

ACK - A Framework for High Level Synthesis of Asynchronous Circuits

Prabhakar Kudva, Hans Jacobson
Ganesh Gopalakrishnan, Al Davis, Erik Brunvand
Robert Thacker, Johan Nyblom

University of Utah
Computer Science Department

Asynchronous Synthesis

- Macromodular Approach:

Martin, Brunvand, Akella, van Berkel

- Syntax directed translation to modules
- Restricted low level module libraries
- Does not exploit progress in logic synthesis
- Differ from synchronous paradigm

- Direct Synthesis Approach:

Davis et al., Nowick, Yun, Siegel, Stevens

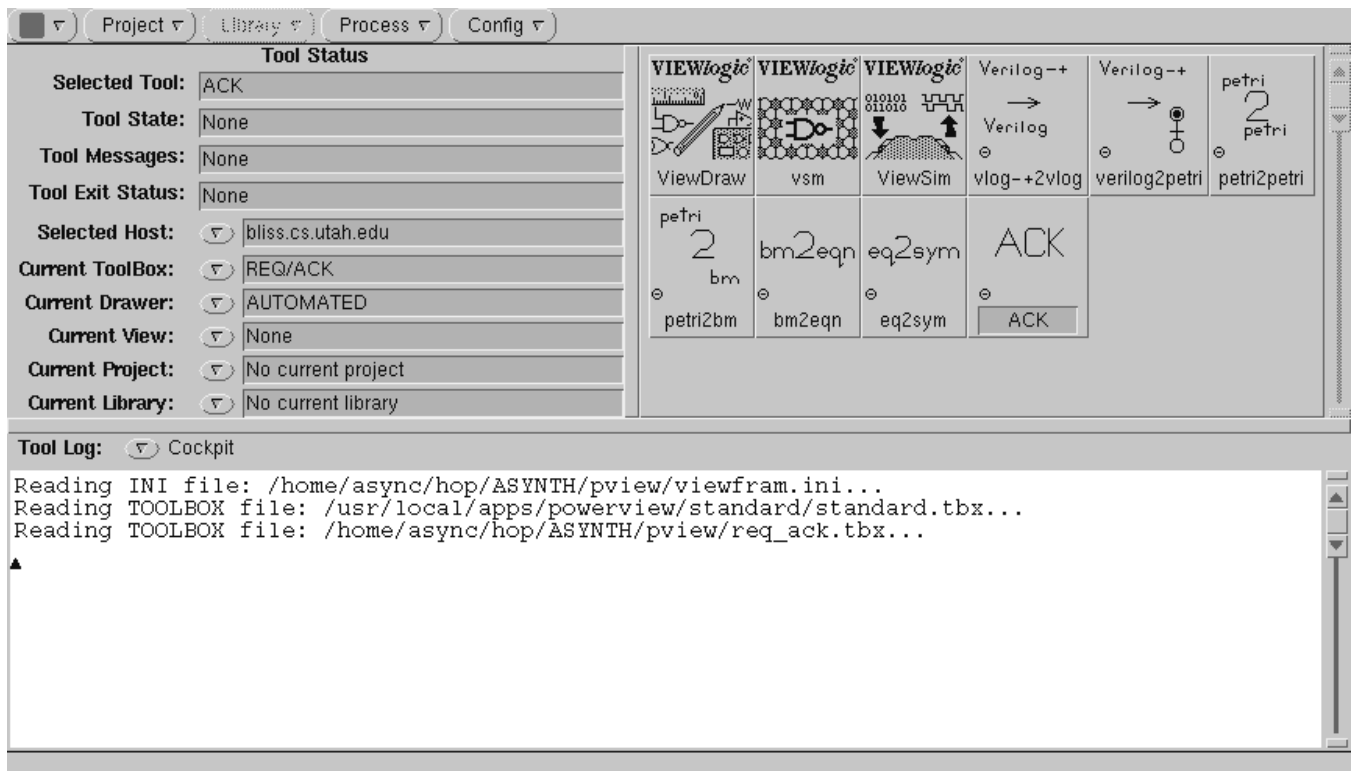
- Synthesis of customized AFSMs
- Similar to synchronous methods

Motivation for ACK

- Synthesis flow similar to synchronous allow use of standard commercial and public domain tools
- a key to acceptability of asynchronous design
- Ability to specify overall system structure rather than individual controller structure
- Take advantage of high level as well as low level synthesis techniques
- Protocol independent description allows targeting of both two and four phase implementations

The ACK Interface

- Requirements:
 - Easy to use
 - All tools in one framework
- Our solution:
 - Graphical user interface
 - Embedded in synchronous framework



Standard HDL Interface

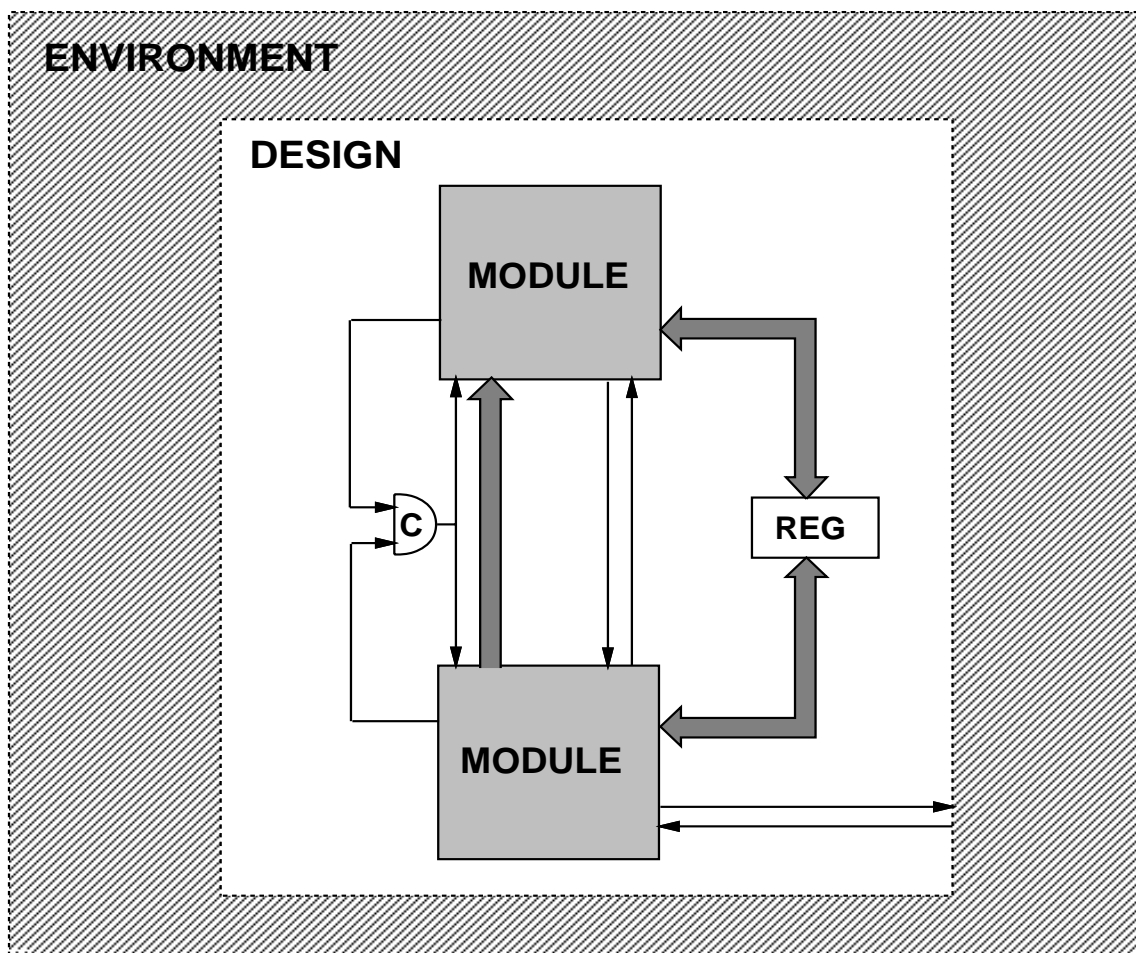
- Requirements for asynchronous language:
 - Channels
 - Process description
 - Direct concurrency support

- Requirement from designers:
 - Standardized language

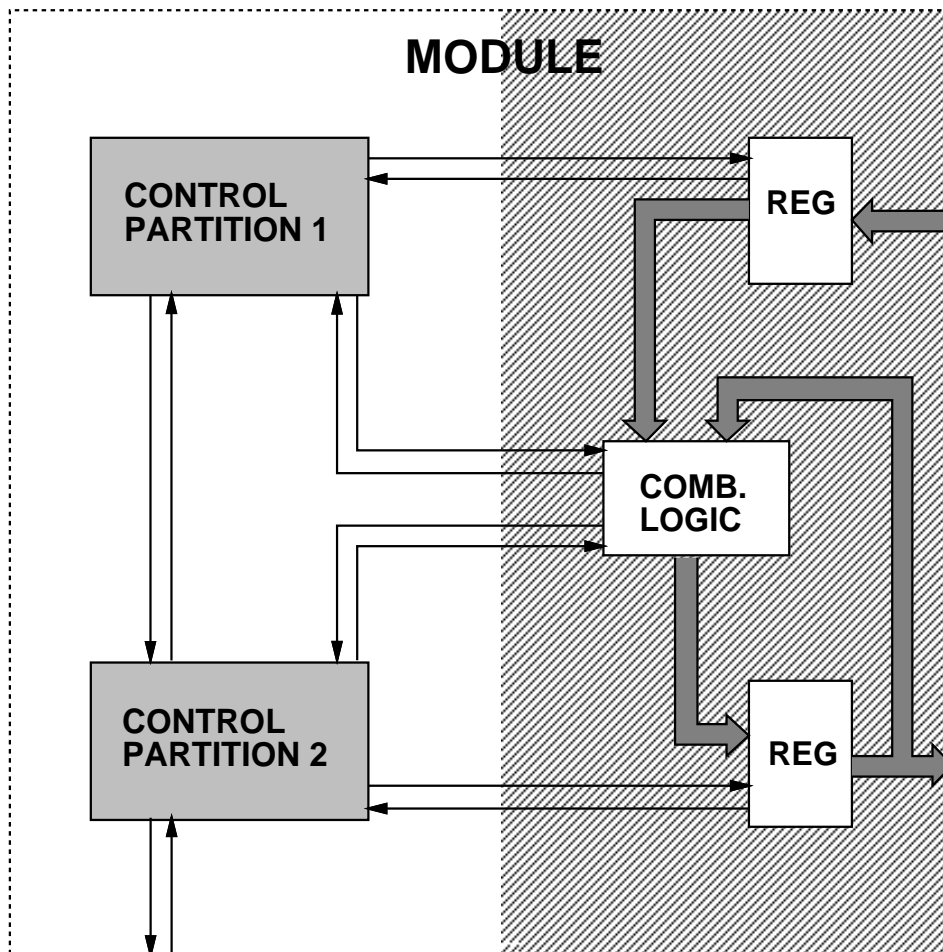
- Our solution:
 - Verilog-+
 - * Synthesizable subset of Verilog
 - * Extended with channels
 - Allows behavioral simulation

