inline], [virtual]
```

adds this component's DC stamp to the target stamp.

Parameters

<i>destination</i>	The stamp to be added to.
<i>solutionMatrix</i>	A vector containing all past solutions to the circuit
<i>numCurrents</i>	The number of currents used by the transient simulation

Reimplemented from [Component< T >](#).

Definition at line 36 of file [VoltageSource.hpp](#).

Here is the call graph for this function:

**6.32.2.2 addStaticStampTo()**

```
template<typename T >
void VoltageSource< T >::addStaticStampTo (
    Stamp< T > & destination ) const [inline], [virtual]
```

Adds this component's static stamp to the target stamp.

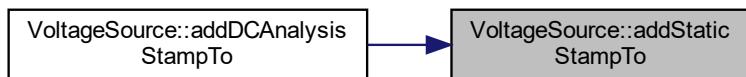
Parameters

<i>destination</i>	The stamp to be added to.
--------------------	---------------------------

Reimplemented from [Component< T >](#).

Definition at line 18 of file [VoltageSource.hpp](#).

Here is the caller graph for this function:



6.32.2.3 addToElements()

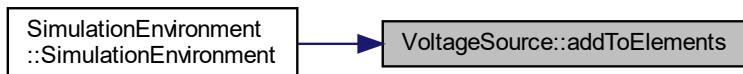
```
template<typename T>
static void VoltageSource< T >::addToElements (
    const std::string & line,
    CircuitElements< T > & elements,
    size_t & numNodes,
    size_t & numCurrents,
    size_t & numDCCurrents ) [inline], [static]
```

Definition at line 42 of file [VoltageSource.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.32.3 Member Data Documentation

6.32.3.1 currentIndex

```
template<typename T>
size_t VoltageSource< T >::currentIndex = 0
```

Definition at line 16 of file [VoltageSource.hpp](#).

6.32.3.2 n1

```
template<typename T >
size_t VoltageSource< T >::n1 = 0
```

Definition at line 14 of file [VoltageSource.hpp](#).

6.32.3.3 n2

```
template<typename T >
size_t VoltageSource< T >::n2 = 0
```

Definition at line 15 of file [VoltageSource.hpp](#).

6.32.3.4 value

```
template<typename T >
T VoltageSource< T >::value = 0
```

Definition at line 12 of file [VoltageSource.hpp](#).

The documentation for this struct was generated from the following file:

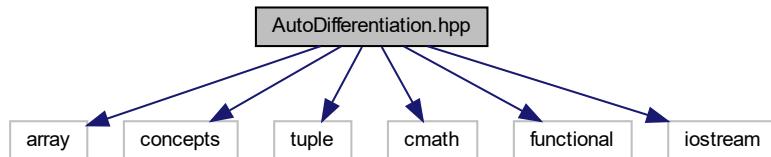
- [VoltageSource.hpp](#)

Chapter 7

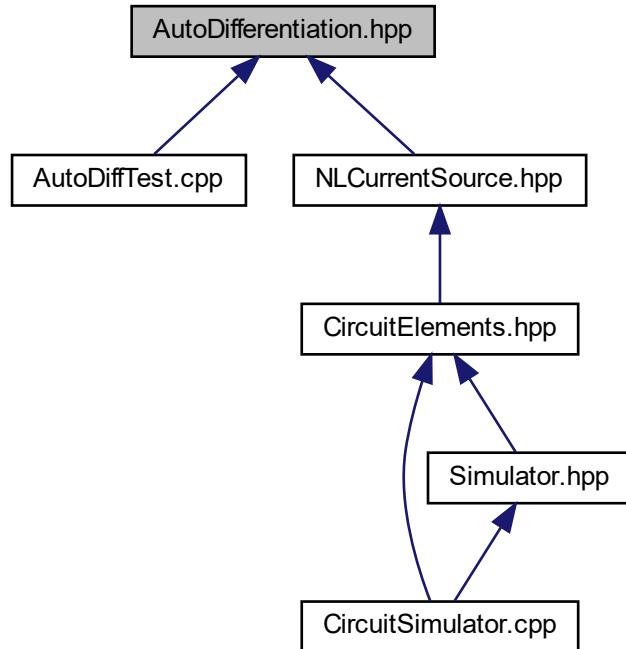
File Documentation

7.1 AutoDifferentiation.hpp File Reference

```
#include <array>
#include <concepts>
#include <tuple>
#include <cmath>
#include <functional>
#include <iostream>
Include dependency graph for AutoDifferentiation.hpp:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [AutoDifferentiation](#)
a namespace to hold the messiness of my auto-differentiator

Functions

- template<typename ValType , size_t NumVars>
[AutoDifferentiation::requires](#) (std::is_arithmetic< ValType >::value &&NumVars >=0) struct DiffVar
- template<typename ValType , size_t NumVars, typename F1 , typename F2 >
[AutoDifferentiation::requires](#) (std::is_arithmetic< ValType >::value &&NumVars >=0) const expr auto diff← Func(const DiffVar< ValType >
- [AutoDifferentiation::while](#) (it1 !=toRet.diffVars.end() &&it2 !=arg.diffVars.end())
- DiffVar< ValType, NumVars > [AutoDifferentiation::toRet](#) (derivEval)
- template<typename ValType , size_t NumVars, typename OT >
[AutoDifferentiation::requires](#) (std::is_arithmetic< ValType >::value &&NumVars >=0) &&(lstd
- DiffVar< ValType, NumVars > [AutoDifferentiation::toRet](#) (func(arg.var, exponent.var))
- DiffVar< ValType, NumVars > [AutoDifferentiation::toRet](#) (funcResult)

Variables

- template<typename RT , typename LT , typename OT >
concept **AutoDifferentiation::MultipliableResult**
- template<typename RT , typename LT , typename OT >
concept **AutoDifferentiation::AddableResult**
- template<typename RT , typename LT , typename OT >
concept **AutoDifferentiation::SubtractableResult**
- template<typename RT , typename LT , typename OT >
concept **AutoDifferentiation::DivisibleResult**
- NumVars & **AutoDifferentiation::arg**
- NumVars F1 **AutoDifferentiation::func**
- NumVars F1 F2 **AutoDifferentiation::deriv**
- auto **AutoDifferentiation::it1** = toRet.diffVars.begin()
- auto **AutoDifferentiation::it2** = arg.diffVars.begin()
- auto **AutoDifferentiation::derivEval** = deriv(arg.var)
- return **AutoDifferentiation::toRet**
- toRet **AutoDifferentiation::var** = derivEval
- NumVars DiffVar< ValType, NumVars > **AutoDifferentiation::exponent**
- auto **AutoDifferentiation::it3** = exponent.diffVars.begin()

7.2 AutoDifferentiation.hpp

```

00001 #ifndef _AUTODIFFERENTIATION_HPP_INC_
00002 #define _AUTODIFFERENTIATION_HPP_INC_
00003 #include <array>
00004 #include <concepts>
00005 #include <tuple>
00006 #include <cmath>
00007 #include <functional>
00008 #include <iostream>
00009
00017 namespace AutoDifferentiation {
00018
00019 template<typename RT, typename LT, typename OT>
00020 concept MultipliableResult = requires(LT lhs, OT rhs) {
00021     std::is_same<RT, decltype(lhs *= rhs)>::value;
00022 };
00023
00024 template<typename RT, typename LT, typename OT>
00025 concept AddableResult = requires(LT lhs, OT rhs) {
00026     std::is_same<RT, decltype(lhs += rhs)>::value;
00027 };
00028
00029 template<typename RT, typename LT, typename OT>
00030 concept SubtractableResult = requires(LT lhs, OT rhs) {
00031     std::is_same<RT, decltype(lhs -= rhs)>::value;
00032 };
00033
00034 template<typename RT, typename LT, typename OT>
00035 concept DivisibleResult = requires(LT lhs, OT rhs) {
00036     std::is_same<RT, decltype(lhs /= rhs)>::value;
00037 };
00038
00039
00040 template<typename ValType, size_t NumVars>
00041 requires(std::is_arithmetic<ValType>::value && NumVars >= 0) struct DiffVar {
00042     using ThisDiff = DiffVar<ValType, NumVars>;
00043
00044     ValType var = 0;
00045     std::array<ValType, NumVars> diffVars = {0};
00046
00047     template<typename... VariadicType>
00048     requires(... && std::is_convertible<VariadicType, ValType>::
00049             value) constexpr explicit DiffVar(ValType _var,
00050                                         VariadicType... _diffVars)
00051         : var(_var), diffVars {{static_cast<ValType>(_diffVars)...}} {
00052     }
00053
00054     constexpr DiffVar(ValType _var, std::array<ValType, NumVars> _diffVars)
00055         : var(_var), diffVars(_diffVars) {
00056     }

```

```

00057
00058     template<typename OtherValType>
00059     requires(std::is_convertible<OtherValType, ValType>::value)
00060             value) constexpr explicit DiffVar(const OtherValType & other) {
00061         var = static_cast<ValType>(other);
00062     }
00063
00064     constexpr explicit operator ValType() const {
00065         return var;
00066     }
00067
00068     constexpr ValType & operator[](size_t index) {
00069         if (index == 0) {
00070             return var;
00071         } else {
00072             return diffVars[index - 1];
00073         }
00074     }
00075
00076     constexpr const ValType & operator[](size_t index) const {
00077         if (index == 0) {
00078             return var;
00079         } else {
00080             return diffVars[index - 1];
00081         }
00082     }
00083
00084     template<typename OtherValType>
00085     requires(std::is_convertible<OtherValType, ValType>::value) constexpr int
00086     operator<=>(OtherValType & rhs) {
00087         return var <=> static_cast<ValType>(rhs);
00088     }
00089
00090     constexpr ThisDiff operator+=(const ThisDiff & rhs) {
00091         var += rhs.var;
00092         auto it1 = diffVars.begin();
00093         auto it2 = rhs.diffVars.begin();
00094         while (it1 != diffVars.end() && it2 != rhs.diffVars.end()) {
00095             *it1 += *it2;
00096             ++it1;
00097             ++it2;
00098         }
00099         return *this;
00100     }
00101
00102     template<typename OtherValType>
00103     requires(std::is_convertible<OtherValType, ValType>::value) constexpr ThisDiff
00104     operator+=(const OtherValType & rhs) {
00105         var += static_cast<ValType>(rhs);
00106         return *this;
00107     }
00108
00109     constexpr ThisDiff operator==(const ThisDiff & rhs) {
00110         var -= rhs.var;
00111         auto it1 = diffVars.begin();
00112         auto it2 = rhs.diffVars.begin();
00113         while (it1 != diffVars.end() && it2 != rhs.diffVars.end()) {
00114             *it1 -= *it2;
00115             ++it1;
00116             ++it2;
00117         }
00118         return *this;
00119     }
00120
00121     template<typename OtherValType>
00122     requires(std::is_convertible<OtherValType, ValType>::value) constexpr ThisDiff
00123     operator-=(const OtherValType & rhs) {
00124         var -= static_cast<ValType>(rhs);
00125         return *this;
00126     }
00127
00128     constexpr ThisDiff operator-() const {
00129         auto toRet = *this;
00130         toRet.var = -var;
00131         auto it1 = toRet.diffVars.begin();
00132         while (it1 != toRet.diffVars.end()) {
00133             *it1 = -*it1;
00134             ++it1;
00135         }
00136         return toRet;
00137     }
00138
00139
00140     constexpr ThisDiff operator*=(const ThisDiff & rhs) {
00141         auto it1 = diffVars.begin();
00142         auto it2 = rhs.diffVars.begin();
00143         while (it1 != diffVars.end() && it2 != rhs.diffVars.end()) {

```

```

00144         *it1 = (rhs.var * *it1 + var * *it2);
00145         ++it1;
00146         ++it2;
00147     }
00148     var *= rhs.var;
00149     return *this;
00150 }
00151
00152 template<typename OtherValType>
00153 requires(std::is_convertible<OtherValType, ValType>::value) constexpr ThisDiff
00154 operator*=(const OtherValType & rhs) {
00155     auto it1 = diffVars.begin();
00156     while (it1 != diffVars.end()) {
00157         *it1 = static_cast<ValType>(rhs) * *it1;
00158         ++it1;
00159     }
00160     var *= static_cast<ValType>(rhs);
00161     return *this;
00162 }
00163
00164 constexpr ThisDiff operator/=(const ThisDiff & rhs) {
00165     auto it1 = diffVars.begin();
00166     auto it2 = rhs.diffVars.begin();
00167     while (it1 != diffVars.end() && it2 != rhs.diffVars.end()) {
00168         *it1 = (rhs.var * *it1 - var * *it2) / (rhs.var * rhs.var);
00169         ++it1;
00170         ++it2;
00171     }
00172     var /= rhs.var;
00173     return *this;
00174 }
00175
00176 template<typename OtherValType>
00177 requires(std::is_convertible<OtherValType, ValType>::value) constexpr ThisDiff
00178 operator/=(const OtherValType & rhs) {
00179     auto it1 = diffVars.begin();
00180     while (it1 != diffVars.end()) {
00181         *it1 = *it1 / static_cast<ValType>(rhs);
00182         ++it1;
00183     }
00184     var /= static_cast<ValType>(rhs);
00185     return *this;
00186 }
00187
00188 // Friends
00189 // Binary Operators
00190 // Addition
00191 template<typename LT, typename RT>
00192 requires AddableResult<ThisDiff, LT, RT> constexpr friend auto
00193 operator+(LT lhs, const RT & rhs) {
00194     lhs += rhs;
00195     return lhs;
00196 }
00197
00198 template<typename LT, typename RT>
00199     requires AddableResult<ThisDiff, RT, LT> &&
00200     (!AddableResult<ThisDiff, LT, RT>) constexpr friend auto
00201 operator+(const LT & lhs, RT rhs) {
00202     rhs += lhs;
00203     return rhs;
00204 }
00205
00206 // Subtraction
00207 template<typename LT, typename RT>
00208 requires SubtractableResult<ThisDiff, LT, RT> constexpr friend auto
00209 operator-(LT lhs, const RT & rhs) {
00210     lhs -= rhs;
00211     return lhs;
00212 }
00213
00214 template<typename LT, typename RT>
00215     requires SubtractableResult<ThisDiff, RT, LT> &&
00216     (!SubtractableResult<ThisDiff, LT, RT>) constexpr friend auto
00217 operator-(<b>const LT & lhs, RT rhs) {
00218     rhs -= lhs;
00219     return -rhs;
00220 }
00221
00222 template<typename LT, typename RT>
00223     requires(!SubtractableResult<ThisDiff, LT, RT>) &&
00224     AddableResult<ThisDiff, LT, RT> constexpr friend auto
00225 operator-(<b>LT lhs, const RT & rhs) {
00226     lhs += -rhs;
00227     return lhs;
00228 }
00229
00230 template<typename LT, typename RT>

```

```

00231     requires (!SubtractableResult<ThisDiff, RT, LT>) &&
00232     AddableResult<ThisDiff, RT, LT> &&
00233     (!AddableResult<ThisDiff, LT, RT>) constexpr friend auto
00234     operator-(const LT & lhs, RT rhs) {
00235     rhs += -lhs;
00236     return -rhs;
00237 }
00238
00239 // Multiplication
00240 template<typename LT, typename RT>
00241     requires MultipliableResult<ThisDiff, LT, RT> constexpr friend auto
00242     operator*(LT lhs, const RT & rhs) {
00243     lhs *= rhs;
00244     return lhs;
00245 }
00246
00247 template<typename LT, typename RT>
00248     requires MultipliableResult<ThisDiff, RT, LT> &&
00249     (!MultipliableResult<ThisDiff, LT, RT>) constexpr friend auto
00250     operator*(const LT & lhs, RT rhs) {
00251     rhs *= lhs;
00252     return rhs;
00253 }
00254
00255 // Division
00256 template<typename OT>
00257     requires (!DivisibleResult<ThisDiff, OT, ThisDiff>) &&
00258     (std::is_convertible<OT, ValType>::value) constexpr friend auto
00259     operator/(const OT & lhs, const ThisDiff & rhs) {
00260     auto lhsDiff = static_cast<ThisDiff>(lhs);
00261     lhsDiff /= rhs;
00262     return lhsDiff;
00263 }
00264
00265 template<typename OT>
00266     requires(std::is_convertible<OT, ValType>::value) constexpr friend auto
00267     operator/(<ThisDiff lhs, const OT & rhs) {
00268     lhs /= rhs;
00269     return lhs;
00270 }
00271
00272 constexpr friend std::ostream &
00273     operator<<(std::ostream & output, const ThisDiff & v) {
00274     output << "Value: " << v.var;
00275     size_t n = 0;
00276     for (auto diffVar : v.diffVars) {
00277         output << std::endl << "Derivative " << n++ << ":" << diffVar;
00278     }
00279
00280     return output;
00281 }
00282 };
00283
00284
00285 template<typename ValType, size_t NumVars, typename F1, typename F2>
00286     requires(std::is_arithmetic<ValType>::value && NumVars >=
00287     0) constexpr auto diffFunc(const DiffVar<ValType, NumVars> & arg, F1 func,
00288                             F2 deriv) {
00289     DiffVar<ValType, NumVars> toRet(func(arg.var));
00290
00291     auto it1 = toRet.diffVars.begin();
00292     auto it2 = arg.diffVars.begin();
00293     auto derivEval = deriv(arg.var);
00294     while (it1 != toRet.diffVars.end() && it2 != arg.diffVars.end()) {
00295         *it1 = *it2 * derivEval;
00296         ++it1;
00297         ++it2;
00298     }
00299     return toRet;
00300 }
00301
00302 template<typename ValType, size_t NumVars>
00303     requires(std::is_arithmetic<ValType>::value && NumVars >=
00304     0) auto sin(const DiffVar<ValType, NumVars> & arg) {
00305     return diffFunc(
00306         arg, [](ValType v) { return std::sin(v); },
00307         [](ValType v) { return std::cos(v); });
00308 }
00309
00310 template<typename ValType, size_t NumVars>
00311     requires(std::is_arithmetic<ValType>::value && NumVars >=
00312     0) auto cos(const DiffVar<ValType, NumVars> & arg) {
00313     return diffFunc(
00314         arg, [](ValType v) { return std::cos(v); },
00315         [](ValType v) { return -std::sin(v); });
00316 }
00317

```

```

00318 template<typename ValType, size_t NumVars>
00319 requires(std::is_arithmetic<ValType>::value && NumVars >=
00320             0) auto tan(const DiffVar<ValType, NumVars> & arg) {
00321     return diffFunc(
00322         arg, [](ValType v) { return std::tan(v); },
00323         [](ValType v) { return 1 / (std::cos(v) * std::cos(v)); });
00324 }
00325
00326 template<typename ValType, size_t NumVars>
00327 requires(std::is_arithmetic<ValType>::value && NumVars >=
00328             0) auto sinh(const DiffVar<ValType, NumVars> & arg) {
00329     return diffFunc(
00330         arg, [](ValType v) { return std::sinh(v); },
00331         [](ValType v) { return std::cosh(v); });
00332 }
00333
00334 template<typename ValType, size_t NumVars>
00335 requires(std::is_arithmetic<ValType>::value && NumVars >=
00336             0) auto cosh(const DiffVar<ValType, NumVars> & arg) {
00337     return diffFunc(
00338         arg, [](ValType v) { return std::cosh(v); },
00339         [](ValType v) { return std::sinh(v); });
00340 }
00341
00342 template<typename ValType, size_t NumVars>
00343 requires(std::is_arithmetic<ValType>::value && NumVars >=
00344             0) auto tanh(const DiffVar<ValType, NumVars> & arg) {
00345     return diffFunc(
00346         arg, [](ValType v) { return std::tanh(v); },
00347         [](ValType v) { return 1 / (std::cosh(v) * std::cosh(v)); });
00348 }
00349
00350 template<typename ValType, size_t NumVars>
00351 requires(std::is_arithmetic<ValType>::value && NumVars >=
00352             0) auto exp(const DiffVar<ValType, NumVars> & arg) {
00353     auto derivEval = std::exp(arg.var);
00354     DiffVar<ValType, NumVars> toRet(derivEval);
00355
00356     auto it1 = toRet.diffVars.begin();
00357     auto it2 = arg.diffVars.begin();
00358     while (it1 != toRet.diffVars.end() && it2 != arg.diffVars.end()) {
00359         *it1 = *it2 * derivEval;
00360         ++it1;
00361         ++it2;
00362     }
00363     toRet.var = derivEval;
00364     return toRet;
00365 }
00366
00367 template<typename ValType, size_t NumVars, typename OT>
00368 requires(std::is_arithmetic<ValType>::value && NumVars >= 0) &&
00369 (!std::is_same<OT, DiffVar<ValType, NumVars> >::value) auto pow(
00370     const DiffVar<ValType, NumVars> & arg, OT exponent) {
00371     return diffFunc(
00372         arg, [&exponent](ValType v) { return std::pow(v, exponent); },
00373         [&exponent](ValType v) { return exponent * std::pow(v, exponent - 1); });
00374 }
00375
00376 template<typename ValType, size_t NumVars>
00377 requires(std::is_arithmetic<ValType>::value && NumVars >=
00378             0) auto pow(const DiffVar<ValType, NumVars> & arg,
00379                         DiffVar<ValType, NumVars> exponent) {
00380     auto func = [](ValType v, ValType exponent) { return std::pow(v, exponent); };
00381
00382     auto deriv = [](ValType f, ValType fPrime, ValType g, ValType gPrime) {
00383         return std::pow(f, g - 1) * (g * fPrime + f * std::log(f) * gPrime);
00384     };
00385
00386     DiffVar<ValType, NumVars> toRet(func(arg.var, exponent.var));
00387
00388     auto it1 = toRet.diffVars.begin();
00389     auto it2 = arg.diffVars.begin();
00390     auto it3 = exponent.diffVars.begin();
00391     while (it1 != toRet.diffVars.end() && it2 != arg.diffVars.end()) {
00392         *it1 = deriv(arg.var, *it2, exponent.var, *it3);
00393         ++it1;
00394         ++it2;
00395         ++it3;
00396     }
00397     return toRet;
00398 }
00399
00400 template<typename ValType, size_t NumVars>
00401 requires(std::is_arithmetic<ValType>::value && NumVars >=
00402             0) auto sqrt(const DiffVar<ValType, NumVars> & arg) {
00403     auto funcResult = std::pow(arg.var, 0.5);
00404     DiffVar<ValType, NumVars> toRet(funcResult);

```

```

00405
00406     auto it1 = toRet.diffVars.begin();
00407     auto it2 = arg.diffVars.begin();
00408     auto derivEval = 0.5 / funcResult;
00409     while (it1 != toRet.diffVars.end() && it2 != arg.diffVars.end()) {
00410         *it1 = *it2 * derivEval;
00411         ++it1;
00412         ++it2;
00413     }
00414     return toRet;
00415 }
00416
00417
00418 } // namespace AutoDifferentiation
00419 #endif

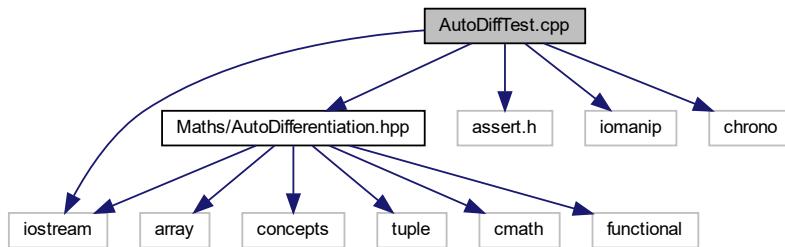
```

7.3 AutoDiffTest.cpp File Reference

```

#include <iostream>
#include "Maths/AutoDifferentiation.hpp"
#include <assert.h>
#include <iomanip>
#include <chrono>
Include dependency graph for AutoDiffTest.cpp:

```



Classes

- struct `BJTResults< T >`
- struct `TransistorTestResult< T >`

Functions

- void `testBasicOutput_NoCheck ()`
- template<typename T >
void `testBJTModelControl` (const T &base_v_be, const T &base_v_bc, `BJTResults< T >` &results)
- template<typename T >
void `testBJTModelAutoDiff` (const T &base_v_be, const T &base_v_bc, `BJTResults< T >` &results)
- template<typename T >
void `testBJTModel ()`
- template<typename T >
void `transistorTestControl` (T V_gs_in, T V_ds_in, `TransistorTestResult< T >` &toRet)
- template<typename T >
void `transistorTestAutoDiff` (T V_gs_in, T V_ds_in, `AD::DiffVar< T, 2 >` &toRet)
- template<typename T >
void `transistorTest ()`
- int `main` (int argc, char *argv[])

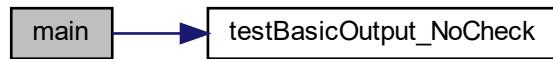
7.3.1 Function Documentation

7.3.1.1 main()

```
int main (
    int argc,
    char * argv[ ] )
```

Definition at line 333 of file [AutoDiffTest.cpp](#).

Here is the call graph for this function:



7.3.1.2 testBasicOutput_NoCheck()

```
void testBasicOutput_NoCheck ( )
```

Definition at line 10 of file [AutoDiffTest.cpp](#).

Here is the caller graph for this function:

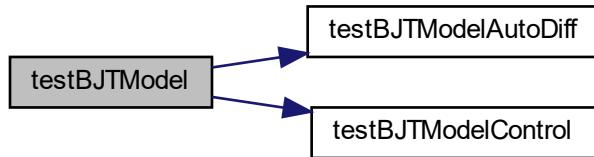


7.3.1.3 testBJTModel()

```
template<typename T >
void testBJTModel ( )
```

Definition at line 119 of file [AutoDiffTest.cpp](#).

Here is the call graph for this function:

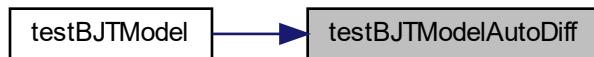


7.3.1.4 testBJTModelAutoDiff()

```
template<typename T >
void testBJTModelAutoDiff (
    const T & base_v_be,
    const T & base_v_bc,
    BJTResults< T > & results )
```

Definition at line 79 of file [AutoDiffTest.cpp](#).

Here is the caller graph for this function:

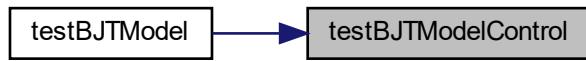


7.3.1.5 testBJTModelControl()

```
template<typename T >
void testBJTModelControl (
    const T & base_v_be,
    const T & base_v_bc,
    BJTResults< T > & results )
```

Definition at line 38 of file [AutoDiffTest.cpp](#).

Here is the caller graph for this function:

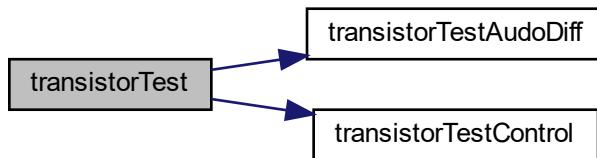


7.3.1.6 transistorTest()

```
template<typename T >
void transistorTest ( )
```

Definition at line 282 of file [AutoDiffTest.cpp](#).

Here is the call graph for this function:

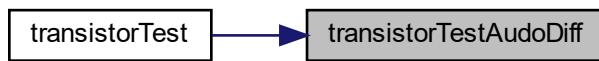


7.3.1.7 transistorTestAutoDiff()

```
template<typename T >
void transistorTestAutoDiff (
    T V_gs_in,
    T V_ds_in,
    AD::DiffVar< T, 2 > & toRet )
```

Definition at line 255 of file [AutoDiffTest.cpp](#).

Here is the caller graph for this function:

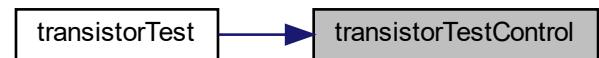


7.3.1.8 transistorTestControl()

```
template<typename T >
void transistorTestControl (
    T V_gs_in,
    T V_ds_in,
    TransistorTestResult< T > & toRet )
```

Definition at line 211 of file [AutoDiffTest.cpp](#).

Here is the caller graph for this function:



7.4 AutoDiffTest.cpp

```

00001 #include <iostream>
00002 #include "Maths/AutoDifferentiation.hpp"
00003 #include <assert.h>
00004 #include <iomanip>
00005 #include <chrono>
00006
00007 namespace AD = AutoDifferentiation;
00008
00009 void
00010 testBasicOutput_NoCheck() {
00011     constexpr AD::DiffVar<double, 2> x(1, 1, 0);
00012     constexpr AD::DiffVar<double, 2> y(2, 0, 1);
00013
00014     constexpr auto f = x * y;
00015     auto sin = AD::sin(x);
00016     auto sin2 = AD::sin(2 * x);
00017     auto tanh = AD::tanh(1 + 2 * x);
00018
00019     std::cout << "f\n" << f << std::endl;
00020     std::cout << "sin\n" << sin << std::endl;
00021     std::cout << "sin2\n" << sin2 << std::endl;
00022     std::cout << "tanh2\n" << tanh << std::endl;
00023 }
00024
00025 template<typename T>
00026 struct BJTResults {
00027     T g_ee;
00028     T g_ec;
00029     T g_ce;
00030     T g_cc;
00031
00032     T I_e;
00033     T I_c;
00034 };
00035
00036 template<typename T>
00037 void
00038 testBJTModelControl(const T & base_v_be, const T & base_v_bc,
00039                         BJTResults<T> & results) {
00040     // The control
00041
00042     constexpr T alpha_f = 0.99;
00043     constexpr T alpha_r = 0.02;
00044
00045     constexpr T I_es = 2e-14;
00046     constexpr T V_Te = 26e-3;
00047     constexpr T I_cs = 99e-14;
00048     constexpr T V_Tc = 26e-3;
00049
00050     T control_v_be = base_v_be;
00051     T control_v_bc = base_v_bc;
00052
00053     T control_i_e = -I_es * (std::exp(control_v_be / V_Te) - 1) +
00054         alpha_r * I_cs * (std::exp(control_v_bc / V_Tc) - 1);
00055     T control_i_c = alpha_f * I_es * (std::exp(control_v_be / V_Te) - 1) -
00056         I_cs * (std::exp(control_v_bc / V_Tc) - 1);
00057
00058     T control_g_ee = (I_es / V_Te) * std::exp(control_v_be / V_Te);
00059     T control_g_ec = alpha_r * (I_cs / V_Tc) * std::exp(control_v_bc / V_Tc);
00060     T control_g_ce = alpha_f * (I_es / V_Te) * std::exp(control_v_be / V_Te);
00061     T control_g_cc = (I_cs / V_Tc) * std::exp(control_v_bc / V_Tc);
00062
00063     T control_I_e = control_i_e + control_g_ee * control_v_be -
00064         control_g_ec * control_v_bc;
00065     T control_I_c = control_i_c - control_g_ce * control_v_be +
00066         control_g_cc * control_v_bc;
00067
00068     results.g_ee = control_g_ee;
00069     results.g_ec = control_g_ec;
00070     results.g_ce = control_g_ce;
00071     results.g_cc = control_g_cc;
00072
00073     results.I_e = control_I_e;
00074     results.I_c = control_I_c;
00075 }
00076
00077 template<typename T>
00078 void
00079 testBJTModelAutoDiff(const T & base_v_be, const T & base_v_bc,
00080                         BJTResults<T> & results) {
00081     // AutoDifferentiation
00082     constexpr T alpha_f = 0.99;
00083     constexpr T alpha_r = 0.02;
00084
00085     constexpr T I_es = 2e-14;

```

```

00086     constexpr T V_Te = 26e-3;
00087     constexpr T I_cs = 99e-14;
00088     constexpr T V_Tc = 26e-3;
00089     using ADT = AD::DiffVar<T, 2>;
00090
00091     ADT ad_v_be = ADT(base_v_be, 1, 0);
00092     ADT ad_v_bc = ADT(base_v_bc, 0, 1);
00093
00094     ADT ad_i_e = -I_es * (AD::exp(ad_v_be / V_Te) - 1) +
00095         alpha_r * I_cs * (AD::exp(ad_v_bc / V_Tc) - 1);
00096     ADT ad_i_c = alpha_f * I_es * (AD::exp(ad_v_be / V_Te) - 1) -
00097         I_cs * (AD::exp(ad_v_bc / V_Tc) - 1);
00098
00099 // Najm defines the g_ee and g_cc to be the negated partial
00100     T ad_g_ee = -ad_i_e[1];
00101     T ad_g_ec = ad_i_e[2];
00102     T ad_g_ce = ad_i_c[1];
00103     T ad_g_cc = -ad_i_c[2];
00104
00105     T ad_I_e = ad_i_e[0] + ad_g_ee * ad_v_be[0] - ad_g_ec * ad_v_bc[0];
00106     T ad_I_c = ad_i_c[0] - ad_g_ce * ad_v_be[0] + ad_g_cc * ad_v_bc[0];
00107
00108     results.g_ee = ad_g_ee;
00109     results.g_ec = ad_g_ec;
00110     results.g_ce = ad_g_ce;
00111     results.g_cc = ad_g_cc;
00112
00113     results.I_e = ad_I_e;
00114     results.I_c = ad_I_c;
00115 }
00116
00117 template<typename T>
00118 [[clang::optnone]] void
00119 testBJTModel() {
00120     // Model constants
00121     constexpr T I_es = 2e-14;
00122     constexpr T V_Te = 26e-3;
00123     constexpr T I_cs = 99e-14;
00124     constexpr T V_Tc = 26e-3;
00125
00126     BJTResults<T> controlResults;
00127     BJTResults<T> autoDiffResults;
00128
00129     T V_bc_crit = V_Tc * std::log(V_Tc / (I_cs * std::sqrt(2)));
00130     T V_be_crit = V_Te * std::log(V_Te / (I_es * std::sqrt(2)));
00131
00132     std::array<T, 6> base_v_be_vec = {0, 1, 2, 3, 4, 5};
00133     std::array<T, 6> base_v_bc_vec = {0, 1, 2, 3, 4, 5};
00134 ;
00135
00136     size_t controlAccumulate = 0;
00137     size_t autoDiffAccumulate = 0;
00138
00139     for (auto base_v_be : base_v_be_vec) {
00140         for (auto base_v_bc : base_v_bc_vec) {
00141             std::cout << "base_v_be=" << std::setw(10) << base_v_be
00142             << ", base_v_bc=" << std::setw(10) << base_v_bc << std::endl;
00143
00144         base_v_be = std::min(V_be_crit, base_v_be);
00145         base_v_bc = std::min(V_bc_crit, base_v_bc);
00146
00147         auto startTime = std::chrono::high_resolution_clock::now();
00148         for (size_t i = 0; i < 10000; i++) {
00149             testBJTModelControl(base_v_be, base_v_bc, controlResults);
00150         }
00151         auto endTime = std::chrono::high_resolution_clock::now();
00152         auto timeTaken = std::chrono::duration_cast<std::chrono::microseconds>(
00153             endTime - startTime)
00154             .count();
00155         controlAccumulate += timeTaken;
00156
00157         std::cout << std::setw(15) << timeTaken << " us | ";
00158
00159         startTime = std::chrono::high_resolution_clock::now();
00160         for (size_t i = 0; i < 10000; i++) {
00161             testBJTModelAutoDiff(base_v_be, base_v_bc, autoDiffResults);
00162         }
00163         endTime = std::chrono::high_resolution_clock::now();
00164         timeTaken = std::chrono::duration_cast<std::chrono::microseconds>(
00165             endTime - startTime)
00166             .count();
00167         autoDiffAccumulate += timeTaken;
00168
00169         std::cout << std::setw(15) << timeTaken << " us" << std::endl;
00170
00171     // Comparisons
00172

