ditioned on value of *c* is also shown..... 157
Figure 6.8. Conditional constructs in parallel paths 158
Figure 6.9. A bus access mechanism that selectively "masks" bus grants based on values of control tokens that are evaluated at run time 159
Figure 6.10. Bus access lists and the annotated list corresponding to Fig. 6.6.. 161
Figure 6.11. Quasi-static schedule for the data-dependent iteration graph of Fig. 6.2(b). 164
Figure 6.12. A possible access order list corresponding to the quasi-static schedule of Fig. 6.11. 165
Figure 7.1. An example of how execution time guarantees can be used to reduce buffer size bounds. 168

strongly connected into a strongly connected graph. 133
Figure 5.9. An illustration of a possible solution obtained by algorithm Convert-to-SC-graph. 134
Figure 5.10. The synchronization graph, after redundant synchronization edges are removed, induced by a four-processor schedule of a music synthesizer based on the Karplus-Strong algorithm. 136
Figure 5.11. A possible solution obtained by applying Convert-to-SC-graph to the example of Figure 5.10. 137
Figure 5.13. An example used to illustrate a solution obtained by algorithm DetermineDelays. 138
Figure 5.12. An algorithm for determining the delays on the edges introduced by algorithm Convert-to-SC-graph. 139
Figure 5.14. An example of resynchronization. 143
Figure 5.15. The complete synchronization optimization algorithm. 145
Figure 6.1. BDF actors SWITCH and SELECT 148
Figure 6.2. (a) Conditional (if-then-else) dataflow graph. The branch outcome is determined at run time by actor B. (b) Graph representing data-dependent iteration. The termination condition for the loop is determined by actor D. 149
Figure 6.3. Acyclic precedence graphs corresponding to the if-then-else graph of Fig. 6.2. (a) corresponds to the TRUE assignment of the control token, (b) to the FALSE assignment. 150
Figure 6.4. Quasi-static schedule for a conditional construct (adapted from [Lee88b]) 152
Figure 6.5. Programs on three processors for the quasi-static schedule of Fig. 6.4 153
Figure 6.6. Transaction order corresponding to the TRUE and FALSE branches 155
Figure 6.7. Bus access list that is stored in the schedule RAM for the quasi-static schedule of Fig. 6.6. Loading operation of the schedule counter con-

Figure 3.19. Schedule for the FFT example. 78
Figure 4.1. Fully-static schedule on five processors 80
Figure 4.2. Self-timed schedule 81
Figure 4.3. Schedule evolution when the transaction order of Fig. 3.1 is enforced 81
Figure 4.4. The IPC graph for the schedule in Fig. 4.1. 83
Figure 4.5. Transaction ordering constraints 89
Figure 4.6. Modified schedule S' 95
Figure 4.7. G_{ipc}, actor C has execution time t_c, constant over all invocations of C *97*
Figure 4.8. $T_{ST}(t_C)$ *98*
Figure 4.9. G_{ipc} with transaction ordering constraints represented as dashed lines 105
Figure 4.10. $T_{ST}(t_C)$ and $T_{OT}(t_C)$ *105*
Figure 5.1. (a) An HSDFG (b) A three-pro(a) An HSDFG (b) A three-processor self-timed schedule for (a). (c) An illustration of execution under the placement of barriers. 110
Figure 5.2. Self-timed execution 113
Figure 5.3. An IPC graph with a feedforward edge: (a) original graph (b) imposing bounded buffers. 115
Figure 5.4. x_2 is an example of a redundant synchronization edge. 124
Figure 5.5. An algorithm that optimally removes redundant synchronization edges. 127
Figure 5.6. (a) A multi-resolution QMF filter bank used to illustrate the benefits of removing redundant synchronizations. (b) The precedence graph for (a). (c) A self-timed, two-processor, parallel schedule for (a). (d) The initial synchronization graph for (c) 130
Figure 5.7. The synchronization graph of Fig. 5.6(d) after all redundant synchronization edges are removed. 132
Figure 5.8. An algorithm for converting a synchronization graph that is not

List of Figures

Figure 1.1. Fully static schedule 16
Figure 1.2. Fully-static schedule on five processors 17
Figure 1.3. Steps in a self-timed scheduling strategy 20
Figure 3.1. One possible transaction order derived from the fully-static schedule 41
Figure 3.2. Block diagram of the OMA prototype 49
Figure 3.3. Modified design 50
Figure 3.4. Details of the "TA" line mechanism (only one processor is shown) 51
Figure 3.5. Top-level schematic of the OMA prototype 54
Figure 3.6. Using processor bus arbitration signals for controlling bus access. 56
Figure 3.7. Ordered Transaction Controller implementation 58
Figure 3.8. Presettable counter implementation 59
Figure 3.9. Host interface 61
Figure 3.10. Processing element 62
Figure 3.11. Xilinx configuration at run time 64
Figure 3.12. Connecting multiple boards 67
Figure 3.13. Schematics hierarchy of the four processor OMA architecture 68
Figure 3.14. OMA prototype board photograph 69
Figure 3.15. Steps required for downloading code (*tcl* script omaDoAll)..... 70
Figure 3.16. Hierarchical specification of the Karplus-Strong algorithm in 28 voices. 74
Figure 3.17. Four processor schedule for the Karplus-Strong algorithm in 28 voices. Three processors are assigned 8 voices each, the fourth (Proc 1) is assigned 4 voices along with the noise source. 75
Figure 3.18. (a) Hierarchical block diagram for a 15 band analysis and synthesis filter bank. (b) Schedule on four processors (using Sih's DL heuristic [Sih90])..... 77