The Design Structure

- Structural level design:
 - Modules communicating via
 - * Channels
 - * Handshakes
 - * Shared variables

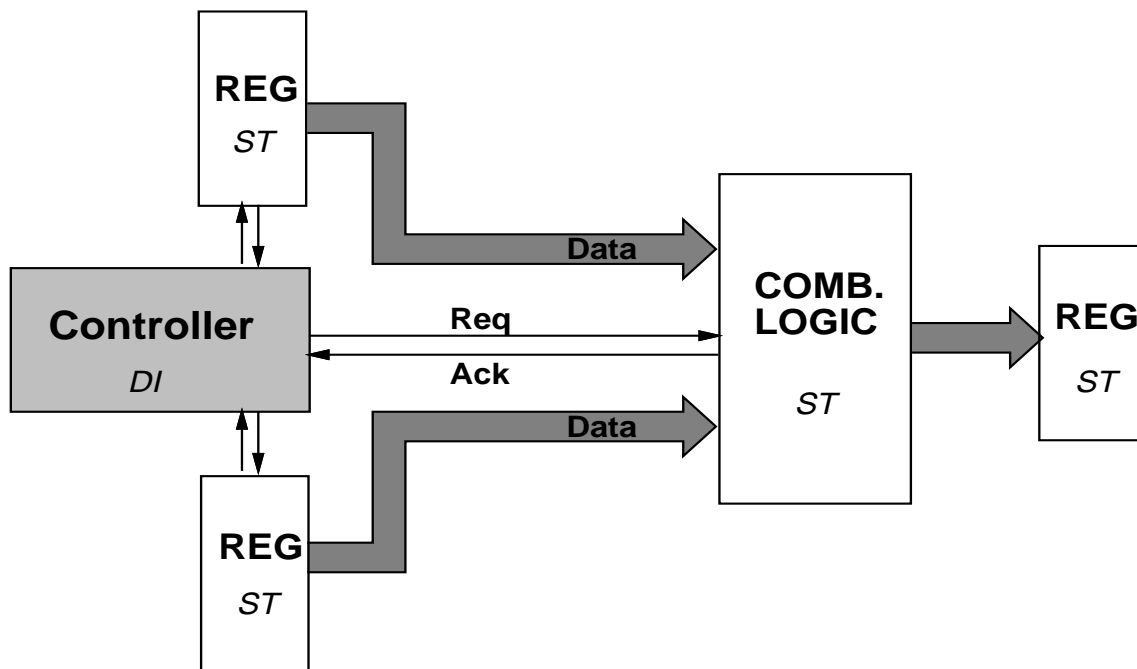


- Module:
 - Communication interface
 - Local resources
 - Partitions



Delay Models and Environment Assumptions

- Control:
 - Huffman style
 - Delay insensitive
- Datapath:
 - Self-timed blocks
 - Bundled data between blocks

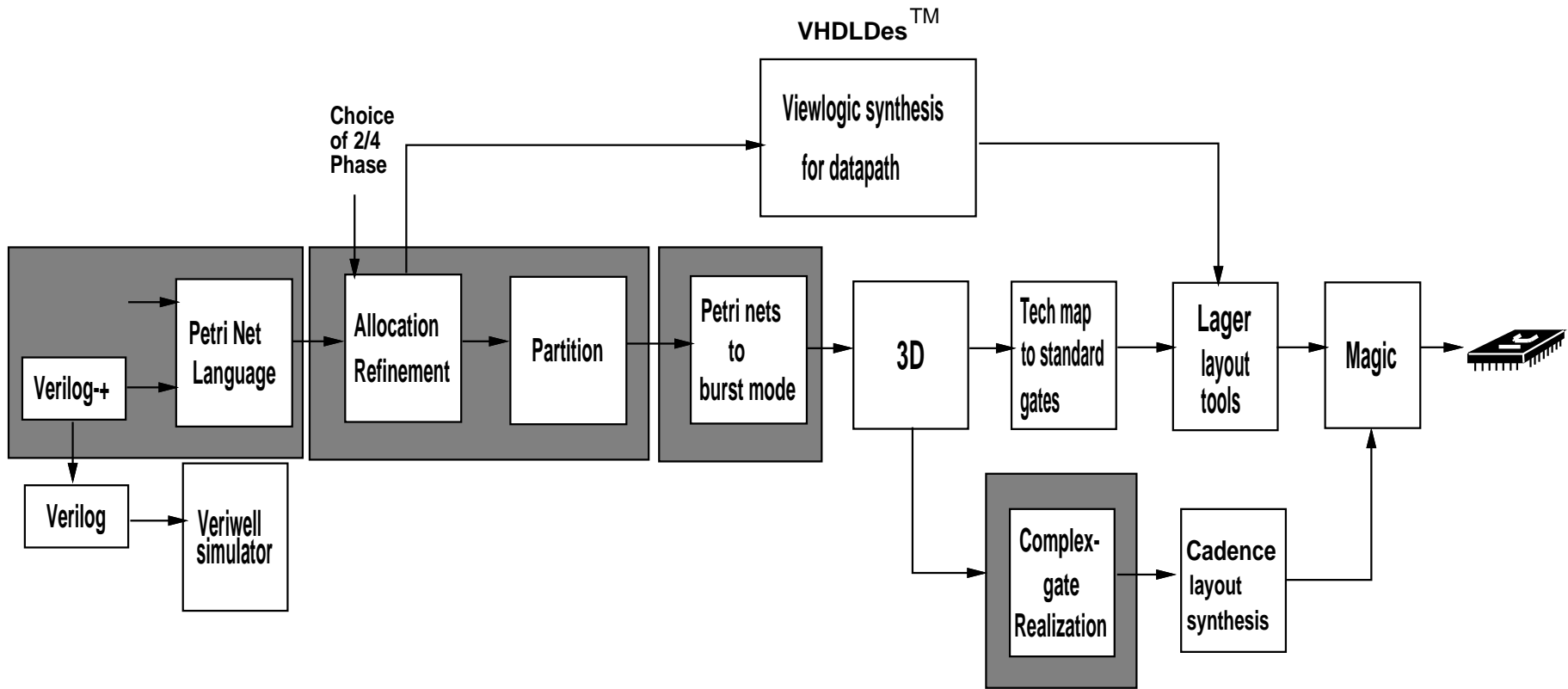


Communication Protocols

- Two phase:
 - Advantages:
 - * Well suited for protocol based designs
 - * All transitions do useful work
 - Disadvantages:
 - * Slow or large datapath

- Four phase:
 - Advantages:
 - * Well suited for datapath intensive designs
 - * Fast and small datapath
 - Disadvantages:
 - * Not all transitions do useful work

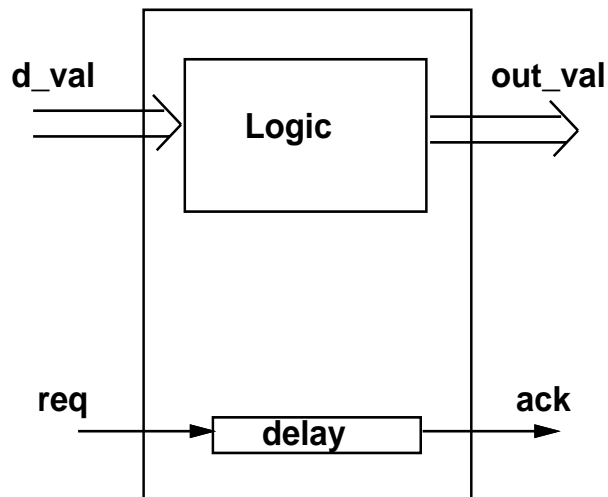
ACK - System Flow



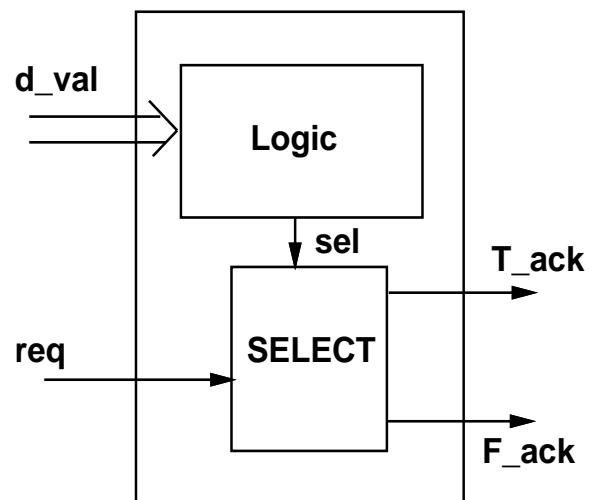
Allocation and Refinement

- **Allocation**

- For the structural description:
 - * Allocate and connect physical resources
- For each module:
 - * Allocate physical resources
 - local variable - register
 - computation - data block
 - data dependent choice - predicate block



Model for Data Block (DB)