```

```

00173     std::cout << "g_ee | control=" << std::setw(15) << controlResults.g_ee
00174         << " | autoDiff=" << std::setw(15) << autoDiffResults.g_ee
00175         << std::endl;
00176     std::cout << "g_ec | control=" << std::setw(15) << controlResults.g_ec
00177         << " | autoDiff=" << std::setw(15) << autoDiffResults.g_ec
00178         << std::endl;
00179     std::cout << "g_ce | control=" << std::setw(15) << controlResults.g_ce
00180         << " | autoDiff=" << std::setw(15) << autoDiffResults.g_ce
00181         << std::endl;
00182     std::cout << "g_cc | control=" << std::setw(15) << controlResults.g_cc
00183         << " | autoDiff=" << std::setw(15) << autoDiffResults.g_cc
00184         << std::endl;
00185
00186     std::cout << "I_e | control=" << std::setw(15) << controlResults.I_e
00187         << " | autoDiff=" << std::setw(15) << autoDiffResults.I_e
00188         << std::endl;
00189     std::cout << "I_c | control=" << std::setw(15) << controlResults.I_c
00190         << " | autoDiff=" << std::setw(15) << autoDiffResults.I_c
00191         << std::endl;
00192
00193     assert(std::abs(controlResults.I_e - autoDiffResults.I_e) < 1e-12);
00194     assert(std::abs(controlResults.I_c - autoDiffResults.I_c) < 1e-12);
00195 }
00196 }
00197
00198 std::cout << std::setw(15) << controlAccumulate << " us | ";
00199 std::cout << std::setw(15) << autoDiffAccumulate << " us" << std::endl;
00200 }
00201
00202 template<typename T>
00203 struct TransistorTestResult {
00204     T var;
00205     T diff1;
00206     T diff2;
00207 };
00208
00209 template<typename T>
00210 void
00211 transistorTestControl(T V_gs_in, T V_ds_in, TransistorTestResult<T> & toRet) {
00212     constexpr T alpha = 1.3;
00213     constexpr T beta0 = 0.42;
00214     constexpr T gamma = 0.0005;
00215     constexpr T delta = 0.3;
00216     constexpr T xi = 0.06;
00217     constexpr T lambda = 1.5;
00218     constexpr T mu = 0.0;
00219     constexpr T zeta = 0.18;
00220     constexpr T Vto = -2.4;
00221
00222     T V_gs = V_gs_in;
00223     T V_ds = V_ds_in;
00224
00225     auto beta = beta0;
00226     auto Vgst = V_gs - (1 + beta * beta) * Vto + gamma * V_ds;
00227     auto Veff = 0.5 * (Vgst + std::pow(std::pow(Vgst, 2) + delta * delta, 0.5));
00228     auto power = lambda / (1 + mu * std::pow(V_ds, 2) + xi * Veff);
00229     auto area = alpha * V_ds * (1 + zeta * Veff);
00230     auto f1 = std::tanh(area);
00231     auto Ids_lim = beta * std::pow(Veff, power);
00232     auto Idrain = Ids_lim * f1;
00233     toRet.var = Idrain;
00234
00235     auto dVeff_dVgs = 0.5 * (1 + Vgst * std::pow(Vgst * Vgst + delta * delta, -0.5));
00236     auto dpower_dVgs = -lambda * xi * dVeff_dVgs * std::pow(power / lambda, 2);
00237     auto df_dVgs = std::pow(1 / cosh(area), 2) * alpha * V_ds * zeta * dVeff_dVgs;
00238     toRet.diff1 =
00239         (power * (dVeff_dVgs / Veff) + std::log(Veff) * dpower_dVgs) +
00240         Ids_lim * df_dVgs;
00241
00242     auto dVeff_dVds = 0.5 * (gamma + std::pow(Vgst * Vgst + delta * delta, -0.5)) *
00243         Vgst * gamma;
00244     auto dpower_dVds = -lambda * (2 * mu * V_ds + xi * dVeff_dVds) *
00245         std::pow(power / lambda, 2);
00246     auto df_dVds = std::pow(1 / cosh(area), 2) * alpha *
00247         (1 + zeta * (V_ds * dVeff_dVds + Veff));
00248     toRet.diff2 = Idrain *
00249         (power * (dVeff_dVds / Veff) + std::log(Veff) * dpower_dVds) +
00250         Ids_lim * df_dVds;
00251 }
00252
00253 template<typename T>
00254 void
00255 transistorTestAutoDiff(T V_gs_in, T V_ds_in, AD::DiffVar<T, 2> & toRet) {
00256     constexpr T alpha = 1.3;
00257     constexpr T beta0 = 0.42;
00258     constexpr T gamma = 0.0005;
00259     constexpr T delta = 0.3;

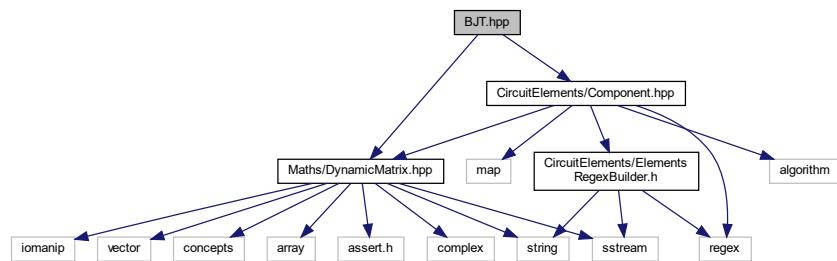
```

```

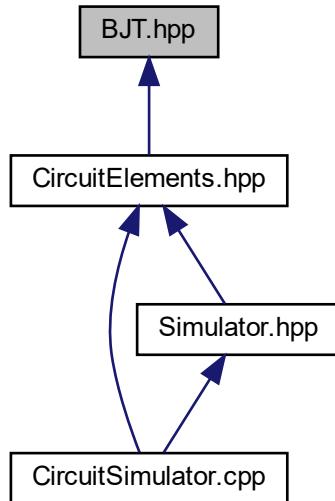
00260     constexpr T xi = 0.06;
00261     constexpr T lambda = 1.5;
00262     constexpr T mu = 0.0;
00263     constexpr T zeta = 0.18;
00264     constexpr T Vto = -2.4;
00265
00266     using ADT = AD::DiffVar<T, 2>;
00267     ADT V_gs(V_gs_in, 1, 0);
00268     ADT V_ds(V_ds_in, 0, 1);
00269
00270     auto beta = beta0;
00271     auto Vgst = V_gs - (1 + beta * beta) * Vto + gamma * V_ds;
00272     auto Veff = 0.5 * (Vgst + ADT::sqrt(ADT::pow(Vgst, 2) + delta * delta));
00273     auto power = lambda / (1 + mu * ADT::pow(V_ds, 2) + xi * Veff);
00274     auto area = alpha * V_ds * (1 + zeta * Veff);
00275     auto f1 = ADT::tanh(area);
00276     auto Ids_lim = beta * ADT::pow(Veff, power);
00277     toRet = Ids_lim * f1;
00278 }
00279
00280 template<typename T>
00281 [[clang::optnone]] void
00282 transistorTest() {
00283     using ADT = AD::DiffVar<T, 2>;
00284     std::vector<T> V_gs_in_vec = {0, 1, 2, 3};
00285     std::vector<T> V_ds_in_vec = {0, 1, 2, 3};
00286     size_t controlAccumulate = 0;
00287     size_t autoDiffAccumulate = 0;
00288     for (auto V_gs_in : V_gs_in_vec) {
00289         for (auto V_ds_in : V_gs_in_vec) {
00290             ADT autoDiffResult(0);
00291             TransistorTestResult<T> res;
00292             auto startTime = std::chrono::high_resolution_clock::now();
00293             for (size_t i = 0; i < 10000; i++) {
00294                 transistorTestAutoDiff(V_gs_in, V_ds_in, autoDiffResult);
00295             }
00296             auto endTime = std::chrono::high_resolution_clock::now();
00297             auto timeTaken = std::chrono::duration_cast<std::chrono::microseconds>(
00298                     endTime - startTime)
00299                     .count();
00300             autoDiffAccumulate += timeTaken;
00301             std::cout << std::setw(15) << timeTaken << " us | ";
00302
00303             startTime = std::chrono::high_resolution_clock::now();
00304             for (size_t i = 0; i < 10000; i++) {
00305                 transistorTestControl(V_gs_in, V_ds_in, res);
00306             }
00307             endTime = std::chrono::high_resolution_clock::now();
00308             timeTaken = std::chrono::duration_cast<std::chrono::microseconds>(
00309                     endTime - startTime)
00310                     .count();
00311             controlAccumulate += timeTaken;
00312             std::cout << std::setw(15) << timeTaken << " us" << std::endl;
00313
00314             std::cout << "V_gs: " << V_gs_in << " V_ds: " << V_ds_in << " "
00315             << std::endl;
00316             std::cout << std::setw(15) << res.var << " ";
00317             std::cout << std::setw(15) << res.diff1 << " ";
00318             std::cout << std::setw(15) << res.diff2 << std::endl;
00319             std::cout << std::setw(15) << autoDiffResult[0] << " ";
00320             std::cout << std::setw(15) << autoDiffResult[1] << " ";
00321             std::cout << std::setw(15) << autoDiffResult[2] << std::endl
00322             << std::endl;
00323             assert(std::abs(res.var - autoDiffResult[0]) < 1e-12);
00324             assert(std::abs(res.diff1 - autoDiffResult[1]) < 1e-12);
00325             assert(std::abs(res.diff2 - autoDiffResult[2]) < 1e-12);
00326         }
00327     }
00328     std::cout << std::setw(15) << autoDiffAccumulate << " us | ";
00329     std::cout << std::setw(15) << controlAccumulate << " us" << std::endl;
00330 }
00331
00332 int
00333 main(int argc, char * argv[]) {
00334     testBasicOutput_NoCheck();
00335
00336     testBJTModel<double>();
00337
00338     transistorTest<double>();
00339
00340     return 0;
00341 }
```

7.5 BJT.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for BJT.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `BJTN< T >`
A simple NPN BJT model.
- struct `BJTP< T >`
A simple PNP BJT model.

7.6 BJT.hpp

```

00001 #ifndef _BJT_HPP_INC_
00002 #define _BJT_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00009 template<typename T>
0010 struct BJT : public Component<T> {
0011 public:
0012     size_t c = 0;
0013     size_t b = 0;
0014     size_t e = 0;
0015
0016     T alpha_f = 0.99;
0017     T alpha_r = 0.02;
0018
0019     const T I_es = 2e-14;
0020     const T V_Te = 26e-3;
0021     const T I_cs = 99e-14;
0022     const T V_Tc = 26e-3;
0023
0024     T V_bc_crit = V_Tc * std::log(V_Tc / (I_cs * std::sqrt(2)));
0025     T V_be_crit = V_Te * std::log(V_Te / (I_es * std::sqrt(2)));
0026
0027
0028     void
0029     addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
0030                         const size_t currentSolutionIndex, T timestep = 0) const {
0031         const size_t bp = b - 1;
0032         const size_t cp = c - 1;
0033         const size_t ep = e - 1;
0034
0035         T v_be = 0;
0036         T v_bc = 0;
0037
0038         if (b > 0) {
0039             v_be = solutionMatrix(bp, currentSolutionIndex);
0040             v_bc = solutionMatrix(bp, currentSolutionIndex);
0041         }
0042
0043         if (e > 0) {
0044             v_be -= solutionMatrix(ep, currentSolutionIndex);
0045         }
0046
0047         if (c > 0) {
0048             v_bc -= solutionMatrix(cp, currentSolutionIndex);
0049         }
0050
0051         v_be = std::min(V_be_crit, v_be);
0052         v_bc = std::min(V_bc_crit, v_bc);
0053
0054         T i_e = -I_es * (std::exp(v_be / V_Te) - 1) +
0055             alpha_r * I_cs * (std::exp(v_bc / V_Tc) - 1);
0056         T i_c = alpha_f * I_es * (std::exp(v_be / V_Te) - 1) -
0057             I_cs * (std::exp(v_bc / V_Tc) - 1);
0058 // T i_b = - (i_e + i_c); This is unused
0059
0060         T g_ee = (I_es / V_Te) * std::exp(v_be / V_Te);
0061         T g_ec = alpha_r * (I_cs / V_Tc) * std::exp(v_bc / V_Tc);
0062         T g_ce = alpha_f * (I_es / V_Te) * std::exp(v_be / V_Te);
0063         T g_cc = (I_cs / V_Tc) * std::exp(v_bc / V_Tc);
0064
0065         T I_e = i_e + g_ee * v_be - g_ec * v_bc;
0066         T I_c = i_c - g_ce * v_be + g_cc * v_bc;
0067
0068         if (e > 0) {
0069             stamp.G(ep, ep) += g_ee;
0070             stamp.s(ep, 0) += -I_e;
0071
0072             if (c > 0) {
0073                 stamp.G(ep, cp) += -g_ec;
0074             }
0075
0076             if (b > 0) {
0077                 stamp.G(ep, bp) += (g_ec - g_ee);
0078             }
0079         }
0080
0081         if (c > 0) {
0082             stamp.G(cp, cp) += g_cc;
0083             stamp.s(cp, 0) += -I_c;
0084
0085             if (e > 0) {
0086                 stamp.G(cp, ep) += -g_ce;
0087             }
0088

```

```

00089         if (b > 0) {
00090             stamp.G(cp, bp) += (g_ce - g_cc);
00091         }
00092     }
00093
00094     if (b > 0) {
00095         stamp.G(bp, bp) += g_cc + g_ee - g_ce - g_ec;
00096         stamp.s(bp, 0) += I_e + I_c;
00097
00098         if (e > 0) {
00099             stamp.G(bp, ep) += g_ce - g_ee;
00100         }
00101
00102         if (c > 0) {
00103             stamp.G(bp, cp) += (g_ec - g_cc);
00104         }
00105     }
00106 }
00107
00108 void updateStoredState(const Matrix<T> & solutionMatrix,
00109                         const size_t currentSolutionIndex, T timestep,
00110                         size_t sizeG_A) {
00111 }
00112
00113 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00114                           size_t numCurrents) const {
00115     addNonLinearStampTo(stamp, solutionVector, 0, 0);
00116 }
00117
00118 static void
00119 addToElements(const std::string & line, CircuitElements<T> & elements,
00120                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00121     std::regex BJTRegex = generateRegex("QN", "n n n");
00122     BJT<T> bjt;
00123     std::smatch matches;
00124
00125     std::regex_match(line, matches, BJTRegex);
00126
00127     bjt.designator = "QN";
00128     bjt.designator += matches.str(1);
00129
00130     bjt.c = std::stoi(matches.str(2));
00131     bjt.b = std::stoi(matches.str(3));
00132     bjt.e = std::stoi(matches.str(4));
00133
00134     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00135     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00136     numNodes = std::max(numNodes, std::stoull(matches.str(4)));
00137
00138
00139     elements.nonLinearElements.emplace_back(std::make_shared<BJTN<T>>(bjt));
00140     elements.nodeComponentMap.insert(
00141         {{bjt.b, elements.nonLinearElements.back()},
00142          {bjt.c, elements.nonLinearElements.back()},
00143          {bjt.e, elements.nonLinearElements.back()}});
00144     }
00145 };
00146
00147 template<typename T>
00148 struct BJT<T> : public Component<T> {
00149 public:
00150     size_t c = 0;
00151     size_t b = 0;
00152     size_t e = 0;
00153
00154     T alpha_f = 0.99;
00155     T alpha_r = 0.02;
00156
00157     const T I_es = 2e-14;
00158     const T V_Te = 26e-3;
00159     const T I_cs = 99e-14;
00160     const T V_Tc = 26e-3;
00161
00162     T V_bc_crit = V_Tc * std::log(V_Tc / (I_es * std::sqrt(2)));
00163     T V_be_crit = V_Te * std::log(V_Te / (I_es * std::sqrt(2)));
00164
00165
00166     void
00167     addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00168                         const size_t currentSolutionIndex, T timestep = 0) const {
00169         const size_t bp = b - 1;
00170         const size_t cp = c - 1;
00171         const size_t ep = e - 1;
00172
00173         T v_be = 0;
00174         T v_bc = 0;
00175
00176     }
00177
00178 }
```

```

00179     if (b > 0) {
00180         v_be = solutionMatrix(bp, currentSolutionIndex);
00181         v_bc = solutionMatrix(bp, currentSolutionIndex);
00182     }
00183
00184     if (e > 0) {
00185         v_be -= solutionMatrix(ep, currentSolutionIndex);
00186     }
00187
00188     if (c > 0) {
00189         v_bc -= solutionMatrix(cp, currentSolutionIndex);
00190     }
00191
00192     v_be = std::max(-V_be_crit, v_be);
00193     v_bc = std::max(-V_bc_crit, v_bc);
00194
00195     T i_F = I_cs * (std::exp(-v_bc / V_Tc) - 1);
00196     T i_R = I_es * (std::exp(-v_be / V_Te) - 1);
00197     T di_F = -(I_cs / V_Tc) * std::exp(-v_bc / V_Tc);
00198     T di_R = -(I_es / V_Te) * std::exp(-v_be / V_Te);
00199
00200     T i_e = i_R - alpha_f * i_F;
00201     T i_b = (alpha_f - 1) * i_F + (alpha_r - 1) * i_R;
00202     T i_c = i_F - alpha_r * i_R;
00203
00204     T g_ee = di_F;
00205     T g_ec = -alpha_r * di_R;
00206     T g_ce = -alpha_f * di_F;
00207     T g_cc = di_R;
00208     T g_be = (alpha_r - 1) * di_R;
00209     T g_bc = (alpha_f - 1) * di_F;
00210
00211     T I_e = i_e - g_ee * v_be - g_ec * v_bc;
00212     T I_c = i_c - g_ce * v_be - g_cc * v_bc;
00213     T I_b = i_b - g_be * v_be - g_bc * v_bc;
00214
00215     if (e > 0) {
00216         stamp.G(ep, ep) += -g_ee;
00217         stamp.s(ep, 0) += -I_e;
00218
00219         if (c > 0) {
00220             stamp.G(ep, cp) += -g_ec;
00221         }
00222
00223         if (b > 0) {
00224             stamp.G(ep, bp) += g_ec + g_ee;
00225         }
00226     }
00227
00228     if (c > 0) {
00229         stamp.G(cp, cp) += -g_cc;
00230         stamp.s(cp, 0) += -I_c;
00231
00232         if (e > 0) {
00233             stamp.G(cp, ep) += -g_ce;
00234         }
00235
00236         if (b > 0) {
00237             stamp.G(cp, bp) += g_ce + g_cc;
00238         }
00239     }
00240
00241     if (b > 0) {
00242         stamp.G(bp, bp) += g_be + g_bc;
00243         stamp.s(bp, 0) += -I_b;
00244
00245         if (e > 0) {
00246             stamp.G(bp, ep) += -g_be;
00247         }
00248
00249         if (c > 0) {
00250             stamp.G(bp, cp) += -g_bc;
00251         }
00252     }
00253 }
00254
00255 void updateStoredState(const Matrix<T> & solutionMatrix,
00256                         const size_t currentSolutionIndex, T timestep,
00257                         size_t sizeG_A) {
00258 }
00259
00260 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00261                           size_t numCurrents) const {
00262     addNonLinearStampTo(stamp, solutionVector, 0, 0);
00263 }
00264
00265 static void

```

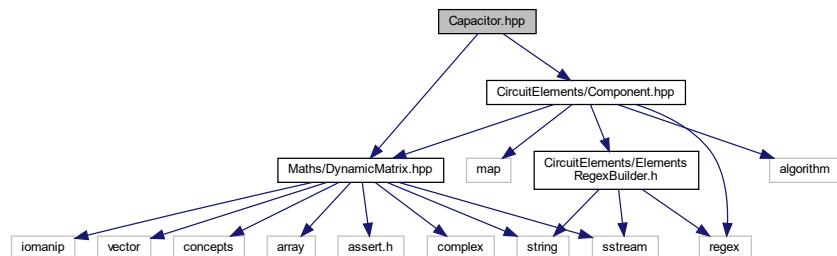
```

00266     addToElements(const std::string & line, CircuitElements<T> & elements,
00267             size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00268         std::regex BJTRegex = generateRegex("QP", "n n n");
00269         BJT<T> bjt;
00270         std::smatch matches;
00271
00272         std::regex_match(line, matches, BJTRegex);
00273
00274         bjt.designator = "QP";
00275         bjt.designator += matches.str(1);
00276
00277         bjt.c = std::stoi(matches.str(2));
00278         bjt.b = std::stoi(matches.str(3));
00279         bjt.e = std::stoi(matches.str(4));
00280
00281         numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00282         numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00283         numNodes = std::max(numNodes, std::stoull(matches.str(4)));
00284
00285
00286         elements.nonLinearElements.emplace_back(std::make_shared<BJTP<T>>(bjt));
00287         elements.nodeComponentMap.insert(
00288             {{bjt.b, elements.nonLinearElements.back()},
00289              {bjt.c, elements.nonLinearElements.back()},
00290              {bjt.e, elements.nonLinearElements.back()}});
00291     }
00292 };
00293 #endif

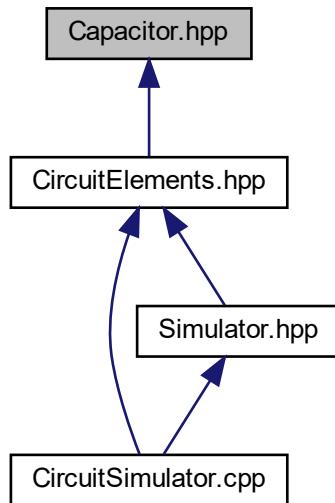
```

7.7 Capacitor.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for Capacitor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [Capacitor< T >](#)
An ideal capacitor model.

7.8 Capacitor.hpp

```

00001 #ifndef _CAPACITOR_HPP_
00002 #define _CAPACITOR_HPP_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00006
00010 template<typename T>
00011 struct Capacitor : public Component<T> {
00012     public:
00013         T value = 0;
00014
00015         size_t n1 = 0;
00016         size_t n2 = 0;
00017         T lastCurrent = 0;
00018
00019         bool trapezoidalRule = true;
00020         void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00021                               const size_t currentSolutionIndex, T timestep) const {
00022             size_t n1p = n1 - 1;
00023             size_t n2p = n2 - 1;
00024
00025             T u0 = 0;
00026             if (n1) {
00027                 u0 = solutionMatrix(n1p, currentSolutionIndex - 1);
00028             }
00029
00030             if (n2) {
00031                 u0 -= solutionMatrix(n2p, currentSolutionIndex - 1);
00032             }
00033
00034             T G_eq = 0;
00035             T I_eq = 0;
  
```

```

00037
00038     if (trapezoidalRule) {
00039         G_eq = 2 * value / timestep;
00040         I_eq = lastCurrent + G_eq * u0;
00041     } else {
00042         G_eq = value / timestep;
00043         I_eq = value * u0 / timestep;
00044     }
00045
00046     if (n1) {
00047         stamp.G(nlp, nlp) += G_eq;
00048         stamp.s(nlp, 0) += I_eq;
00049     }
00050
00051     if (n2) {
00052         stamp.G(n2p, n2p) += G_eq;
00053         stamp.s(n2p, 0) += -I_eq;
00054     }
00055
00056     if (n1 && n2) {
00057         stamp.G(nlp, n2p) += -G_eq;
00058         stamp.G(n2p, nlp) += -G_eq;
00059     }
00060 }
00061
00062 void updateStoredState(const Matrix<T> & solutionMatrix,
00063                         const size_t currentSolutionIndex, T timestep,
00064                         size_t sizeG_A) {
00065     if (trapezoidalRule) {
00066         size_t nlp = n1 - 1;
00067         size_t n2p = n2 - 1;
00068         T u0 = 0;
00069         T u1 = 0;
00070         if (n1) {
00071             u0 = solutionMatrix(nlp, currentSolutionIndex - 1);
00072             u1 = solutionMatrix(nlp, currentSolutionIndex);
00073         }
00074
00075         if (n2) {
00076             u0 -= solutionMatrix(n2p, currentSolutionIndex - 1);
00077             u1 -= solutionMatrix(n2p, currentSolutionIndex);
00078         }
00079
00080         T G_eq = 2 * value / timestep;
00081         lastCurrent = G_eq * u1 - (lastCurrent + G_eq * u0);
00082     }
00083 }
00084
00085 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00086                           size_t numCurrents) const {
00087     // open circuit. Maybe add large connection to ref if unstable
00088     if (n1 > 0) {
00089         stamp.G(n1 - 1, n1 - 1) += 1e-9;
00090     }
00091     if (n2 > 0) {
00092         stamp.G(n2 - 1, n2 - 1) += 1e-9;
00093     }
00094 }
00095
00096 static void
00097 addToElements(const std::string & line, CircuitElements<T> & elements,
00098               size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00099     // std::regex capacitorRegex( R"(^C(.*)\s(\d+?)\s(\d+?)\s(.+)\s\$)" );
00100     std::regex capacitorRegex = generateRegex("C", "n n w");
00101     Capacitor<T> capacitor;
00102     std::smatch matches;
00103
00104     std::regex_match(line, matches, capacitorRegex);
00105
00106     capacitor.designator = "C";
00107     capacitor.designator += matches.str(1);
00108
00109     capacitor.n1 = std::stoi(matches.str(2));
00110     capacitor.n2 = std::stoi(matches.str(3));
00111     capacitor.trapezoidalRule = true;
00112
00113     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00114     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00115
00116     if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00117         capacitor.value = std::stod(matches.str(4));
00118     } else {
00119         static_assert("Unsupported Type");
00120     }
00121
00122     elements.dynamicElements.emplace_back(
00123         std::make_shared<Capacitor<T>>(capacitor));

```

```

00124
00125     elements.nodeComponentMap.insert(
00126         {{capacitor.n1, elements.dynamicElements.back()},
00127          {capacitor.n2, elements.dynamicElements.back()}});
00128 }
00129 };
00130
00131
00132 #endif

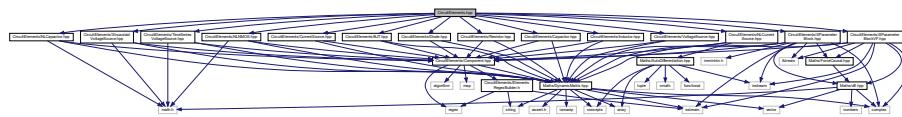
```

7.9 CircuitElements.hpp File Reference

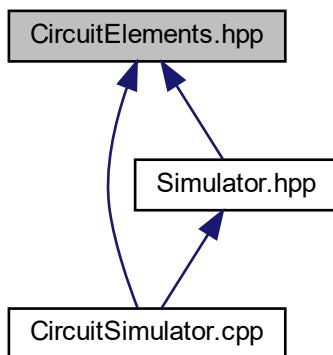
```

#include "CircuitElements/Resistor.hpp"
#include "CircuitElements/Capacitor.hpp"
#include "CircuitElements/NLCapacitor.hpp"
#include "CircuitElements/Inductor.hpp"
#include "CircuitElements/VoltageSource.hpp"
#include "CircuitElements/SinusoidalVoltageSource.hpp"
#include "CircuitElements/TimeSeriesVoltageSource.hpp"
#include "CircuitElements/CurrentSource.hpp"
#include "CircuitElements/BJT.hpp"
#include "CircuitElements/Diode.hpp"
#include "CircuitElements/SParameterBlock.hpp"
#include "CircuitElements/SParameterBlockVF.hpp"
#include "CircuitElements/NLNMOS.hpp"
#include "CircuitElements/NLCurrentSource.hpp"
Include dependency graph for CircuitElements.hpp:

```



This graph shows which files directly or indirectly include this file:



Classes

- struct [CircuitElements< T >](#)
a glorified container for the different types of components.

Enumerations

- enum class [SolutionStage { StaticSolution = 0 , DynamicSolution = 1 , NonLinearSolution = 2 }](#)
An enum to track how far we want to go in a solution.
- enum class [ComponentType { Resistor , Capacitor , Inductor , VoltageSource , CurrentSource , BJT , Diode }](#)
An enum for component types. Current unused.

7.9.1 Enumeration Type Documentation

7.9.1.1 ComponentType

```
enum ComponentType [strong]
```

An enum for component types. Current unused.

Enumerator

Resistor	
Capacitor	
Inductor	
VoltageSource	
CurrentSource	
BJT	
Diode	

Definition at line 31 of file [CircuitElements.hpp](#).

7.9.1.2 SolutionStage

```
enum SolutionStage [strong]
```

An enum to track how far we want to go in a solution.

Enumerator

StaticSolution	
DynamicSolution	
NonLinearSolution	

Definition at line 24 of file [CircuitElements.hpp](#).

7.10 CircuitElements.hpp

```

00001 #ifndef _CIRCUITELEMENTS_HPP_INC_
00002 #define _CIRCUITELEMENTS_HPP_INC_
00003 #include "CircuitElements/Resistor.hpp"
00004 #include "CircuitElements/Capacitor.hpp"
00005 #include "CircuitElements/NLCapacitor.hpp"
00006 #include "CircuitElements/Inductor.hpp"
00007 #include "CircuitElements/VoltageSource.hpp"
00008 #include "CircuitElements/SinusoidalVoltageSource.hpp"
00009 #include "CircuitElements/TimeSeriesVoltageSource.hpp"
00010 #include "CircuitElements/CurrentSource.hpp"
00011 #include "CircuitElements/BJT.hpp"
00012 #include "CircuitElements/Diode.hpp"
00013 #include "CircuitElements/SParameterBlock.hpp"
00014 #include "CircuitElements/SParameterBlockVF.hpp"
00015 #include "CircuitElements/NLNMOS.hpp"
00016 #include "CircuitElements/NLCurrentSource.hpp"
00017
00018 #ifdef WITH_MATLAB
00019 #include "MatlabEngine.hpp"
00020 #include "MatlabDataArray.hpp"
00021 #endif
00022
00024 enum class SolutionStage {
00025     StaticSolution = 0,
00026     DynamicSolution = 1,
00027     NonLinearSolution = 2
00028 };
00029
00031 enum class ComponentType {
00032     Resistor,
00033     Capacitor,
00034     Inductor,
00035     VoltageSource,
00036     CurrentSource,
00037     BJT,
00038     Diode
00039 };
00040
00041
00045 template<typename T>
00046 struct CircuitElements {
00049     Stamp<T> staticStamp;
00052     Stamp<T> dynamicStamp;
00056     Stamp<T> nonLinearStamp;
00060     Stamp<T> dcStamp;
00061
00062 #ifdef WITH_MATLAB
00063     std::shared_ptr<matlab::engine::MATLABEngine> matlabEngine;
00064 #endif
00065
00067     std::vector<std::shared_ptr<Component<T>>> staticElements;
00069     std::vector<std::shared_ptr<Component<T>>> dynamicElements;
00071     std::vector<std::shared_ptr<Component<T>>> nonLinearElements;
00072
00074     bool staticStampIsFresh = false;
00076     bool dynamicStampIsFresh = false;
00078     bool nonLinearStampIsFresh = false;
00079
00081     std::multimap<size_t, std::shared_ptr<Component<T>>> nodeComponentMap;
00082
00088 CircuitElements(size_t numNodes = 0, size_t numCurrents = 0,
00089                     size_t numDCCurrents = 0)
00090     : staticStamp(numNodes, numCurrents), dynamicStamp(numNodes, numCurrents),
00091       nonLinearStamp(numNodes, numCurrents),
00092       dcStamp(numNodes, numCurrents + numDCCurrents) {
00093 }
00094
00100 void
00101 setNewStampSize(size_t numNodes, size_t numCurrents, size_t numDCCurrents = 0) {
00102     staticStamp = Stamp<T>(numNodes, numCurrents);
00103     dynamicStamp = Stamp<T>(numNodes, numCurrents);
00104     nonLinearStamp = Stamp<T>(numNodes, numCurrents);
00105     dcStamp = Stamp<T>(numNodes, numCurrents + numDCCurrents);
00106     staticStampIsFresh = false;
00107     dynamicStampIsFresh = false;
00108     nonLinearStampIsFresh = false;
00109 }
00110
00114     Stamp<T> & generateStaticStamp() {

```

```

00115     staticStamp.clear();
00116
00117     for (const auto & component : staticElements) {
00118         staticStamp.addStaticStamp(component);
00119     }
00120
00121     for (const auto & component : dynamicElements) {
00122         staticStamp.addStaticStamp(component);
00123     }
00124
00125     for (const auto & component : nonLinearElements) {
00126         staticStamp.addStaticStamp(component);
00127     }
00128
00129     staticStampIsFresh = true;
00130     return staticStamp;
00131 }
00132
00141 Stamp<T> & generateDynamicStamp(const Matrix<T> & solutionMatrix,
00142                                     const size_t currentSolutionIndex, T timestep) {
00143     if (!staticStampIsFresh) {
00144         generateStaticStamp();
00145     }
00146     dynamicStamp = staticStamp;
00147
00148     for (const auto & component : dynamicElements) {
00149         dynamicStamp.addDynamicStamp(component, solutionMatrix,
00150                                         currentSolutionIndex, timestep);
00151     }
00152
00153     for (const auto & component : nonLinearElements) {
00154         dynamicStamp.addDynamicStamp(component, solutionMatrix,
00155                                         currentSolutionIndex, timestep);
00156     }
00157
00158     dynamicStampIsFresh = true;
00159     return dynamicStamp;
00160 }
00161
00170 Stamp<T> &
00171 generateNonLinearStamp(const Matrix<T> & solutionMatrix,
00172                         const size_t currentSolutionIndex, T timestep) {
00173     if (!dynamicStampIsFresh) {
00174         generateDynamicStamp(solutionMatrix, currentSolutionIndex, timestep);
00175     }
00176     nonLinearStamp = dynamicStamp;
00177
00178     for (const auto & component : nonLinearElements) {
00179         nonLinearStamp.addNonLinearStamp(component, solutionMatrix,
00180                                         currentSolutionIndex, timestep);
00181     }
00182
00183     nonLinearStampIsFresh = true;
00184     return nonLinearStamp;
00185 }
00186
00195 Stamp<T> &
00196 generateCompleteStamp(SolutionStage stage, const Matrix<T> & solutionMatrix,
00197                         const size_t currentSolutionIndex, T timestep) {
00198     switch (stage) {
00199         case SolutionStage::StaticSolution:
00200             if (!staticStampIsFresh) {
00201                 generateStaticStamp();
00202             }
00203             return staticStamp;
00204             break;
00205
00206         case SolutionStage::DynamicSolution:
00207             if (!dynamicStampIsFresh) {
00208                 generateDynamicStamp(solutionMatrix, currentSolutionIndex,
00209                                     timestep)
00210             }
00211             return dynamicStamp;
00212             break;
00213
00214         case SolutionStage::NonLinearSolution:
00215             if (!nonLinearStampIsFresh) {
00216                 generateNonLinearStamp(solutionMatrix, currentSolutionIndex,
00217                                     timestep)
00218             }
00219             return nonLinearStamp;
00220             break;
00221     }
00222 }
00223
00233 Stamp<T> &
00234 generateDCStamp(const Matrix<T> & solutionVector, size_t numCurrents) {

```

```

00235     dcStamp.clear();
00236
00237     for (const auto & component : staticElements) {
00238         dcStamp.addDCAnalysisStamp(component, solutionVector, numCurrents);
00239     }
00240
00241     for (const auto & component : dynamicElements) {
00242         dcStamp.addDCAnalysisStamp(component, solutionVector, numCurrents);
00243     }
00244
00245     for (const auto & component : nonLinearElements) {
00246         dcStamp.addDCAnalysisStamp(component, solutionVector, numCurrents);
00247     }
00248
00249     return dcStamp;
00250 }
00251
00259 void updateTimeStep(const Matrix<T> & solutionMatrix,
00260                      const size_t currentSolutionIndex, T timestep) {
00261     dynamicStampIsFresh = false;
00262     nonLinearStampIsFresh = false;
00263     for (const auto & component : dynamicElements) {
00264         component->updateStoredState(solutionMatrix, currentSolutionIndex,
00265                                         timestep, staticStamp.sizeG_A);
00266     }
00267     for (const auto & component : nonLinearElements) {
00268         component->updateStoredState(solutionMatrix, currentSolutionIndex,
00269                                         timestep, staticStamp.sizeG_A);
00270     }
00271 }
00272
00278 void updateDCStoredState(const Matrix<T> & solutionVector, size_t numCurrents) {
00279     for (const auto & component : staticElements) {
00280         component->updateDCStoredState(solutionVector, dcStamp.sizeG_A,
00281                                         numCurrents);
00282     }
00283     for (const auto & component : dynamicElements) {
00284         component->updateDCStoredState(solutionVector, dcStamp.sizeG_A,
00285                                         numCurrents);
00286     }
00287     for (const auto & component : nonLinearElements) {
00288         component->updateDCStoredState(solutionVector, dcStamp.sizeG_A,
00289                                         numCurrents);
00290     }
00291 }
00292 };
00293
00294 #endif

```

7.11 CircuitSimulator.cpp File Reference

```

#include <iostream>
#include "Maths/DynamicMatrix.hpp"
#include "CircuitElements/CircuitElements.hpp"
#include "CircuitSimulator/Simulator.hpp"
#include <thread>
#include <complex>
#include <chrono>
Include dependency graph for CircuitSimulator.cpp:

```



Functions

- int **main** (int argc, char *argv[])

main function to launch the circuit simulator

7.11.1 Function Documentation

7.11.1.1 main()

```
int main (
    int argc,
    char * argv[ ] )
```

main function to launch the circuit simulator

Contains platform specific code to handle directory changes

Parameters

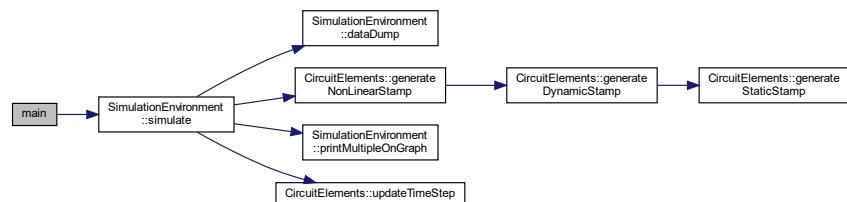
<i>argc</i>	
<i>argv[]</i>	

Returns

statuscode

Definition at line 29 of file [CircuitSimulator.cpp](#).

