

Contents

Abstract	xiii
Table of Contents	xv
List of Symbols	xix
List of Figures	xxi
List of Tables	xxv
1 Introduction	1
1.1 Embedded system design flow	1
1.2 Behavioural transformations	4
1.2.1 Code motion transformations	4
1.2.2 Loop transformations	5
1.2.3 Arithmetic transformations	6
1.2.4 High-level to RTL transformations	6
1.2.5 Sequential to parallel transformations	7
1.3 Motivations and objectives	8
1.3.1 Problem statements	10
1.4 Contributions	13
1.5 Organization of the thesis	15
2 Literature Survey	17
2.1 Code motion transformations	17
2.1.1 Applications of code motion transformations	17
2.1.2 Verification of code motion transformations	20
2.2 High-level to RTL and RTL transformations	22
2.2.1 Applications of High-level to RTL and RTL transformations . .	22
2.2.2 Verification of High-level to RTL and RTL transformations . .	23

2.3	Loop transformations and arithmetic transformations	25
2.3.1	Applications of loop transformations	25
2.3.2	Applications of arithmetic transformations	27
2.3.3	Verification of loop and arithmetic transformations	29
2.4	Parallelizing transformations	31
2.4.1	Applications of parallelizing transformations	31
2.4.2	Verification of parallelizing transformations	34
2.5	Conclusion	35
3	Verification of Code Motion Transformations	37
3.1	Introduction	37
3.2	Basic equivalence checking method	38
3.2.1	FSMDs and its paths	38
3.2.2	Normalization of arithmetic expressions	42
3.3	Equivalence problem formulation	44
3.3.1	Path cover and equivalence of FSMDs	45
3.3.2	A method to handle uniform code motions	46
3.4	Verification of non-uniform code motions	48
3.4.1	An example of non-uniform code motion	48
3.4.2	A scheme for verifying non-uniform code motions	49
3.4.3	Strong and weak equivalence of paths	53
3.4.4	Formulation of the path extension procedure	56
3.4.5	Encoding and model checking the data-flow properties	57
3.4.6	The equivalence checking method	58
3.4.7	Illustration of working of the equivalence checking method	64
3.4.8	Justification of the initial cutpoints	66
3.5	Multicycle and pipelined execution of operations	68
3.6	Correctness and complexity	70
3.6.1	Correctness	70
3.6.2	Complexity	74
3.7	Experimental results	75
3.7.1	Limitations of the method	81
3.8	Conclusion	82
4	Verification of RTL Generation Phase	83
4.1	Introduction	83
4.2	Verification challenges	86
4.3	Construction of FSMDs from RTL designs	87
4.3.1	Representation of the datapath description	87
4.3.2	A Method of obtaining the micro-operations for a control as- sertion pattern	89
4.3.3	Identification of RT operations realized by a set of micro- operations	91
4.3.4	Multicycle, pipelined and chained operations	93
4.4	The Overall construction framework of FSMD	95
4.4.1	Handling of multicycle operations	96

4.4.2	Handling of pipelined operations	99
4.4.3	Handling chained operations	101
4.4.4	Verification during construction of FSMMD	102
4.5	Correctness and complexity of the algorithm	103
4.5.1	Correctness and complexity of the module <i>findRewriteSeq</i>	103
4.5.2	Correctness and complexity of the module <i>RTL-1</i>	111
4.5.3	Correctness and complexity of the modules <i>Multicycle</i> and <i>Pipelined</i>	112
4.5.4	Correctness and complexity of the module <i>RTL-0</i>	113
4.6	Verification by equivalence checking	116
4.7	Verification of low power RTL transformations	117
4.7.1	Alternative datapath architecture	118
4.7.2	Restructuring of multiplexer networks to enhance data correlation	121
4.7.3	Restructuring of multiplexer networks to eliminate glitchy control signals	122
4.7.4	Clocking of control signals	123
4.7.5	Glitch reduction using delays	124
4.8	Experimental results	126
4.9	Conclusion	132
5	Verification of Loop Transformations	133
5.1	Introduction	133
5.2	Array data dependence graphs	135
5.2.1	Representation of data dependencies of the behaviour	139
5.2.2	Transitive dependence	142
5.2.3	Recurrence in ADDG	146
5.2.4	Construction of the ADDG from a sequential behaviour	148
5.3	Slices	148
5.4	Equivalence of ADDGs	154
5.4.1	Normalization of the characteristic formula of a slice	156
5.4.2	Some simplification rules for data transformations	158
5.4.3	Equivalence problem formulation	161
5.5	A case study	165
5.6	Correctness and complexity	167
5.6.1	Complexity	168
5.7	Error diagnosis	171
5.8	Experimental results	172
5.9	Conclusion	174
6	Verification of Parallelizing Transformations	175
6.1	Introduction	175
6.2	Verification framework	176
6.2.1	Kahn process networks	176
6.2.2	Verification approach	176
6.3	Modelling a KPN as an ADDG	179

6.3.1	Modelling KPN processes as ADDGs	179
6.3.2	Computation of the dependence mappings involving FIFOs	182
6.3.3	Composition of ADDGs of KPN processes	192
6.3.4	Correctness of the composition operation	195
6.4	Deadlock detection in a KPN	199
6.4.1	Deadlock due to insufficient communication	200
6.4.2	Deadlock due to circular dependence in a KPN	201
6.5	Verification of KPN level transformations	205
6.5.1	Channel merging	205
6.5.2	Channel splitting	213
6.5.3	Process splitting	217
6.5.4	Process merging	223
6.5.5	Message vectorization	226
6.5.6	Computation migration	230
6.6	Experimental results	233
6.7	Conclusion	234
7	Conclusion and Future Scopes	237
7.1	Summary of contributions	238
7.2	Scope for future work	242
7.2.1	Enhancement of the present work	242
7.2.2	Scope of application to other research areas	242
7.3	Conclusion	244
	Bibliography	249