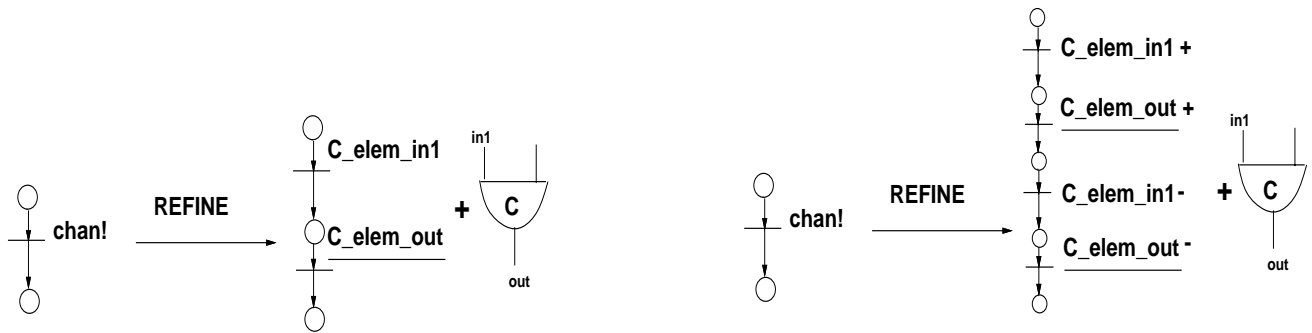
Model for Predicate Block (PB)

• Refinement

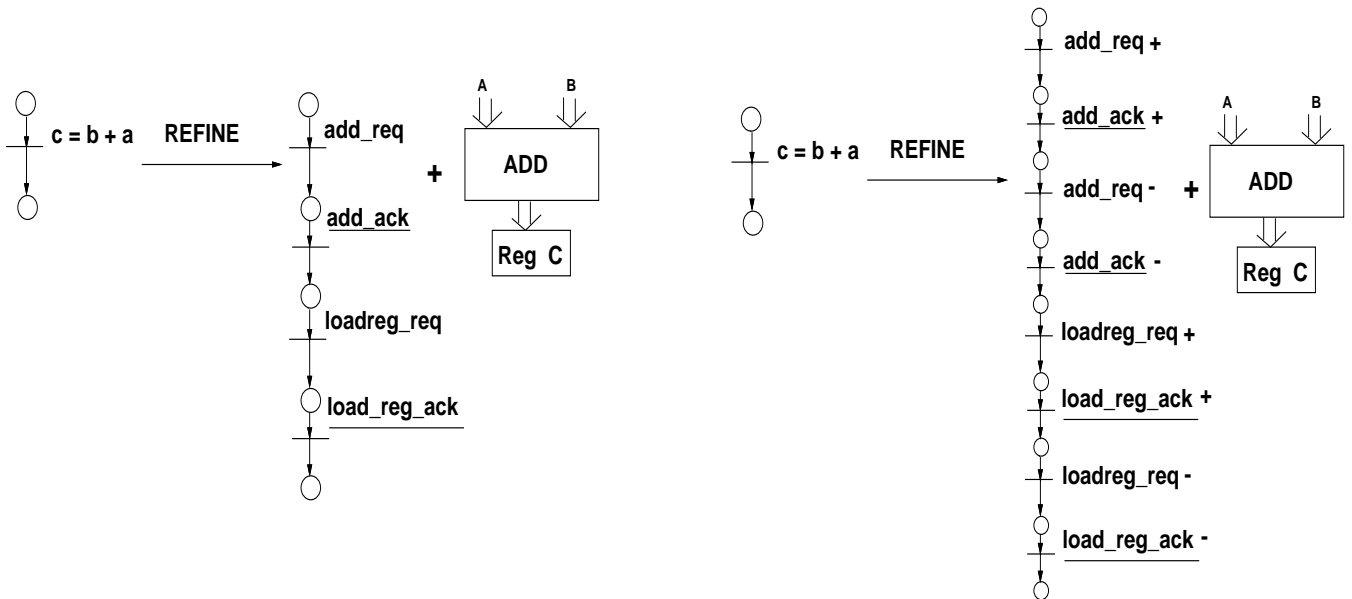
– For each module:

- * Generate control graph acting on allocated resources

CHANNEL COMMUNICATION



COMPUTATION



Two phase

Four phase

Partitioning

- Why Partition?
 - Behavioral description often large and centralized
 - Asynchronous synthesis of large designs is slow
 - Exploit spatial and temporal locality
 - Reduce controller area and propagation delay

- Why assisted partitioning?
 - Partitioning by hand is error prone
 - Complex signal sharing arrangements sometimes necessary

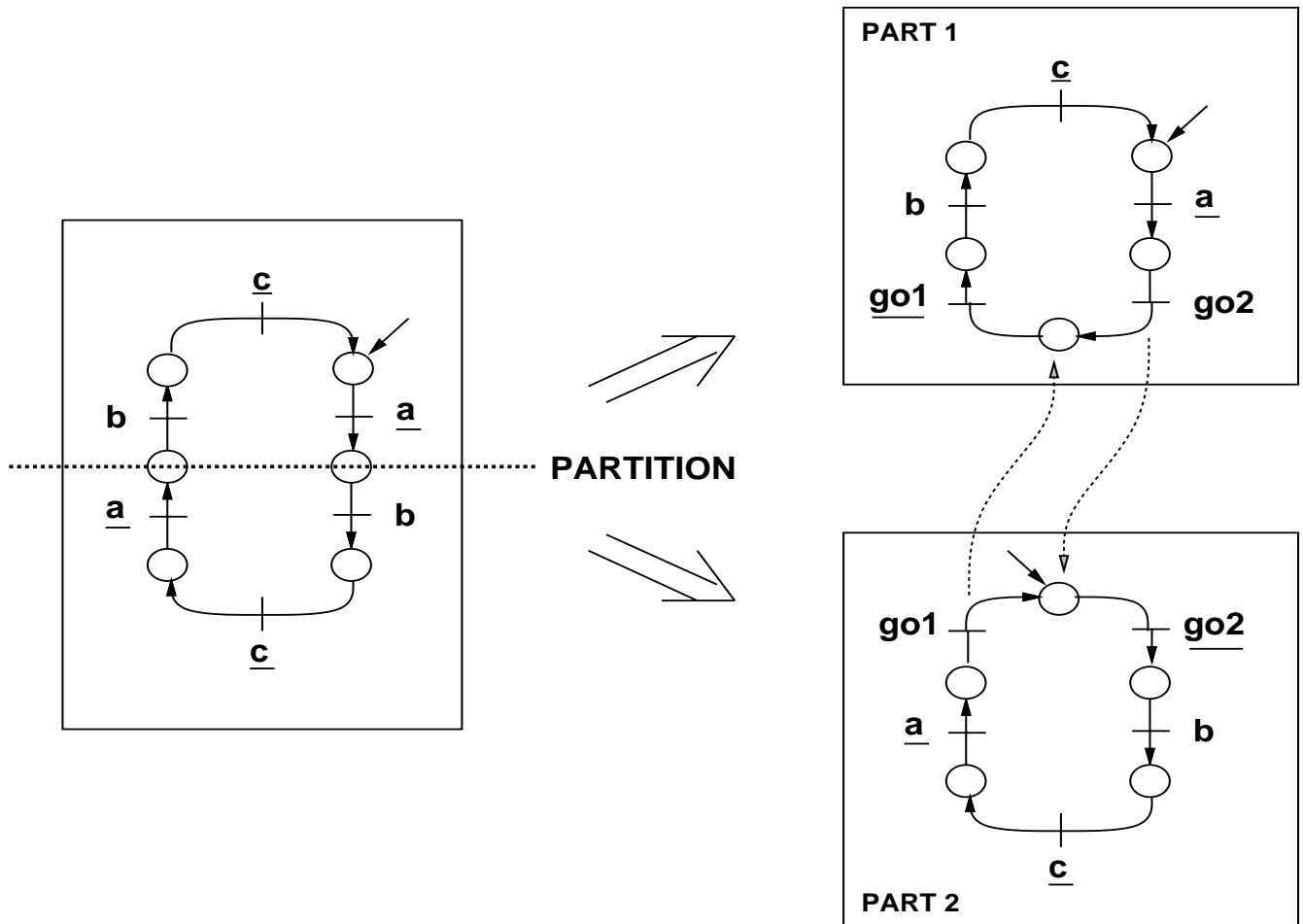
- Problem:
 - Partitioning of incompletely specified machines

- Requirement:
 - Correctly handle control flow and signal sharings between partitions
 - Composite behavior of distributed controllers same as for centralized