Here is the call graph for this function:



7.12 CircuitSimulator.cpp

```
00001 #ifdef WITH_PYTHON
00002 #include "matplotlibcpp.h"
00003 #endif
00004 #include <iostream>
00005 #include "Maths/DynamicMatrix.hpp"
00006 #include "CircuitElements/CircuitElements.hpp"
00007 #include "CircuitSimulator/Simulator.hpp"
00008 #include <thread>
00009 #include <complex>
00010 #include <chrono>
00011
00012 #ifdef _WIN32
00013 #include <windows.h>
00014 #include <direct.h>
00015 #include <shobjidl.h>
00016 #include <ShObjIdl_core.h>
00017 #endif
00018
```

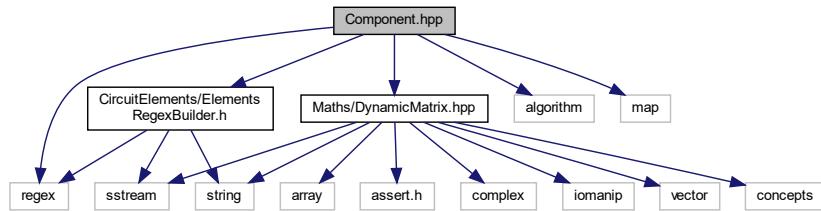
```

00019
00028 int
00029 main(int argc, char * argv[]) {
00030 #ifdef _WIN32
00031     // credit for this way to get exe path:
00032     // https://gist.github.com/karolisjan/f9b8ac3ae2d41ec0ce70f2feac6bdaf
00033     char buffer[MAX_PATH];
00034     GetModuleFileNameA(NULL, buffer, MAX_PATH);
00035     std::string::size_type pos = std::string(buffer).find_last_of("\\\\");
00036     _chdir(std::string(buffer).substr(0, pos).c_str());
00037
00038     // modification to get it as a wstring
00039     wchar_t bufferW[MAX_PATH];
00040     GetModuleFileNameW(NULL, bufferW, MAX_PATH);
00041     std::string::size_type posW = std::wstring(bufferW).find_last_of(L"\\\\");
00042     std::wstring exePathW = std::wstring(bufferW).substr(0, posW);
00043 #endif
00044
00045
00046     std::string filePath = "Netlists/Diode Test.netlist";
00047     if (argc > 1) {
00048         filePath = argv[1];
00049         std::cout << "Using netlist: " << filePath;
00050     } else {
00051 #ifdef _WIN32
00052         // example code from:
00053         // https://docs.microsoft.com/en-us/windows/win32/learnwin32/example--the-open-dialog-box
00054         // modified to use ascii
00055
00056         HRESULT hr = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED |
00057                                     COINIT_DISABLE_OLE1DDE);
00058         if (SUCCEEDED(hr)) {
00059             IFileOpenDialog * pFileOpen;
00060             IShellItem * defaultFolder;
00061
00062             // Create the FileOpenDialog object.
00063             hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_ALL,
00064                                   IID_IFileOpenDialog,
00065                                   reinterpret_cast<void **>(&pFileOpen));
00066             std::wstring defaultNetlistPath = exePathW + L"\\Netlists";
00067             SHCreateItemFromParsingName(defaultNetlistPath.c_str(), NULL,
00068                                         IID_IShellItem,
00069                                         reinterpret_cast<void **>(&defaultFolder));
00070             pFileOpen->SetDefaultFolder(defaultFolder);
00071             pFileOpen->SetFolder(defaultFolder);
00072
00073             if (SUCCEEDED(hr)) {
00074                 // Show the Open dialog box.
00075                 hr = pFileOpen->Show(NULL);
00076
00077                 // Get the file name from the dialog box.
00078                 if (SUCCEEDED(hr)) {
00079                     IShellItem * pItem;
00080                     hr = pFileOpen->GetResult(&pItem);
00081                     if (SUCCEEDED(hr)) {
00082                         PWSTR pszFilePath;
00083                         hr = pItem->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
00084
00085                         // Display the file name to the user.
00086                         if (SUCCEEDED(hr)) {
00087                             // MessageBoxW(NULL, pszFilePath, L"File Path", MB_OK);
00088                             WideCharToMultiByte(CP_UTF8, 0, pszFilePath, -1, buffer,
00089                                                 MAX_PATH, NULL, NULL);
00090                             filePath = std::string(buffer);
00091                             CoTaskMemFree(pszFilePath);
00092                         }
00093                         pItem->Release();
00094                     }
00095                 }
00096                 pFileOpen->Release();
00097             }
00098             CoUninitialize();
00099         }
00100         std::cout << "Using netlist: " << filePath;
00101     } else
00102         std::cout << "No path given. Defaulting to using netlist: " << filePath;
00103 #endif
00104     }
00105     std::cout << std::endl;
00106
00107     SimulationEnvironment<double> env(filePath);
00108
00109     env.simulate();
00110
00111     return 0;
00112 }
```

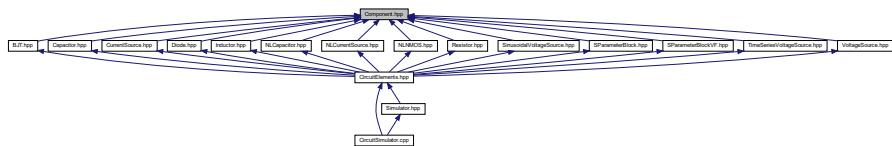
7.13 Component.hpp File Reference

```
#include "Maths/DynamicMatrix.hpp"
#include "CircuitElements/ElementsRegexBuilder.h"
#include <regex>
#include <algorithm>
#include <map>
```

Include dependency graph for Component.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct **Stamp< T >**
A helper struct to store the preallocated stamps for MNA.
- struct **Component< T >**
A template base class to define the fundamental things a component should define.

7.14 Component.hpp

```
00001 #ifndef _COMPONENT_HPP_INC_
00002 #define _COMPONENT_HPP_INC_
00003 #include "Maths/DynamicMatrix.hpp"
00004 #include "CircuitElements/ElementsRegexBuilder.h"
00005 #include <regex>
00006 #include <algorithm>
00007 #include <map>
00008
00009 template<typename T>
00010 struct Component;
00011
00022 template<typename T>
00023 struct Stamp {
00024     size_t sizeG_A;
00025     size_t sizeG_D;
00026
00027     Matrix<T> G;
00028     Matrix<T> s;
00029
00036     Stamp(size_t _sizeG_A, size_t _sizeG_D)
```

```

00037     : sizeG_A(_sizeG_A), sizeG_D(_sizeG_D),
00038     G(_sizeG_A + _sizeG_D, _sizeG_A + _sizeG_D, 0),
00039     s(_sizeG_A + _sizeG_D, 1, 0) {
00040 }
00041
00043 void clear() {
00044     G.fill(0);
00045     s.fill(0);
00046 }
00047
00052 void add(const Stamp<T> & rhs) {
00053     G.add(rhs.G, G);
00054     s.add(rhs.s, s);
00055 }
00056
00060 void addStaticStamp(const std::shared_ptr<Component<T> > & rhs) {
00061     rhs->addStaticStampTo(*this);
00062 }
00063
00067 void addDynamicStamp(const std::shared_ptr<Component<T> > & rhs,
00068                         const Matrix<T> & solutionMatrix,
00069                         const size_t currentSolutionIndex, T timestep) {
00070     rhs->addDynamicStampTo(*this, solutionMatrix, currentSolutionIndex,
00071                             timestep);
00072 }
00073
00077 void addNonLinearStamp(const std::shared_ptr<Component<T> > & rhs,
00078                         const Matrix<T> & solutionMatrix,
00079                         const size_t currentSolutionIndex, T timestep = 0) {
00080     rhs->addNonLinearStampTo(*this, solutionMatrix, currentSolutionIndex,
00081                             timestep);
00082 }
00083
00087 void
00088 addDCAnalysisStamp(const std::shared_ptr<Component<T> > & rhs,
00089                      const Matrix<T> & solutionMatrix, const size_t numCurrents) {
00090     rhs->addDCAnalysisStampTo(*this, solutionMatrix, numCurrents);
00091 }
00092
00096 Matrix<T> solve() {
00097     return G.leftDivide(s);
00098 }
00099 };
00100
00101 template<typename T>
00102 struct CircuitElements;
00107 template<typename T>
00108 struct Component {
00109     std::string designator = "";
00111
00115     virtual void addStaticStampTo(Stamp<T> & destination) const {
00116     }
00117
00124     virtual void
00125     addDynamicStampTo(Stamp<T> & destination, const Matrix<T> & solutionMatrix,
00126                         const size_t currentSolutionIndex, T timestep) const {
00127     }
00128
00135     virtual void
00136     addNonLinearStampTo(Stamp<T> & destination, const Matrix<T> & solutionMatrix,
00137                         const size_t currentSolutionIndex, T timestep = 0) const {
00138         throw std::exception("not implemented");
00139     }
00140
00148     virtual void updateStoredState(const Matrix<T> & solutionMatrix,
00149                                     const size_t currentSolutionIndex, T timestep,
00150                                     size_t numCurrents) {
00151     }
00152
00158     virtual void
00159     addDCAnalysisStampTo(Stamp<T> & destination, const Matrix<T> & solutionVector,
00160                          size_t numCurrents) const {
00161         throw std::exception("not implemented");
00162     }
00163
00170     virtual void updateDCStoredState(const Matrix<T> & solutionVector,
00171                                     size_t sizeG_A, size_t numCurrents) {
00172     }
00173
00177     virtual void setTimestep(T timestep) {
00178     }
00179
00188     static void
00189     addToElements(const std::string & line, CircuitElements<T> & elements,
00190                   size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00191         throw std::exception("not implemented");
00192     }

```

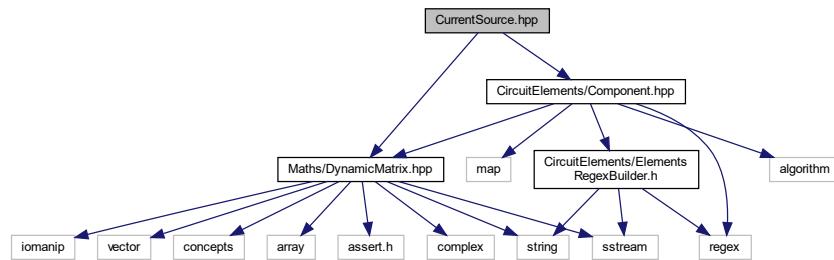
```

00193
00194     virtual ~Component() {};
00195 };
00196
00197 #endif

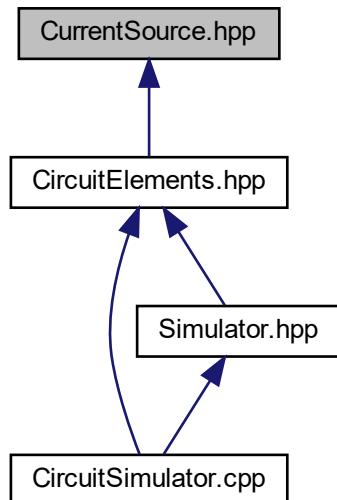
```

7.15 CurrentSource.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for CurrentSource.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `CurrentSource< T >`
- An ideal current source.*

7.16 CurrentSource.hpp

```

00001 #ifndef _CURRENTSOURCE_HPP_INC_
00002 #define _CURRENTSOURCE_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00009 template<typename T>
0010 struct CurrentSource : public Component<T> {
0011 public:
0012     T value = 0;
0013
0014     size_t n1 = 0;
0015     size_t n2 = 0;
0016
0017     void addStaticStampTo(Stamp<T> & stamp) const {
0018         // the p means prime (') and is used for the index - 1
0019         size_t nlp = n1 - 1;
0020         size_t n2p = n2 - 1;
0021
0022         if (n1) {
0023             stamp.s(nlp, 0) += -value;
0024         }
0025
0026         if (n2) {
0027             stamp.s(n2p, 0) += value;
0028         }
0029     }
0030
0031     void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
0032                             size_t numCurrents) const {
0033         addStaticStampTo(stamp);
0034     }
0035
0036     static void
0037     addToElements(const std::string & line, CircuitElements<T> & elements,
0038                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
0039         // std::regex currentSourceRegex( R"(^I(.*)\s(\d+)\s(\d+)\s(.+)\s?)" );
0040         std::regex currentSourceRegex = generateRegex("I", "n n w");
0041         CurrentSource<T> currentSource;
0042         std::smatch matches;
0043
0044         std::regex_match(line, matches, currentSourceRegex);
0045         currentSource.n1 = std::stoi(matches.str(2));
0046         currentSource.n2 = std::stoi(matches.str(3));
0047
0048         numNodes = std::max(numNodes, std::stoull(matches.str(2)));
0049         numNodes = std::max(numNodes, std::stoull(matches.str(3)));
0050
0051         if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
0052             currentSource.value = std::stod(matches.str(4));
0053         } else {
0054             static_assert("Unsupported Type");
0055         }
0056
0057         elements.staticElements.emplace_back(
0058             std::make_shared<CurrentSource<T>>(currentSource));
0059     }
0060 };
0061
0062 #endif

```

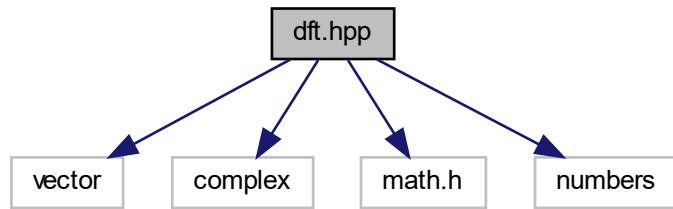
7.17 dft.hpp File Reference

```

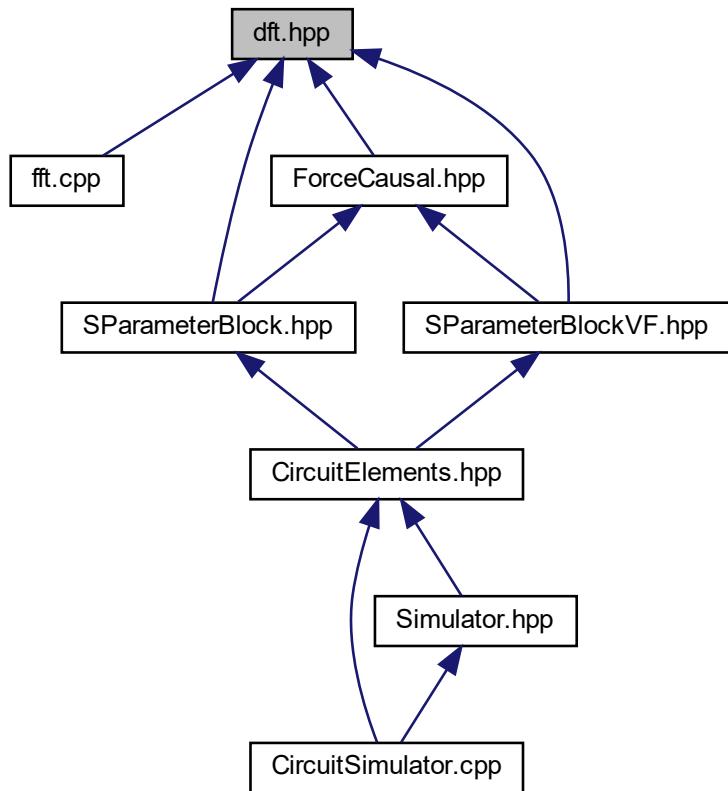
#include <vector>
#include <complex>
#include <math.h>
#include <numbers>

```

Include dependency graph for dft.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- template<typename T >
 `std::complex< T > nthRootOfUnity (T fraction)`

- template<typename T >
std::complex< T > **nthRootOfUnity** (int numerator, size_t denominator)
- template<typename T >
std::vector< std::complex< T > > **dft** (const std::vector< T > &inputData)
- template<typename T >
std::vector< std::complex< T > > **idft** (const std::vector< std::complex< T > > &inputData)
- template<typename T >
std::vector< std::complex< T > > **fft** (const std::vector< T > &inputData)
- template<typename T >
std::vector< std::complex< T > > **ifft** (const std::vector< std::complex< T > > &inputData)
- template<typename T , typename Iter , typename U = T, int dir = 1>
void **_fftHelperRadix2** (const std::vector< T > &inputData, Iter result, Iter scratch, size_t offset, size_t stride)

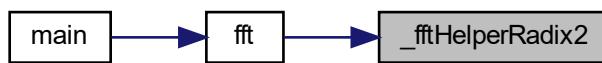
7.17.1 Function Documentation

7.17.1.1 _fftHelperRadix2()

```
template<typename T , typename Iter , typename U = T, int dir = 1>
void _fftHelperRadix2 (
    const std::vector< T > & inputData,
    Iter result,
    Iter scratch,
    size_t offset,
    size_t stride )
```

Definition at line 76 of file [dft.hpp](#).

Here is the caller graph for this function:



7.17.1.2 dft()

```
template<typename T >
std::vector<std::complex<T> > dft (
    const std::vector< T > & inputData )
```

Definition at line 24 of file [dft.hpp](#).

Here is the caller graph for this function:



7.17.1.3 fft()

```
template<typename T >
std::vector<std::complex<T> > fft (
    const std::vector< T > & inputData )
```

Definition at line 54 of file [dft.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

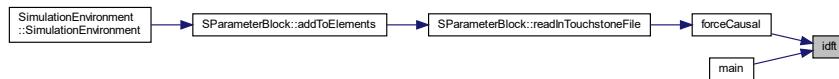


7.17.1.4 idft()

```
template<typename T >
std::vector<std::complex<T> > idft (
    const std::vector< std::complex< T > > & inputData )
```

Definition at line 37 of file [dft.hpp](#).

Here is the caller graph for this function:



7.17.1.5 ifft()

```
template<typename T >
std::vector<std::complex<T> > ifft (
    const std::vector< std::complex< T > > & inputData )
```

Definition at line 63 of file [dft.hpp](#).

Here is the caller graph for this function:



7.17.1.6 nthRootOfUnity() [1/2]

```
template<typename T >
std::complex<T> nthRootOfUnity (
    int numerator,
    size_t denominator )
```

Definition at line 16 of file [dft.hpp](#).

7.17.1.7 nthRootOfUnity() [2/2]

```
template<typename T >
std::complex<T> nthRootOfUnity (
    T fraction )
```

Definition at line 10 of file [dft.hpp](#).

7.18 dft.hpp

```
00001 #ifndef _FFT_H_INC_
00002 #define _FFT_H_INC_
00003 #include <vector>
00004 #include <complex>
00005 #include <math.h>
00006 #include <numbers>
00007
00008 template<typename T>
00009 std::complex<T>
0010 nthRootOfUnity(T fraction) {
0011     return std::exp(std::complex<T>(0, -2 * std::numbers::pi * fraction));
0012 }
0013
0014 template<typename T>
0015 std::complex<T>
0016 nthRootOfUnity(int numerator, size_t denominator) {
0017     return std::exp(std::complex<T>(0, -2 * std::numbers::pi * numerator /
0018                                         static_cast<double>(denominator)));
0019 }
0020
0021
0022 template<typename T>
0023 std::vector<std::complex<T> >
0024 dft(const std::vector<T> & inputData) {
0025     const size_t length = inputData.size();
0026     std::vector<std::complex<T> > toRet(length, std::complex<T>(0.0, 0.0));
0027     for (size_t k = 0; k < length; k++) {
0028         for (size_t n = 0; n < length; n++) {
0029             toRet[k] += inputData[n] * nthRootOfUnity<T>(k * n, length);
0030         }
0031     }
0032     return toRet;
0033 }
0034
0035 template<typename T>
0036 std::vector<std::complex<T> >
0037 idft(const std::vector<std::complex<T> > & inputData) {
0038     std::vector<std::complex<T> > toRet;
0039     const size_t length = inputData.size();
0040     toRet.resize(length);
0041     for (size_t n = 0; n < length; n++) {
0042         for (size_t k = 0; k < length; k++) {
0043             toRet[n] += inputData[k] *
0044                         nthRootOfUnity<T>(-static_cast<int>(k * n), length);
0045         }
0046         toRet[n] /= length;
0047     }
0048     return toRet;
0049 }
0050
0051
0052 template<typename T>
0053 std::vector<std::complex<T> >
0054 fft(const std::vector<T> & inputData) {
0055     std::vector<std::complex<T> > result(inputData.size(), std::complex<T>(0, 0));
0056     std::vector<std::complex<T> > scratch(inputData.size(), std::complex<T>(0, 0));
0057     _fftHelperRadix2(inputData, result.begin(), scratch.begin(), 0, 0);
0058     return result;
0059 }
0060
0061 template<typename T>
0062 std::vector<std::complex<T> >
0063 ifft(const std::vector<std::complex<T> > & inputData) {
0064     std::vector<std::complex<T> > result(inputData.size(), std::complex<T>(0, 0));
0065     std::vector<std::complex<T> > scratch(inputData.size(), std::complex<T>(0, 0));
0066     _fftHelperRadix2<std::complex<T>, decltype(result.begin()), T,
0067                 -1>(inputData, result.begin(), scratch.begin(), 0, 0);
0068     for (auto & num : result) {
0069         num /= result.size();
```

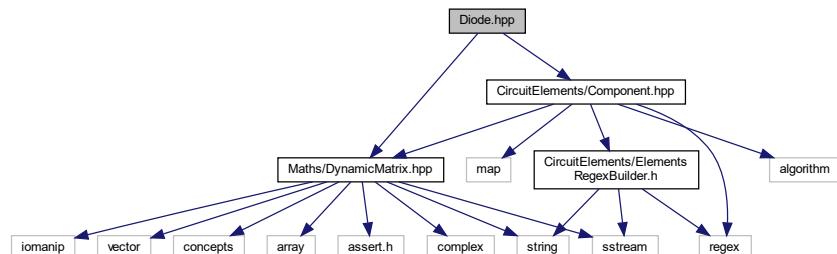
```

00070     }
00071     return result;
00072 }
00073
00074 template<typename T, typename Iter, typename U = T, int dir = 1>
00075 void
00076 _fftHelperRadix2(const std::vector<T> & inputData, Iter result, Iter scratch,
00077                      size_t offset, size_t stride) {
00078     const size_t len = (inputData.size() » stride);
00079     if (len > 2) {
00080         _fftHelperRadix2<T, Iter, U, dir>(inputData, scratch, result, offset,
00081                                         stride + 1);
00082
00083         _fftHelperRadix2<T, Iter, U, dir>(inputData, scratch + (len / 2),
00084                                         result + (len / 2), offset + (1 « stride),
00085                                         stride + 1);
00086
00087         for (size_t i = 0; i < len / 2; i++) {
00088             result[i] = scratch[i] +
00089                         nthRootOfUnity<U>(dir * i, len) * scratch[i + len / 2];
00090             result[i + len / 2] = scratch[i] - nthRootOfUnity<U>(dir * i, len) *
00091                                     scratch[i + len / 2];
00092         }
00093     } else {
00094         result[0] = inputData[offset] + inputData[offset + (1 « stride)];
00095         result[1] = inputData[offset] - inputData[offset + (1 « stride)];
00096     }
00097 }
00098
00099 #endif

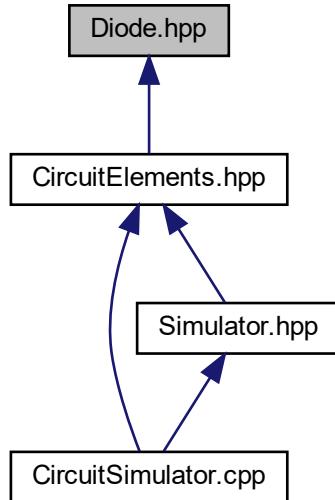
```

7.19 Diode.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for Diode.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `Diode< T >`
An ebbers moll diode model.

7.20 Diode.hpp

```

00001 #ifndef _DIODE_HPP_
00002 #define _DIODE_HPP_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00006
00010 template<typename T>
00011 struct Diode : public Component<T> {
00012     public:
00013         size_t n1 = 0;
00014         size_t n2 = 0;
00015
00016         const T I_sat = 2.52e-9;
00017         const T V_T = 25.8563e-3;
00018         const T eta = 2;
00019
00020         T V_crit = eta * V_T * std::log(eta * V_T / (I_sat * std::sqrt(2)));
00021
00022
00023     void
00024     addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00025                         const size_t currentSolutionIndex, T timestep = 0) const {
00026         const size_t n1p = n1 - 1;
00027         const size_t n2p = n2 - 1;
00028
00029         T v = 0;
00030
00031         if (n1 > 0) {
00032             v = solutionMatrix(n1p, currentSolutionIndex);
00033         }
00034
00035         if (n2 > 0) {
00036             v -= solutionMatrix(n2p, currentSolutionIndex);
  
```

```

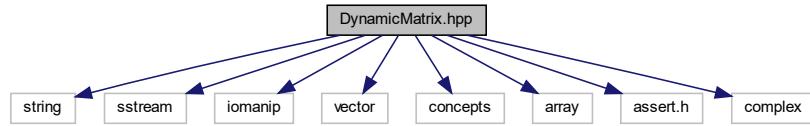
00037         }
00038
00039     v = std::min(V_crit, v);
00040
00041     T G_eq = (I_sat / (eta * V_T)) * std::exp(v / (eta * V_T));
00042     T I_eq = I_sat * (std::exp(v / (eta * V_T)) - 1) - G_eq * v;
00043
00044     if (n1 > 0) {
00045         stamp.G(nlp, nlp) += G_eq;
00046         stamp.s(nlp, 0) += -I_eq;
00047     }
00048
00049     if (n2 > 0) {
00050         stamp.G(n2p, n2p) += G_eq;
00051         stamp.s(n2p, 0) += +I_eq;
00052     }
00053
00054     if (n1 > 0 && n2 > 0) {
00055         stamp.G(nlp, n2p) += -G_eq;
00056         stamp.G(n2p, nlp) += -G_eq;
00057     }
00058 }
00059
00060 void updateStoredState(const Matrix<T> & solutionMatrix,
00061                         const size_t currentSolutionIndex, T timestep,
00062                         size_t sizeG_A) {
00063 }
00064
00065 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00066                           size_t numCurrents) const {
00067     addNonLinearStampTo(stamp, solutionVector, 0, 0);
00068 }
00069
00070 static void
00071 addToElements(const std::string & line, CircuitElements<T> & elements,
00072                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00073     std::regex diodeRegex = generateRegex("D", "n n");
00074     Diode<T> diode;
00075     std::smatch matches;
00076
00077     std::regex_match(line, matches, diodeRegex);
00078
00079     diode.designator = "D";
00080     diode.designator += matches.str(1);
00081
00082     diode.n1 = std::stoi(matches.str(2));
00083     diode.n2 = std::stoi(matches.str(3));
00084
00085     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00086     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00087
00088     elements.nonLinearElements.emplace_back(std::make_shared<Diode<T>>(diode));
00089     elements.nodeComponentMap.insert(
00090         {{diode.n1, elements.nonLinearElements.back()},
00091          {diode.n2, elements.nonLinearElements.back()}});
00092 }
00093 };
00094
00095 #endif

```

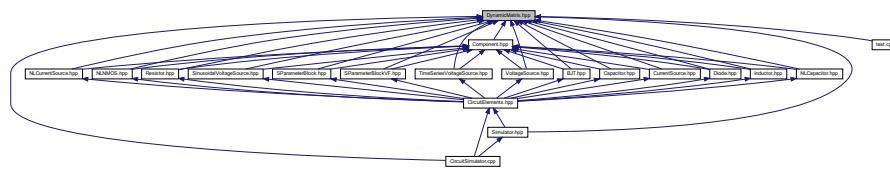
7.21 DynamicMatrix.hpp File Reference

```
#include <string>
#include <sstream>
#include <iomanip>
#include <vector>
#include <concepts>
#include <array>
#include <assert.h>
#include <complex>
```

Include dependency graph for DynamicMatrix.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct `Matrix< T >`
A matrix class with support for LU-decomposition, and left division.
- struct `LUPair< T >`
A helper class to store the L U and pivot matrices. Mainly useful in solving systems of equations.

Functions

- template<typename T >
`std::complex< double > operator>` (const `std::complex< T >` &a, const `std::complex< T >` &b)
- template<typename T >
`std::complex< double > operator<` (const `std::complex< T >` &a, const `std::complex< T >` &b)
- template<typename T >
`std::complex< double > operator>=` (const `std::complex< T >` &a, const `std::complex< T >` &b)
- template<typename T >
`std::complex< double > operator<=` (const `std::complex< T >` &a, const `std::complex< T >` &b)

Variables

- template<typename T >
concept `arithmetic`

7.21.1 Function Documentation

7.21.1.1 **operator<()**

```
template<typename T >
std::complex<double> operator< (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 14 of file [DynamicMatrix.hpp](#).

7.21.1.2 **operator<=()**

```
template<typename T >
std::complex<double> operator<= (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 26 of file [DynamicMatrix.hpp](#).

7.21.1.3 **operator>()**

```
template<typename T >
std::complex<double> operator> (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 14 of file [DynamicMatrix.hpp](#).

7.21.1.4 **operator>=()**

```
template<typename T >
std::complex<double> operator>= (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 26 of file [DynamicMatrix.hpp](#).

7.21.2 Variable Documentation

7.21.2.1 arithmetic

```
template<typename T >
concept arithmetic
```

Initial value:

```
= requires(T a, T b) {
    {a == b};
    {a != b};
    {a >= b};
    {a <= b};
    {a * b};
    {a / b};
    {a + b};
    {a - b};
}
```

Definition at line 37 of file [DynamicMatrix.hpp](#).

7.22 DynamicMatrix.hpp

```
00001 #ifndef _MATRIX_HPP_INC_
00002 #define _MATRIX_HPP_INC_
00003 #include <string>
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <vector>
00007 #include <concepts>
00008 #include <array>
00009 #include <assert.h>
00010 #include <complex>
00011
00012 template<typename T>
00013 std::complex<double>
00014 operator>(const std::complex<T> & a, const std::complex<T> & b) {
00015     return std::norm(a) > std::norm(b);
00016 }
00017
00018 template<typename T>
00019 std::complex<double>
00020 operator<(const std::complex<T> & a, const std::complex<T> & b) {
00021     return std::norm(a) < std::norm(b);
00022 }
00023
00024 template<typename T>
00025 std::complex<double>
00026 operator>=(const std::complex<T> & a, const std::complex<T> & b) {
00027     return std::norm(a) >= std::norm(b);
00028 }
00029
00030 template<typename T>
00031 std::complex<double>
00032 operator<=(const std::complex<T> & a, const std::complex<T> & b) {
00033     return std::norm(a) <= std::norm(b);
00034 }
00035
00036 template<typename T>
00037 concept arithmetic = requires(T a, T b) {
00038     {a == b};
00039     {a != b};
00040     {a >= b};
00041     {a <= b};
00042     {a * b};
00043     {a / b};
00044     {a + b};
00045     {a - b};
00046 };
00047
00048 template<typename T>
00049 struct LUPair;
00050
00054 template<typename T>
00055 requires arithmetic<T> struct Matrix {
00056     std::vector<T> data;
00057     size_t M;
00058     size_t N;
00059
00060     Matrix(size_t M, size_t N) : M(M), N(N) {
00061         data.resize(M * N);
```

```

00062     }
00063
00064     Matrix(size_t M, size_t N, T initialValue) : M(M), N(N) {
00065         data.resize(M * N, initialValue);
00066     }
00067
00068     T & operator()(size_t m, size_t n) {
00069         return data[m * N + n];
00070     }
00071
00072     const T & operator()(size_t m, size_t n) const {
00073         return data[m * N + n];
00074     }
00075
00076     void fill(T fillVal) {
00077         for (auto & entry : data) {
00078             entry = fillVal;
00079         }
00080     }
00081
00082     void rowAddition(size_t destinationRow, size_t sourceRow, T scalingFactor) {
00083         assert(0 <= destinationRow && destinationRow <= N);
00084         assert(0 <= sourceRow && sourceRow <= M);
00085         for (size_t n = 0; n < N; n++) {
00086             data[destinationRow * N + n] += scalingFactor * data[sourceRow * N + n];
00087         }
00088     }
00089
00090     void swapRows(size_t row1, size_t row2) {
00091         std::swap_ranges(data.begin() + row1 * N, data.begin() + (row1 + 1) * N,
00092                           data.begin() + row2 * N);
00093     }
00094
00095     Matrix<T> transpose() {
00096         Matrix<T> toRet(N, M);
00097         for (size_t m = 0; m < M; m++) {
00098             for (size_t n = 0; n < N; n++) {
00099                 toRet.data[n * M + m] = data[m * N + n];
00100             }
00101         }
00102         return toRet;
00103     }
00104
00105     Matrix<T> multiply(const Matrix<T> & rhs) const {
00106         assert(N == rhs.M);
00107         Matrix<T> toRet(M, rhs.N);
00108         multiply(rhs, toRet, 0);
00109         return toRet;
00110     }
00111
00112     void multiply(const Matrix<T> & rhs, Matrix<T> & dest) const {
00113         // for SIMD/cache reasons, the order of operations is slightly weird. This is
00114         // to minimise row changes which may cause cache misses. This is due to the
00115         // fact that in this model rows are represented contiguously in memory, so
00116         // streaming instructions and local caching can be used to our advantage
00117         //
00118         // It is also worth stating that there is strong potential for multithreading
00119         // here, and especially if paired with the Strassen Method for matrix
00120         // multiplication. However for small sizes, naive multiplication can be better
00121         // due to less memory allocations
00122
00123         for (size_t m = 0; m < M; m++) {
00124             for (size_t k = 0; k < N; k++) {
00125                 for (size_t n = 0; n < dest.N; n++) {
00126                     dest.data[m * destination.N + n] += data[m * N + k] *
00127                                         rhs.data[k * N + n];
00128                 }
00129             }
00130         }
00131     }
00132
00133     Matrix<T> add(const Matrix<T> & rhs) const {
00134         assert(N == rhs.N && M == rhs.M);
00135         Matrix<T, M, N> toRet(M, N);
00136         add(rhs, toRet);
00137         return toRet;
00138     }
00139
00140     void add(const Matrix<T> & rhs, Matrix<T> & dest) const {
00141         for (size_t m = 0; m < M; m++) {
00142             for (size_t n = 0; n < N; n++) {
00143                 dest.data[m * N + n] = data[m * N + n] + rhs.data[m * N + n];
00144             }
00145         }
00146     }
00147
00148     Matrix<T> subtract(const Matrix<T> & rhs) const {

```

```

00149     assert(N == rhs.N && M == rhs.M);
00150     Matrix<T> toRet(M, N);
00151     add(rhs, toRet);
00152     return toRet;
00153 }
00154
00155 void subtract(const Matrix<T> & rhs, Matrix<T> & dest) const {
00156     for (size_t m = 0; m < M; m++) {
00157         for (size_t n = 0; n < N; n++) {
00158             dest.data[m * N + n] = data[m * N + n] - rhs.data[m * N + n];
00159         }
00160     }
00161 }
00162
00163 std::string toString() const {
00164     std::stringstream toRet;
00165     for (size_t m = 0; m < M; m++) {
00166         for (size_t n = 0; n < N; n++) {
00167             toRet << std::setw(5) << std::setprecision(2) << data[m * N + n]
00168             << " ";
00169         }
00170         toRet << std::endl;
00171     }
00172 }
00173
00174     return toRet.str();
00175 }
00176
00177 LUPair<T> luPair() const {
00178     assert(N == M);
00179
00180     LUPair<T> toRet(M);
00181     luPair(toRet);
00182     return toRet;
00183 }
00184
00185 void luPair(LUPair<T> & dest) const {
00186     dest.u = *this;
00187     dest.l.fill(0.0);
00188     for (size_t n = 0; n < N; n++) {
00189         dest.l.data[n * N + n] = 1.0;
00190         dest.p[n] = n;
00191     }
00192
00193     for (size_t r = 0; r < M - 1; r++) {
00194         // find largest in column
00195         size_t largestRow = r;
00196         auto maxV = abs(dest.u.data[r * N + r]);
00197         for (size_t r2 = r + 1; r2 < M; r2++) {
00198             if (abs(dest.u.data[r2 * N + r]) > maxV) {
00199                 maxV = abs(dest.u.data[r2 * N + r]);
00200                 largestRow = r2;
00201             }
00202         }
00203
00204         // swap rows in U and indices in p
00205         dest.u.swapRows(r, largestRow);
00206         std::swap(dest.p[r], dest.p[largestRow]);
00207         // swap subdiagonal entries in L
00208         for (size_t n = 0; n < r; n++) {
00209             std::swap(dest.l.data[r * N + n], dest.l.data[largestRow * N + n]);
00210         }
00211
00212         // Gaussian elimination
00213         for (size_t m = r + 1; m < M; m++) {
00214             // TODO: potential for multithreading here
00215             // need to take into account how many processors there are
00216             // https://en.cppreference.com/w/cpp/thread/thread/hardware\_concurrency
00217             T multiplier = dest.u.data[m * N + r] / dest.u.data[r * N + r];
00218             dest.u.rowAddition(m, r, -multiplier);
00219             dest.l.data[m * N + r] = multiplier;
00220         }
00221         // std::cout << "After Gaussian\n" << dest.toString();
00222     }
00223 }
00224
00225 Matrix<T> leftDivide(const Matrix<T> & rhs) const {
00226     auto lu = luPair();
00227     Matrix<T> scratchSpace(M, 1);
00228     Matrix<T> toRet(M, 1);
00229     leftDivide(rhs, lu, scratchSpace, toRet);
00230     return toRet;
00231 }
00232
00233 void leftDivide(const Matrix<T> & rhs, const LUPair<T> & lu,
00234                   Matrix<T> & scratchSpace, Matrix<T> & dest) const {
00235     leftDivide(rhs, lu, scratchSpace, dest.data.begin(), dest.data.end());

```

```

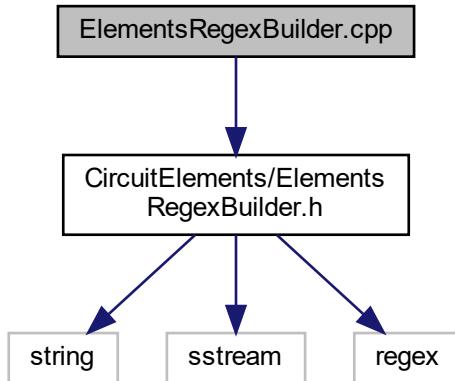
00236     }
00237
00238     template<typename Iterator>
00239     void
00240     leftDivide(const Matrix<T> & rhs, const LUPair<T> & lu, Matrix<T> & scratchSpace,
00241                 Iterator destBegin, Iterator destEnd) const {
00242         assert(destEnd - destBegin == scratchSpace.M);
00243         for (size_t m = 0; m < M; m++) {
00244             destBegin[m] = rhs.data[lu.p[m]];
00245         }
00246
00247         // scratchSpace: solve LY = Pb for y using substitution
00248         for (size_t m = 0; m < M; m++) {
00249             T val = destBegin[m];
00250             for (size_t n = 0; n < m; n++) {
00251                 val -= scratchSpace.data[n] * lu.l.data[m * N + n];
00252             }
00253             scratchSpace.data[m] = val / lu.l.data[m * N + m];
00254         }
00255
00256         // stage2: solve Ux = Y for x using substitution
00257         for (size_t m = 0; m < M; m++) {
00258             T val = scratchSpace.data[M - m - 1];
00259             for (size_t n = 0; n < m; n++) {
00260                 val -= destBegin[(M - n - 1)] *
00261                     lu.u.data[(M - m - 1) * N + N - n - 1];
00262             }
00263             destBegin[M - m - 1] = val / lu.u.data[(M - m - 1) * N + M - m - 1];
00264         }
00265     }
00266 };
00267
00272 template<typename T>
00273 struct LUPair {
00274     Matrix<T> l;
00275     Matrix<T> u;
00276     std::vector<size_t> p;
00277     size_t M;
00278
00279     LUPair(size_t _M) : l(_M, _M, 0), u(_M, _M), p(_M), M(_M) {
00280     }
00281
00282     LUPair(const LUPair<T> & other)
00283         : l(other.l), u(other.u), p(other.p), M(other.M) {
00284     }
00285
00286
00287     std::string toString() {
00288         std::stringstream toRet;
00289         toRet << "U\n" << u.toString();
00290         toRet << "L\n" << l.toString();
00291         toRet << "p\n";
00292         for (size_t i = 0; i < p.size(); i++) {
00293             toRet << std::setw(5) << p[i] << " ";
00294         }
00295         toRet << std::endl;
00296         return toRet.str();
00297     }
00298 };
00299 // -----
00300 // ----- Implementation
00301 // -----
00302
00303
00304 #endif

```

7.23 ElementsRegexBuilder.cpp File Reference

```
#include "CircuitElements/ElementsRegexBuilder.h"
```

Include dependency graph for ElementsRegexBuilder.cpp:



Functions

- std::regex [generateRegex](#) (std::string identifier, std::string simplifiedMatching, bool startAnchor, bool endAnchor)

a helper function to aid in the construction of regexes for parsing netlist files

7.23.1 Function Documentation

7.23.1.1 generateRegex()

```
std::regex generateRegex (
    std::string identifier,
    std::string simplifiedMatching,
    bool startAnchor = true,
    bool endAnchor = true )
```

a helper function to aid in the construction of regexes for parsing netlist files

Parameters

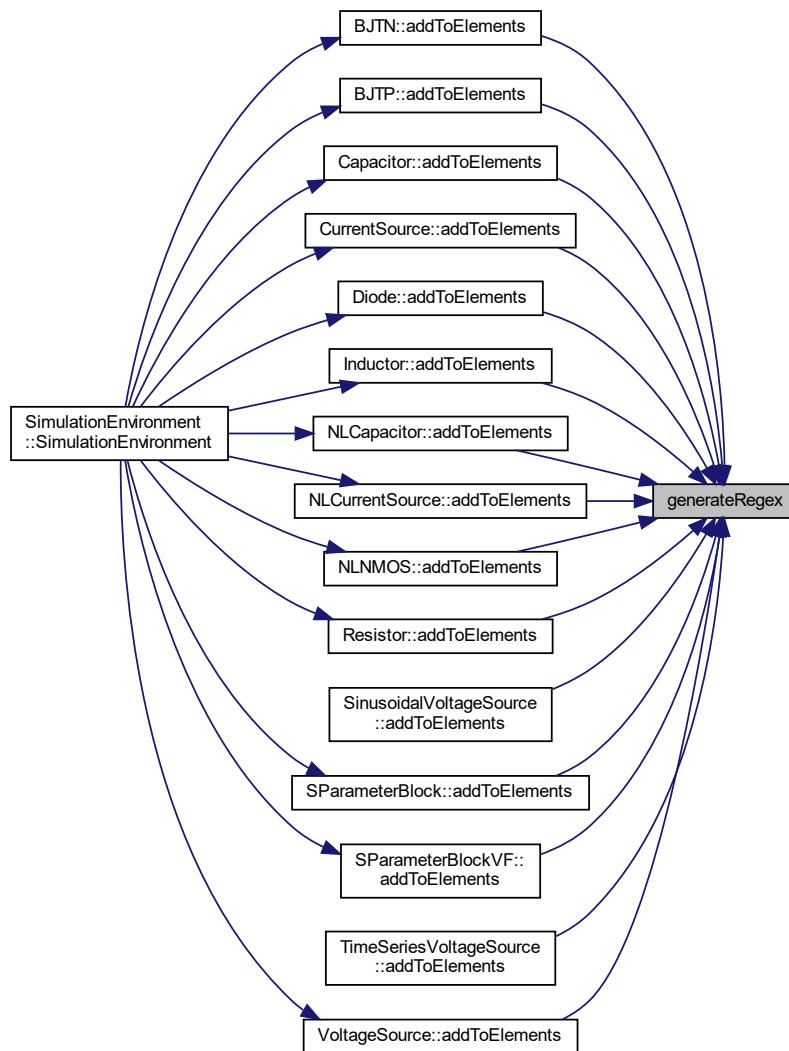
<i>identifier</i>	The designator for the component. e.g. "R" for resistor
<i>simplifiedMatching</i>	The string that contains the data for what we want matched. For example: n = int/size_t w = word (works for floats etc) ? = everything after is optional c = char s = space
<i>startAnchor</i>	What prepends the regex, defaults to "^".
<i>endAnchor</i>	What appends the regex, defaults to "\$".

Returns

The complete Regex.

Definition at line 22 of file [ElementsRegexBuilder.cpp](#).

Here is the caller graph for this function:



7.24 ElementsRegexBuilder.cpp

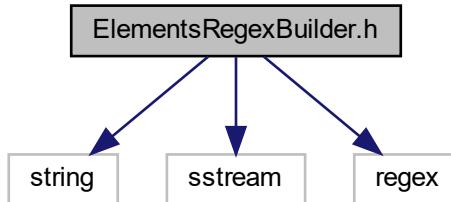
```

00001 #include "CircuitElements/ElementsRegexBuilder.h"
00002 // identifier is the sequence of letters representing the component
00003 // simplifiedMatching:
00004 // a string detailing how many, and the order of size_ts, and Values, and strings, and when options
00005 // start
00006 // startAnchor "^"
00007 // endAnchor "$"
00008 // n = int/size_t
00009 // w = word (works for floats etc)
00010 // ? = everything after is optional
  
```

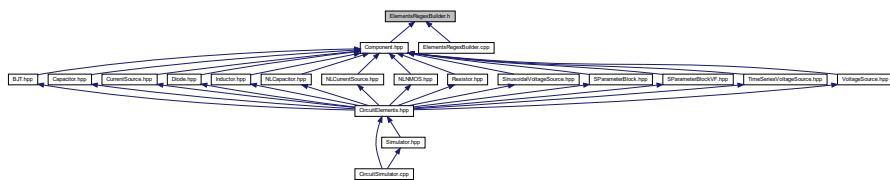
```
00011 // c = char
00012 // s = space
00013 //
00014 // Capacitor example:
00015 // ( "C", "n n w" )
00016 // Resistor example:
00017 // ( "R", "n n w ? w" )
00018 //
00019 //
00020
00021 std::regex
00022 generateRegex( std::string identifier, std::string simplifiedMatching,
00023                 bool startAnchor,
00024                 bool endAnchor ) {
00025     constexpr char spaceRegex[] = R"(\s)";
00026     constexpr char startAnchorRegex[] = R"(^)";
00027     constexpr char endAnchorRegex[] = R"(\s?$)";
00028     constexpr char optionalSpaceRegex[] = R"(\s?)";
00029
00030     constexpr char charRegex[] = R"((.))";
00031     constexpr char size_tRegex[] = R"((\d+))";
00032     constexpr char emptyWordRegex[] = R"((.*?))";
00033     constexpr char wordRegex[] = R"((.+?))";
00034     constexpr char optionalCharRegex[] = R"((?:\s(.))?)";
00035     constexpr char optionalWordRegex[] = R"((?:\s(.+?))?)";
00036     constexpr char optionalSize_tRegex[] = R"((?:\s(\d+?))?)";
00037
00038     std::string built( " " );
00039
00040     if ( startAnchor ) {
00041         built += startAnchorRegex;
00042     }
00043
00044     built += identifier + emptyWordRegex;
00045
00046     bool option = false;
00047     for ( char letter : simplifiedMatching ) {
00048         switch( letter ) {
00049             case 'n':
00050                 if ( option ) {
00051                     built += optionalSize_tRegex;
00052                 } else {
00053                     built += spaceRegex;
00054                     built += size_tRegex;
00055                 }
00056                 break;
00057             case 'w':
00058                 if ( option ) {
00059                     built += optionalWordRegex;
00060                 } else {
00061                     built += spaceRegex;
00062                     built += wordRegex;
00063                 }
00064                 break;
00065             case 'c':
00066                 if ( option ) {
00067                     built += optionalCharRegex;
00068                 } else {
00069                     built += spaceRegex;
00070                     built += charRegex;
00071                 }
00072                 break;
00073             case 's':
00074                 if ( option ) {
00075                     built += optionalSpaceRegex;
00076                 } else {
00077                     built += spaceRegex;
00078                 }
00079                 break;
00080             case '?':
00081                 option = true;
00082                 break;;
00083             case ' ':
00084                 break;
00085             default:
00086                 break;
00087         }
00088     }
00089
00090     if ( endAnchor ) {
00091         built += endAnchorRegex;
00092     }
00093
00094     return std::regex( built.c_str() );
00095 }
```

7.25 ElementsRegexBuilder.h File Reference

```
#include <string>
#include <sstream>
#include <regex>
Include dependency graph for ElementsRegexBuilder.h:
```



This graph shows which files directly or indirectly include this file:



Functions

- `std::regex generateRegex (std::string identifier, std::string simplifiedMatching, bool startAnchor=true, bool endAnchor=true)`
a helper function to aid in the construction of regexes for parsing netlist files

7.25.1 Function Documentation

7.25.1.1 generateRegex()

```
std::regex generateRegex (
    std::string identifier,
    std::string simplifiedMatching,
    bool startAnchor = true,
    bool endAnchor = true )
```

a helper function to aid in the construction of regexes for parsing netlist files

Parameters

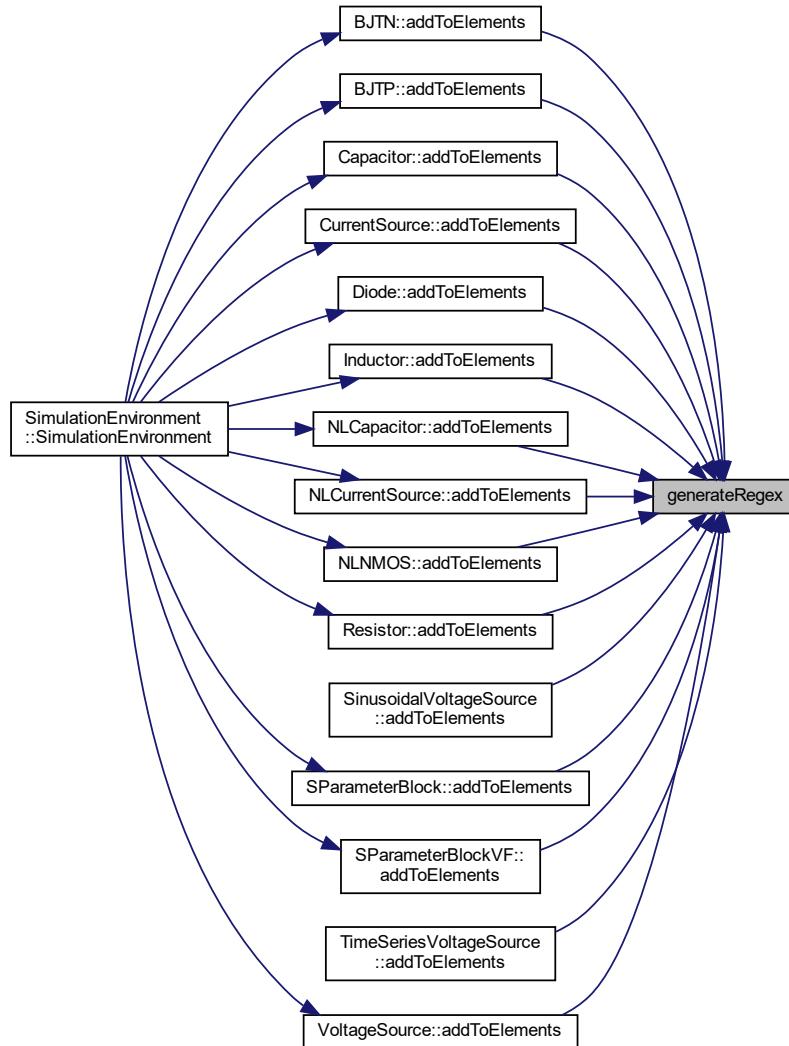
<i>identifier</i>	The designator for the component. e.g. "R" for resistor
<i>simplifiedMatching</i>	The string that contains the data for what we want matched. For example: n = int/size_t w = word (works for floats etc) ? = everything after is optional c = char s = space
<i>startAnchor</i>	What prepends the regex, defaults to "^".
<i>endAnchor</i>	What appends the regex, defaults to "\$".

Returns

The complete Regex.

Definition at line 22 of file [ElementsRegexBuilder.cpp](#).

Here is the caller graph for this function:



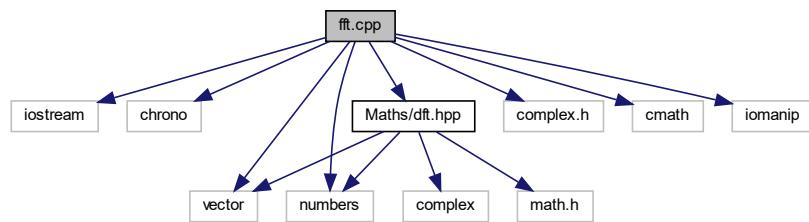
7.26 ElementsRegexBuilder.h

```

00001 #ifndef _ELEMENTSREGEXBUILDER_HPP_
00002 #define _ELEMENTSREGEXBUILDER_HPP_
00003 #include <string>
00004 #include <sstream>
00005 #include <regex>
00006
00007 // identifier is the sequence of letters representing the component
00008 // simplifiedMatching:
00009 // a string detailing how many, and the order of size_ts, and Values, and strings,
00010 // and when options start startAnchor "^" endAnchor "$"
00026 std::regex
00027 generateRegex(std::string identifier, std::string simplifiedMatching,
00028                 bool startAnchor = true, bool endAnchor = true);
00029
00030 #endif
  
```

7.27 fft.cpp File Reference

```
#include <iostream>
#include <chrono>
#include <vector>
#include <complex.h>
#include <cmath>
#include <numbers>
#include <iomanip>
#include "Maths/dft.hpp"
Include dependency graph for fft.cpp:
```



Functions

- double [myFunctionToSample \(double t\)](#)
- int [main \(\)](#)

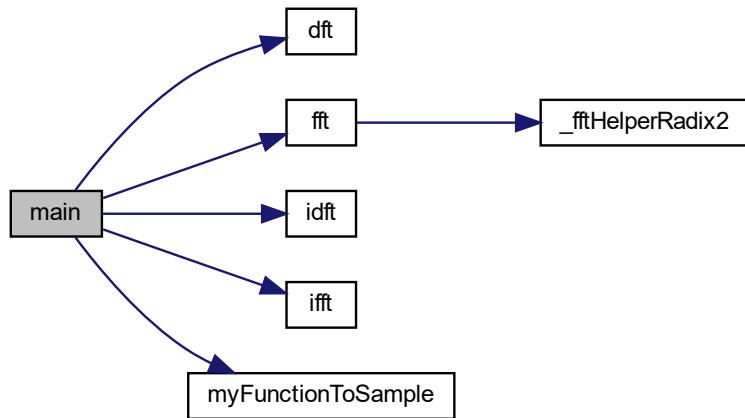
7.27.1 Function Documentation

7.27.1.1 main()

```
int main ( )
```

Definition at line 18 of file [fft.cpp](#).

Here is the call graph for this function:



7.27.1.2 myFunctionToSample()

```
double myFunctionToSample (
    double t )
```

Definition at line 11 of file [fft.cpp](#).

Here is the caller graph for this function:



7.28 fft.cpp

```
00001 #include <iostream>
00002 #include <chrono>
00003 #include <vector>
00004 #include <complex.h>
00005 #include <cmath>
00006 #include <numbers>
00007 #include <iomanip>
00008 #include "Maths/dft.hpp"
00009
00010 double
00011 myFunctionToSample(double t) {
00012     constexpr double freqSine = 1;
```

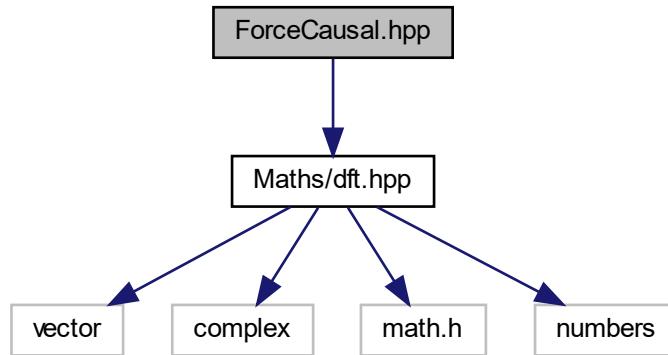
```

00013     return std::sin(2 * std::numbers::pi * freqSine * t);
00014 }
00015
00016
00017 [[clang::optnone]] int
00018 main() {
00019     // generate data
00020     constexpr double T = 2;
00021     constexpr double f_s = 8;
00022     constexpr size_t numPoints = T * f_s;
00023
00024
00025     std::vector<double> inputData; // = { 0, 1, 0, -1, 0, 1, 0, -1 };
00026     for (int i = 0; i < numPoints; i++) {
00027         inputData.push_back(myFunctionToSample(static_cast<double>(i) / f_s));
00028     }
00029
00030     // print data as csv
00031     for (const auto & num : inputData) {
00032         std::cout << std::setprecision(5) << num << ", ";
00033     }
00034     std::cout << std::endl;
00035     std::cout << std::endl;
00036     std::cout << std::endl;
00037
00038     auto dftres = dft(inputData);
00039     auto fftres = fft(inputData);
00040
00041     for (const auto & num : dftres) {
00042         std::cout << std::setprecision(5) << std::abs(num) << ", ";
00043     }
00044     std::cout << std::endl;
00045     std::cout << std::endl;
00046     std::cout << std::endl;
00047
00048     for (const auto & num : fftres) {
00049         std::cout << std::setprecision(5) << std::abs(num) << ", ";
00050     }
00051     std::cout << std::endl;
00052     std::cout << std::endl;
00053     std::cout << std::endl;
00054
00055
00056     auto idftres = idft(dftres);
00057     for (const auto & num : idftres) {
00058         std::cout << std::setprecision(5) << std::real(num) << ", ";
00059     }
00060     std::cout << std::endl;
00061     std::cout << std::endl;
00062     std::cout << std::endl;
00063
00064     auto ifftres = ifft(dftres);
00065     for (const auto & num : ifftres) {
00066         std::cout << std::setprecision(5) << std::real(num) << ", ";
00067     }
00068     std::cout << std::endl;
00069     std::cout << std::endl;
00070     std::cout << std::endl;
00071     return 0;
00072 }
```

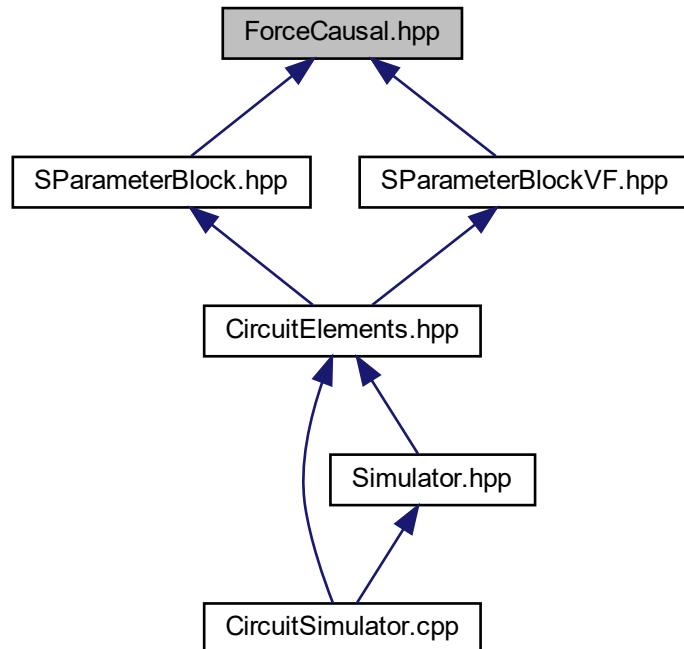
7.29 ForceCausal.hpp File Reference

```
#include "Maths/dft.hpp"
```

Include dependency graph for ForceCausal.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct `ForceCausal::CausalData< T >`
a helper struct to return the result of the forced causal IDFT

Namespaces

- `ForceCausal`

Functions

- template<typename T >
`std::complex< T > ForceCausal::F (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau, T k, size_t n)`
- template<typename T >
`T ForceCausal::K (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau)`
- template<typename T >
`T ForceCausal::f0 (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau)`
- template<typename T >
`T ForceCausal::f0derivative (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tau, T step)`
- template<typename T >
`T ForceCausal::getTau (const std::vector< T > &freq, const std::vector< std::complex< T > > &data, T tol=1e-7, size_t maxIter=30, T step=1e-8)`
- template<typename T >
`ForceCausal::CausalData< T > forceCausal (const std::vector< T > &freq, const std::vector< std::complex< T > > &data)`

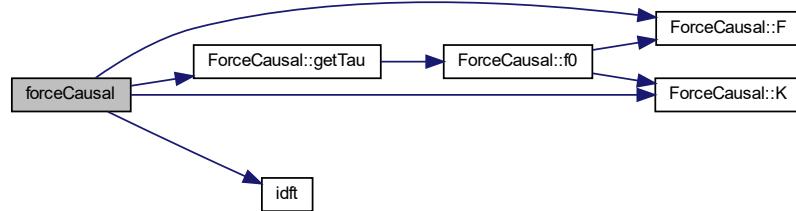
7.29.1 Function Documentation

7.29.1.1 forceCausal()

```
template<typename T >
ForceCausal::CausalData<T> forceCausal (
    const std::vector< T > & freq,
    const std::vector< std::complex< T > > & data )
```

Definition at line 78 of file [ForceCausal.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.30 ForceCausal.hpp

```

00001 #ifndef _FORCECAUSAL_HPP_INC_
00002 #define _FORCECAUSAL_HPP_INC_
00003 #include "Maths/dft.hpp"
00004
00005 namespace ForceCausal {
00006
00007 // TODO: Maybe this could be made more efficient by worrying about vectorisation
00008 // of the calculations. I.E storing precomputed F values etc
00009
00010 template<typename T>
00011 std::complex<T>
00012 F(const std::vector<T> & freq, const std::vector<std::complex<T> > & data, T tau,
00013     T k, size_t n) {
00014     return (data[n] - k) *
00015         exp(std::complex<T>(0, -2 * std::numbers::pi * freq[n] * tau));
00016 }
00017
00018 template<typename T>
00019 T
00020 K(const std::vector<T> & freq, const std::vector<std::complex<T> > & data, T tau) {
00021     return std::real(data.back()) -
00022         std::imag(data.back()) /
00023         std::tan(2 * std::numbers::pi * freq.back() * tau);
00024 }
00025
00026 template<typename T>
00027 T
00028 f0(const std::vector<T> & freq, const std::vector<std::complex<T> > & data, T tau) {
00029     std::complex<T> toRet = 0;
00030     T k = K(freq, data, tau);
00031     for (size_t i = 1; i < freq.size() - 1; i++) {
00032         toRet += 2 * (std::real(F(freq, data, tau, k, i)));
00033     }
00034     toRet += F(freq, data, tau, k, 0);
00035     toRet += std::real(F(freq, data, tau, k, freq.size() - 1));
00036     toRet *= 1e3 / (2 * freq.size() - 2);
00037     return std::real(toRet);
00038 }
00039
00040 template<typename T>
00041 T
00042 f0derivative(const std::vector<T> & freq, const std::vector<std::complex<T> > & data,
00043                 T tau, T step) {
00044     return (f0(freq, data, tau + step) - f0(freq, data, tau)) / step;
00045 }
00046
00047
00048 template<typename T>
00049 T
00050 getTau(const std::vector<T> & freq, const std::vector<std::complex<T> > & data,
00051           T tol = 1e-7, size_t maxIter = 30, T step = 1e-8) {
00052     T currentGuess = 1e-8;
00053     for (size_t i = 0; i < maxIter; i++) {
00054         T f0Curr = f0(freq, data, currentGuess);
00055         if (f0Curr * f0Curr < tol) {
00056             break;
00057         }
00058         T diff = (f0(freq, data, currentGuess + step) - f0Curr) / step;
00059         currentGuess = currentGuess - f0Curr / diff;
00060     }
00061     return currentGuess;
00062 }
00063
00064 template<typename T>
00065 struct CausalData {
00066     T tau;
00067     T Ts;
00068     std::vector<T> data;
00069 };
00070
00071 } // namespace ForceCausal
00072
00073
00074 } // namespace ForceCausal
00075
00076 template<typename T>
00077 ForceCausal::CausalData<T>
00078 forceCausal(const std::vector<T> & freq,
00079               const std::vector<std::complex<T> > & data) {
00080     ForceCausal::CausalData<T> toRet;
00081     toRet.data = std::vector<T>(2 * freq.size() - 2);
00082     toRet.Ts = 1.0 / (toRet.data.size() * (freq[1] - freq[0]));
00083     // Add in conjugate Symmetric Data
00084     std::vector<std::complex<T> > hermitianData(2 * freq.size() - 2);
00085     T k = 0;
00086
00087     if (abs(std::imag(data.back())) < 1e-5) {
00088         toRet.tau = 0;

```

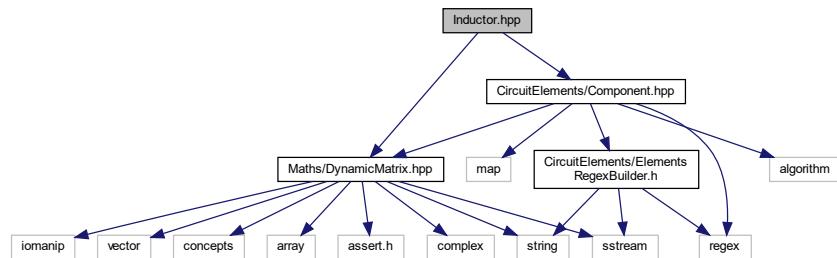
```

00089     for (size_t i = 0; i < freq.size() - 1; i++) {
00090         hermitianData[i] = data[i];
00091     }
00092
00093     for (size_t i = 1; i < freq.size(); i++) {
00094         hermitianData[hermitianData.size() - i] = std::conj(data[i]);
00095     }
00096 }
00097
00098 } else {
00099     toRet.tau = ForceCausal::getTau(freq, data);
00100     k = ForceCausal::K(freq, data, toRet.tau);
00101
00102     for (size_t i = 0; i < freq.size() - 1; i++) {
00103         hermitianData[i] = ForceCausal::F(freq, data, toRet.tau, k, i);
00104     }
00105
00106     for (size_t i = 1; i < freq.size(); i++) {
00107         hermitianData[hermitianData.size() - i] = std::conj(
00108             ForceCausal::F(freq, data, toRet.tau, k, i));
00109     }
00110 }
00111
00112 auto idftVal = idft(hermitianData);
00113 for (size_t i = 0; i < idftVal.size(); i++) {
00114     toRet.data[i] = std::real(idftVal[i]);
00115 }
00116
00117 if (abs(std::imag(data.back())) >= 1e-5) {
00118     toRet.data[0] = k;
00119 }
00120 return toRet;
00121 }
00122
00123
00124 #endif

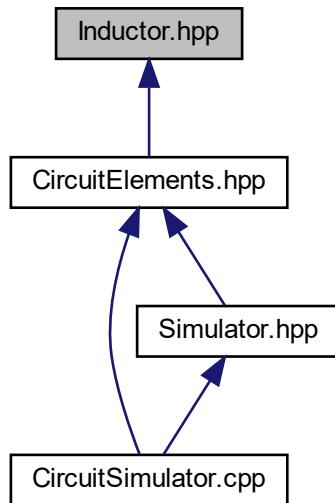
```

7.31 Inductor.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for Inductor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `Inductor< T >`
An ideal inductor model.

7.32 Inductor.hpp

```

00001 #ifndef _INDUCTOR_HPP_INC_
00002 #define _INDUCTOR_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00006
00010 template<typename T>
00011 struct Inductor : public Component<T> {
00012     public:
00013         T value = 0;
00014
00015         size_t n1 = 0;
00016         size_t n2 = 0;
00017         T lastCurrent = 0;
00018
00019         size_t dcCurrentIndex = 0;
00020
00021         bool trapezoidalRule = true;
00022         void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00023                               const size_t currentSolutionIndex, T timestep) const {
00024             size_t nlp = n1 - 1;
00025             size_t n2p = n2 - 1;
00026
00027             T u0 = 0;
00028             if (n1) {
00029                 u0 = solutionMatrix(nlp, currentSolutionIndex - 1);
00030             }
00031
00032             if (n2) {
00033                 u0 -= solutionMatrix(n2p, currentSolutionIndex - 1);
00034             }
00035
00036
  
```

```

00037     T G_eq = 0;
00038     T I_eq = 0;
00039
00040     if (trapezoidalRule) {
00041         G_eq = timestep / (2 * value);
00042         I_eq = lastCurrent + G_eq * u0;
00043     } else {
00044         G_eq = timestep / timestep;
00045         I_eq = lastCurrent;
00046     }
00047
00048     if (n1) {
00049         stamp.G(nlp, nlp) += G_eq;
00050         stamp.s(nlp, 0) += -I_eq;
00051     }
00052
00053     if (n2) {
00054         stamp.G(n2p, n2p) += G_eq;
00055         stamp.s(n2p, 0) += I_eq;
00056     }
00057
00058     if (n1 && n2) {
00059         stamp.G(nlp, n2p) += -G_eq;
00060         stamp.G(n2p, nlp) += -G_eq;
00061     }
00062 }
00063
00064 void updateStoredState(const Matrix<T> & solutionMatrix,
00065                         const size_t currentSolutionIndex, T timestep,
00066                         size_t sizeG_A) {
00067     size_t nlp = n1 - 1;
00068     size_t n2p = n2 - 1;
00069     T u0 = 0;
00070     T ul = 0;
00071     if (n1) {
00072         u0 = solutionMatrix(nlp, currentSolutionIndex - 1);
00073         ul = solutionMatrix(nlp, currentSolutionIndex);
00074     }
00075
00076     if (n2) {
00077         u0 -= solutionMatrix(n2p, currentSolutionIndex - 1);
00078         ul -= solutionMatrix(n2p, currentSolutionIndex);
00079     }
00080
00081     if (trapezoidalRule) {
00082         T G_eq = timestep / (2 * value);
00083         lastCurrent = G_eq * ul + (lastCurrent + G_eq * u0);
00084     } else {
00085         T G_eq = timestep / value;
00086         lastCurrent = G_eq * ul + lastCurrent;
00087     }
00088 }
00089
00090 void updateDCStoredState(const Matrix<T> & solutionVector, size_t sizeG_A,
00091                         size_t numCurrents) {
00092     lastCurrent = solutionVector(sizeG_A + numCurrents + dccurrentIndex - 1, 0);
00093 }
00094
00095 void addDCAAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00096                           size_t numCurrents) const {
00097     // Short circuit
00098     size_t nlp = n1 - 1;
00099     size_t n2p = n2 - 1;
00100     size_t dcCurrentIndexp = dccurrentIndex - 1;
00101
00102     if (n1 > 0) {
00103         stamp.G(nlp, stamp.sizeG_A + numCurrents + dcCurrentIndexp) += 1;
00104         stamp.G(stamp.sizeG_A + numCurrents + dcCurrentIndexp, nlp) += 1;
00105     }
00106
00107     if (n2 > 0) {
00108         stamp.G(n2p, stamp.sizeG_A + numCurrents + dcCurrentIndexp) += -1;
00109         stamp.G(stamp.sizeG_A + numCurrents + dcCurrentIndexp, n2p) += -1;
00110     }
00111 }
00112
00113 static void
00114 addToElements(const std::string & line, CircuitElements<T> & elements,
00115               size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00116     // std::regex inductorRegex(R"^(L(.+?)\s(\d+)\s(\d+)\s(.+)\s)$");
00117     std::regex inductorRegex = generateRegex("L", "n n w");
00118     Inductor<T> inductor;
00119     std::smatch matches;
00120
00121     std::regex_match(line, matches, inductorRegex);
00122     inductor.designator = "L";

```

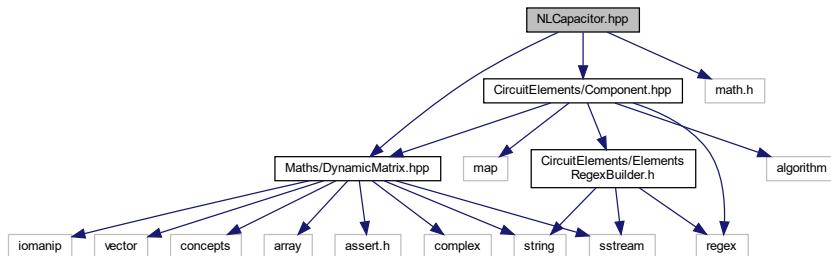
```

00124     inductor.designator += matches.str(1);
00125
00126     inductor.n1 = std::stoi(matches.str(2));
00127     inductor.n2 = std::stoi(matches.str(3));
00128     inductor.trapezoidalRule = true;
00129     inductor.dcCurrentIndex = ++numDCCurrents;
00130
00131     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00132     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00133
00134     if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00135         inductor.value = std::stod(matches.str(4));
00136     } else {
00137         static_assert("Unsupported Type");
00138     }
00139
00140     elements.dynamicElements.emplace_back(
00141         std::make_shared<Inductor<T>>(inductor));
00142
00143     elements.nodeComponentMap.insert(
00144         {{inductor.n1, elements.dynamicElements.back()},
00145          {inductor.n2, elements.dynamicElements.back()}});
00146 }
00147 };
00148
00149 #endif

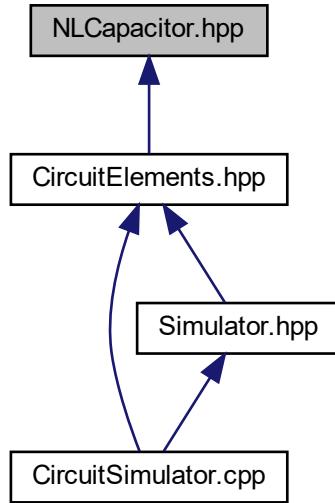
```

7.33 NLCapacitor.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include <math.h>
Include dependency graph for NLCapacitor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `NLCapacitor< T >`
*a non-linear capacitor model of the form $C = C_p + C_o * (1.0 + \tanh(P_{10} + P_{11} * u))$*

7.34 NLCapacitor.hpp

```

00001 #ifndef _NLCAPACITOR_HPP_
00002 #define _NLCAPACITOR_HPP_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005 #include <math.h>
00006
00010 template<typename T>
00011 struct NLCapacitor : public Component<T> {
00012     public:
00013         size_t n1 = 0;
00014         size_t n2 = 0;
00015
00016         T C_p = 0;
00017         T C_o = 0;
00018         T P_10 = 0;
00019         T P_11 = 0;
00020
00021         T u_last = 0;
00022         T i_last = 0;
00023
00024         T C_last = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u_last));
00025
00026     void
00027     addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00028                         const size_t currentSolutionIndex, T timestep = 0) const {
00029         const size_t n1p = n1 - 1;
00030         const size_t n2p = n2 - 1;
00031
00032         T u = 0;
00033
00034         if (n1 > 0) {
00035             u = solutionMatrix(n1p, currentSolutionIndex);
00036         }
  
```

```

00037
00038     if (n2 > 0) {
00039         u -= solutionMatrix(n2p, currentSolutionIndex);
00040     }
00041
00042     T C = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u));
00043
00044     T dC = C_o * P_11 / std::pow(std::cosh(P_10 + P_11 * u), 2);
00045
00046     T i = C * (2.0 * (u - u_last) / timestep - i_last / C_last);
00047
00048     T di = dC * (2.0 * (u - u_last) / timestep - i_last / C_last) +
00049         2.0 * C / timestep;
00050
00051     T G_eq = di;
00052
00053     T I_eq = -G_eq * u + i;
00054
00055     if (n1 > 0) {
00056         stamp.G(nlp, nlp) += G_eq;
00057         stamp.s(nlp, 0) += -I_eq;
00058
00059         if (n2 > 0) {
00060             stamp.G(nlp, n2p) += -G_eq;
00061         }
00062     }
00063
00064     if (n2 > 0) {
00065         stamp.G(n2p, n2p) += G_eq;
00066         stamp.s(n2p, 0) += +I_eq;
00067
00068         if (n1 > 0) {
00069             stamp.G(n2p, nlp) += -G_eq;
00070         }
00071     }
00072 }
00073
00074 void updateStoredState(const Matrix<T> & solutionMatrix,
00075                         const size_t currentSolutionIndex, T timestep,
00076                         size_t sizeG_A) {
00077     const size_t nlp = n1 - 1;
00078     const size_t n2p = n2 - 1;
00079
00080     T u = 0;
00081
00082     if (n1 > 0) {
00083         u = solutionMatrix(nlp, currentSolutionIndex);
00084     }
00085
00086     if (n2 > 0) {
00087         u -= solutionMatrix(n2p, currentSolutionIndex);
00088     }
00089
00090     T C = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u));
00091
00092     i_last = C * (2.0 * (u - u_last) / timestep - i_last / C_last);
00093
00094     C_last = C;
00095
00096     u_last = u;
00097 }
00098
00099 void updateDCStoredState(const Matrix<T> & solutionVector, size_t sizeG_A,
00100                         size_t numCurrents) {
00101     const size_t nlp = n1 - 1;
00102     const size_t n2p = n2 - 1;
00103
00104     T u = 0;
00105
00106     if (n1 > 0) {
00107         u = solutionVector(nlp, 0);
00108     }
00109
00110     if (n2 > 0) {
00111         u -= solutionVector(n2p, 0);
00112     }
00113
00114     T C = C_p + C_o * (1.0 + std::tanh(P_10 + P_11 * u));
00115
00116     i_last = 0;
00117
00118     C_last = C;
00119
00120     u_last = u;
00121 }
00122
00123 void addDCAalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,

```

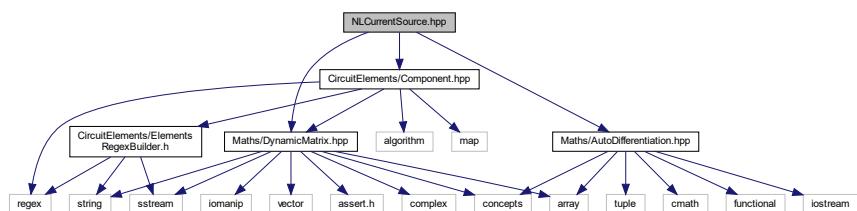
```

00124             size_t numCurrents) const {
00125     // open circuit
00126     if (n1 > 0) {
00127         stamp.G(n1 - 1, n1 - 1) += 1e-9;
00128     }
00129     if (n2 > 0) {
00130         stamp.G(n2 - 1, n2 - 1) += 1e-9;
00131     }
00132 }
00133
00134 static void
00135 addToElements(const std::string & line, CircuitElements<T> & elements,
00136                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00137     std::regex capacitorRegex = generateRegex("CN", "n n w w w w");
00138     NLCapacitor<T> cap;
00139     std::smatch matches;
00140
00141     std::regex_match(line, matches, capacitorRegex);
00142
00143     cap.designator = "CN";
00144     cap.designator += matches.str(1);
00145
00146     cap.n1 = std::stoi(matches.str(2));
00147     cap.n2 = std::stoi(matches.str(3));
00148
00149     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00150     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00151
00152     if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00153         cap.C_p = std::stod(matches.str(4));
00154         cap.C_o = std::stod(matches.str(5));
00155         cap.P_10 = std::stod(matches.str(6));
00156         cap.P_11 = std::stod(matches.str(7));
00157         cap.C_last = cap.C_p +
00158             cap.C_o *
00159             (1.0 + std::tanh(cap.P_10 + cap.P_11 * cap.u_last));
00160     } else {
00161         static_assert("Unsupported Type");
00162     }
00163
00164
00165     elements.nonLinearElements.emplace_back(
00166         std::make_shared<NLCapacitor<T>>(cap));
00167     elements.nodeComponentMap.insert(
00168         {{cap.n1, elements.nonLinearElements.back()},
00169          {cap.n2, elements.nonLinearElements.back()}});
00170 }
00171 };
00172 #endif

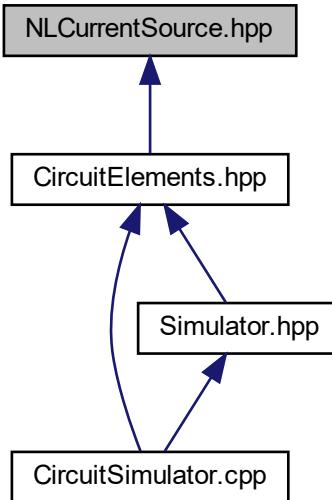
```

7.35 NLCurrentSource.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include "Maths/AutoDifferentiation.hpp"
Include dependency graph for NLCurrentSource.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [NLCURRENTSOURCE< T >](#)
An non-linear current source for the COBRA transistor model.