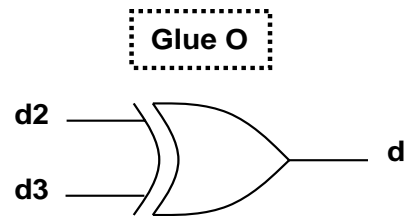
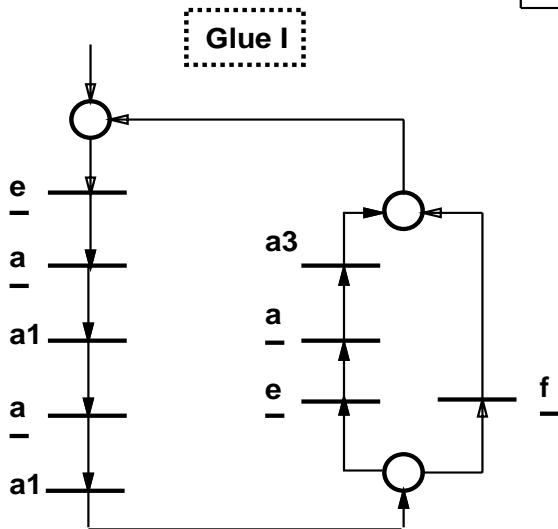
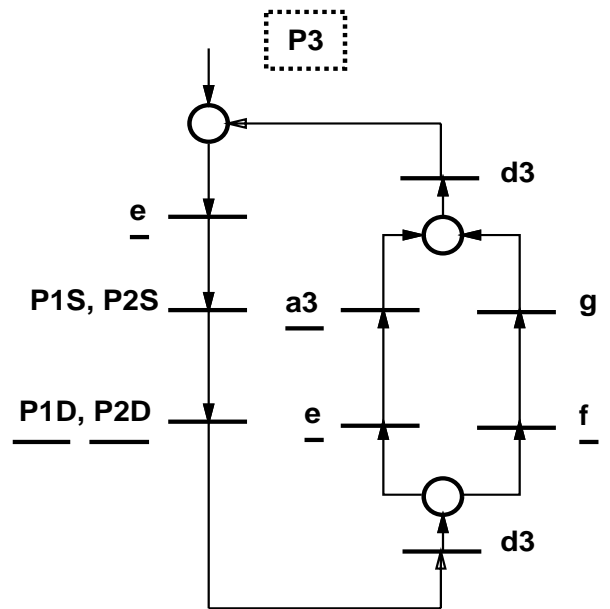
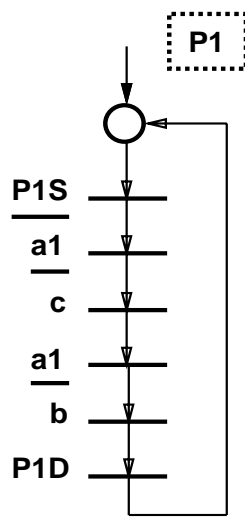
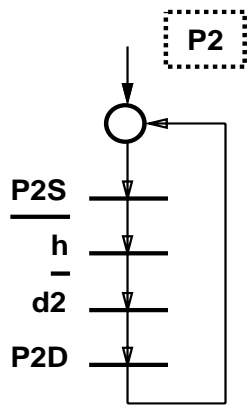
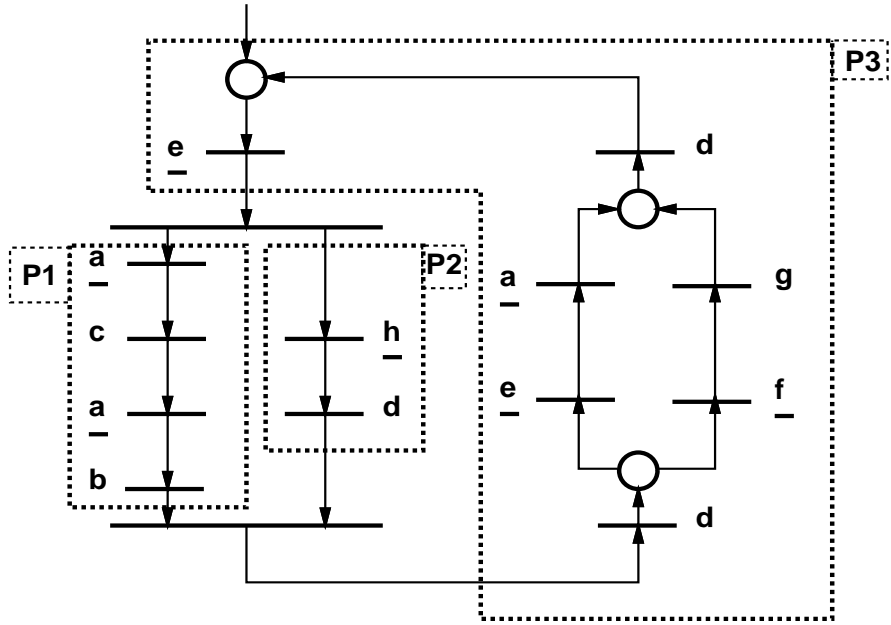
- Solution:
 - Handshakes for sequential control flow between partitions
 - Input and output state machines to handle distribution of shared signals

- Partitioning approach:

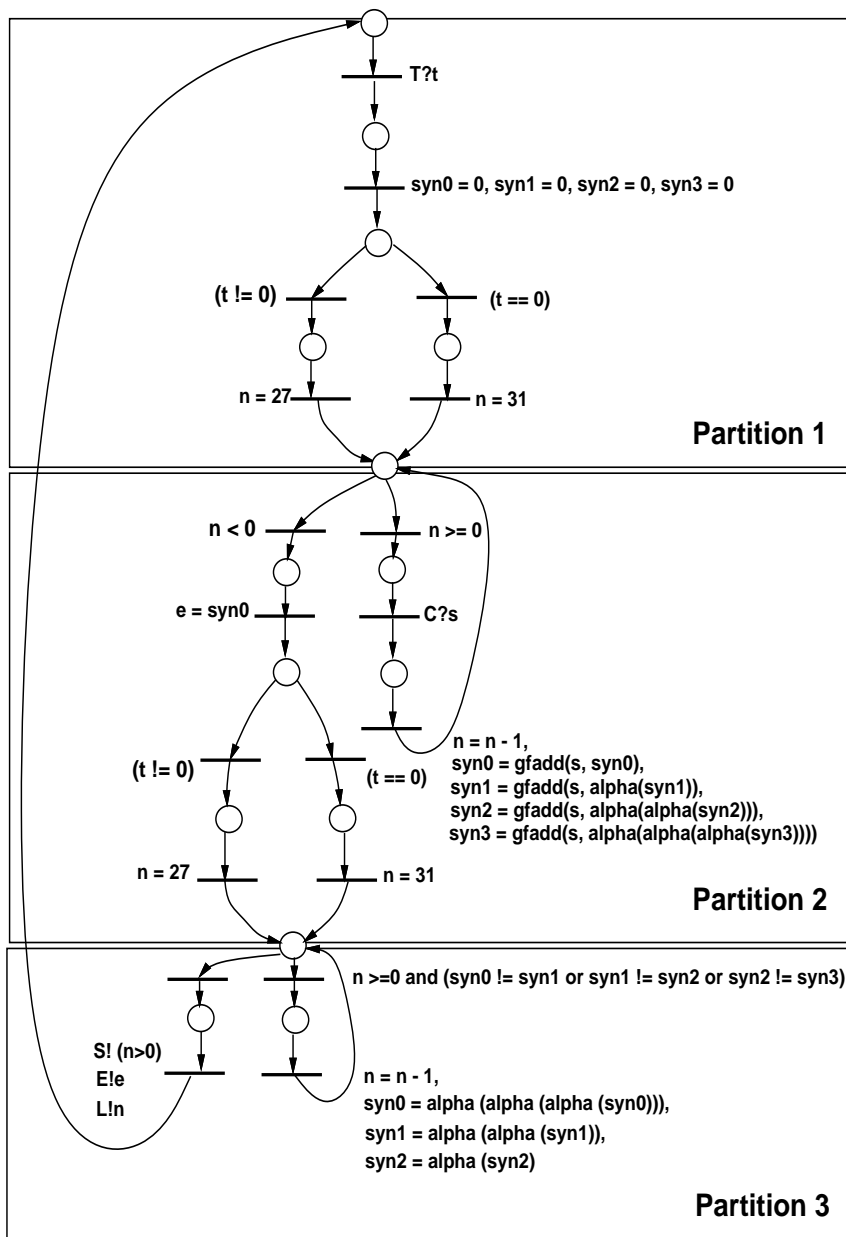
- Splits sequential controller flow
- Sequential flow is ensured by handshaking between partitions under fundamental mode assumption



- Illustration of Partitioning



- Example of Partitioning - CD Player Error Detector
 - Divided into three partitions to
 - * Minimize overhead for loops
 - * Make design synthesizable
 - Resulted in 5 Input and Output State Machines for shared variables



Results for Partitioning

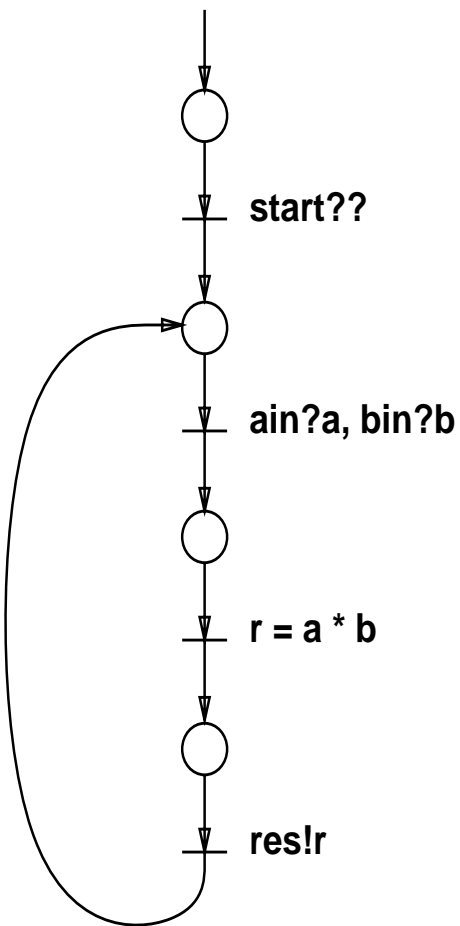
| Controller | Num of BM transitions | Size of IO set | Synthesis time (secs) | Num of literals |
|--------------------------|-----------------------|----------------|-----------------------|-----------------|
| <i>CD Error Detector</i> | | | | |
| Centralized | 1824 | 68 | did not finish | - |
| Partition 1 | 110 | 18 | 800 | 94 |
| Partition 2 | 32 | 29 | 220 | 96 |
| Partition 3 | 28 | 25 | 140 | 122 |
| ISM 1,2,3 | 81 | 13 | 230 | 63 |
| ISM 4 | 16 | 6 | 90 | 14 |
| ISM 5 | 7 | 14 | 80 | 92 |
| <i>Barcode Reader</i> | | | | |
| Centralized | 960 | 49 | did not finish | - |
| Partition 1 | 26 | 14 | 34 | 14 |
| Partition 2 | 40 | 8 | 25 | 3 |
| Partition 3 | 72 | 19 | 500 | 13 |
| Partition 4 | 26 | 23 | 53 | 43 |
| ISM 1,2 | 56 | 8 | 22 | 14 |
| ISM 3 | 64 | 9 | 28 | 53 |
| <i>GCD</i> | | | | |
| Centralized | 126 | 25 | 33420 | 207 |
| Partition 1 | 22 | 15 | 30 | 33 |
| Partition 2 | 72 | 18 | 90 | 12 |
| ISM 1,2 | 48 | 7 | 25 | 14 |
| <i>Factorial</i> | | | | |
| Centralized | 44 | 20 | 620 | 88 |
| Partition 1 | 12 | 13 | 24 | 6 |
| Partition 2 | 28 | 15 | 36 | 9 |
| ISM 1,2 | 16 | 5 | 20 | 14 |
| <i>Loop Example</i> | | | | |
| Centralized | 32 | 5 | 38 | 199 |
| Partition 1 | 14 | 5 | 20 | 58 |
| Partition 2 | 12 | 6 | 16 | 3 |