7.36 NLCURRENTSOURCE.hpp

```

00001 #ifndef _NLCURRENTSOURCE_HPP_
00002 #define _NLCURRENTSOURCE_HPP_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005 #include "Maths/AutoDifferentiation.hpp"
00006
00010 template<typename T>
00011 struct NLCURRENTSOURCE : public Component<T> {
00012 public:
00013     T value = 0;
00014
00015     size_t n1 = 0;
00016     size_t n2 = 0;
00017
00018     size_t r1_pos = 0;
00019     size_t r1_neg = 0;
00020     size_t r2_pos = 0;
00021     size_t r2_neg = 0;
00022
00023     void
00024     addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00025                           const size_t currentSolutionIndex, T timestep = 0) const {
00026         constexpr T alpha = 1.3;
00027         constexpr T beta0 = 0.42;
00028         constexpr T gamma = 0.0005;
00029         constexpr T delta = 0.3;
00030         constexpr T xi = 0.06;
00031         constexpr T lambda = 1.5;
00032         constexpr T mu = 0.0;
00033         constexpr T zeta = 0.18;
00034         constexpr T Vto = -2.4;
00035
00036         size_t nlp = n1 - 1;
  
```

```

00037     size_t n2p = n2 - 1;
00038     size_t rlp_pos = r1_pos - 1;
00039     size_t rlp_neg = r1_neg - 1;
00040     size_t r2p_pos = r2_pos - 1;
00041     size_t r2p_neg = r2_neg - 1;
00042
00043     T u = 0;
00044     T r1 = 0;
00045     T r2 = 0;
00046
00047     if (n1 > 0) {
00048         u = solutionMatrix(nlp, currentSolutionIndex);
00049     }
00050
00051     if (n2 > 0) {
00052         u -= solutionMatrix(n2p, currentSolutionIndex);
00053     }
00054
00055     if (rl_pos > 0) {
00056         r1 = solutionMatrix(rlp_pos, currentSolutionIndex);
00057     }
00058
00059     if (rl_neg > 0) {
00060         r1 -= solutionMatrix(rlp_neg, currentSolutionIndex);
00061     }
00062
00063     if (r2_pos > 0) {
00064         r2 = solutionMatrix(r2p_pos, currentSolutionIndex);
00065     }
00066
00067     if (r2_neg > 0) {
00068         r2 -= solutionMatrix(r2p_neg, currentSolutionIndex);
00069     }
00070
00071     namespace AD = AutoDifferentiation;
00072
00073     using ADT = AD::DiffVar<T, 2>;
00074     ADT V_gs(r1, 1, 0);
00075     ADT V_ds(r2, 0, 1);
00076
00077     auto beta = beta0;
00078     auto Vgst = V_gs - (1 + beta * beta) * Vto + gamma * V_ds;
00079     auto Veff = 0.5 * (Vgst + ADT::sqrt(ADT::pow(Vgst, 2) + delta * delta));
00080     auto power = lambda / (1 + mu * ADT::pow(V_ds, 2) + xi * Veff);
00081     auto area = alpha * V_ds * (1 + zeta * Veff);
00082     auto f1 = ADT::tanh(area);
00083     auto Ids_lim = beta * ADT::pow(Veff, power);
00084     auto Idrain = Ids_lim * f1;
00085     auto I_ds = Idrain[0] - Idrain[1] * r1 - Idrain[2] * r2;
00086
00087     if (n1 > 0) {
00088         stamp.s(nlp, 0) += -I_ds;
00089         if (rl_pos > 0) {
00090             stamp.G(nlp, rlp_pos) += Idrain[1];
00091         }
00092         if (rl_neg > 0) {
00093             stamp.G(nlp, rlp_neg) += -Idrain[1];
00094         }
00095         if (r2_pos > 0) {
00096             stamp.G(nlp, r2p_pos) += Idrain[2];
00097         }
00098         if (r2_neg > 0) {
00099             stamp.G(nlp, r2p_neg) += -Idrain[2];
00100         }
00101     }
00102
00103     if (n2 > 0) {
00104         stamp.s(n2p, 0) += +I_ds;
00105         if (rl_pos > 0) {
00106             stamp.G(n2p, rlp_pos) += -Idrain[1];
00107         }
00108         if (rl_neg > 0) {
00109             stamp.G(n2p, rlp_neg) += Idrain[1];
00110         }
00111         if (r2_pos > 0) {
00112             stamp.G(n2p, r2p_pos) += -Idrain[2];
00113         }
00114         if (r2_neg > 0) {
00115             stamp.G(n2p, r2p_neg) += Idrain[2];
00116         }
00117     }
00118 }
00119
00120 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00121                           size_t numCurrents) const {
00122     addNonLinearStampTo(stamp, solutionVector, 0, 0);
00123 }
```

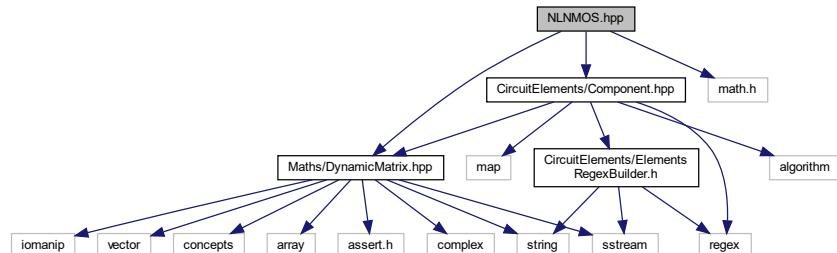
```

00124
00125
00126     static void
00127         addToElements(const std::string & line, CircuitElements<T> & elements,
00128             size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00129             std::regex currentSourceRegex = generateRegex("I", "n n n n n n");
00130             NLCurrentSource<T> currentSource;
00131             std::smatch matches;
00132
00133             std::regex_match(line, matches, currentSourceRegex);
00134             currentSource.n1 = std::stoi(matches.str(2));
00135             currentSource.n2 = std::stoi(matches.str(3));
00136             currentSource.r1_pos = std::stoi(matches.str(4));
00137             currentSource.r1_neg = std::stoi(matches.str(5));
00138             currentSource.r2_pos = std::stoi(matches.str(6));
00139             currentSource.r2_neg = std::stoi(matches.str(7));
00140
00141             numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00142             numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00143             numNodes = std::max(numNodes, std::stoull(matches.str(4)));
00144             numNodes = std::max(numNodes, std::stoull(matches.str(5)));
00145             numNodes = std::max(numNodes, std::stoull(matches.str(6)));
00146             numNodes = std::max(numNodes, std::stoull(matches.str(7)));
00147
00148             elements.nonLinearElements.emplace_back(
00149                 std::make_shared<NLCurrentSource<T>>(currentSource));
00150         }
00151     };
00152
00153 #endif

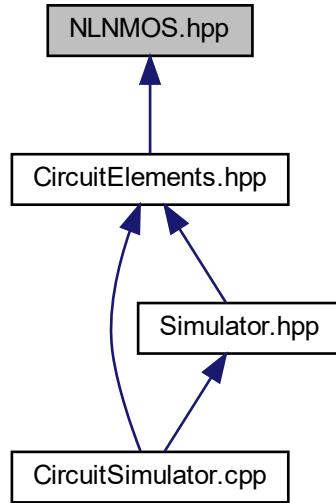
```

7.37 NLNMOS.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include <math.h>
Include dependency graph for NLNMOS.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct **NLN莫斯< T >**
a non-linear FET model

7.38 NLNMOS.hpp

```

00001 #ifndef _NLNMOS_HPP_INC_
00002 #define _NLNMOS_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005 #include <math.h>
00006
00009 template<typename T>
00010 struct NLNMOS : public Component<T> {
00011     public:
00012         size_t d = 0;
00013         size_t g = 0;
00014         size_t s = 0;
00015
00016     // constant params of the model
00017     const T C_GSp = 0.01;
00018     const T C_GSo = 0.5;
00019     const T P_S10 = 0;
00020     const T P_S11 = 0.5;
00021     const T C_GDp = 0.5;
00022     const T C_GDo = 1;
00023     const T P_D10 = -1;
00024     const T P_D11 = 0.4;
00025
00026     const T beta_DS = 1.3;
00027     const T alpha_DS = 0.42;
00028
00029     T u_gd_last = 0;
00030     T u_gs_last = 0;
00031
00032
00033     T i_gd_last = 0;
00034     T i_gs_last = 0;
00035
  
```

```

00036 T C_GD_last = C_GDp + C_GDo * (1.0 + std::tanh(P_D10 + P_D11 * u_gd_last));
00037 T C_GS_last = C_GSp + C_GSo * (1.0 + std::tanh(P_S10 + P_S11 * u_gs_last));
00038
00039 void
00040 addNonLinearStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00041                         const size_t currentSolutionIndex, T timestep = 0) const {
00042     const size_t gp = g - 1;
00043     const size_t dp = d - 1;
00044     const size_t sp = s - 1;
00045
00046     T u_gs = 0;
00047     T u_gd = 0;
00048
00049     if (g > 0) {
00050         u_gs = solutionMatrix(gp, currentSolutionIndex);
00051         u_gd = solutionMatrix(gp, currentSolutionIndex);
00052     }
00053
00054     if (s > 0) {
00055         u_gs -= solutionMatrix(sp, currentSolutionIndex);
00056     }
00057
00058     if (d > 0) {
00059         u_gd -= solutionMatrix(dp, currentSolutionIndex);
00060     }
00061
00062     T C_GD = C_GDp + C_GDo * (1.0 + std::tanh(P_D10 + P_D11 * u_gd));
00063     T C_GS = C_GSp + C_GSo * (1.0 + std::tanh(P_S10 + P_S11 * u_gs));
00064
00065     T dC_GD = C_GDo * P_D11 / std::pow(std::cosh(P_D10 + P_D11 * u_gd), 2);
00066     T dC_GS = C_GSo * P_S11 / std::pow(std::cosh(P_S10 + P_S11 * u_gs), 2);
00067
00068     T i_ds = beta_DS * std::tanh(alpha_DS * (u_gs - u_gd));
00069     T di_ds_d = -beta_DS * alpha_DS /
00070                 std::pow(std::cosh(alpha_DS * (u_gs - u_gd)), 2);
00071     T di_ds_s = beta_DS * alpha_DS /
00072                 std::pow(std::cosh(alpha_DS * (u_gs - u_gd)), 2);
00073
00074     T i_gd = C_GD *
00075             (2.0 * (u_gd - u_gd_last) / timestep - i_gd_last / C_GD_last);
00076     T i_gs = C_GS *
00077             (2.0 * (u_gs - u_gs_last) / timestep - i_gs_last / C_GS_last);
00078
00079     T i_d = -i_gd + i_ds;
00080     T i_s = -i_gs - i_ds;
00081     T i_g = i_gs + i_gd;
00082
00083     T di_gd = dC_GD *
00084             (2.0 * (u_gd - u_gd_last) / timestep - i_gd_last / C_GD_last) +
00085             2.0 * C_GD / timestep;
00086     T di_gs = dC_GS *
00087             (2.0 * (u_gs - u_gs_last) / timestep - i_gs_last / C_GS_last) +
00088             2.0 * C_GS / timestep;
00089
00090     T g_dd = -di_gd + di_ds_d;
00091     T g_sd = -di_ds_d;
00092     T g_gd = di_gd;
00093
00094     T g_ds = di_ds_d;
00095     T g_ss = -di_gs - di_ds_s;
00096     T g_gs = di_gs;
00097
00098     T I_d = i_d - g_dd * u_gd - g_ds * u_gs;
00099     T I_s = i_s - g_sd * u_gd - g_ss * u_gs;
00100     T I_g = i_g - g_gd * u_gd - g_gs * u_gs;
00101
00102     if (d > 0) {
00103         stamp.G(dp, dp) += -g_dd;
00104         stamp.s(dp, 0) += -I_d;
00105
00106         if (s > 0) {
00107             stamp.G(dp, sp) += -g_ds;
00108         }
00109
00110         if (g > 0) {
00111             stamp.G(dp, gp) += g_dd + g_ds;
00112         }
00113     }
00114
00115     if (s > 0) {
00116         stamp.G(sp, sp) += -g_ss;
00117         stamp.s(sp, 0) += -I_s;
00118
00119         if (d > 0) {
00120             stamp.G(sp, dp) += -g_sd;
00121         }
00122     }

```

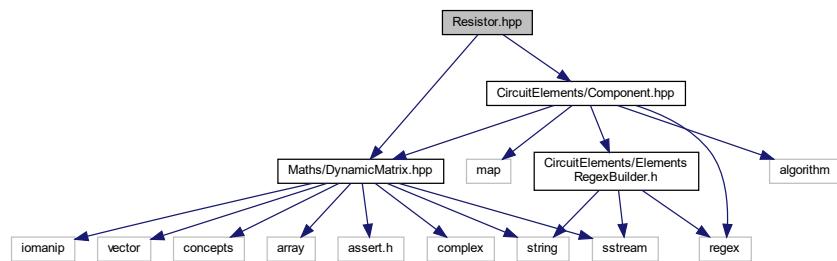
```

00123         if (g > 0) {
00124             stamp.G(sp, gp) += g_sd + g_ss;
00125         }
00126     }
00127
00128     if (g > 0) {
00129         stamp.G(gp, gp) += g_gd + g_gs;
00130         stamp.s(gp, 0) += -I_g;
00131
00132         if (d > 0) {
00133             stamp.G(gp, dp) += -g_gd;
00134         }
00135
00136         if (s > 0) {
00137             stamp.G(gp, sp) += -g_gs;
00138         }
00139     }
00140 }
00141
00142 void updateStoredState(const Matrix<T> & solutionMatrix,
00143                         const size_t currentSolutionIndex, T timestep,
00144                         size_t sizeG_A) {
00145     const size_t gp = g - 1;
00146     const size_t dp = d - 1;
00147     const size_t sp = s - 1;
00148
00149     T u_gs = 0;
00150     T u_gd = 0;
00151
00152     if (g > 0) {
00153         u_gs = solutionMatrix(gp, currentSolutionIndex);
00154         u_gd = solutionMatrix(gp, currentSolutionIndex);
00155     }
00156
00157     if (s > 0) {
00158         u_gs -= solutionMatrix(sp, currentSolutionIndex);
00159     }
00160
00161     if (d > 0) {
00162         u_gd -= solutionMatrix(dp, currentSolutionIndex);
00163     }
00164
00165     T C_GDp = C_GDp + C_GDo * (1.0 + std::tanh(P_D10 + P_D11 * u_gd));
00166     T C_GSp = C_GSp + C_GSo * (1.0 + std::tanh(P_S10 + P_S11 * u_gs));
00167
00168     i_gd_last = C_GD *
00169                 (2.0 * (u_gd - u_gd_last) / timestep - i_gd_last / C_GD_last);
00170     i_gs_last = C_GS *
00171                 (2.0 * (u_gs - u_gs_last) / timestep - i_gs_last / C_GS_last);
00172
00173     C_GD_last = C_GD;
00174     C_GS_last = C_GS;
00175
00176     u_gd_last = u_gd;
00177     u_gs_last = u_gs;
00178 }
00179
00180 static void
00181 addToElements(const std::string & line, CircuitElements<T> & elements,
00182               size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00183     std::regex BJTRegex = generateRegex("QMN", "n n n");
00184     NLNMOS<T> nmos;
00185     std::smatch matches;
00186
00187     std::regex_match(line, matches, BJTRegex);
00188
00189     nmos.designator = "QMN";
00190     nmos.designator += matches.str(1);
00191
00192     nmos.d = std::stoi(matches.str(2));
00193     nmos.g = std::stoi(matches.str(3));
00194     nmos.s = std::stoi(matches.str(4));
00195
00196     numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00197     numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00198     numNodes = std::max(numNodes, std::stoull(matches.str(4)));
00199
00200
00201     elements.nonLinearElements.emplace_back(std::make_shared<NLNMOS<T>>(nmos));
00202     elements.nodeComponentMap.insert(
00203         {{nmos.d, elements.nonLinearElements.back()},
00204          {nmos.g, elements.nonLinearElements.back()},
00205          {nmos.s, elements.nonLinearElements.back()}});
00206 }
00207 };
00208 #endif

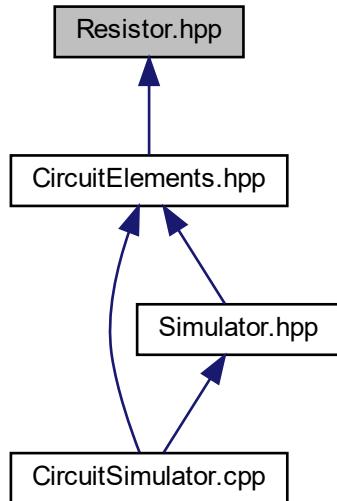
```

7.39 Resistor.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for Resistor.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `Resistor< T >`
An ideal resistor model.

7.40 Resistor.hpp

```

00001 #ifndef _RESISTOR_HPP_INC_
00002 #define _RESISTOR_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00009 template<typename T>
0010 struct Resistor : public Component<T> {
0011 public:
0012     T value = 0;
0013
0014     size_t n1 = 0;
0015     size_t n2 = 0;
0016     size_t currentIndex = 0;
0017
0018     bool group1 = true;
0019
0020     void addStaticStampTo(Stamp<T> & stamp) const {
0021         // the p means prime (') and is used for the index - 1
0022         size_t n1p = n1 - 1;
0023         size_t n2p = n2 - 1;
0024         size_t currentIndexp = currentIndex - 1;
0025
0026         if (group1) {
0027             T conductance = 1 / value;
0028
0029             if (n1) {
0030                 stamp.G(n1p, n1p) += conductance;
0031             }
0032             if (n2) {
0033                 stamp.G(n2p, n2p) += conductance;
0034             }
0035             if (n1 && n2) {
0036                 stamp.G(n1p, n2p) += -conductance;
0037                 stamp.G(n2p, n1p) += -conductance;
0038             }
0039         } else {
0040             if (n1) {
0041                 stamp.G(n1p, stamp.sizeG_A + currentIndex) += 1;
0042                 stamp.G(stamp.sizeG_A + currentIndex, n1p) += 1;
0043             }
0044             if (n2) {
0045                 stamp.G(n2p, stamp.sizeG_A + currentIndex) += -1;
0046                 stamp.G(stamp.sizeG_A + currentIndex, n2p) += -1;
0047             }
0048
0049             stamp.G(stamp.sizeG_A + currentIndex,
0050                     stamp.sizeG_A + currentIndex) += -value;
0051         }
0052     }
0053
0054     void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
0055                             size_t numCurrents) const {
0056         addStaticStampTo(stamp);
0057     }
0058
0059     static void
0060     addToElements(const std::string & line, CircuitElements<T> & elements,
0061                  size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
0062         // std::regex resistorRegex(
0063         // R"(^R(.*)?\s(\d+?)\s(\d+?)\s(.+?)\s(.+)\s(.+)\s(.+?)\s(.+?)\s(.+?)$)"
0064         // );
0065         std::regex resistorRegex = generateRegex("R", "n n w ? c");
0066         Resistor<T> resistor;
0067         std::smatch matches;
0068
0069         std::regex_match(line, matches, resistorRegex);
0070
0071         resistor.designator = "R";
0072         resistor.designator += matches.str(1);
0073
0074         resistor.n1 = std::stoi(matches.str(2));
0075         resistor.n2 = std::stoi(matches.str(3));
0076
0077         numNodes = std::max(numNodes, std::stoull(matches.str(2)));
0078         numNodes = std::max(numNodes, std::stoull(matches.str(3)));
0079
0080         if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
0081             resistor.value = std::stod(matches.str(4));
0082         } else {
0083             static_assert("Unsupported Type");
0084         }
0085
0086         if (matches.size() < 6 && matches.str(5) != "") {
0087             resistor.group1 = false;
0088             resistor.currentIndex = ++numCurrents;
0089         }
0090     }
0091 }
```

```

00089     } else {
00090         resistor.group1 = true;
00091     }
00092
00093     elements.staticElements.emplace_back(
00094         std::make_shared<Resistor<T>>(resistor));
00095
00096     elements.nodeComponentMap.insert(
00097         {{resistor.n1, elements.staticElements.back()},
00098          {resistor.n2, elements.staticElements.back()}});
00099 }
00100 };
00101
00102 #endif

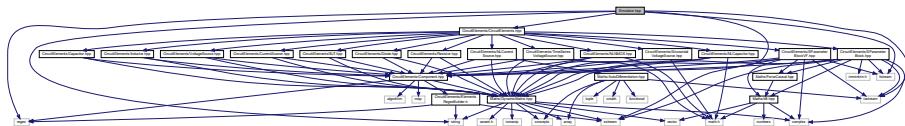
```

7.41 Simulator.hpp File Reference

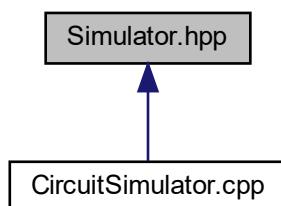
```

#include "CircuitElements/CircuitElements.hpp"
#include "Maths/DynamicMatrix.hpp"
#include <string>
#include <regex>
#include <iostream>
#include <fstream>
Include dependency graph for Simulator.hpp:

```



This graph shows which files directly or indirectly include this file:



Classes

- class [SimulationEnvironment< VT >](#)
The main class to hold all of the relevant simulation data.

Enumerations

- enum class `LineType` {
 Resistor = 'R' , Capacitor = 'C' , Inductor = 'L' , CurrentSource = 'I' ,
 VoltageSource = 'V' , SParameterBlock = 'S' , Transistor = 'Q' , Diode = 'D' ,
 Comment = " " , Directive = '.' }

The first character of each line for each component type.

- enum class `SourceType` { TimeSeries = 'T' , Sinusoidal = 'S' }

The type of (voltage) source.

7.41.1 Enumeration Type Documentation

7.41.1.1 LineType

```
enum LineType [strong]
```

The first character of each line for each component type.

Enumerator

Resistor	
Capacitor	
Inductor	
CurrentSource	
VoltageSource	
SParameterBlock	
Transistor	
Diode	
Comment	
Directive	

Definition at line 23 of file [Simulator.hpp](#).

7.41.1.2 SourceType

```
enum SourceType [strong]
```

The type of (voltage) source.

Enumerator

TimeSeries	
Sinusoidal	

Definition at line 37 of file [Simulator.hpp](#).

7.42 Simulator.hpp

```

00089     case LineType::Inductor:
00090         Inductor<VT>::addElements(line, elements, numNodes,
00091                                     numCurrents, numDCCurrents);
00092         break;
00093     case LineType::CurrentSource:
00094         if (line[1] == 'N') {
00095             NLCcurrentSource<VT>::addElements(line, elements, numNodes,
00096                                             numCurrents,
00097                                             numDCCurrents);
00098         } else {
00099             CurrentSource<VT>::addElements(line, elements, numNodes,
00100                                             numCurrents, numDCCurrents);
00101         }
00102         break;
00103     case LineType::VoltageSource:
00104         switch (static_cast<SourceType>(line[1])) {
00105             case SourceType::TimeSeries:
00106                 TimeSeriesVoltageSource<
00107                     VT>::addElements(line, elements, numNodes,
00108                                     numCurrents, numDCCurrents);
00109                 break;
00110             case SourceType::Sinusoidal:
00111                 SinusoidalVoltageSource<
00112                     VT>::addElements(line, elements, numNodes,
00113                                     numCurrents, numDCCurrents);
00114                 break;
00115             default:
00116                 VoltageSource<VT>::addElements(line, elements,
00117                                             numNodes, numCurrents,
00118                                             numDCCurrents);
00119                 break;
00120         }
00121         break;
00122     case LineType::SParameterBlock:
00123         if (line[1] == 'V' && (line[2] == 'P' || line[2] == 'F')) {
00124 #ifndef WITH_MATLAB
00125             if (line[2] == 'F') {
00126                 std::throw(
00127                     "ERROR: Matlab not available at compile time");
00128             }
00129 #endif
00130             SParameterBlockVF<VT>::addElements(line, elements,
00131                                             numNodes, numCurrents,
00132                                             numDCCurrents);
00133         } else {
00134             SParameterBlock<VT>::addElements(line, elements, numNodes,
00135                                             numCurrents,
00136                                             numDCCurrents);
00137         }
00138         break;
00139     case LineType::Transistor:
00140         if (line[1] == 'N') {
00141             BJTN<VT>::addElements(line, elements, numNodes,
00142                                     numCurrents, numDCCurrents);
00143         } else if (line[1] == 'P') {
00144             BJTP<VT>::addElements(line, elements, numNodes,
00145                                     numCurrents, numDCCurrents);
00146         } else if (line[1] == 'M') {
00147             if (line[2] == 'N') {
00148                 NLNMOS<VT>::addElements(line, elements, numNodes,
00149                                     numCurrents, numDCCurrents);
00150             } else {
00151                 std::cout << "Other Transistors not implemented yet"
00152                               << std::endl;
00153             }
00154         } else {
00155             std::cout << "Other Transistors not implemented yet"
00156                               << std::endl;
00157         }
00158         break;
00159     case LineType::Diode:
00160         Diode<VT>::addElements(line, elements, numNodes, numCurrents,
00161                                     numDCCurrents);
00162         break;
00163     case LineType::Comment:
00164         break;
00165     case LineType::Directive:
00166         std::regex_match(line, matches, transientRegex);
00167
00168         if (matches.size()) {
00169             if constexpr (std::is_same_v<VT, double> ||
00170                         std::is_same_v<VT, float>) {
00171                 initialTime = std::stod(matches.str(1));
00172                 timestep = std::stod(matches.str(3));
00173                 finalTime = std::stod(matches.str(2));
00174                 steps = (finalTime - initialTime) / timestep;
00175             } else {

```

```

00176                     static_assert("Unsupported Type");
00177                 }
00178             break;
00179         }
00180
00181         std::regex_match(line, matches, graphRegex);
00182         if (matches.size()) {
00183             parseGraph(matches.str(1));
00184             break;
00185         }
00186
00187         std::regex_match(line, matches, noDCRegex);
00188         if (matches.size()) {
00189             performDCAAnalysis = false;
00190             break;
00191         }
00192
00193         std::regex_match(line, matches, outputFileRegex);
00194         if (matches.size()) {
00195             outputPath = matches.str(1);
00196             break;
00197         }
00198
00199         std::cout << "Unsupported Directive" << std::endl;
00200
00201         break;
00202     }
00203 }
00204
00205 elements.setNewStampSize(numNodes, numCurrents, numDCCurrents);
00206
00207 size_t sizeMat = elements.staticStamp.G.M;
00208
00209 solutionMat = Matrix<VT>(sizeMat, steps, 0);
00210
00211 luPair = LUPair<VT>(sizeMat);
00212 scratchSpace = Matrix<VT>(sizeMat, 1);
00213
00214 for (auto & comp : elements.staticElements) {
00215     comp->setTimestep(timestep);
00216 }
00217 for (auto & comp : elements.dynamicElements) {
00218     comp->setTimestep(timestep);
00219 }
00220 for (auto & comp : elements.nonLinearElements) {
00221     comp->setTimestep(timestep);
00222 }
00223
00224 if (performDCAAnalysis) {
00225     setDCOpPoint();
00226 }
00227 }
00228
00229 void setDCOpPoint() {
00230     Matrix<VT> dcSoln = Matrix<VT>(solutionMat.M + numDCCurrents, 1);
00231     Matrix<VT> scratchSpace = Matrix<VT>(solutionMat.M + numDCCurrents, 1);
00232     LUPair<VT> luPair = LUPair<VT>(solutionMat.M + numDCCurrents);
00233
00234     auto simStartTime = std::chrono::high_resolution_clock::now();
00235     for (size_t nr = 0; nr < 35; nr++) {
00236         auto & stamp = elements.generateDCStamp(dcSoln, numCurrents);
00237         stamp.G.luPair(luPair);
00238         stamp.G.leftDivide(stamp.s, luPair, scratchSpace, dcSoln);
00239     }
00240
00241     for (size_t k = 0; k < solutionMat.M; k++) {
00242         solutionMat(k, 0) = dcSoln(k, 0);
00243     }
00244
00245     elements.updateDCStoredState(dcSoln, numCurrents);
00246
00247     auto simEndTime = std::chrono::high_resolution_clock::now();
00248     auto timeTaken = std::chrono::duration_cast<std::chrono::nanoseconds>(
00249         simEndTime - simStartTime)
00250         .count();
00251
00252
00253     std::cout << "DC OP in: " << timeTaken * 1e-6 << " ms (" << timeTaken
00254         << " ns)" << std::endl;
00255 }
00256
00257 void simulate() {
00258     constexpr VT convergedThreshold = 1e-12;
00259     constexpr size_t maxNR = 32;
00260     Matrix<VT> tempSoln = Matrix<VT>(solutionMat.M, 1);
00261     VT maxDiff;
00262     VT singleVarDiff;

```

```

00271     auto simStartTime = std::chrono::high_resolution_clock::now();
00272     for (size_t n = 1; n < steps; n++) {
00273         size_t nr;
00274         for (nr = 0; nr < maxNR; nr++) {
00275             auto & stamp = elements.generateNonLinearStamp(solutionMat, n,
00276                                                 timestep);
00277             stamp.G.luPair(luPair);
00278             stamp.G.leftDivide(stamp.s, luPair, scratchSpace, tempSoln);
00279
00280             maxDiff = 0;
00281             for (size_t k = 0; k < solutionMat.M; k++) {
00282                 singleVarDiff = std::abs(solutionMat(k, n) - tempSoln(k, 0));
00283                 maxDiff = std::max(maxDiff, singleVarDiff);
00284             }
00285
00286             for (size_t k = 0; k < solutionMat.M; k++) {
00287 #ifdef _DEBUG
00288                 if (std::isnan(tempSoln(k, 0))) {
00289                     std::cout << "Simulation Error: NaN found in solution"
00290                         << std::endl;
00291                 }
00292 #endif
00293                 solutionMat(k, n) = tempSoln(k, 0);
00294             }
00295             if (maxDiff < convergedThreshold) {
00296                 break;
00297             }
00298             elements.nonLinearStampIsFresh = false;
00299         }
00300
00301 #ifdef _DEBUG
00302         if (nr < maxNR) {
00303             std::cout << "NR terminated at: " << nr << " steps" << std::endl;
00304         }
00305 #endif
00306
00307         elements.updateTimeStep(solutionMat, n, timestep);
00308         if (n == 1) {
00309             elements.staticStampIsFresh = false; // for VF s-param model update
00310         }
00311     }
00312     auto simEndTime = std::chrono::high_resolution_clock::now();
00313     auto timeTaken = std::chrono::duration_cast<std::chrono::nanoseconds>(
00314         simEndTime - simStartTime)
00315         .count();
00316 // std::cout << "Time taken for simulation: " << timeTaken << std::endl;
00317 std::cout << timeTaken * 1e-6 << " ms (" << timeTaken << " ns)" << std::endl;
00318 std::ofstream runtimeFile("RunTimes.txt", std::ofstream::app);
00319 runtimeFile << netlistPath << " " << timeTaken << std::endl;
00320
00321     dataDump();
00322
00323     size_t graphNum = 1;
00324     for (auto nodes : nodesToGraph) {
00325         printMultipleOnGraph(nodes, std::to_string(graphNum++));
00326     }
00327 }
00328
00329 void printGraph(size_t node) {
00330 #ifdef WITH_PYTHON
00331     namespace plt = matplotlibcpp;
00332     assert(node > 0);
00333     std::vector<double> timeVector(steps);
00334     std::vector<double> voltageNode(steps);
00335     for (size_t n = 0; n < steps; n++) {
00336         voltageNode[n] = solutionMat(node - 1, n);
00337         timeVector[n] = n * timestep;
00338     }
00339 // Set the size of output image to 1200x780 pixels
00340 plt::figure_size(1200, 780);
00341 // Plot line from given x and y data. Color is selected automatically.
00342 plt::plot(timeVector, voltageNode);
00343
00344 std::string filename = "Node " + std::to_string(node) + ".png";
00345 std::string filenameEps = "Node " + std::to_string(node) + ".eps";
00346 // std::cout << "Saving result to " << filename << std::endl;
00347 plt::save(filename.c_str());
00348 plt::save(filenameEps.c_str());
00349 plt::close();
00350
00351 #endif
00352 }
00353
00354 #endif
00355 }
00356
00357 void printMultipleOnGraph(std::vector<size_t> nodeVec, std::string suffix = "") {
00358 #ifdef WITH_PYTHON
00359     namespace plt = matplotlibcpp;
00360     plt::figure_size(1200, 780);
00361     for (auto node : nodeVec) {