Burst Mode Generation

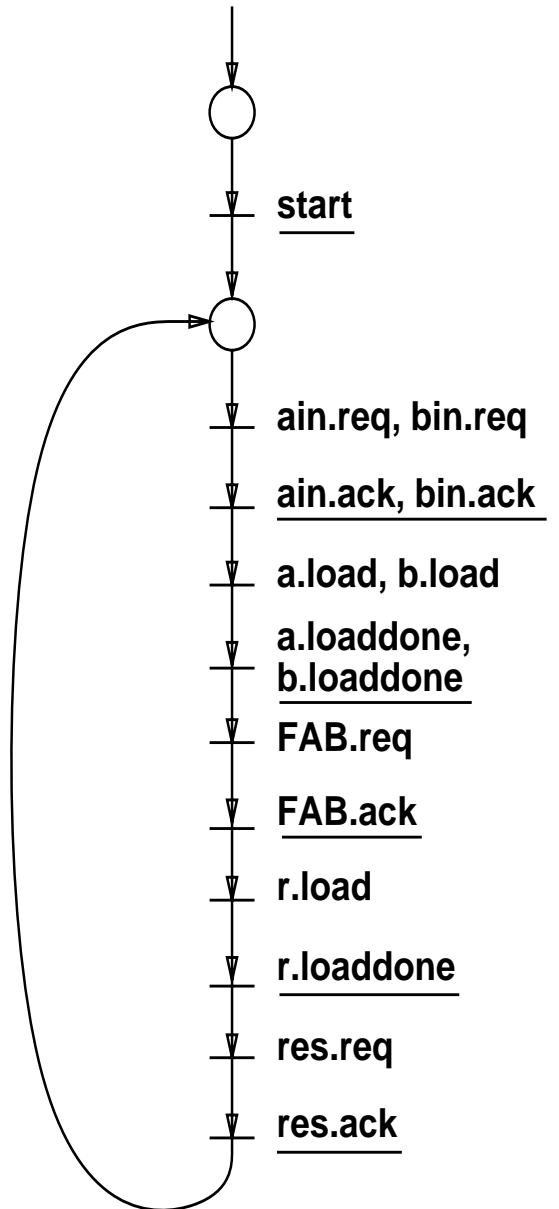
- Burst mode specification:
 - MIC
 - Huffman style
 - Maximal set property
 - Each state has unique entry point

- Burst mode generation:
 - Two phase
 - * Refined Petri Net \rightarrow State Graph
 - * State Graph \rightarrow Burst Mode Graph
 - Four phase
 - * Refined Petri Net \rightarrow State Graph
 - * State Graph \rightarrow Burst Mode Graph
 - * Reshuffle Burst Mode Graph