```

```

00367         assert(node > 0);
00368         std::vector<double> timeVector(steps);
00369         std::vector<double> voltageNode(steps);
00370         for (size_t n = 0; n < steps; n++) {
00371             voltageNode[n] = solutionMat(node - 1, n);
00372             timeVector[n] = n * timestep;
00373         }
00374         // Set the size of output image to 1200x780 pixels
00375         // Plot line from given x and y data. Color is selected automatically.
00376         std::map<std::string, std::string> kwArgs;
00377         kwArgs["label"] = "Node " + std::to_string(node);
00378         plt::plot(timeVector, voltageNode, kwArgs);
00379     }
00380     plt::legend();
00381     std::string name = "Graph";
00382     name += suffix;
00383
00384     plt::save(name + ".png");
00385     plt::save(name + ".eps");
00386     plt::close();
00387 #endif
00388 }
00389
00390 void dataDump() {
00391 #ifdef WITH_MATLAB
00392     // Create matlab data array factory
00393     matlab::data::ArrayFactory factory;
00394
00395     std::vector<std::string> varNames = {"t"};
00396
00397 #endif
00398
00399     std::ofstream outputFile(outputFilePath);
00400     outputFile << "time";
00401     for (int i = 1; i <= numNodes; i++) {
00402         outputFile << "\t"
00403             << "n" << i;
00404
00405 #ifdef WITH_MATLAB
00406         if (matlabDesktop) {
00407             varNames.emplace_back(std::string("n") + std::to_string(i));
00408         }
00409 #endif
00410     }
00411
00412     for (int i = 1; i <= numCurrents; i++) {
00413         outputFile << "\t"
00414             << "i" << i;
00415 #ifdef WITH_MATLAB
00416         if (matlabDesktop) {
00417             varNames.emplace_back(std::string("i") + std::to_string(i));
00418         }
00419 #endif
00420     }
00421
00422 #ifdef WITH_MATLAB
00423     auto sArray = factory.createStructArray({solutionMat.N, 1}, varNames);
00424 #endif
00425     for (size_t n = 0; n < solutionMat.N; n++) {
00426         outputFile << std::endl;
00427         outputFile << std::setprecision(9) << n * timestep;
00428 #ifdef WITH_MATLAB
00429         if (matlabDesktop) {
00430             sArray[n][varNames[0]] = factory.createArray({1, 1}, {n * timestep});
00431         }
00432 #endif
00433         for (int i = 0; i < numNodes + numCurrents; i++) {
00434             outputFile << "\t" << std::setprecision(9) << solutionMat(i, n);
00435 #ifdef WITH_MATLAB
00436         if (matlabDesktop) {
00437             sArray[n][varNames[i + 1]] = factory
00438                 .createArray({1, 1},
00439                             {solutionMat(i,
00440                                         n)}));
00441         }
00442 #endif
00443     }
00444 }
00445     outputFile.close();
00446
00447 #ifdef WITH_MATLAB
00448     if (matlabDesktop) {
00449         matlabEngine->setVariable(u"solutionData", sArray,
00450                                     matlab::engine::WorkspaceType::GLOBAL);
00451     }
00452 #endif
00453 }
00454
00455

```

```

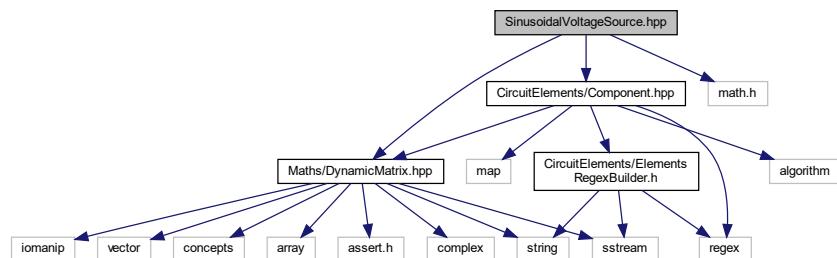
00456 private:
00457 #ifdef WITH_MATLAB
00458     std::shared_ptr<matlab::engine::MATLABEngine> matlabEngine;
00459     bool matlabDesktop = false;
00460 #endif
00461
00462
00463     void parseGraph(std::string line) {
00464         std::stringstream sstr(line);
00465         nodesToGraph.emplace_back(std::vector<size_t>());
00466         std::vector<size_t> & toGraph = nodesToGraph.back();
00467         size_t nodeNum = 0;
00468         sstr >> nodeNum;
00469
00470         while (!sstr.fail()) {
00471             toGraph.emplace_back(nodeNum);
00472             sstr.ignore(1);
00473             sstr >> nodeNum;
00474         }
00475     }
00476
00477     std::string outputFilePath = "datadump.txt";
00478     std::string netlistPath = "";
00479
00480     double initialTime;
00481     double timestep;
00482     double finalTime;
00483     size_t steps;
00484
00485     size_t numNodes = 1;
00486     size_t numCurrents = 0;
00487     size_t numDCCurrents = 0;
00488     bool performDCAnalysis = true;
00489     CircuitElements<VT> elements;
00490
00491     LUPair<VT> luPair;
00492     Matrix<VT> scratchSpace;
00493
00494     std::vector<std::vector<size_t> > nodesToGraph;
00495
00496     Matrix<VT> solutionMat;
00497 }
00498
00499 #endif
00500
00501
00502
00503
00504
00505
00506
00507
00508

```

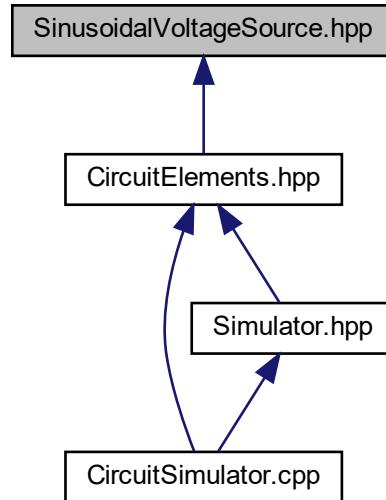
7.43 SinusoidalVoltageSource.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include <math.h>

Include dependency graph for SinusoidalVoltageSource.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct **SinusoidalVoltageSource< T >**
A sinusoidal voltage source model.

7.44 SinusoidalVoltageSource.hpp

```

00001 #ifndef _SINUSOIDALVOLTAGESOURCE_INC_
00002 #define _SINUSOIDALVOLTAGESOURCE_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005 #include <math.h>
00006
00010 template<typename T>
00011 struct SinusoidalVoltageSource : public Component<T> {
00012     size_t n1 = 0;
00013     size_t n2 = 0;
00014     size_t currentIndex = 0;
00015
00016     T V = 1;
00017     T phase = 0;
00018     T frequency = 1;
00019     T offset = 0;
00020     bool degrees = true;
00021
00022     void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00023                           const size_t currentSolutionIndex, T timestep) const {
00024         size_t n1p = n1 - 1;
00025         size_t n2p = n2 - 1;
00026         size_t currentIndexp = currentIndex - 1;
00027
00028         if (n1) {
00029             stamp.G(n1p, stamp.sizeG_A + currentIndexp) += 1;
00030             stamp.G(stamp.sizeG_A + currentIndexp, n1p) += 1;
00031         }
00032
00033         if (n2) {
00034             stamp.G(n2p, stamp.sizeG_A + currentIndexp) += -1;
00035             stamp.G(stamp.sizeG_A + currentIndexp, n2p) += -1;
00036     }
  
```

```

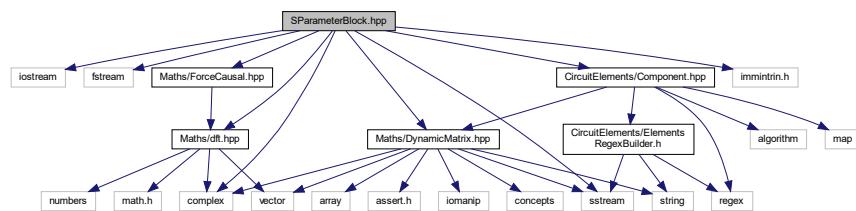
00037
00038     if (degrees) {
00039         stamp.s(stamp.sizeG_A + currentIndex,
00040                 0) += offset + V * std::sin(2 * std::numbers::pi * frequency *
00041                                         currentSolutionIndex * timestep +
00042                                         std::numbers::pi * phase / 180);
00043     } else {
00044         stamp.s(stamp.sizeG_A + currentIndex,
00045                 0) += offset + V * std::sin(2 * std::numbers::pi * frequency *
00046                                         currentSolutionIndex * timestep +
00047                                         phase);
00048     }
00049 }
00050
00051 void updateStoredState(const Matrix<T> & solutionMatrix,
00052                         const size_t currentSolutionIndex, T timestep,
00053                         size_t sizeG_A) {
00054 }
00055
00056 void addDCAalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00057                           size_t numCurrents) const {
00058     addDynamicStampTo(stamp, solutionVector, 0, 0);
00059 }
00060
00061 static void
00062 addElements(const std::string & line, CircuitElements<T> & elements,
00063               size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00064     // n1 n2 amp freq off phase
00065     std::regex sinusoidalVoltageSourceRegex = generateRegex("VS",
00066                                                 "n n ? w w w w");
00067
00068     SinusoidalVoltageSource<T> voltageSource;
00069     std::smatch matches;
00070
00071     std::regex_match(line, matches, sinusoidalVoltageSourceRegex);
00072
00073     voltageSource.designator = "VS";
00074     voltageSource.designator += matches.str(1);
00075
00076     voltageSource.n1 = std::stoi(matches.str(2));
00077     voltageSource.n2 = std::stoi(matches.str(3));
00078
00079     numNodes = std::max(numNodes, std::stoul(matches.str(2)));
00080     numNodes = std::max(numNodes, std::stoul(matches.str(3)));
00081
00082     if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00083         voltageSource.V = std::stod(matches.str(4));
00084     } else {
00085         static_assert("Unsupported Type");
00086     }
00087
00088     if (matches.size() > 4 && matches.str(5) != "") {
00089         voltageSource.frequency = std::stod(matches.str(5));
00090     }
00091
00092     if (matches.size() > 5 && matches.str(6) != "") {
00093         voltageSource.offset = std::stod(matches.str(6));
00094     }
00095
00096     if (matches.size() > 6 && matches.str(7) != "") {
00097         voltageSource.phase = std::stod(matches.str(7));
00098     }
00099
00100    voltageSource.currentIndex = ++numCurrents;
00101
00102    elements.dynamicElements.emplace_back(
00103        std::make_shared<SinusoidalVoltageSource<T>>(voltageSource));
00104    elements.nodeComponentMap.insert(
00105        {{voltageSource.n1, elements.dynamicElements.back()},
00106         {voltageSource.n2, elements.dynamicElements.back()}});
00107 }
00108 };
00109
00110 #endif

```

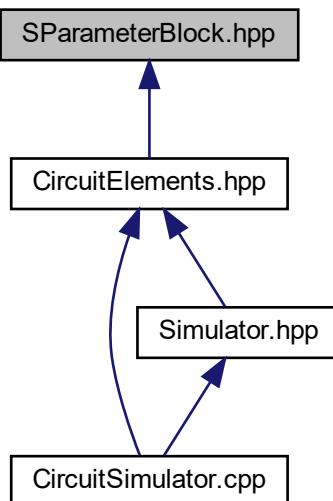
7.45 SParameterBlock.hpp File Reference

```
#include <iostream>
#include <fstream>
#include <sstream>
```

```
#include <complex>
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include "Maths/dft.hpp"
#include "Maths/ForceCausal.hpp"
#include <immintrin.h>
Include dependency graph for SParameterBlock.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `SParameterPort< T >`
a helper struct to store the information for the ports of an S-Parameter Block
- struct `SParamLengthOffset`
- struct `SParameterSequence< T >`
a helper struct to store the DTIR sequence for the bloc
- struct `SParameterBlock< T >`
A DTIR based model of an s-parameter block.

7.46 SParameterBlock.hpp

```

00113     // if ( 0 <= kprime && kprime < 1.0 ) {
00114     //   std::cout << "WARNING: SIMULATION Timestep TOO LARGE FOR CONVOLUTION" <<
00115     //   std::endl;
00116     //}
00117     T index = n - kprime;
00118     // TODO: Maybe look into changing the bounds of the convolution to avoid
00119     // having this checked too much. Maybe wrap in debug ifdef.
00120     if (index <= 0) {
00121         return 0.0;
00122     }
00123
00124     size_t floor = static_cast<size_t>(index);
00125     if (floor <= 0 || floor + 1 >= n) {
00126         return 0.0;
00127     }
00128
00129     T mix = index - static_cast<T>(floor);
00130
00131     T toRet;
00132
00133     T uppArr[3] = {0};
00134     T lowArr[3] = {0};
00135
00136     if (port[portIndex].positive != 0) {
00137         uppArr[0] = solutionMatrix(port[portIndex].positive - 1, floor + 1);
00138         lowArr[0] = solutionMatrix(port[portIndex].positive - 1, floor);
00139     }
00140
00141     if (port[portIndex].negative != 0) {
00142         uppArr[1] = solutionMatrix(port[portIndex].negative - 1, floor + 1);
00143         lowArr[1] = solutionMatrix(port[portIndex].negative - 1, floor);
00144     }
00145
00146     uppArr[2] = solutionMatrix(sizeG_A + port[portIndex].current - 1, floor + 1);
00147     lowArr[2] = solutionMatrix(sizeG_A + port[portIndex].current - 1, floor);
00148
00149     toRet = (uppArr[0] - lowArr[0]) * mix + lowArr[0] -
00150         (uppArr[1] - lowArr[1]) * mix - lowArr[1] +
00151         (uppArr[2] - lowArr[2]) * mix + lowArr[2] * z_ref;
00152
00153     return toRet;
00154 }
00155
00156 T V_p(size_t p, const Matrix<T> & solutionMatrix, const size_t n,
00157     T simulationTimestep, size_t sizeG_A) const {
00158     // V_p = beta * sum of ports ( history of port )
00159     T toRet = 0;
00160     // TODO: This may finally be a good place for multithreading
00161     // We could have one thread per port perhaps. Mainly as the data overlap is
00162     // minimal. This would require having a different "toRet" for each thread and
00163     // summing at the end before returning perhaps
00164     for (size_t c = 0; c < port.size(); c++) {
00165         // TODO: Optimise the convolution here
00166         // size_t len = std::min( n, s.sParamLength );
00167         for (size_t k = 1; k < s.length(p, c); k++) {
00168             toRet += aWaveConvValue(c, solutionMatrix, n, s.time(p, c, k),
00169                                     simulationTimestep, sizeG_A) *
00170                     s.data(p, c, k);
00171         }
00172     }
00173     return port[p].beta * toRet;
00174 }
00175
00176 T R_p(size_t p) const {
00177     return port[p].R;
00178 }
00179
00180 T beta_p(size_t p) const {
00181     return port[p].beta;
00182 }
00183
00184 void addStaticStampTo(Stamp<T> & stamp) const {
00185     for (size_t p = 0; p < port.size(); p++) {
00186         size_t np = port[p].positive - 1;
00187         size_t nn = port[p].negative - 1;
00188         size_t curr = port[p].current - 1;
00189         // Voltage source and resistance
00190         stamp.G(stamp.sizeG_A + curr, stamp.sizeG_A + curr) += -port[p].R;
00191         if (port[p].positive != 0) {
00192             stamp.G(stamp.sizeG_A + curr, np) += 1;
00193             stamp.G(np, stamp.sizeG_A + curr) += 1;
00194         }
00195         if (port[p].negative != 0) {
00196             stamp.G(stamp.sizeG_A + curr, nn) += -1;
00197             stamp.G(nn, stamp.sizeG_A + curr) += -1;
00198         }
00199     }
00200 }
```

```

00210         // controlled sources
00211         for (size_t c = 0; c < port.size(); c++) {
00212             if (c != p) {
00213                 T alpha = port[p].beta * port[p].s0[c];
00214                 if (port[c].positive != 0) {
00215                     stamp.G(stamp.sizeG_A + curr,
00216                             port[c].positive - 1) += -alpha;
00217                 }
00218                 if (port[c].negative != 0) {
00219                     stamp.G(stamp.sizeG_A + curr, port[c].negative - 1) += alpha;
00220                 }
00221                 stamp.G(stamp.sizeG_A + curr,
00222                         stamp.sizeG_A + port[c].current - 1) += -z_ref * alpha;
00223             }
00224         }
00225     }
00226 }
00227
00228 void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00229                         const size_t currentSolutionIndex,
00230                         T simulationTimestep) const {
00231     for (size_t p = 0; p < port.size(); p++) {
00232         size_t curr = port[p].current - 1;
00233         // V_p
00234         stamp.s(stamp.sizeG_A + curr,
00235                 0) += V_p(p, solutionMatrix, currentSolutionIndex,
00236                             simulationTimestep, stamp.sizeG_A);
00237     }
00238 }
00239
00240 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00241                           size_t numCurrents) const {
00242     for (size_t p = 0; p < port.size(); p++) {
00243         size_t np = port[p].positive - 1;
00244         size_t nn = port[p].negative - 1;
00245         size_t curr = port[p].current - 1;

00246         // Voltage source and resistance
00247         T sppSum = 0;
00248         for (size_t k = 0; k < s.length(p, p); k++) {
00249             sppSum += s.data(p, p, k);
00250         }
00251         T Rprime = port[p].beta * z_ref * (1 + sppSum) /
00252             (1 - port[p].beta * sppSum);
00253         stamp.G(stamp.sizeG_A + curr, stamp.sizeG_A + curr) += Rprime;
00254         if (port[p].positive != 0) {
00255             stamp.G(stamp.sizeG_A + curr, np) += 1;
00256             stamp.G(np, stamp.sizeG_A + curr) += 1;
00257         }

00258         if (port[p].negative != 0) {
00259             stamp.G(stamp.sizeG_A + curr, nn) += -1;
00260             stamp.G(nn, stamp.sizeG_A + curr) += -1;
00261         }

00262         // controlled sources
00263         for (size_t c = 0; c < port.size(); c++) {
00264             if (c != p) {
00265                 T alpha = port[p].beta * port[p].s0[c];
00266                 T alphaPrime = 0;

00267                 for (size_t k = 0; k < s.length(p, c); k++) {
00268                     alphaPrime += s.data(p, c, k);
00269                 }

00270                 alphaPrime = port[p].beta * alphaPrime;
00271                 alphaPrime += alpha;
00272                 alphaPrime = alphaPrime / (1 - port[p].beta * sppSum);

00273                 if (port[c].positive != 0) {
00274                     stamp.G(stamp.sizeG_A + curr,
00275                             port[c].positive - 1) += -alphaPrime;
00276                 }
00277                 if (port[c].negative != 0) {
00278                     stamp.G(stamp.sizeG_A + curr,
00279                             port[c].negative - 1) += alphaPrime;
00280                 }
00281                 stamp.G(stamp.sizeG_A + curr, stamp.sizeG_A + port[c].current -
00282                         1) += -z_ref * alphaPrime;
00283             }
00284         }
00285     }
00286
00287     // V_p = 0
00288 }
```

```

00297     }
00298
00299     void updateStoredState(const Matrix<T> & solutionMatrix,
00300             const size_t currentSolutionIndex, T timestep,
00301             size_t sizeG_A) {
00302     }
00303
00304
00305     void readInTouchstoneFile() {
00306         using CT = std::complex<T>;
00307         using FSP = std::vector<CT>;
00308         std::ifstream file(touchstoneFilePath);
00309
00310         std::vector<T> freqs;
00311         std::vector<std::vector<FSP> > freqSParams(s.numPorts);
00312         z_ref = 50;
00313
00314         std::string line;
00315         while (file.peek() == '#' || file.peek() == '!') {
00316             std::getline(file, line);
00317         }
00318
00319         T val1;
00320         T val2;
00321         while (!file.eof()) {
00322             file >> val1;
00323             if (file.fail()) {
00324                 break;
00325             }
00326             freqs.emplace_back(val1);
00327             for (size_t a = 0; a < s.numPorts; a++) {
00328                 freqSParams[a].resize(s.numPorts);
00329             }
00330
00331             for (size_t b = 0; b < s.numPorts; b++) {
00332                 for (size_t a = 0; a < s.numPorts; a++) {
00333                     file >> val1 >> val2;
00334                     freqSParams[a][b].emplace_back(val1, val2);
00335                 }
00336             }
00337         }
00338     }
00339
00340     /*
00341     // std::vector< T > symFreqs( 2 * freqs.size() - 1 );
00342     // T maxFreq = freqs.back();
00343     /*     for ( size_t k = 0; k < freqs.size(); k++ ) {
00344         symFreqs[ k ] = freqs[ k ];
00345         symFreqs[ freqs.size() + k ] = maxFreq + freqs[ k ];
00346     } */
00347
00348
00349     // s.sParamLength = ( 2 * freqs.size() - 2 );
00350     s.sParamLengthOffset.resize(s.numPorts * s.numPorts);
00351     for (size_t a = 0; a < s.numPorts; a++) {
00352         port[a].s0.resize(s.numPorts);
00353         for (size_t b = 0; b < s.numPorts; b++) {
00354             auto causal = forceCausal(freqs, freqSParams[a][b]);
00355
00356             T thresholdToKeep = 1;
00357             for (auto entry : causal.data) {
00358                 thresholdToKeep = std::max(std::abs(entry), thresholdToKeep);
00359             }
00360
00361             thresholdToKeep = thresholdToKeep * fracMaxToKeep;
00362
00363             s.offset(a, b) = s._data.size();
00364             for (size_t n = 0; n < causal.data.size(); n++) {
00365                 if (n == 0 || std::abs(causal.data[n]) > thresholdToKeep) {
00366                     s._data.emplace_back(causal.data[n]);
00367                     s._time.emplace_back(n == 0 ? 0
00368                                         : n * causal.Ts - causal.tau);
00369                     std::cout << s._time.back() << " " << s._data.back()
00370                                         << std::endl;
00371                 }
00372             }
00373             s.length(a, b) = s._data.size() - s.offset(a, b);
00374             std::cout << "Pruned " << ((2 * freqs.size() - 2) - s.length(a, b))
00375                     << " DTIR entries out of " << (2 * freqs.size() - 2)
00376                     << " less than " << thresholdToKeep << "("
00377                     << fracMaxToKeep * 100 << "% of max val)" << std::endl;
00378
00379             port[a].s0[b] = s.data(a, b, 0);
00380         }
00381         port[a].beta = 1.0 / (1 - s.data(a, a, 0));
00382         port[a].R = port[a].beta * 50 * (1 + s.data(a, a, 0));
00383     }
00384 }
00385
00386 static void

```

```

00387     addToElements(const std::string & line, CircuitElements<T> & elements,
00388                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00389     // std::regex SParameterBlockInitialRegex( R"(^S(.*)\s(\d+)\s)" );
00390     std::regex SParameterBlockInitialRegex = generateRegex("S", "w n s", true,
00391                                                 false);
00392     std::smatch matches;
00393     std::regex_search(line, matches, SParameterBlockInitialRegex);
00394
00395     SParameterBlock<T> block;
00396
00397     block.designator = "S";
00398     block.designator += matches.str(1);
00399
00400     block.fractionMaxToKeep = std::stod(matches.str(2));
00401
00402     size_t numPorts = std::stoul(matches.str(3));
00403     block.s.numPorts = numPorts;
00404     block.port = std::vector<SParameterPort<T>>(numPorts);
00405
00406     auto strIter = matches.suffix().first;
00407     std::regex SParameterBlockPortRegex(R"^( (\d+) \s (\d+) \s )");
00408     for (size_t p = 0; p < numPorts; p++) {
00409         std::regex_search(strIter, line.cend(), matches,
00410                           SParameterBlockPortRegex);
00411         block.port[p].positive = std::stoi(matches.str(1));
00412         block.port[p].negative = std::stoi(matches.str(2));
00413
00414         numNodes = std::max(numNodes, std::stoull(matches.str(1)));
00415         numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00416
00417         block.port[p].current = ++numCurrents;
00418
00419         strIter = matches.suffix().first;
00420     }
00421     std::regex endOfLineRegex(R"^(.*$)");
00422     std::regex_search(strIter, line.cend(), matches, endOfLineRegex);
00423     block.touchstoneFilePath = matches.str(1);
00424     block.readInTouchstoneFile();
00425
00426     elements.dynamicElements.emplace_back(
00427         std::make_shared<SParameterBlock<T>>(block));
00428     for (size_t p = 0; p < numPorts; p++) {
00429         elements.nodeComponentMap.insert(
00430             {block.port[p].positive, elements.dynamicElements.back()},
00431             {block.port[p].negative, elements.dynamicElements.back()});
00432     }
00433 }
00434 };
00435
00436 #endif

```

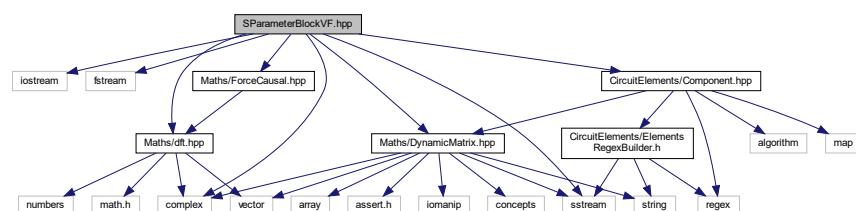
7.47 SParameterBlockVF.hpp File Reference

```

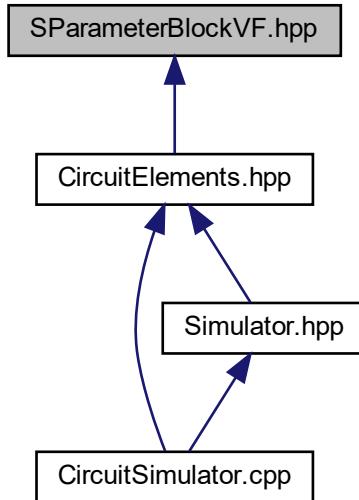
#include <iostream>
#include <fstream>
#include <sstream>
#include <complex>
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include "Maths/dft.hpp"
#include "Maths/ForceCausal.hpp"

```

Include dependency graph for SParameterBlockVF.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct **SParamVFDDataFrom< T >**
- struct **SParameterPortVF< T >**
 - a helper struct to store the information for the ports of an S-Parameter Block
- struct **SParameterBlockVF< T >**
 - A vectorfitting based model of an s-parameter block.

7.48 SParameterBlockVF.hpp

```

00001 #ifndef _SPARAMETERBLOCKVF_HPP_
00002 #define _SPARAMETERBLOCKVF_HPP_
00003 #include <iostream>
00004 #include <fstream>
00005 #include <sstream>
00006 #include <complex>
00007
00008 #include "CircuitElements/Component.hpp"
00009 #include "Maths/DynamicMatrix.hpp"
00010 #include "Maths/dft.hpp"
00011 #include "Maths/ForceCausal.hpp"
00012
00013 #ifdef WITH_MATLAB
00014 #include "MatlabEngine.hpp"
00015 #include "MatlabdataArray.hpp"
00016 #endif
00017
00018 template<typename T>
00019 struct SParamVFDDataFrom {
00020     size_t numPoles = 0;
00021
00022     std::vector<std::complex<T>> pole;
00023     std::vector<std::complex<T>> residue;
00024     std::complex<T> remainder = 0;
00025
00026     std::vector<std::complex<T>> lambda_p;
00027     std::vector<std::complex<T>> mu_p;
00028     std::vector<std::complex<T>> nu_p;
  
```

```

00032     std::vector<std::complex<T> > exp_alpha;
00034
00036     std::complex<T> lambda = 0;
00038     std::complex<T> mu = 0;
00040     std::complex<T> nu = 0;
00041
00043     std::vector<std::complex<T> > x;
00044 };
00045
00050 template<typename T>
00051 struct SParameterPortVF {
00053     size_t positive = 0;
00055     size_t negative = 0;
00057     size_t current = 0;
00058
00059     std::complex<T> beta = 0;
00062
00065     std::vector<std::complex<T> > alpha;
00066
00068     std::complex<T> R = 0;
00069
00070     std::vector<SParamVFDataFrom<T> > from;
00071 };
00072
00073
00077 template<typename T>
00078 struct SParameterBlockVF : public Component<T> {
00079     std::vector<SParameterPortVF<T> > port;
00080     size_t numPorts = 0;
00081     bool firstOrder = true;
00082
00083     T z_ref = 0;
00084
00085     std::complex<T> history_p(const size_t p, const Matrix<T> & solutionMatrix,
00086                               const size_t currentSolutionIndex, T timestep,
00087                               const size_t sizeG_A) const {
00088         std::complex<T> toRet = 0;
00089         for (size_t c = 0; c < numPorts; c++) {
00090             for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00091                 toRet += port[p].from[c].x[rho] * port[p].from[c].exp_alpha[rho];
00092             }
00093
00094             toRet += port[p].from[c].mu *
00095                 awave_p(c, solutionMatrix, currentSolutionIndex - 1, sizeG_A);
00096             if (currentSolutionIndex > 1) {
00097                 toRet += port[p].from[c].nu * awave_p(c, solutionMatrix,
00098                                               currentSolutionIndex - 2,
00099                                               sizeG_A);
00100             }
00101         }
00102         return 2.0 * toRet * std::sqrt(z_ref);
00103     }
00104
00105     T
00106     V_p(const size_t p, const Matrix<T> & solutionMatrix,
00107           const size_t currentSolutionIndex, T timestep, const size_t sizeG_A) const {
00108         return std::real(
00109             history_p(p, solutionMatrix, currentSolutionIndex, timestep, sizeG_A) *
00110             port[p].beta);
00111     }
00112
00113     T awave_p(const size_t p, const Matrix<T> & solutionMatrix,
00114               const size_t currentSolutionIndex, const size_t sizeG_A) const {
00115         size_t np = port[p].positive - 1;
00116         size_t nn = port[p].negative - 1;
00117         size_t curr = port[p].current - 1;
00118         size_t n = currentSolutionIndex;
00119         T toRet = 0;
00120
00121         if (port[p].positive) {
00122             toRet += solutionMatrix(np, n);
00123         }
00124
00125         if (port[p].negative) {
00126             toRet += -solutionMatrix(nn, n);
00127         }
00128
00129         return (toRet + solutionMatrix(sizeG_A + curr, n) * z_ref) /
00130             (2 * std::sqrt(z_ref));
00131     }
00132
00133     T bwave_p(const size_t p, const Matrix<T> & solutionMatrix,
00134               const size_t currentSolutionIndex, const size_t sizeG_A) const {
00135         size_t np = port[p].positive - 1;
00136         size_t nn = port[p].negative - 1;

```

```

00137     size_t curr = port[p].current - 1;
00138     size_t n = currentSolutionIndex;
00139     T toRet = 0;
00140
00141     if (port[p].positive) {
00142         toRet += solutionMatrix(np, n);
00143     }
00144
00145     if (port[p].negative) {
00146         toRet += -solutionMatrix(nn, n);
00147     }
00148
00149     return (toRet - solutionMatrix(sizeG_A + curr, n) * z_ref) /
00150         (2 * std::sqrt(z_ref));
00151 }
00152
00153 void addStaticStampTo(Stamp<T> & stamp) const {
00154     for (size_t p = 0; p < port.size(); p++) {
00155         size_t np = port[p].positive - 1;
00156         size_t nn = port[p].negative - 1;
00157         size_t curr = port[p].current - 1;
00158         // Voltage source and resistance
00159         //
00160         stamp.G(stamp.sizeG_A + curr,
00161                 stamp.sizeG_A + curr) += std::real(-port[p].R);
00162
00163         if (port[p].positive != 0) {
00164             stamp.G(stamp.sizeG_A + curr, np) += 1;
00165             stamp.G(np, stamp.sizeG_A + curr) += 1;
00166         }
00167
00168         if (port[p].negative != 0) {
00169             stamp.G(stamp.sizeG_A + curr, nn) += -1;
00170             stamp.G(nn, stamp.sizeG_A + curr) += -1;
00171         }
00172         // controlled sources
00173         for (size_t c = 0; c < port.size(); c++) {
00174             if (c != p) {
00175                 if (port[c].positive != 0) {
00176                     stamp.G(stamp.sizeG_A + curr,
00177                             port[c].positive -
00178                                 1) += std::real(-port[p].alpha[c]);
00179                 }
00180                 if (port[c].negative != 0) {
00181                     stamp.G(stamp.sizeG_A + curr,
00182                             port[c].negative - 1) += std::real(port[p].alpha[c]);
00183                 }
00184                 stamp.G(stamp.sizeG_A + curr,
00185                         stamp.sizeG_A + port[c].current -
00186                             1) += std::real(-z_ref * port[p].alpha[c]);
00187             }
00188         }
00189     }
00190 }
00191
00192 void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00193                         const size_t currentSolutionIndex,
00194                         T simulationTimestep) const {
00195     for (size_t p = 0; p < port.size(); p++) {
00196         size_t curr = port[p].current - 1;
00197         // V_p
00198         stamp.s(stamp.sizeG_A + curr,
00199                 0) += V_p(p, solutionMatrix, currentSolutionIndex,
00200                           simulationTimestep, stamp.sizeG_A);
00201     }
00202 }
00203
00204 void updateStoredState(const Matrix<T> & solutionMatrix,
00205                         const size_t currentSolutionIndex, T timestep,
00206                         size_t sizeG_A) {
00207     for (size_t p = 0; p < numPorts; p++) {
00208         for (size_t c = 0; c < numPorts; c++) {
00209             for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00210                 port[p].from[c].x[rho] = port[p].from[c].x[rho] *
00211                     port[p].from[c].exp_alpha[rho] +
00212                     port[p].from[c].lambda_p[rho] *
00213                     awave_p(c, solutionMatrix,
00214                             currentSolutionIndex,
00215                             sizeG_A) +
00216                     port[p].from[c].mu_p[rho] *
00217                     awave_p(c, solutionMatrix,
00218                             currentSolutionIndex - 1,
00219                             sizeG_A);
00220             if (!firstOrder) {
00221                 port[p].from[c].x[rho] += port[p].from[c].nu_p[rho] *
00222                     awave_p(c, solutionMatrix,
00223                             currentSolutionIndex - 2,

```

```

00224
00225         }
00226     }
00227   }
00228 }
00229
00230   if (firstOrder && currentSolutionIndex >= 1) {
00231     setSecondOrder(timestep);
00232   }
00233 }
00234
00235 void setConstants(T timestep) {
00236   for (size_t p = 0; p < numPorts; p++) {
00237     port[p].beta = 1.0 - port[p].from[p].lambda - port[p].from[p].remainder;
00238     port[p].beta = 1.0 / port[p].beta;
00239
00240     port[p].R = 1.0 + port[p].from[p].lambda + port[p].from[p].remainder;
00241
00242     port[p].R = z_ref * port[p].R * port[p].beta;
00243
00244     for (size_t c = 0; c < numPorts; c++) {
00245       if (c == p) {
00246         port[p].alpha[c] = 0;
00247       } else {
00248         port[p].alpha[c] = port[p].from[c].lambda +
00249                           port[p].from[c].remainder;
00250         port[p].alpha[c] = port[p].alpha[c] * port[p].beta;
00251       }
00252     }
00253   }
00254 }
00255
00256 void setFirstOrder(T timestep) {
00257   firstOrder = true;
00258
00259   for (size_t p = 0; p < numPorts; p++) {
00260     for (size_t c = 0; c < numPorts; c++) {
00261       port[p].from[c].lambda = 0;
00262       port[p].from[c].mu = 0;
00263       port[p].from[c].nu = 0;
00264       for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00265         const auto & pole = port[p].from[c].pole[rho];
00266         const auto & residue = port[p].from[c].residue[rho];
00267         const auto a = pole * timestep;
00268         const auto ea = std::exp(a);
00269         port[p].from[c].lambda_p[rho] = -(residue / pole) *
00270                                       (1.0 + (1.0 - ea) / (a));
00271         port[p].from[c].lambda += port[p].from[c].lambda_p[rho];
00272
00273         port[p].from[c].mu_p[rho] = -(residue / pole) *
00274                                       ((ea - 1.0) / a - ea);
00275         port[p].from[c].mu += port[p].from[c].mu_p[rho];
00276
00277         port[p].from[c].nu_p[rho] = 0;
00278       }
00279     }
00280   }
00281
00282   setConstants(timestep);
00283 }
00284
00285 void setSecondOrder(T timestep) {
00286   firstOrder = false;
00287
00288   for (size_t p = 0; p < numPorts; p++) {
00289     for (size_t c = 0; c < numPorts; c++) {
00290       port[p].from[c].lambda = 0;
00291       port[p].from[c].mu = 0;
00292       port[p].from[c].nu = 0;
00293       for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00294         const auto & pole = port[p].from[c].pole[rho];
00295         const auto & residue = port[p].from[c].residue[rho];
00296         const auto a = pole * timestep;
00297         const auto ea = std::exp(a);
00298         port[p].from[c].lambda_p[rho] = -(residue / pole) *
00299                                       (((1.0 - ea) / (a * a) +
00300                                         (3.0 - ea) / (2.0 * a) + 1.0));
00301         port[p].from[c].lambda += port[p].from[c].lambda_p[rho];
00302
00303         port[p].from[c].mu_p[rho] = -(residue / pole) *
00304                                       (-2.0 * (1.0 - ea) / (a * a) -
00305                                         (2.0 / a) - ea);
00306         port[p].from[c].mu += port[p].from[c].mu_p[rho];
00307
00308         port[p].from[c].nu_p[rho] = -(residue / pole) *
00309                                       (((1.0 - ea) / (a * a) +
00310                                         (1.0 + ea) / (2.0 * a)));
00310

```

```

00311             port[p].from[c].nu += port[p].from[c].nu_p[rho];
00312         }
00313     }
00314 }
00315
00316     setConstants(timestep);
00317 }
00318
00319 void setTimestep(T timestep) {
00320     for (size_t p = 0; p < numPorts; p++) {
00321         for (size_t c = 0; c < numPorts; c++) {
00322             for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00323                 port[p].from[c].exp_alpha[rho] = std::exp(
00324                     port[p].from[c].pole[rho] * timestep);
00325             }
00326         }
00327     }
00328     if (firstOrder) {
00329         setFirstOrder(timestep);
00330     } else {
00331         setSecondOrder(timestep);
00332     }
00333 }
00334 }
00335 #ifdef WITH_MATLAB
00336 void
00337 performVectorFit(std::string filePath, size_t numPorts,
00338                     std::shared_ptr<matlab::engine::MATLABEngine> matlabEngine) {
00339     matlab::data::ArrayFactory factory;
00340
00341     matlabEngine->eval(u"addpath('./Matlab');");
00342     std::vector<matlab::data::Array> args;
00343     args.push_back(factory.createCharArray(filePath));
00344     auto result = matlabEngine->feval(u"CPPVectFitAdaptor", 3, args);
00345
00346     assert(result[0].getDimensions()[0] == numPorts);
00347     assert(result[0].getDimensions()[1] == numPorts);
00348
00349     z_ref = static_cast<T>(result[1][0]);
00350     for (size_t a = 0; a < numPorts; a++) {
00351         port[a].alpha.resize(numPorts);
00352         port[a].from.resize(numPorts);
00353         for (size_t b = 0; b < numPorts; b++) {
00354             auto structArrayResult = static_cast<
00355                 matlab::data::TypedArray<matlab::data::Struct>>(result[0]);
00356             auto structResult = static_cast<matlab::data::Struct>(
00357                 structArrayResult[a][b]);
00358
00359             port[a].from[b].numPoles = structResult["poles"]
00360                             .getNumberOfElements();
00361             port[a].from[b].lambda_p.resize(port[a].from[b].numPoles);
00362             port[a].from[b].mu_p.resize(port[a].from[b].numPoles);
00363             port[a].from[b].nu_p.resize(port[a].from[b].numPoles);
00364             port[a].from[b].exp_alpha.resize(port[a].from[b].numPoles);
00365             port[a].from[b].x.resize(port[a].from[b].numPoles);
00366             port[a].from[b].pole.resize(port[a].from[b].numPoles);
00367             port[a].from[b].residue.resize(port[a].from[b].numPoles);
00368
00369             for (size_t p = 0; p < port[a].from[b].numPoles; p++) {
00370                 port[a].from[b].pole[p] = static_cast<std::complex<T>>(
00371                     static_cast<matlab::data::TypedArray<std::complex<T>>>(
00372                         structResult["poles"])[p]);
00373                 port[a].from[b].residue[p] = static_cast<std::complex<T>>(
00374                     static_cast<matlab::data::TypedArray<std::complex<T>>>(
00375                         structResult["residues"])[p]);
00376             }
00377             port[a].from[b].remainder = static_cast<std::complex<T>>(
00378                 static_cast<matlab::data::TypedArray<std::complex<T>>>(
00379                     structResult["remainder"])[0]);
00380         }
00381     }
00382 }
00383 #endif
00384
00385 void readInPRR(std::string filePath, size_t numPorts) {
00386     std::ifstream file(filePath);
00387
00388
00389     std::string line;
00390     while (file.peek() == '#' || file.peek() == '!') {
00391         std::getline(file, line);
00392     }
00393
00394     T rval;
00395     T cval;
00396
00397     std::stringstream polesLine;

```

```

00398     std::stringstream residuesLine;
00399
00400     file >> z_ref;
00401
00402     while (!file.eof()) {
00403         for (size_t a = 0; a < numPorts; a++) {
00404             port[a].alpha.resize(numPorts);
00405             port[a].from.resize(numPorts);
00406             for (size_t c = 0; c < numPorts; c++) {
00407                 file >> rval >> cval;
00408                 port[a].from[c].remainder = std::complex(rval, cval);
00409
00410             if (file.fail())
00411                 break;
00412         }
00413
00414         std::string line,
00415             polesLine; // have to clear the end of the remainder line
00416         std::getline(file, line);
00417         polesLine = std::stringstream(line);
00418         std::getline(file, line);
00419         residuesLine = std::stringstream(line);
00420
00421         while (!polesLine.eof() && !residuesLine.eof()) {
00422             polesLine >> rval >> cval;
00423             port[a].from[c].pole.emplace_back(rval, cval);
00424
00425             residuesLine >> rval >> cval;
00426             port[a].from[c].residue.emplace_back(rval, cval);
00427         }
00428
00429         port[a].from[c].numPoles = port[a].from[c].pole.size();
00430         port[a].from[c].lambda_p.resize(port[a].from[c].numPoles);
00431         port[a].from[c].mu_p.resize(port[a].from[c].numPoles);
00432         port[a].from[c].nu_p.resize(port[a].from[c].numPoles);
00433         port[a].from[c].exp_alpha.resize(port[a].from[c].numPoles);
00434         port[a].from[c].x.resize(port[a].from[c].numPoles);
00435
00436     }
00437
00438     if (file.fail())
00439         break;
00440     }
00441 }
00442
00443 }
00444
00445 void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00446                           size_t numCurrents) const {
00447     size_t numPorts = port.size();
00448     std::vector<std::complex<T> > xSum(port.size() * port.size());
00449     for (size_t p = 0; p < port.size(); p++) {
00450         for (size_t c = 0; c < port.size(); c++) {
00451             for (size_t rho = 0; rho < port[p].from[c].numPoles; rho++) {
00452                 xSum[p * numPorts + c] += -port[p].from[c].lambda_p[rho] +
00453                                         port[p].from[c].mu_p[rho]) /
00454                                         (port[p].from[c].exp_alpha[rho] - 1.0);
00455             }
00456             xSum[p * numPorts + c] += port[p].from[c].remainder;
00457         }
00458     }
00459
00460     for (size_t p = 0; p < port.size(); p++) {
00461         size_t np = port[p].positive - 1;
00462         size_t nn = port[p].negative - 1;
00463         size_t curr = port[p].current - 1;
00464
00465         std::complex<T> beta = 1.0 / (1.0 - xSum[p * numPorts + p]);
00466
00467         stamp.G(stamp.sizeG_A + curr, stamp.sizeG_A + curr) += std::real(
00468             -z_ref * (1.0 + xSum[p * numPorts + p]) * beta);
00469
00470         if (port[p].positive != 0) {
00471             stamp.G(stamp.sizeG_A + curr, np) += 1;
00472             stamp.G(np, stamp.sizeG_A + curr) += 1;
00473         }
00474
00475         if (port[p].negative != 0) {
00476             stamp.G(stamp.sizeG_A + curr, nn) += -1;
00477             stamp.G(nn, stamp.sizeG_A + curr) += -1;
00478         }
00479     // controlled sources
00480     for (size_t c = 0; c < port.size(); c++) {
00481         if (c != p) {
00482             if (port[c].positive != 0) {
00483                 stamp.G(stamp.sizeG_A + curr,
00484                         port[c].positive -

```

```

00485                         1) += std::real(-xSum[p * numPorts + c] * beta);
00486                     }
00487                     if (port[c].negative != 0) {
00488                         stamp.G(stamp.sizeG_A + curr,
00489                             port[c].negative -
00490                             1) += std::real(xSum[p * numPorts + c] * beta);
00491                     }
00492                     stamp.G(stamp.sizeG_A + curr,
00493                         stamp.sizeG_A + port[c].current -
00494                             1) += std::real(-z_ref * xSum[p * numPorts + c] *
00495                             beta);
00496                 }
00497             }
00498         }
00499     }
00500
00501     static void
00502     addToElements(const std::string & line, CircuitElements<T> & elements,
00503         size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00504         // std::regex SParameterBlockInitialRegex( R"(^S(.*)?\s(\d+?)\s)" );
00505         std::regex SParameterBlockInitialRegex = generateRegex("SV", "n s", true,
00506                                         false);
00507         std::smatch matches;
00508         std::regex_search(line, matches, SParameterBlockInitialRegex);
00509
00510         SParameterBlockVF<T> block;
00511
00512         block.designator = "SV";
00513         block.designator += matches.str(1);
00514
00515         size_t numPorts = std::stoul(matches.str(2));
00516         block.numPorts = numPorts;
00517         block.port = std::vector<SParameterPortVF<T> >(numPorts);
00518
00519         auto strIter = matches.suffix().first;
00520         std::regex SParameterBlockPortRegex(R"^(^(\d+?)\s(\d+?)\s)" );
00521         for (size_t p = 0; p < numPorts; p++) {
00522             std::regex_search(strIter, line.cend(), matches,
00523                             SParameterBlockPortRegex);
00524             block.port[p].positive = std::stoi(matches.str(1));
00525             block.port[p].negative = std::stoi(matches.str(2));
00526
00527             numNodes = std::max(numNodes, std::stoull(matches.str(1)));
00528             numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00529
00530             block.port[p].current = +numCurrents;
00531
00532             strIter = matches.suffix().first;
00533         }
00534         std::regex endOfLineRegex(R"^(.*$)" );
00535         std::regex_search(strIter, line.cend(), matches, endOfLineRegex);
00536         if (line[2] == 'F') {
00537 #ifdef WITH_MATLAB
00538             block.performVectorFit(matches.str(1), numPorts, elements.matlabEngine);
00539 #endif
00540         } else {
00541             block.readInPRR(matches.str(1), numPorts);
00542         }
00543
00544         elements.dynamicElements.emplace_back(
00545             std::make_shared<SParameterBlockVF<T> >(block));
00546         for (size_t p = 0; p < numPorts; p++) {
00547             elements.nodeComponentMap.insert(
00548                 {block.port[p].positive, elements.dynamicElements.back()},
00549                 {block.port[p].negative, elements.dynamicElements.back()});
00550         }
00551     }
00552 };
00553
00554 #endif

```

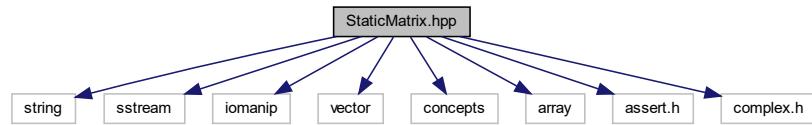
7.49 StaticMatrix.hpp File Reference

```

#include <string>
#include <sstream>
#include <iomanip>
#include <vector>
#include <concepts>
#include <array>

```

```
#include <assert.h>
#include <complex.h>
Include dependency graph for StaticMatrix.hpp:
```



Classes

- struct [StaticRow< T, N, ST >](#)
A compile-time sized matrix row.
- struct [StaticMatrix< T, M, N, ST >](#)
A compile-time sized matrix.
- struct [StaticLUPair< T, M >](#)
A compile-time sized L U and pivot grouping.

Typedefs

- template<typename T, size_t N = 1>
 using [storageType](#) = typename std::conditional< N *sizeof(T)< 20000, std::array< T, N >, std::vector< T > >::type

Functions

- template<typename T >
`std::complex< double > operator>` (const std::complex< T > &a, const std::complex< T > &b)
- template<typename T >
`std::complex< double > operator<` (const std::complex< T > &a, const std::complex< T > &b)
- template<typename T >
`std::complex< double > operator>=` (const std::complex< T > &a, const std::complex< T > &b)
- template<typename T >
`std::complex< double > operator<=` (const std::complex< T > &a, const std::complex< T > &b)

Variables

- template<typename T >
concept [arithmetic](#)

7.49.1 Typedef Documentation

7.49.1.1 storageType

```
template<typename T , size_t N = 1>
using storageType = typename std::conditional< N * sizeof(T) < 20000, std::array<T, N>, std::vector<T> >::type
```

Definition at line 13 of file [StaticMatrix.hpp](#).

7.49.2 Function Documentation

7.49.2.1 operator<()

```
template<typename T >
std::complex<double> operator< (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 18 of file [StaticMatrix.hpp](#).

7.49.2.2 operator<=()

```
template<typename T >
std::complex<double> operator<= (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 30 of file [StaticMatrix.hpp](#).

7.49.2.3 operator>()

```
template<typename T >
std::complex<double> operator> (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 18 of file [StaticMatrix.hpp](#).

7.49.2.4 operator>=()

```
template<typename T >
std::complex<double> operator>= (
    const std::complex< T > & a,
    const std::complex< T > & b )
```

Definition at line 30 of file [StaticMatrix.hpp](#).

7.49.3 Variable Documentation

7.49.3.1 arithmetic

```
template<typename T >
concept arithmetic
```

Initial value:

```
= requires(T a, T b) {
    {a == b};
    {a != b};
    {a >= b};
    {a <= b};
    {a * b};
    {a / b};
    {a + b};
    {a - b};
}
```

Definition at line 41 of file [StaticMatrix.hpp](#).

7.50 StaticMatrix.hpp

```
00001 #ifndef _MATRIX_HPP_INC_
00002 #define _MATRIX_HPP_INC_
00003 #include <string>
00004 #include <iostream>
00005 #include <iomanip>
00006 #include <vector>
00007 #include <concepts>
00008 #include <array>
00009 #include <assert.h>
00010 #include <complex.h>
00011
00012 template<typename T, size_t N = 1>
00013 using storageType = typename std::conditional<
00014     N * sizeof(T) < 20000, std::array<T, N>, std::vector<T> >::type;
00015
00016 template<typename T>
00017 std::complex<double>
00018 operator>(const std::complex<T> & a, const std::complex<T> & b) {
00019     return std::norm(a) > std::norm(b);
00020 }
00021
00022 template<typename T>
00023 std::complex<double>
00024 operator<(const std::complex<T> & a, const std::complex<T> & b) {
00025     return std::norm(a) < std::norm(b);
00026 }
00027
00028 template<typename T>
00029 std::complex<double>
00030 operator>=(const std::complex<T> & a, const std::complex<T> & b) {
00031     return std::norm(a) >= std::norm(b);
00032 }
00033
```

```

00034 template<typename T>
00035     std::complex<double>
00036     operator<=(const std::complex<T> & a, const std::complex<T> & b) {
00037         return std::norm(a) <= std::norm(b);
00038     }
00039
00040 template<typename T>
00041 concept arithmetic = requires(T a, T b) {
00042     {a == b};
00043     {a != b};
00044     {a >= b};
00045     {a <= b};
00046     {a * b};
00047     {a / b};
00048     {a + b};
00049     {a - b};
00050 };
00051
00052 template<typename T, size_t N, typename ST = storageType<T, N> >
00053 requires arithmetic<T> struct StaticRow {
00054     ST columns;
00055
00056     static constexpr size_t sizeN = N;
00057
00058     StaticRow() {
00059         if constexpr (std::is_same<ST, std::vector<T> >::value) {
00060             columns.resize(N);
00061         }
00062     }
00063
00064     StaticRow(T initialValue) {
00065         if constexpr (std::is_same<ST, std::vector<T> >::value) {
00066             columns.resize(N);
00067         }
00068         columns.fill(initialValue);
00069     }
00070
00071     void fill(T value) {
00072         columns.fill(value);
00073     }
00074
00075     T & operator[](size_t index) {
00076         return columns[index];
00077     }
00078
00079     const T & operator[](size_t index) const {
00080         return columns[index];
00081     }
00082
00083     T dot(StaticRow<T, N> other) {
00084         T toRet = 0;
00085         for (size_t i = 0; i < sizeN; i++) {
00086             toRet += columns[i] * other.columns[i];
00087         }
00088         return toRet;
00089     }
00090 };
00091
00092 template<typename T, size_t M>
00093 struct StaticLUPair;
00094
00095
00096 template<typename T, size_t M, size_t N = M,
00097     typename ST = storageType<StaticRow<T, N>, M> >
00098 requires arithmetic<T> struct StaticMatrix {
00099     ST rows;
00100
00101     StaticMatrix() {
00102         if constexpr (std::is_same<ST, std::vector<StaticRow<T, N> > >::value) {
00103             rows.resize(M);
00104         }
00105     }
00106
00107     StaticMatrix(T initialValue) {
00108         if constexpr (std::is_same<ST, std::vector<T> >::value) {
00109             rows.resize(M);
00110         }
00111         fill(initialValue);
00112     }
00113
00114     /*
00115     StaticMatrix( const StaticMatrix< T, M, N, ST > & other ) {
00116         if constexpr ( std::is_same< ST, std::vector< T > >::value ) {
00117             rows.resize( M );
00118         }
00119         (*this) = other;
00120     }
00121
00122     }
00123
00124 
```

```

00132     StaticMatrix< T, M, N, ST > & operator=( const StaticMatrix< T, M, N, ST > &
00133     other ) { for ( size_t m = 0; m < M; m++ ) { for ( size_t n = 0; n < N; n++ ) {
00134         rows[ m
00135             ][ n ] = other[ m ][ n ];
00136         }
00137     }
00138     return *this;
00139 }
00140 */
00141
00142 static constexpr size_t sizeM = M;
00143 static constexpr size_t sizeN = N;
00144
00145 void fill(T fillVal) {
00146     for (auto & row : rows) {
00147         row.fill(fillVal);
00148     }
00149 }
00150
00151 StaticRow<T, N> & operator[](size_t index) {
00152     return rows[index];
00153 }
00154
00155 StaticRow<T, N> operator[](size_t index) const {
00156     return rows[index];
00157 }
00158
00159 void rowAddition(size_t destinationRow, size_t sourceRow, T scalingFactor) {
00160     assert(0 <= destinationRow && destinationRow <= N);
00161     assert(0 <= sourceRow && sourceRow <= M);
00162     for (size_t i = 0; i < N; i++) {
00163         rows[destinationRow][i] += scalingFactor * rows[sourceRow][i];
00164     }
00165 }
00166
00167 void swapRows(size_t row1, size_t row2) {
00168     std::swap(rows[row1], rows[row2]);
00169 }
00170
00171 StaticMatrix<T, N, M> transpose() {
00172     StaticMatrix<T, N, M> toRet;
00173     for (size_t m = 0; m < M; m++) {
00174         for (size_t n = 0; n < N; n++) {
00175             toRet[n][m] = rows[m][n];
00176         }
00177     }
00178     return toRet;
00179 }
00180
00181 template<size_t N2>
00182 StaticMatrix<T, M, N2> multiply(const StaticMatrix<T, N, N2> & rhs) const {
00183     StaticMatrix<T, M, N2> toRet(0.0);
00184     multiply(rhs, toRet);
00185     return toRet;
00186 }
00187
00188 template<size_t N2>
00189 void multiply(const StaticMatrix<T, N, N2> & rhs,
00190                 StaticMatrix<T, M, N2> & dest) const {
00191     // for SIMD reasons, the order of operations is slightly weird. This is to
00192     // minimise row changes which may cause cache misses. This is due to the fact
00193     // that in this model rows are represented contiguously in memory,
00194     // so streaming instructions and local caching can be used to our advantage
00195     //
00196     // It is also worth stating that there is strong potential for multithreading
00197     // here, and especially if paired with the Strassen Method for matrix
00198     // multiplication. However for small sizes, naive multiplication can be better
00199     // due to less memory allocations
00200
00201     for (size_t m = 0; m < M; m++) {
00202         for (size_t k = 0; k < N; k++) {
00203             for (size_t n = 0; n < N2; n++) {
00204                 destination[m][n] += rows[m][k] * rows[k][n];
00205             }
00206         }
00207     }
00208 }
00209
00210 StaticMatrix<T, M, N> add(const StaticMatrix<T, M, N> & rhs) const {
00211     StaticMatrix<T, M, N> toRet();
00212     add(rhs, toRet);
00213     return toRet;
00214 }
00215
00216 void add(const StaticMatrix<T, M, N> & rhs, StaticMatrix<T, M, N> & dest) const {
00217     for (size_t m = 0; m < M; m++) {
00218         for (size_t n = 0; n < N; n++) {
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01
```

```

00219             dest[m][n] = rows[m][n] + rhs[m][n];
00220         }
00221     }
00222 }
00223
00224 StaticMatrix<T, M, N> subtract(const StaticMatrix<T, N, N> & rhs) const {
00225     StaticMatrix<T, M, N> toRet();
00226     add(rhs, toRet);
00227     return toRet;
00228 }
00229
00230 void
00231 subtract(const StaticMatrix<T, N, N> & rhs, StaticMatrix<T, M, N> & dest) const {
00232     for (size_t m = 0; m < M; m++) {
00233         for (size_t n = 0; n < N; n++) {
00234             dest[m][n] = rows[m][n] - dest[m][n]
00235         }
00236     }
00237 }
00238
00239 std::string toString() const {
00240     std::stringstream toRet;
00241     for (const auto & row : rows) {
00242         for (const auto & val : row.columns) {
00243             toRet « std::setprecision(2) « val « " ";
00244         }
00245
00246         toRet « std::endl;
00247     }
00248
00249     return toRet.str();
00250 }
00251
00252 StaticLUPair<T, M> luPair() const {
00253     static_assert(N == M, "StaticMatrix must be a square");
00254
00255     StaticLUPair<T, M> toRet;
00256     luPair(toRet);
00257     return toRet;
00258 }
00259
00260 void luPair(StaticLUPair<T, M> & dest) const {
00261     static_assert(N == M, "StaticMatrix must be a square");
00262
00263     dest.u = *this;
00264     dest.l.fill(0.0);
00265     for (size_t n = 0; n < N; n++) {
00266         dest.l[n][n] = 1.0;
00267         dest.p[n] = n;
00268     }
00269
00270     for (size_t r = 0; r < M - 1; r++) {
00271         // find largest in column
00272         size_t largestRow = r;
00273         auto maxV = abs(dest.u[r][r]);
00274         for (size_t r2 = r + 1; r2 < M; r2++) {
00275             if (abs(dest.u[r2][r]) > maxV) {
00276                 maxV = abs(dest.u[r2][r]);
00277                 largestRow = r2;
00278             }
00279         }
00280
00281         // swap rows in U and indices in p
00282         dest.u.swapRows(r, largestRow);
00283         std::swap(dest.p[r], dest.p[largestRow]);
00284         // swap subdiagonal entries in L
00285         for (size_t n = 0; n < r; n++) {
00286             std::swap(dest.l[r][n], dest.l[largestRow][n]);
00287         }
00288
00289         // Gaussian elimination
00290         for (size_t m = r + 1; m < M; m++) {
00291             // TODO: potential for multithreading here
00292             // need to take into account how many processors there are
00293             // https://en.cppreference.com/w/cpp/thread/thread/hardware_concurrency
00294             T multiplier = dest.u[m][r] / dest.u[r][r];
00295             dest.u.rowAddition(m, r, -multiplier);
00296             dest.l[m][r] = multiplier;
00297         }
00298         // std::cout « "After Gaussian\n" « dest.toString();
00299     }
00300 }
00301
00302 StaticMatrix<T, M, 1> leftDivide(const StaticMatrix<T, M, 1> & rhs) const {
00303     auto lu = luPair();
00304     StaticMatrix<T, M, 1> scratchSpace;
00305     StaticMatrix<T, M, 1> toRet;

```

```

00306     leftDivide(rhs, lu, scratchSpace, toRet);
00307     return toRet;
00308 }
00309
00310 void leftDivide(const StaticMatrix<T, M, 1> & rhs, const StaticLUPair<T, M> & lu,
00311                 StaticMatrix<T, M, 1> & scratchSpace,
00312                 StaticMatrix<T, M, 1> & dest) const {
00313     for (size_t i = 0; i < M; i++) {
00314         dest[i] = rhs[lu.p[i]];
00315     }
00316
00317     // scratchSpace: solve LY = Pb for y using substitution
00318     for (size_t m = 0; m < M; m++) {
00319         T val = dest[m][0];
00320         for (size_t n = 0; n < m; n++) {
00321             val -= scratchSpace[n][0] * lu.l[m][n];
00322         }
00323         scratchSpace[m][0] = val / lu.l[m][m];
00324     }
00325
00326     // stage2: solve Ux = Y for x using substitution
00327     for (size_t m = 0; m < M; m++) {
00328         T val = scratchSpace[M - m - 1][0];
00329         for (size_t n = 0; n < m; n++) {
00330             val -= dest[M - n - 1][0] * lu.u[M - m - 1][N - n - 1];
00331         }
00332         dest[M - m - 1][0] = val / lu.u[M - m - 1][M - m - 1];
00333     }
00334 }
00335 };
00336
00341 template<typename T, size_t M>
00342 struct StaticLUPair {
00343     StaticMatrix<T, M, M> l;
00344     StaticMatrix<T, M, M> u;
00345     std::array<size_t, M> p;
00346
00347     std::string toString() {
00348         std::stringstream toRet;
00349         toRet << "U\n" << u.toString();
00350         toRet << "L\n" << l.toString();
00351         toRet << "p\n";
00352         for (size_t i = 0; i < p.size(); i++) {
00353             toRet << std::setw(5) << p[i] << " ";
00354         }
00355         toRet << std::endl;
00356         return toRet.str();
00357     }
00358 };
00359 // -----
00360 // Implementation
00361 // -----
00362
00363
00364 #endif

```

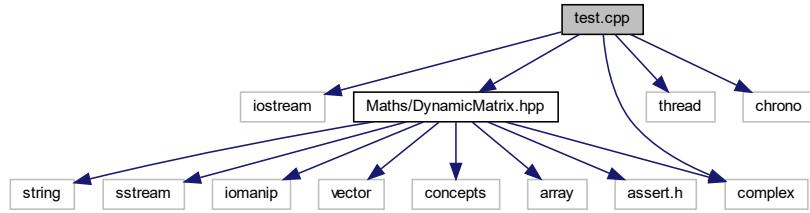
7.51 test.cpp File Reference

```

#include <iostream>
#include "Maths/DynamicMatrix.hpp"
#include <thread>
#include <complex>
#include <chrono>

```

Include dependency graph for test.cpp:



Functions

- int `main ()`

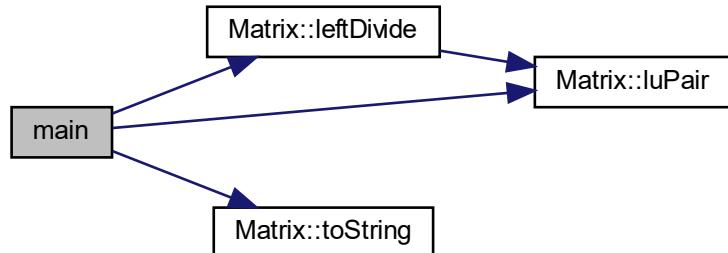
7.51.1 Function Documentation

7.51.1.1 main()

```
int main ( )
```

Definition at line 8 of file [test.cpp](#).

Here is the call graph for this function:



7.52 test.cpp

```
00001 #include <iostream>
00002 #include "Maths/DynamicMatrix.hpp"
00003 #include <thread>
00004 #include <complex>
00005 #include <chrono>
00006
00007 [[clang::optnone]] int
00008 main() {
00009     unsigned int nThreads = std::thread::hardware_concurrency();
00010     std::cout << nThreads << " concurrent threads are supported.\n";
00011
00012     std::cout << "2 x 2 Matrix\n" << std::endl;
00013     Matrix<double> myMat(2, 2);
00014     // std::cin >> d1 >> d2 >> d3 >> d4;
00015     myMat(0, 0) = 1.0;
00016     myMat(0, 1) = 1.0;
00017     myMat(1, 0) = 1.0;
00018     myMat(1, 1) = 2.0;
00019     std::cout << " myMat\n" << myMat.toString();
00020
00021     auto startTime = std::chrono::high_resolution_clock::now();
00022     auto lu = myMat.luPair();
00023     auto endTime = std::chrono::high_resolution_clock::now();
00024     auto timeTaken = std::chrono::duration_cast<std::chrono::milliseconds>(endTime -
00025                                         startTime)
00026                                         .count();
00027
00028     std::cout << timeTaken << "ms" << std::endl << lu.toString();
00029
00030     std::cout << "4 x 4 Matrix\n" << std::endl;
00031     Matrix<double> myMat4(4, 4);
00032     Matrix<std::complex<double>> myCMat4(4, 4);
00033     // std::cin >> d1 >> d2 >> d3 >> d4;
00034     myMat4(0, 0) = 2.0;
00035     myMat4(0, 1) = 1.0;
00036     myMat4(0, 2) = 1.0;
00037     myMat4(0, 3) = 0.0;
00038     myMat4(1, 0) = 4.0;
00039     myMat4(1, 1) = 3.0;
00040     myMat4(1, 2) = 3.0;
00041     myMat4(1, 3) = 1.0;
00042     myMat4(2, 0) = 8.0;
00043     myMat4(2, 1) = 7.0;
00044     myMat4(2, 2) = 9.0;
00045     myMat4(2, 3) = 5.0;
00046     myMat4(3, 0) = 6.0;
00047     myMat4(3, 1) = 7.0;
00048     myMat4(3, 2) = 9.0;
00049     myMat4(3, 3) = 8.0;
00050
00051     Matrix<double> myCol(4, 1);
00052     myCol(0, 0) = 1.0;
00053     myCol(1, 0) = 0.0;
00054     myCol(2, 0) = 3.0;
00055     myCol(3, 0) = 0.0;
00056
00057     myCMat4(0, 0) = 2.0;
00058     myCMat4(0, 1) = 1.0;
00059     myCMat4(0, 2) = 1.0;
00060     myCMat4(0, 3) = 0.0;
00061     myCMat4(1, 0) = 4.0;
00062     myCMat4(1, 1) = 3.0;
00063     myCMat4(1, 2) = 3.0;
00064     myCMat4(1, 3) = 1.0;
00065     myCMat4(2, 0) = 8.0;
00066     myCMat4(2, 1) = 7.0;
00067     myCMat4(2, 2) = 9.0;
00068     myCMat4(2, 3) = 5.0;
00069     myCMat4(3, 0) = 6.0;
00070     myCMat4(3, 1) = 7.0;
00071     myCMat4(3, 2) = 9.0;
00072     myCMat4(3, 3) = 8.0;
00073
00074     std::cout << " myMat\n" << myCMat4.toString();
00075
00076     auto luC4 = myCMat4.luPair();
00077     auto lu4 = myMat4.luPair();
00078     std::cout << lu4.toString();
00079     std::cout << luC4.toString();
00080
00081     startTime = std::chrono::high_resolution_clock::now();
00082     for (unsigned int i = 0; i < 1000000 /* ( unsigned int )-1 */; i++) {
00083         lu4 = myMat4.luPair();
00084     }
00085     endTime = std::chrono::high_resolution_clock::now();
```

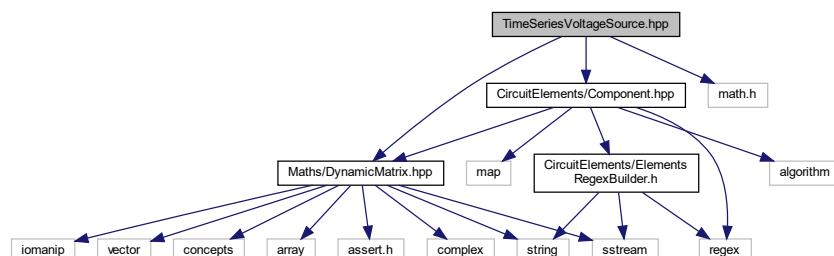
```

00086     timeTaken = std::chrono::duration_cast<std::chrono::milliseconds>(endTime -
00087                                         startTime)
00088                                         .count();
00089     std::cout << 1000000 << " 4 x 4 pairs in " << timeTaken << "ms" << std::endl;
00090
00091     auto solved = myMat4.leftDivide(myCol);
00092     std::cout << solved.toString();
00093
00094     Matrix<double> myMat1000(1000, 1000);
00095     for (size_t i = 0; i < 1000; i++) {
00096         myMat1000(i, 1000 - 1 - i) = i + 10;
00097     }
00098     startTime = std::chrono::high_resolution_clock::now();
00099     auto lu1000 = myMat1000.luPair();
00100     endTime = std::chrono::high_resolution_clock::now();
00101     timeTaken = std::chrono::duration_cast<std::chrono::milliseconds>(endTime -
00102                                         startTime)
00103                                         .count();
00104     std::cout << "1000 x 1000 LU comp in " << timeTaken << "ms" << std::endl;
00105
00106 // std::cout << lu1000.toString();
00107
00108     return 0;
00109 }
```

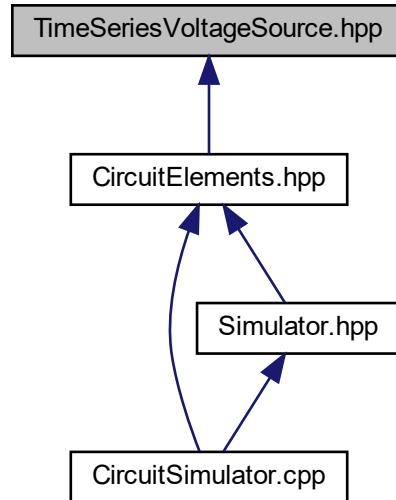
7.53 TimeSeriesVoltageSource.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
#include <math.h>
```

Include dependency graph for TimeSeriesVoltageSource.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct `TimeSeriesVoltageSource< T >`
A time series voltage source model.

7.54 TimeSeriesVoltageSource.hpp

```

00001 #ifndef _TIMESERIESVOLTAGESOURCE_INC_
00002 #define _TIMESERIESVOLTAGESOURCE_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005 #include <math.h>
00006
00010 template<typename T>
00011 struct TimeSeriesVoltageSource : public Component<T> {
00012     size_t n1 = 0;
00013     size_t n2 = 0;
00014     size_t currentIndex = 0;
00015     size_t lastTimeSeriesIndex = 0;
00016
00017     std::vector<T> timeSeries;
00018     std::vector<T> dataSeries;
00019
00020     T lerp(size_t lowIndex, T timeVal) const {
00021         T diffTS = timeSeries[(lowIndex + 1) % timeSeries.size()] -
00022                     timeSeries[lowIndex];
00023         T diffTV = timeVal - timeSeries[lowIndex];
00024         T diffDS = dataSeries[(lowIndex + 1) % timeSeries.size()] -
00025                     dataSeries[lowIndex];
00026         return dataSeries[lowIndex] + diffDS * diffTV / diffTS;
00027     }
00028
00029     void addDynamicStampTo(Stamp<T> & stamp, const Matrix<T> & solutionMatrix,
00030                           const size_t currentSolutionIndex, T timestep) const {
00031         size_t n1p = n1 - 1;
00032         size_t n2p = n2 - 1;
00033         size_t currentIndexp = currentIndex - 1;
00034         size_t timeSeriesIndex = lastTimeSeriesIndex;
00035         T timeMod = std::fmod(currentSolutionIndex * timestep, timeSeries.back());
00036         while (timeMod > timeSeries[(timeSeriesIndex + 1) % timeSeries.size()] ||
  
```

```

00037             (timeSeriesIndex != 0 &&
00038                 timeMod < timeSeries[(timeSeriesIndex - 1) % timeSeries.size()])) {
00039                 timeSeriesIndex = (timeSeriesIndex + 1) % timeSeries.size();
00040             }
00041
00042             if (n1) {
00043                 stamp.G(nlp, stamp.sizeG_A + currentIndexp) += 1;
00044                 stamp.G(stamp.sizeG_A + currentIndexp, nlp) += 1;
00045             }
00046
00047             if (n2) {
00048                 stamp.G(n2p, stamp.sizeG_A + currentIndexp) += -1;
00049                 stamp.G(stamp.sizeG_A + currentIndexp, n2p) += -1;
00050             }
00051
00052             stamp.s(stamp.sizeG_A + currentIndexp, 0) += lerp(timeSeriesIndex, timeMod);
00053         }
00054
00055     void updateStoredState(const Matrix<T> & solutionMatrix,
00056                             const size_t currentSolutionIndex, T timestep,
00057                             size_t sizeG_A) {
00058         while (std::fmod(currentSolutionIndex * timestep, timeSeries.back()) >
00059                timeSeries[(lastTimeSeriesIndex + 1) % timeSeries.size()] ||
00060                (lastTimeSeriesIndex != 0 &&
00061                 std::fmod(currentSolutionIndex * timestep, timeSeries.back()) <
00062                 timeSeries[(lastTimeSeriesIndex - 1) % timeSeries.size()])) {
00063             lastTimeSeriesIndex = (lastTimeSeriesIndex + 1) % timeSeries.size();
00064         }
00065     }
00066
00067     void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
00068                               size_t numCurrents) const {
00069         addDynamicStampTo(stamp, solutionVector, 0, 0);
00070     }
00071
00072     static void
00073     addToElements(const std::string & line, CircuitElements<T> & elements,
00074                   size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00075         // n1 n2 timescale file
00076         std::regex timeSeriesVoltageSourceRegex = generateRegex("VT", "n n w s",
00077                                                               true, false);
00078
00079         TimeSeriesVoltageSource<T> voltageSource;
00080         std::smatch matches;
00081
00082         std::regex_search(line, matches, timeSeriesVoltageSourceRegex);
00083
00084         voltageSource.designator = "VT";
00085         voltageSource.designator += matches.str(1);
00086
00087         voltageSource.n1 = std::stoi(matches.str(2));
00088         voltageSource.n2 = std::stoi(matches.str(3));
00089
00090         numNodes = std::max(numNodes, std::stoul(matches.str(2)));
00091         numNodes = std::max(numNodes, std::stoul(matches.str(3)));
00092
00093         T timescale = 0;
00094         if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00095             timescale = std::stod(matches.str(4));
00096         } else {
00097             static_assert("Unsupported Type");
00098         }
00099
00100         voltageSource.currentIndex = ++numCurrents;
00101
00102         std::regex endOfLineRegex(R"(^(\.*$)");
00103         std::regex_search(matches.suffix().first, line.cend(), matches,
00104                           endOfLineRegex);
00105
00106         std::string seriesPath = matches.str(1);
00107         voltageSource.readInTimeSeries(timescale, seriesPath);
00108
00109         elements.dynamicElements.emplace_back(
00110             std::make_shared<TimeSeriesVoltageSource<T>>(voltageSource));
00111         elements.nodeComponentMap.insert(
00112             {{voltageSource.n1, elements.dynamicElements.back()},
00113              {voltageSource.n2, elements.dynamicElements.back()}});
00114     }
00115
00116
00117     void readInTimeSeries(T timescale, const std::string & seriesPath) {
00118         std::ifstream file(seriesPath);
00119
00120         std::string line;
00121         T time;
00122         T val;
00123

```

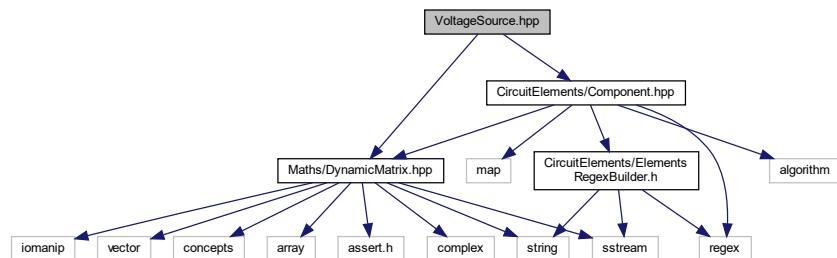
```

00124     while (!file.eof()) {
00125         if (!std::isdigit(file.peek())) {
00126             std::getline(file, line);
00127             continue;
00128         }
00129         file >> time;
00130         while (!std::isdigit(file.peek()) &&
00131             !(file.peek() == '-' || file.peek() == '+')) {
00132             file.ignore(1);
00133         }
00134         file >> val;
00135         timeSeries.emplace_back(time * timescale);
00136         dataSeries.emplace_back(val);
00137     }
00138 }
00139 }
00140 }
00141 }
00142 }
00143 };
00144
00145 #endif

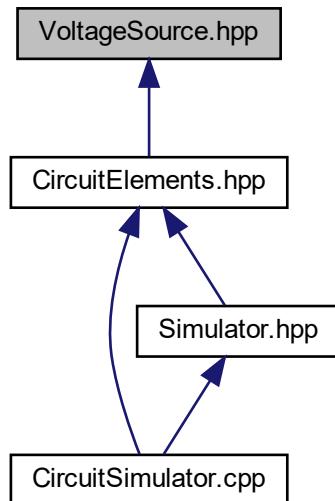
```

7.55 VoltageSource.hpp File Reference

```
#include "CircuitElements/Component.hpp"
#include "Maths/DynamicMatrix.hpp"
Include dependency graph for VoltageSource.hpp:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct [VoltageSource< T >](#)

An ideal voltageSource.