- Two phase case

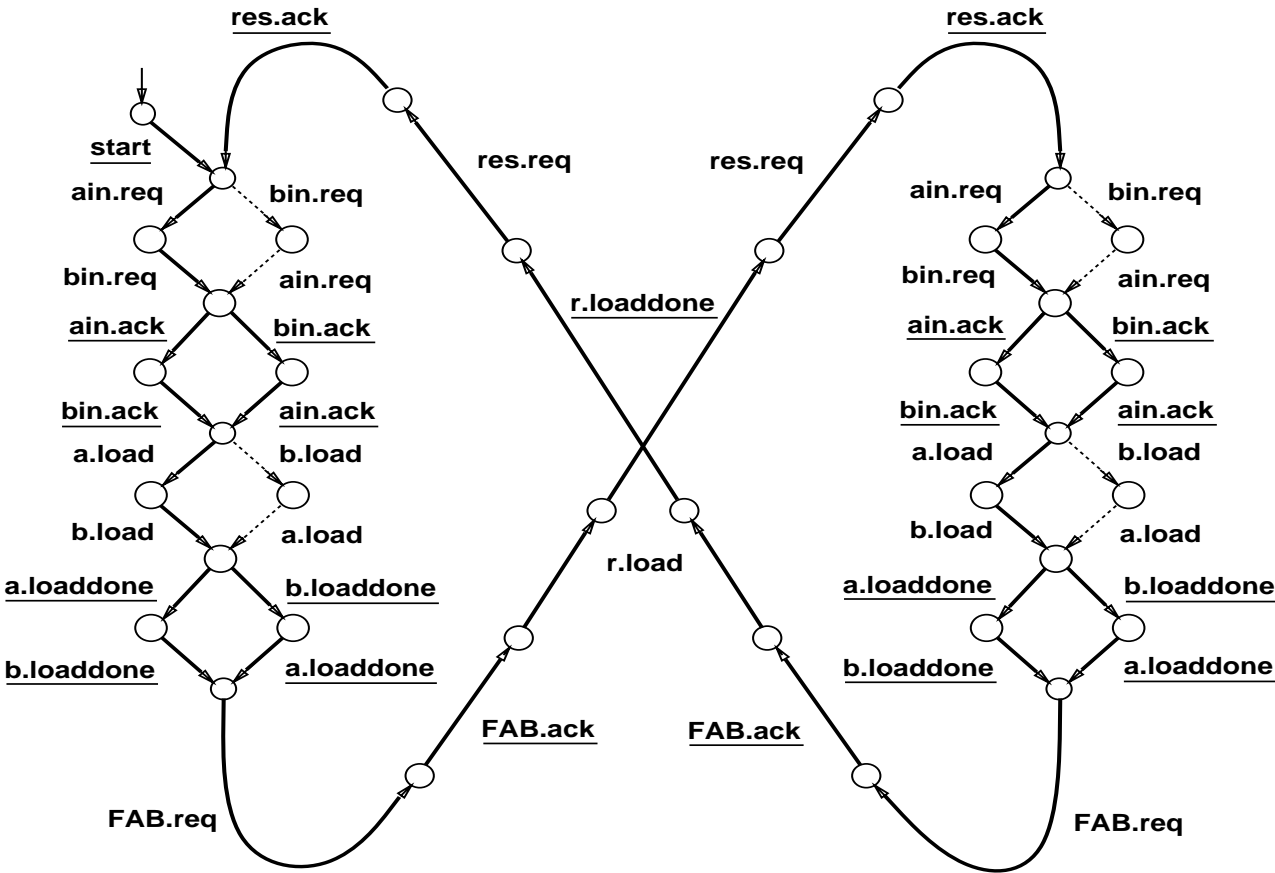


Petri Net description

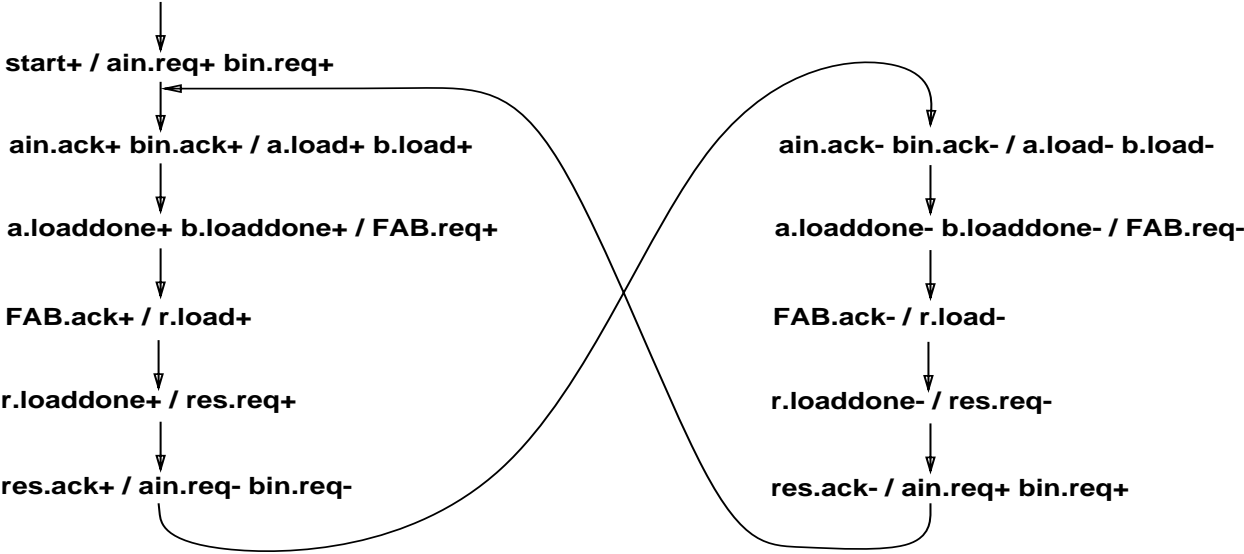


Refined Petri Net controller

Refined Petri Net \Rightarrow State Graph \Rightarrow Burst Mode Graph

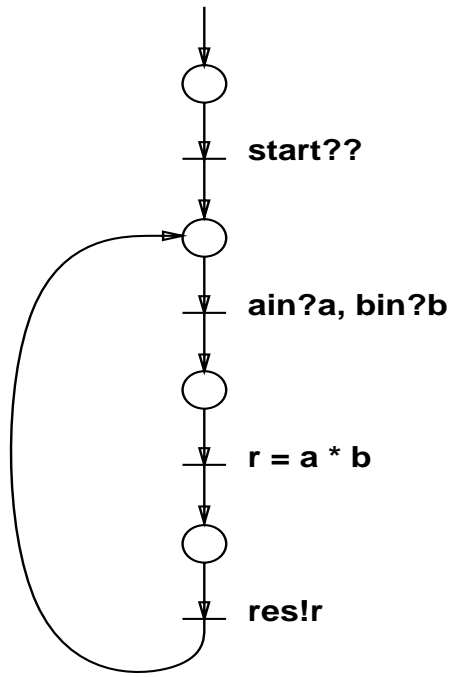


State Graph

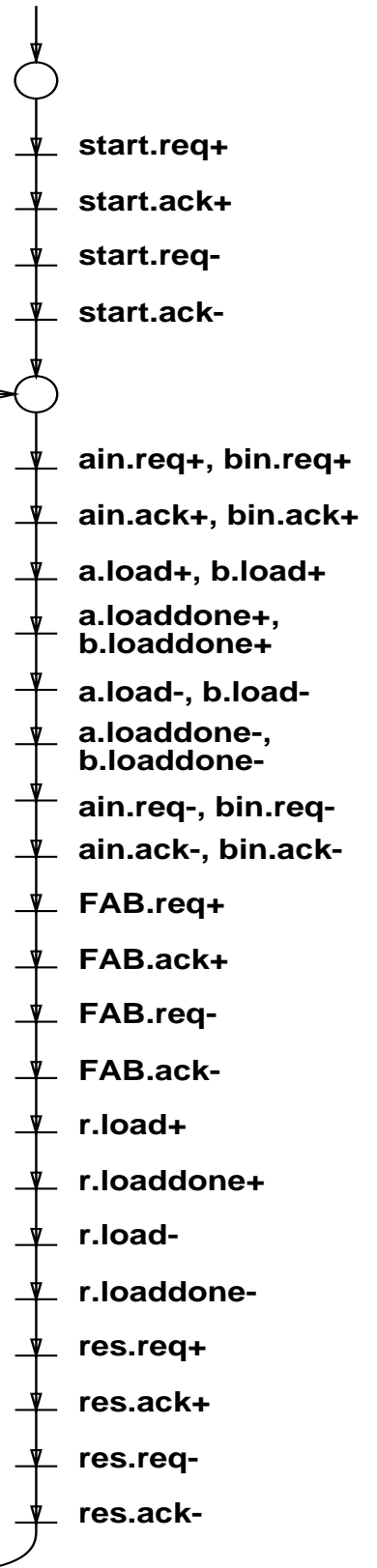


Burst Mode Graph

- Four phase case

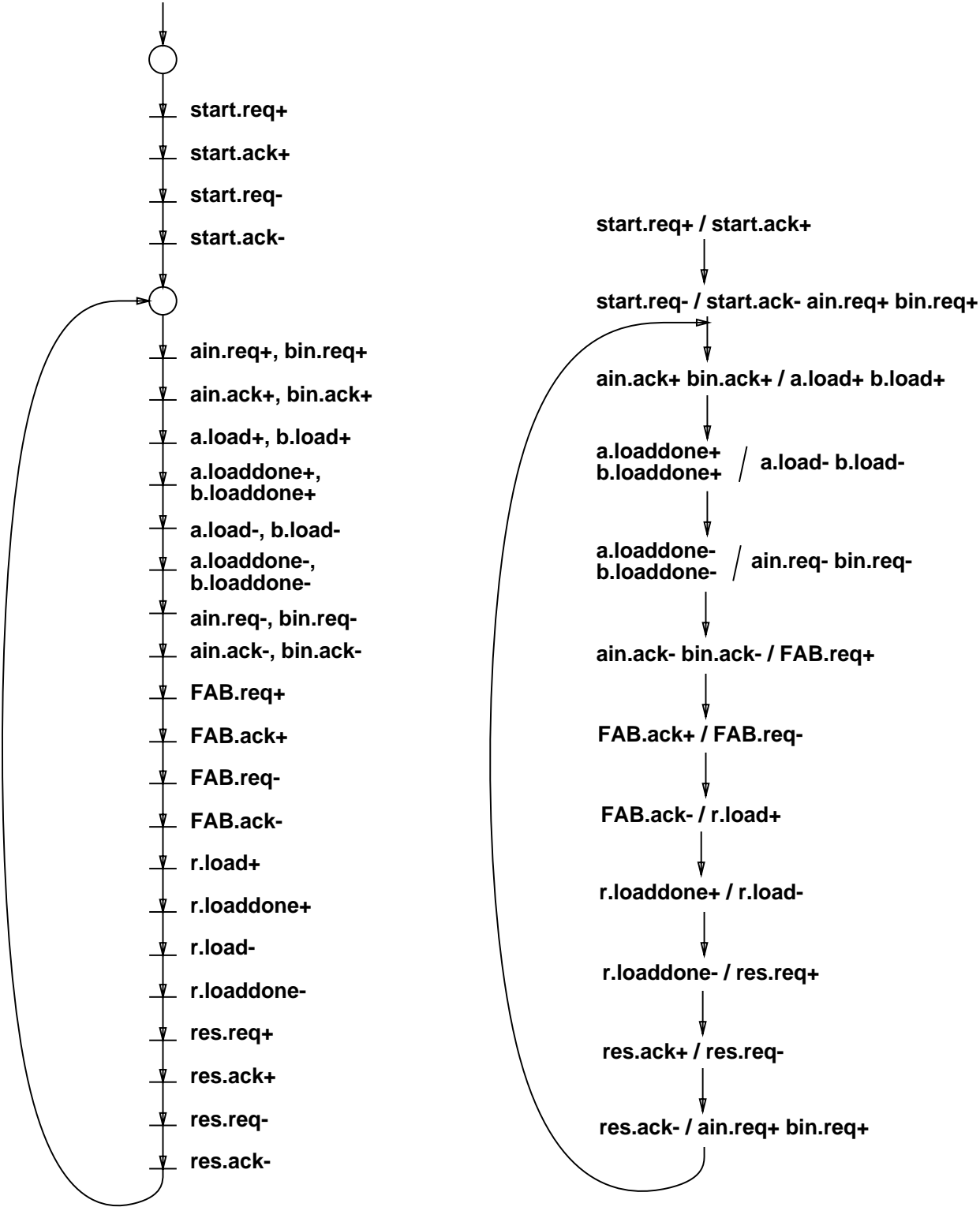


Petri Net description



Refined Petri Net controller

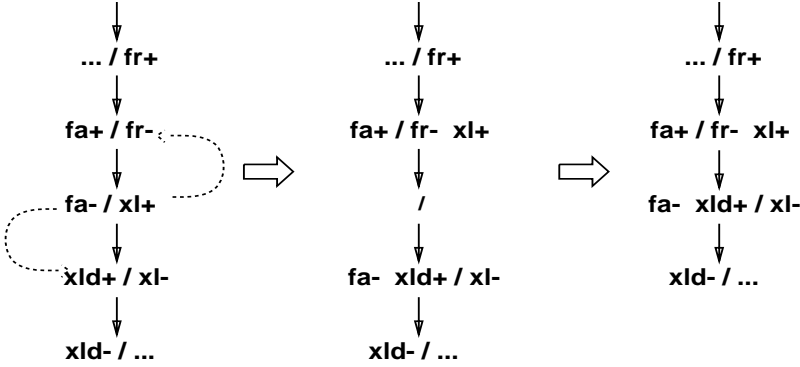
Refined Petri Net \Rightarrow Burst Mode Graph



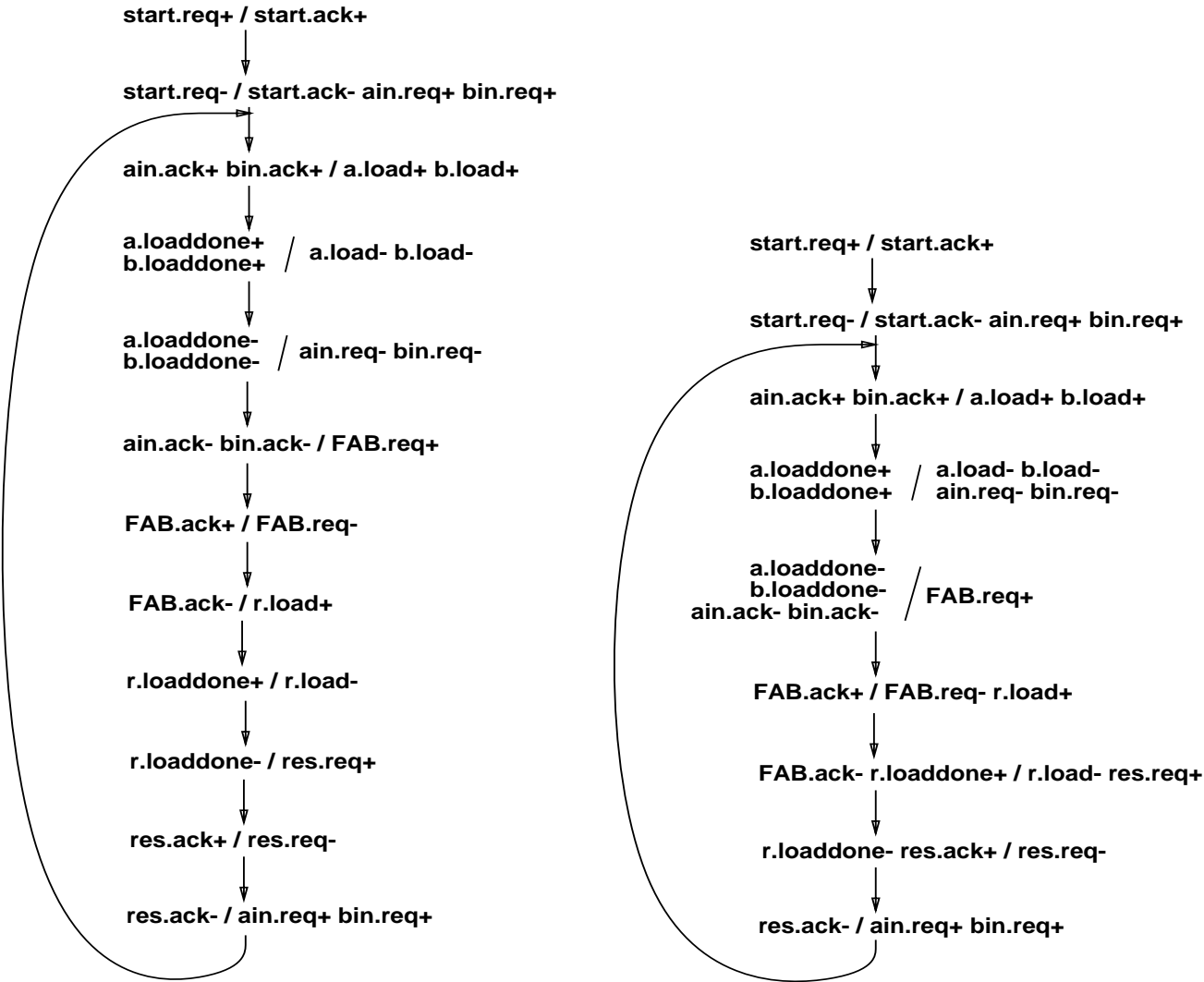
Refined Petri Net controller

Burst Mode Graph

Burst Mode Graph \Rightarrow Reshuffled Burst Mode Graph



Example of shuffling procedure



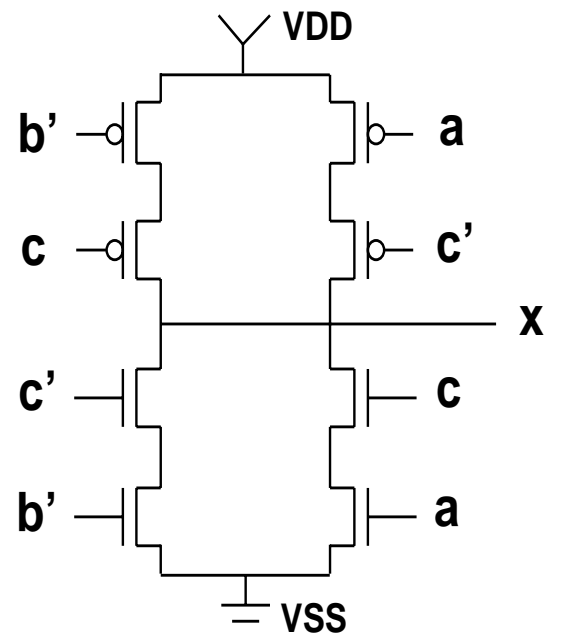
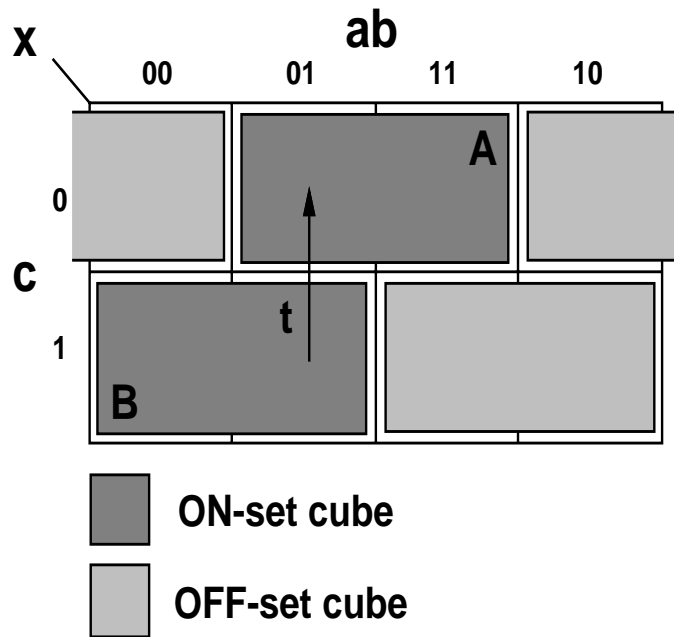
Burst Mode Graph

Reshuffled Burst Mode Graph

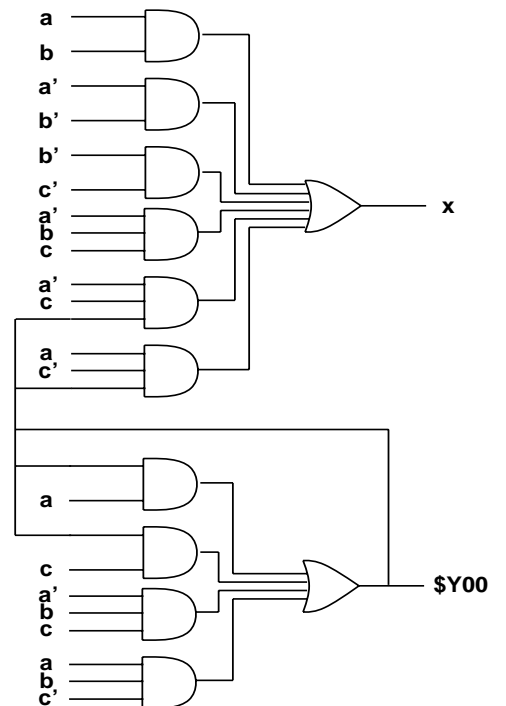
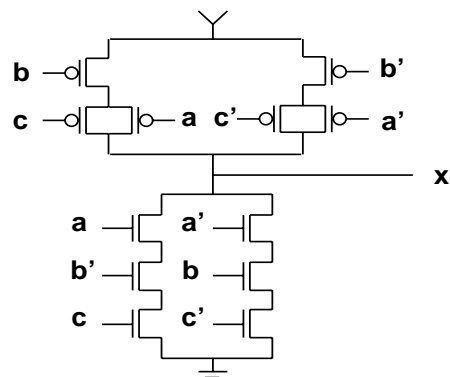
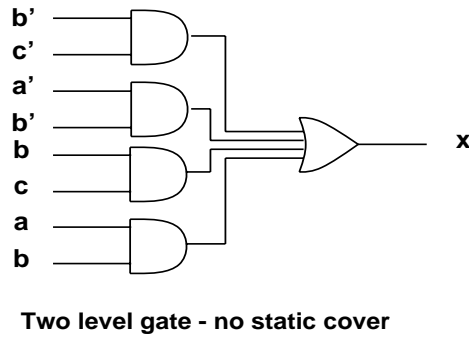
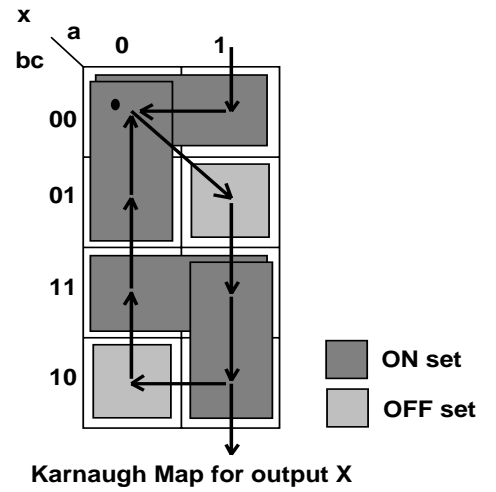
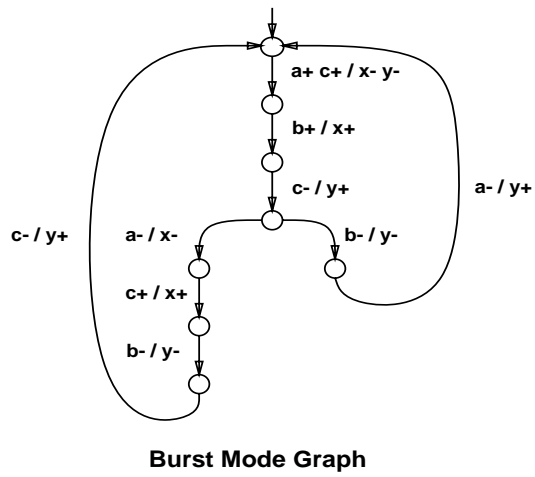
Technology Mapping

- Two level AND-OR gates
- Standard CMOS complex gates
- Customized CMOS complex gates
 - VLSI size decrease - wire delay significant
 - Reduce constraints in hazard free synthesis
 - Purely combinational solution to a larger class of problems
 - Method of logical effort can be applied at transistor level

- SOP/SOP Implementation



• SOP/SOP Single Gate Example



Results for Single Complex Gate

- Reduced synthesis constraints reduce need for adding state variables
- Customized complex gate gives less number of transistors than two level gates
- Delay is comparable to or less than two level gate implementation depending on static hazard occurrence and transistor stack height.

| Name | S.Gate #statevar | C.Gate area | S.Gate area | C.Gate delay | S.Gate delay |
|----------|------------------|-------------|-------------|--------------|--------------|
| comp_tr | 1 | 136 | 513 | 0.35 | 0.64 |
| tri_st | 1 | 148 | 636 | 0.19 | 0.81 |
| store_st | 1 | 223 | 609 | 0.43 | 0.72 |
| dual_si | 0 | 96 | 152 | 0.20 | 0.30 |
| for_lp | 1 | 213 | 332 | 0.38 | 0.53 |
| sim_mod | 1 | 102 | 278 | 0.24 | 0.58 |

- Single complex gate - table 2

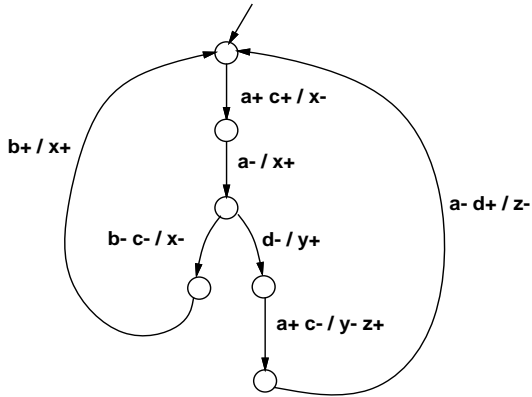
| Circuit Name | <i>Output</i> | <i>C. Gate</i> | <i>Std. Gate</i> |
|----------------|---------------|----------------|------------------|
| chu-ad-opt | dr | 1.1 ns | 1.5 ns |
| | lr | 1.75 | 1.4 |
| van-bek-ad-opt | dr | 0.9 | 1.3 |
| | zr | 0.92 | 1.34 |
| | lr | 1.2 | 0.93 |
| sendr-done | DoneS | 1.3 | 1.36 |
| sbuf-read-ctl | Ack | 1.3 | 1.36 |
| | RamRdSbuf | 0.97 | 1.53 |
| q42 | a4 | 0.94 | 1.56 |
| | r2 | 1.32 | 1.61 |

- **SOP/SOP Multilevel Complex Gate**

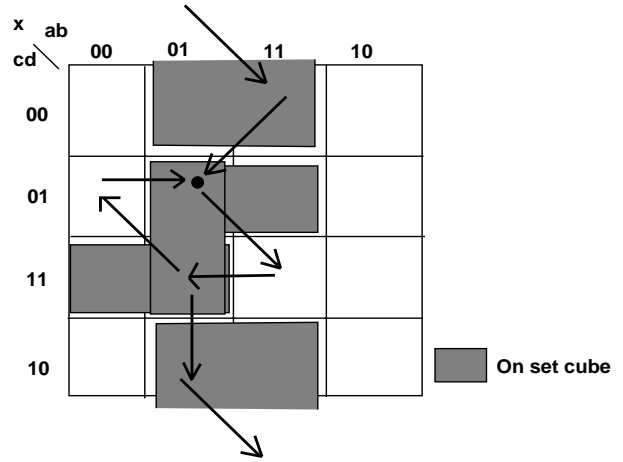
- Why Multilevel?