7.56 VoltageSource.hpp

```

00001 #ifndef _VOLTAGESOURCE_HPP_INC_
00002 #define _VOLTAGESOURCE_HPP_INC_
00003 #include "CircuitElements/Component.hpp"
00004 #include "Maths/DynamicMatrix.hpp"
00005
00009 template<typename T>
00010 struct VoltageSource : public Component<T> {
00011     public:
00012         T value = 0;
00013
00014         size_t n1 = 0;
00015         size_t n2 = 0;
00016         size_t currentIndex = 0;
00017
00018     void addStaticStampTo(Stamp<T> & stamp) const {
00019         // the p means prime (') and is used for the index - 1
00020         size_t n1p = n1 - 1;
00021         size_t n2p = n2 - 1;
00022         size_t currentIndexp = currentIndex - 1;
00023
00024         if (n1) {
00025             stamp.G(n1p, stamp.sizeG_A + currentIndexp) += 1;
00026             stamp.G(stamp.sizeG_A + currentIndexp, n1p) += 1;
00027         }
00028         if (n2) {
00029             stamp.G(n2p, stamp.sizeG_A + currentIndexp) += -1;
00030             stamp.G(stamp.sizeG_A + currentIndexp, n2p) += -1;
00031         }
00032
00033         stamp.s(stamp.sizeG_A + currentIndexp, 0) += value;
00034     }
00035
00036     void addDCAnalysisStampTo(Stamp<T> & stamp, const Matrix<T> & solutionVector,
  
```

```
00037     size_t numCurrents) const {
00038         addStaticStampTo(stamp);
00039     }
00040
00041     static void
00042     addToElements(const std::string & line, CircuitElements<T> & elements,
00043                 size_t & numNodes, size_t & numCurrents, size_t & numDCCurrents) {
00044         // std::regex voltageSourceRegex( R"(^V(.*)\s(\d+?)\s(\d+?)\s(.+)\s?)" );
00045         std::regex voltageSourceRegex = generateRegex("V", "n n w");
00046         VoltageSource<T> voltageSource;
00047         std::smatch matches;
00048
00049         std::regex_match(line, matches, voltageSourceRegex);
00050
00051         voltageSource.designator = "V";
00052         voltageSource.designator += matches.str(1);
00053
00054         voltageSource.n1 = std::stoi(matches.str(2));
00055         voltageSource.n2 = std::stoi(matches.str(3));
00056
00057
00058         numNodes = std::max(numNodes, std::stoull(matches.str(2)));
00059         numNodes = std::max(numNodes, std::stoull(matches.str(3)));
00060
00061         if constexpr (std::is_same_v<T, double> || std::is_same_v<T, float>) {
00062             voltageSource.value = std::stod(matches.str(4));
00063         } else {
00064             static_assert("Unsupported Type");
00065         }
00066
00067         voltageSource.currentIndex = ++numCurrents;
00068
00069         elements.staticElements.emplace_back(
00070             std::make_shared<VoltageSource<T>>(voltageSource));
00071         elements.nodeComponentMap.insert(
00072             {{voltageSource.n1, elements.staticElements.back()},
00073              {voltageSource.n2, elements.staticElements.back()}});
00074     }
00075 };
00076
00077 #endif
```


Index

_data
 SParameterSequence< T >, 146

_fftHelperRadix2
 dft.hpp, 216

_time
 SParameterSequence< T >, 146

~Component
 Component< T >, 51

add
 Matrix< T >, 75
 Stamp< T >, 153
 StaticMatrix< T, M, N, ST >, 160

AddableResult
 AutoDifferentiation, 11

addDCAnalysisStamp
 Stamp< T >, 153

addDCAnalysisStampTo
 BJTN< T >, 20
 BJTP< T >, 26
 Capacitor< T >, 34
 Component< T >, 52
 CurrentSource< T >, 57
 Diode< T >, 61
 Inductor< T >, 67
 NLCapacitor< T >, 82
 NLCURRENTSource< T >, 88
 Resistor< T >, 101
 SinusoidalVoltageSource< T >, 116
 SParameterBlock< T >, 122
 SParameterBlockVF< T >, 130
 TimeSeriesVoltageSource< T >, 170
 VoltageSource< T >, 177

addDynamicStamp
 Stamp< T >, 153

addDynamicStampTo
 Capacitor< T >, 35
 Component< T >, 52
 Inductor< T >, 67
 SinusoidalVoltageSource< T >, 117
 SParameterBlock< T >, 122
 SParameterBlockVF< T >, 130
 TimeSeriesVoltageSource< T >, 171

addNonLinearStamp
 Stamp< T >, 154

addNonLinearStampTo
 BJTN< T >, 21
 BJTP< T >, 27
 Component< T >, 52
 Diode< T >, 62

NLCapacitor< T >, 83
NLCURRENTSource< T >, 90
NLNMOS< T >, 95

addStaticStamp
 Stamp< T >, 154

addStaticStampTo
 Component< T >, 53
 CurrentSource< T >, 58
 Resistor< T >, 102
 SParameterBlock< T >, 123
 SParameterBlockVF< T >, 131
 VoltageSource< T >, 178

addToElements
 BJTN< T >, 22
 BJTP< T >, 28
 Capacitor< T >, 35
 Component< T >, 53
 CurrentSource< T >, 58
 Diode< T >, 63
 Inductor< T >, 68
 NLCapacitor< T >, 83
 NLCURRENTSource< T >, 91
 NLNMOS< T >, 95
 Resistor< T >, 102
 SinusoidalVoltageSource< T >, 117
 SParameterBlock< T >, 123
 SParameterBlockVF< T >, 131
 TimeSeriesVoltageSource< T >, 172
 VoltageSource< T >, 178

alpha
 SParameterPortVF< T >, 141

alpha_DS
 NLNMOS< T >, 96

alpha_f
 BJTN< T >, 23
 BJTP< T >, 29

alpha_r
 BJTN< T >, 23
 BJTP< T >, 29

arg
 AutoDifferentiation, 11

arithmetic
 DynamicMatrix.hpp, 224
 StaticMatrix.hpp, 281

AutoDifferentiation, 9
 AddableResult, 11
 arg, 11
 deriv, 11
 derivEval, 12

DivisibleResult, 12
 exponent, 12
 func, 12
 it1, 12
 it2, 13
 it3, 13
 MultipliableResult, 13
 requires, 10
 SubtractableResult, 13
 toRet, 10, 11, 13
 var, 14
 while, 11
 AutoDifferentiation.hpp, 181, 183
 AutoDiffTest.cpp, 188, 193
 main, 189
 testBasicOutput_NoCheck, 189
 testBJTModel, 189
 testBJTModelAutoDiff, 190
 testBJTModelControl, 190
 transistorTest, 191
 transistorTestAutoDiff, 191
 transistorTestControl, 192
 aware_p
 SParameterBlockVF< T >, 132
 aWaveConvValue
 SParameterBlock< T >, 124
 b
 BJTN< T >, 23
 BJTP< T >, 29
 beta
 SParameterPort< T >, 139
 SParameterPortVF< T >, 141
 beta_DS
 NLNMOS< T >, 96
 beta_p
 SParameterBlock< T >, 124
 BJT
 CircuitElements.hpp, 205
 BJT.hpp, 197, 198
 BJTN< T >, 19
 addDCAnalysisStampTo, 20
 addNonLinearStampTo, 21
 addToElements, 22
 alpha_f, 23
 alpha_r, 23
 b, 23
 c, 23
 e, 24
 I_cs, 24
 I_es, 24
 updateStoredState, 22
 V_bc_crit, 24
 V_be_crit, 24
 V_Tc, 24
 V_Te, 25
 BJTP< T >, 25
 addDCAnalysisStampTo, 26
 addNonLinearStampTo, 27
 addToElements, 28
 alpha_f, 29
 alpha_r, 29
 b, 29
 c, 29
 e, 30
 I_cs, 30
 I_es, 30
 updateStoredState, 28
 V_bc_crit, 30
 V_be_crit, 30
 V_Tc, 30
 V_Te, 31
 BJTResults< T >, 31
 g_cc, 31
 g_ce, 31
 g_ec, 32
 g_ee, 32
 I_c, 32
 I_e, 32
 bwave_p
 SParameterBlockVF< T >, 132
 c
 BJTN< T >, 23
 BJTP< T >, 29
 C_GD_last
 NLNMOS< T >, 97
 C_GDo
 NLNMOS< T >, 97
 C_GDp
 NLNMOS< T >, 97
 C_GS_last
 NLNMOS< T >, 97
 C_GSo
 NLNMOS< T >, 97
 C_GSp
 NLNMOS< T >, 97
 C_last
 NLCapacitor< T >, 85
 C_o
 NLCapacitor< T >, 85
 C_p
 NLCapacitor< T >, 85
 Capacitor
 CircuitElements.hpp, 205
 Simulator.hpp, 257
 Capacitor< T >, 33
 addDCAnalysisStampTo, 34
 addDynamicStampTo, 35
 addToElements, 35
 lastCurrent, 36
 n1, 36
 n2, 37
 trapezoidalRule, 37
 updateStoredState, 36
 value, 37
 Capacitor.hpp, 201, 202
 CircuitElements

CircuitElements< T >, 40
CircuitElements< T >, 39
CircuitElements, 40
dcStamp, 46
dynamicElements, 46
dynamicStamp, 47
dynamicStampIsFresh, 47
generateCompleteStamp, 41
generateDCStamp, 41
generateDynamicStamp, 42
generateNonLinearStamp, 43
generateStaticStamp, 44
nodeComponentMap, 47
nonLinearElements, 47
nonLinearStamp, 48
nonLinearStampIsFresh, 48
setNewStampSize, 44
staticElements, 48
staticStamp, 48
staticStampIsFresh, 48
updateDCStoredState, 45
updateTimeStep, 46
CircuitElements.hpp, 204, 206
BJT, 205
Capacitor, 205
ComponentType, 205
CurrentSource, 205
Diode, 205
DynamicSolution, 205
Inductor, 205
NonLinearSolution, 205
Resistor, 205
SolutionStage, 205
StaticSolution, 205
VoltageSource, 205
CircuitSimulator.cpp, 208, 209
main, 209
clear
 Stamp< T >, 154
columns
 StaticRow< T, N, ST >, 168
Comment
 Simulator.hpp, 257
Component< T >, 49
 ~Component, 51
 addDCAnalysisStampTo, 52
 addDynamicStampTo, 52
 addNonLinearStampTo, 52
 addStaticStampTo, 53
 addToElements, 53
 designator, 55
 setTimestep, 54
 updateDCStoredState, 54
 updateStoredState, 55
Component.hpp, 211
ComponentType
 CircuitElements.hpp, 205
current
 SParameterPort< T >, 139
 SParameterPortVF< T >, 141
currentIndex
 Resistor< T >, 103
 SinusoidalVoltageSource< T >, 118
 TimeSeriesVoltageSource< T >, 174
 VoltageSource< T >, 179
CurrentSource
 CircuitElements.hpp, 205
 Simulator.hpp, 257
CurrentSource< T >, 56
 addDCAnalysisStampTo, 57
 addStaticStampTo, 58
 addToElements, 58
 n1, 59
 n2, 59
 value, 59
CurrentSource.hpp, 213, 214
d
 NLNMOS< T >, 98
data
 ForceCausal::CausalData< T >, 38
 Matrix< T >, 80
 SParameterSequence< T >, 143
dataDump
 SimulationEnvironment< VT >, 107
dataSeries
 TimeSeriesVoltageSource< T >, 174
dcCurrentIndex
 Inductor< T >, 69
dcStamp
 CircuitElements< T >, 46
degrees
 SinusoidalVoltageSource< T >, 119
deriv
 AutoDifferentiation, 11
derivEval
 AutoDifferentiation, 12
designator
 Component< T >, 55
dft
 dft.hpp, 216
dft.hpp, 214, 219
 _fftHelperRadix2, 216
 dft, 216
 fft, 217
 idft, 217
 ifft, 218
 nthRootOfUnity, 218
diff1
 TransistorTestResult< T >, 175
diff2
 TransistorTestResult< T >, 175
Diode
 CircuitElements.hpp, 205
 Simulator.hpp, 257
Diode< T >, 60
 addDCAnalysisStampTo, 61

addNonLinearStampTo, 62
 addToElements, 63
 eta, 64
 I_sat, 64
 n1, 64
 n2, 64
 updateStoredState, 63
 V_crit, 65
 V_T, 65
 Diode.hpp, 220, 221
 Directive
 Simulator.hpp, 257
 DivisibleResult
 AutoDifferentiation, 12
 dot
 StaticRow< T, N, ST >, 168
 dynamicElements
 CircuitElements< T >, 46
 DynamicMatrix.hpp, 222, 225
 arithmetic, 224
 operator<, 223
 operator<=, 224
 operator>, 224
 operator>=, 224
 DynamicSolution
 CircuitElements.hpp, 205
 dynamicStamp
 CircuitElements< T >, 47
 dynamicStampIsFresh
 CircuitElements< T >, 47

e

 BJTN< T >, 24
 BJTP< T >, 30

elements

 SimulationEnvironment< VT >, 111

ElementsRegexBuilder.cpp, 228, 230
 generateRegex, 229

ElementsRegexBuilder.h, 232, 234
 generateRegex, 232

eta

 Diode< T >, 64

exp_alpha

 SParamVFDataFrom< T >, 149

exponent

 AutoDifferentiation, 12

F

 ForceCausal, 14

f0

 ForceCausal, 15

f0derivative

 ForceCausal, 15

fft

 dft.hpp, 217

fft.cpp, 235, 236
 main, 235
 myFunctionToSample, 236

fill

 Matrix< T >, 76
 StaticMatrix< T, M, N, ST >, 160
 StaticRow< T, N, ST >, 168

finalTime

 SimulationEnvironment< VT >, 111

firstOrder

 SParameterBlockVF< T >, 137

ForceCausal, 14

 F, 14
 f0, 15
 f0derivative, 15
 getTau, 16
 K, 16

forceCausal

 ForceCausal.hpp, 239

ForceCausal.hpp, 237, 240
 forceCausal, 239

ForceCausal::CausalData< T >, 37

 data, 38
 tau, 38
 Ts, 38

fracMaxToKeep

 SParameterBlock< T >, 127

frequency

 SinusoidalVoltageSource< T >, 119

from

 SParameterPortVF< T >, 141

func

 AutoDifferentiation, 12

G

 Stamp< T >, 155

g

 NLNMOS< T >, 98

g_cc

 BJTResults< T >, 31

g_ce

 BJTResults< T >, 31

g_ec

 BJTResults< T >, 32

g_ee

 BJTResults< T >, 32

generateCompleteStamp

 CircuitElements< T >, 41

generateDCStamp

 CircuitElements< T >, 41

generateDynamicStamp

 CircuitElements< T >, 42

generateNonLinearStamp

 CircuitElements< T >, 43

generateRegex

 ElementsRegexBuilder.cpp, 229
 ElementsRegexBuilder.h, 232

generateStaticStamp

 CircuitElements< T >, 44

getTau

 ForceCausal, 16

group1

 Resistor< T >, 103

history_p
SParameterBlockVF< T >, 132

I_c
BJTResults< T >, 32

I_cs
BJTN< T >, 24
BJTP< T >, 30

I_e
BJTResults< T >, 32

I_es
BJTN< T >, 24
BJTP< T >, 30

i_gd_last
NLNMOS< T >, 98

i_gs_last
NLNMOS< T >, 98

i_last
NLCapacitor< T >, 86

I_sat
Diode< T >, 64

idft
dft.hpp, 217

ifft
dft.hpp, 218

Inductor
CircuitElements.hpp, 205
Simulator.hpp, 257

Inductor< T >, 65
addDCAnalysisStampTo, 67
addDynamicStampTo, 67
addToElements, 68
dcCurrentIndex, 69
lastCurrent, 69
n1, 70
n2, 70
trapezoidalRule, 70
updateDCStoredState, 68
updateStoredState, 69
value, 70

Inductor.hpp, 241, 242

initialTime
SimulationEnvironment< VT >, 112

it1
AutoDifferentiation, 12

it2
AutoDifferentiation, 13

it3
AutoDifferentiation, 13

K
ForceCausal, 16

I
LUPair< T >, 72
StaticLUPair< T, M >, 157

lambda
SParamVFDataFrom< T >, 149

lambda_p

SParamVFDataFrom< T >, 149

lastCurrent
Capacitor< T >, 36
Inductor< T >, 69

lastTimeSeriesIndex
TimeSeriesVoltageSource< T >, 174

leftDivide
Matrix< T >, 76, 77
StaticMatrix< T, M, N, ST >, 161

length
SParameterSequence< T >, 144
SParamLengthOffset, 147

lerp
TimeSeriesVoltageSource< T >, 172

LineType
Simulator.hpp, 257

LUPair
LUPair< T >, 72

luPair
Matrix< T >, 77
SimulationEnvironment< VT >, 112
StaticMatrix< T, M, N, ST >, 161, 162

LUPair< T >, 71
I, 72
LUPair, 72
M, 72
p, 73
toString, 72
u, 73

M
LUPair< T >, 72
Matrix< T >, 80

main
AutoDiffTest.cpp, 189
CircuitSimulator.cpp, 209
fft.cpp, 235
test.cpp, 286

Matrix
Matrix< T >, 74, 75

Matrix< T >, 73
add, 75
data, 80
fill, 76
leftDivide, 76, 77
luPair, 77
M, 80
Matrix, 74, 75
multiply, 78
N, 80
operator(), 78
rowAddition, 78
subtract, 79
swapRows, 79
toString, 79
transpose, 80

mu
SParamVFDataFrom< T >, 149

mu_p

SParamVFDataFrom< T >, 149
MultipliableResult
 AutoDifferentiation, 13
multiply
 Matrix< T >, 78
 StaticMatrix< T, M, N, ST >, 162, 163
myFunctionToSample
 fft.cpp, 236

N

Matrix< T >, 80
n1
 Capacitor< T >, 36
 CurrentSource< T >, 59
 Diode< T >, 64
 Inductor< T >, 70
 NLCapacitor< T >, 86
 NLCurrentSource< T >, 91
 Resistor< T >, 104
 SinusoidalVoltageSource< T >, 119
 TimeSeriesVoltageSource< T >, 174
 VoltageSource< T >, 179

n2

Capacitor< T >, 37
 CurrentSource< T >, 59
 Diode< T >, 64
 Inductor< T >, 70
 NLCapacitor< T >, 86
 NLCurrentSource< T >, 92
 Resistor< T >, 104
 SinusoidalVoltageSource< T >, 119
 TimeSeriesVoltageSource< T >, 174
 VoltageSource< T >, 180

negative

SParameterPort< T >, 139
 SParameterPortVF< T >, 141

netlistPath

SimulationEnvironment< VT >, 112

NLCapacitor< T >, 81

addDCAnalysisStampTo, 82
 addNonLinearStampTo, 83
 addToElements, 83
 C_last, 85
 C_o, 85
 C_p, 85
 i_last, 86
 n1, 86
 n2, 86
 P_10, 86
 P_11, 86
 u_last, 86
 updateDCStoredState, 84
 updateStoredState, 85

NLCapacitor.hpp, 244, 245

NLCurrentSource< T >, 87

addDCAnalysisStampTo, 88
 addNonLinearStampTo, 90
 addToElements, 91
 n1, 91

n2, 92
 r1_neg, 92
 r1_pos, 92
 r2_neg, 92
 r2_pos, 92
 value, 92

NLCurrentSource.hpp, 247, 248

NLNMOS< T >, 93

addNonLinearStampTo, 95
 addToElements, 95
 alpha_DS, 96
 beta_DS, 96
 C_GD_last, 97
 C_GDo, 97
 C_GDp, 97
 C_GS_last, 97
 C_GSo, 97
 C_GSp, 97
 d, 98
 g, 98
 i_gd_last, 98
 i_gs_last, 98
 P_D10, 98
 P_D11, 98
 P_S10, 99
 P_S11, 99
 s, 99
 u_gd_last, 99
 u_gs_last, 99
 updateStoredState, 96

NLNMOS.hpp, 250, 251

nodeComponentMap

CircuitElements< T >, 47

nodesToGraph

SimulationEnvironment< VT >, 112

nonLinearElements

CircuitElements< T >, 47

NonLinearSolution

CircuitElements.hpp, 205

nonLinearStamp

CircuitElements< T >, 48

nonLinearStampsFresh

CircuitElements< T >, 48

nthRootOfUnity

dft.hpp, 218

nu

SParamVFDataFrom< T >, 150

nu_p

SParamVFDataFrom< T >, 150

numCurrents

SimulationEnvironment< VT >, 112

numDCCurrents

SimulationEnvironment< VT >, 113

numNodes

SimulationEnvironment< VT >, 113

numPoles

SParamVFDataFrom< T >, 150

numPorts

SParameterBlockVF< T >, 137
SParameterSequence< T >, 147

offset
 SinusoidalVoltageSource< T >, 119
 SParameterSequence< T >, 144, 145
 SParamLengthOffset, 148

operator<
 DynamicMatrix.hpp, 223
 StaticMatrix.hpp, 280

operator<=
 DynamicMatrix.hpp, 224
 StaticMatrix.hpp, 280

operator>
 DynamicMatrix.hpp, 224
 StaticMatrix.hpp, 280

operator>=
 DynamicMatrix.hpp, 224
 StaticMatrix.hpp, 280

operator()
 Matrix< T >, 78

operator[]
 StaticMatrix< T, M, N, ST >, 163
 StaticRow< T, N, ST >, 168

outputFilePath
 SimulationEnvironment< VT >, 113

p
 LUPair< T >, 73
 StaticLUPair< T, M >, 157

P_10
 NLCapacitor< T >, 86

P_11
 NLCapacitor< T >, 86

P_D10
 NLNMOS< T >, 98

P_D11
 NLNMOS< T >, 98

P_S10
 NLNMOS< T >, 99

P_S11
 NLNMOS< T >, 99

parseGraph
 SimulationEnvironment< VT >, 107

performDCAnalysis
 SimulationEnvironment< VT >, 113

phase
 SinusoidalVoltageSource< T >, 119

pole
 SParamVFDataFrom< T >, 150

port
 SParameterBlock< T >, 127
 SParameterBlockVF< T >, 138

positive
 SParameterPort< T >, 139
 SParameterPortVF< T >, 142

printGraph
 SimulationEnvironment< VT >, 108

printMultipleOnGraph

R
 SParameterPort< T >, 139
 SParameterPortVF< T >, 142

r1_neg
 NLCurrentSource< T >, 92

r1_pos
 NLCurrentSource< T >, 92

r2_neg
 NLCurrentSource< T >, 92

r2_pos
 NLCurrentSource< T >, 92

R_p
 SParameterBlock< T >, 125

readInPRR
 SParameterBlockVF< T >, 133

readInTimeSeries
 TimeSeriesVoltageSource< T >, 173

readInTouchstoneFile
 SParameterBlock< T >, 125

remainder
 SParamVFDataFrom< T >, 150

requires
 AutoDifferentiation, 10

residue
 SParamVFDataFrom< T >, 151

Resistor
 CircuitElements.hpp, 205
 Simulator.hpp, 257

Resistor< T >, 100
 addDCAnalysisStampTo, 101
 addStaticStampTo, 102
 addToElements, 102
 currentIndex, 103
 group1, 103
 n1, 104
 n2, 104
 value, 104

Resistor.hpp, 254, 255

rowAddition
 Matrix< T >, 78
 StaticMatrix< T, M, N, ST >, 163

rows
 StaticMatrix< T, M, N, ST >, 166

s
 NLNMOS< T >, 99
 SParameterBlock< T >, 127
 Stamp< T >, 155

s0
 SParameterPort< T >, 140

scratchSpace
 SimulationEnvironment< VT >, 113

setConstants
 SParameterBlockVF< T >, 133

setDCOpPoint
 SimulationEnvironment< VT >, 110

setFirstOrder

SParameterBlockVF< T >, 134
 setNewStampSize
 CircuitElements< T >, 44
 setSecondOrder
 SParameterBlockVF< T >, 134
 setTimestep
 Component< T >, 54
 SParameterBlockVF< T >, 135
 simulate
 SimulationEnvironment< VT >, 110
 SimulationEnvironment
 SimulationEnvironment< VT >, 106
 SimulationEnvironment< VT >, 105
 dataDump, 107
 elements, 111
 finalTime, 111
 initialTime, 112
 luPair, 112
 netlistPath, 112
 nodesToGraph, 112
 numCurrents, 112
 numDCCurrents, 113
 numNodes, 113
 outputFilePath, 113
 parseGraph, 107
 performDCAnalysis, 113
 printGraph, 108
 printMultipleOnGraph, 108
 scratchSpace, 113
 setDCOpPoint, 110
 simulate, 110
 SimulationEnvironment, 106
 solutionMat, 114
 steps, 114
 timestep, 114
 Simulator.hpp, 256, 258
 Capacitor, 257
 Comment, 257
 CurrentSource, 257
 Diode, 257
 Directive, 257
 Inductor, 257
 LineType, 257
 Resistor, 257
 Sinusoidal, 257
 SourceType, 257
 SParameterBlock, 257
 TimeSeries, 257
 Transistor, 257
 VoltageSource, 257
 Sinusoidal
 Simulator.hpp, 257
 SinusoidalVoltageSource< T >, 115
 addDCAnalysisStampTo, 116
 addDynamicStampTo, 117
 addToElements, 117
 currentIndex, 118
 degrees, 119
 frequency, 119
 n1, 119
 n2, 119
 offset, 119
 phase, 119
 updateStoredState, 118
 V, 120
 SinusoidalVoltageSource.hpp, 263, 264
 sizeG_A
 Stamp< T >, 155
 sizeG_D
 Stamp< T >, 155
 sizeM
 StaticMatrix< T, M, N, ST >, 166
 sizeN
 StaticMatrix< T, M, N, ST >, 166
 StaticRow< T, N, ST >, 169
 solutionMat
 SimulationEnvironment< VT >, 114
 SolutionStage
 CircuitElements.hpp, 205
 solve
 Stamp< T >, 154
 SourceType
 Simulator.hpp, 257
 SParameterBlock
 Simulator.hpp, 257
 SParameterBlock< T >, 120
 addDCAnalysisStampTo, 122
 addDynamicStampTo, 122
 addStaticStampTo, 123
 addToElements, 123
 aWaveConvValue, 124
 beta_p, 124
 fracMaxToKeep, 127
 port, 127
 R_p, 125
 readInTouchstoneFile, 125
 s, 127
 touchstoneFilePath, 127
 updateStoredState, 125
 V_p, 126
 z_ref, 128
 SParameterBlock.hpp, 265, 267
 SParameterBlockVF< T >, 128
 addDCAnalysisStampTo, 130
 addDynamicStampTo, 130
 addStaticStampTo, 131
 addToElements, 131
 awave_p, 132
 bwave_p, 132
 firstOrder, 137
 history_p, 132
 numPorts, 137
 port, 138
 readInPRR, 133
 setConstants, 133
 setFirstOrder, 134

setSecondOrder, 134
setTimestep, 135
updateStoredState, 136
 V_p , 137
 z_{ref} , 138
SParameterBlockVF.hpp, 271, 272
SParameterPort< T >, 138
 beta, 139
 current, 139
 negative, 139
 positive, 139
 R, 139
 s0, 140
SParameterPortVF< T >, 140
 alpha, 141
 beta, 141
 current, 141
 from, 141
 negative, 141
 positive, 142
 R, 142
SParameterSequence< T >, 142
 _data, 146
 _time, 146
 data, 143
 length, 144
 numPorts, 147
 offset, 144, 145
 sParamLengthOffset, 147
 time, 145, 146
SParamLengthOffset, 147
 length, 147
 offset, 148
sParamLengthOffset
 SParameterSequence< T >, 147
SParamVFDataFrom< T >, 148
 exp_alpha, 149
 lambda, 149
 lambda_p, 149
 mu, 149
 mu_p, 149
 nu, 150
 nu_p, 150
 numPoles, 150
 pole, 150
 remainder, 150
 residue, 151
 x, 151
Stamp
 Stamp< T >, 152
Stamp< T >, 151
 add, 153
 addDCAnalysisStamp, 153
 addDynamicStamp, 153
 addNonLinearStamp, 154
 addStaticStamp, 154
 clear, 154
 G, 155
 s, 155
 sizeG_A, 155
 sizeG_D, 155
 solve, 154
 Stamp, 152
staticElements
 CircuitElements< T >, 48
StaticLUPair< T, M >, 156
 l, 157
 p, 157
 toString, 157
 u, 157
StaticMatrix
 StaticMatrix< T, M, N, ST >, 159
StaticMatrix< T, M, N, ST >, 158
 add, 160
 fill, 160
 leftDivide, 161
 luPair, 161, 162
 multiply, 162, 163
 operator[], 163
 rowAddition, 163
 rows, 166
 sizeM, 166
 sizeN, 166
 StaticMatrix, 159
 subtract, 164
 swapRows, 164
 toString, 165
 transpose, 165
StaticMatrix.hpp, 278, 281
 arithmetic, 281
 operator<, 280
 operator<=, 280
 operator>, 280
 operator>=, 280
 storageType, 279
StaticRow
 StaticRow< T, N, ST >, 167
StaticRow< T, N, ST >, 166
 columns, 168
 dot, 168
 fill, 168
 operator[], 168
 sizeN, 169
 StaticRow, 167
StaticSolution
 CircuitElements.hpp, 205
staticStamp
 CircuitElements< T >, 48
staticStampIsFresh
 CircuitElements< T >, 48
steps
 SimulationEnvironment< VT >, 114
storageType
 StaticMatrix.hpp, 279
subtract
 Matrix< T >, 79

StaticMatrix< T, M, N, ST >, 164
 SubtractableResult
 AutoDifferentiation, 13
 swapRows
 Matrix< T >, 79
 StaticMatrix< T, M, N, ST >, 164
 tau
 ForceCausal::CausalData< T >, 38
 test.cpp, 285, 287
 main, 286
 testBasicOutput_NoCheck
 AutoDiffTest.cpp, 189
 testBJTModel
 AutoDiffTest.cpp, 189
 testBJTModelAutoDiff
 AutoDiffTest.cpp, 190
 testBJTModelControl
 AutoDiffTest.cpp, 190
 time
 SParameterSequence< T >, 145, 146
 TimeSeries
 Simulator.hpp, 257
 timeSeries
 TimeSeriesVoltageSource< T >, 175
 TimeSeriesVoltageSource< T >, 169
 addDCAnalysisStampTo, 170
 addDynamicStampTo, 171
 addToElements, 172
 currentIndex, 174
 dataSeries, 174
 lastTimeSeriesIndex, 174
 lerp, 172
 n1, 174
 n2, 174
 readInTimeSeries, 173
 timeSeries, 175
 updateStoredState, 173
 TimeSeriesVoltageSource.hpp, 288, 289
 timestep
 SimulationEnvironment< VT >, 114
 toRet
 AutoDifferentiation, 10, 11, 13
 toString
 LUPair< T >, 72
 Matrix< T >, 79
 StaticLUPair< T, M >, 157
 StaticMatrix< T, M, N, ST >, 165
 touchstoneFilePath
 SParameterBlock< T >, 127
 Transistor
 Simulator.hpp, 257
 transistorTest
 AutoDiffTest.cpp, 191
 transistorTestAutoDiff
 AutoDiffTest.cpp, 191
 transistorTestControl
 AutoDiffTest.cpp, 192
 TransistorTestResult< T >, 175
 diff1, 175
 diff2, 175
 var, 176
 transpose
 Matrix< T >, 80
 StaticMatrix< T, M, N, ST >, 165
 trapezoidalRule
 Capacitor< T >, 37
 Inductor< T >, 70
 Ts
 ForceCausal::CausalData< T >, 38
 u
 LUPair< T >, 73
 StaticLUPair< T, M >, 157
 u_gd_last
 NLNMOS< T >, 99
 u_gs_last
 NLNMOS< T >, 99
 u_last
 NLCapacitor< T >, 86
 updateDCStoredState
 CircuitElements< T >, 45
 Component< T >, 54
 Inductor< T >, 68
 NLCapacitor< T >, 84
 updateStoredState
 BJTN< T >, 22
 BJTP< T >, 28
 Capacitor< T >, 36
 Component< T >, 55
 Diode< T >, 63
 Inductor< T >, 69
 NLCapacitor< T >, 85
 NLNMOS< T >, 96
 SinusoidalVoltageSource< T >, 118
 SParameterBlock< T >, 125
 SParameterBlockVF< T >, 136
 TimeSeriesVoltageSource< T >, 173
 updateTimeStep
 CircuitElements< T >, 46
 V
 SinusoidalVoltageSource< T >, 120
 V_bc_crit
 BJTN< T >, 24
 BJTP< T >, 30
 V_be_crit
 BJTN< T >, 24
 BJTP< T >, 30
 V_crit
 Diode< T >, 65
 V_p
 SParameterBlock< T >, 126
 SParameterBlockVF< T >, 137
 V_T
 Diode< T >, 65
 V_Tc
 BJTN< T >, 24

BJTP< T >, 30
V_Te
 BJTN< T >, 25
 BJTP< T >, 31
value
 Capacitor< T >, 37
 CurrentSource< T >, 59
 Inductor< T >, 70
 NLCurrentSource< T >, 92
 Resistor< T >, 104
 VoltageSource< T >, 180
var
 AutoDifferentiation, 14
 TransistorTestResult< T >, 176
VoltageSource
 CircuitElements.hpp, 205
 Simulator.hpp, 257
VoltageSource< T >, 176
 addDCAAnalysisStampTo, 177
 addStaticStampTo, 178
 addToElements, 178
 currentIndex, 179
 n1, 179
 n2, 180
 value, 180
VoltageSource.hpp, 291, 292

while
 AutoDifferentiation, 11

x
 SParamVFDataFrom< T >, 151

z_ref
 SParameterBlock< T >, 128
 SParameterBlockVF< T >, 138