- * Give solution to a larger class of problems than single complex gate
- * Have solution to problems with illegally intersecting cubes of dynamic MIC transitions
- * Make purely combinational controllers
- * Reduce fundamental mode delay

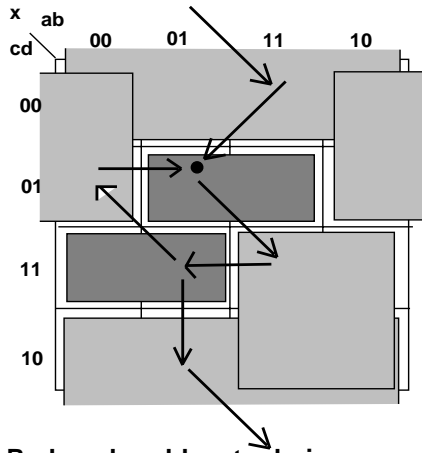
• SOP/SOP Multilevel Gate Example



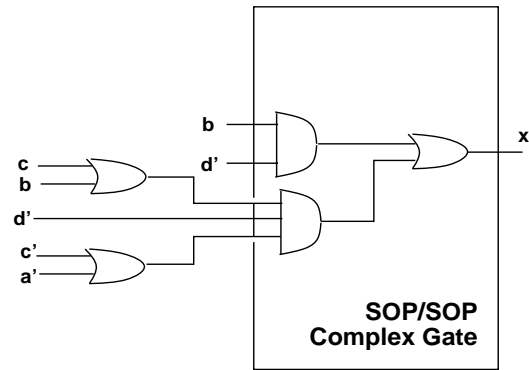
(a) Initial Burst Mode Specification



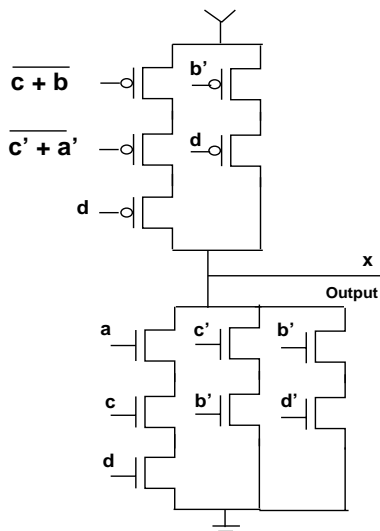
(b) Karnaugh Map for output X



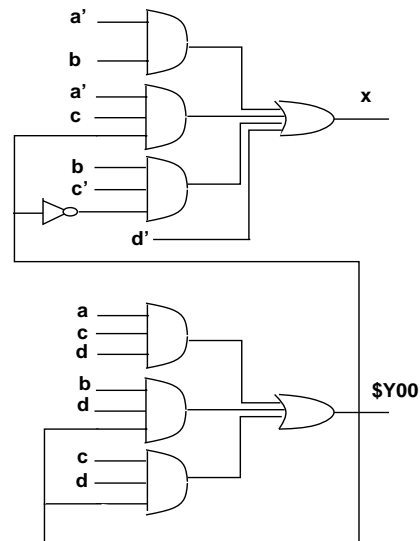
(c) Reduced problem to derive sum of products



(d) Gate level implementation - no static cover



(e) Complex Gate Implementation



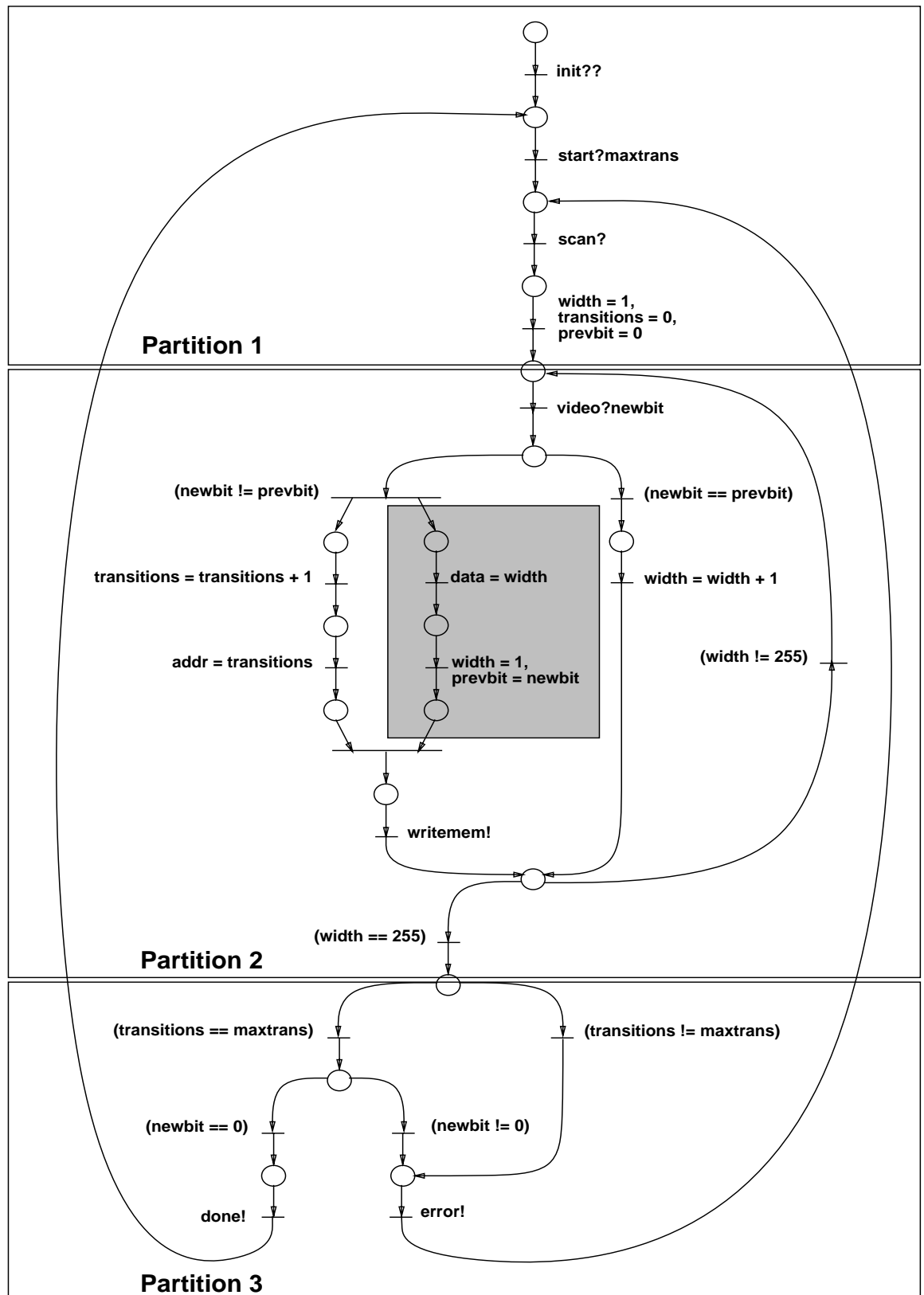
(f) Circuit for X derived by 3D

Results for Multilevel Complex Gate

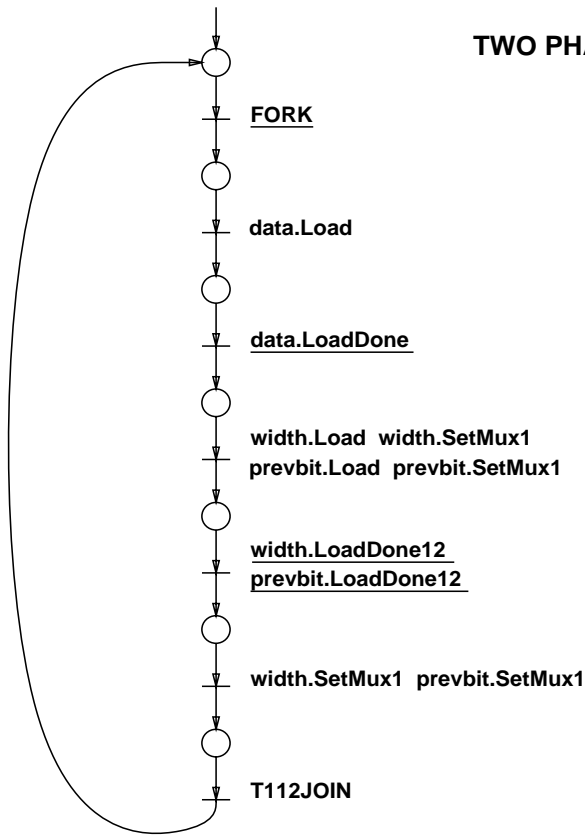
- Reduced synthesis constraints reduce need for adding state variables
- Reduce fundamental mode delay since feedbacks often can be avoided
- Multilevel customized complex gates give less number of transistors than two level gates
- Delay is comparable to or less than two level gate implementation

| Name | S.Gate #statevar | C.Gate area | S.Gate area | C.Gate delay | S.Gate delay |
|----------|---------------------|----------------|----------------|-----------------|-----------------|
| bus_tr | 2 | 271 | 1107 | 0.30 | 0.55 |
| run_dp | 1 | 360 | 1218 | 0.38 | 1.35 |
| si_stack | 1 | 177 | 312 | 0.20 | 0.32 |
| sm_stat | 2 | 187 | 1181 | 0.31 | 0.74 |
| sm_dyn | 2 | 222 | 1109 | 0.23 | 0.59 |
| l_cov | 2 | 173 | 824 | 0.21 | 0.58 |
| comp_dp | 2 | 395 | 1162 | 0.39 | 0.70 |

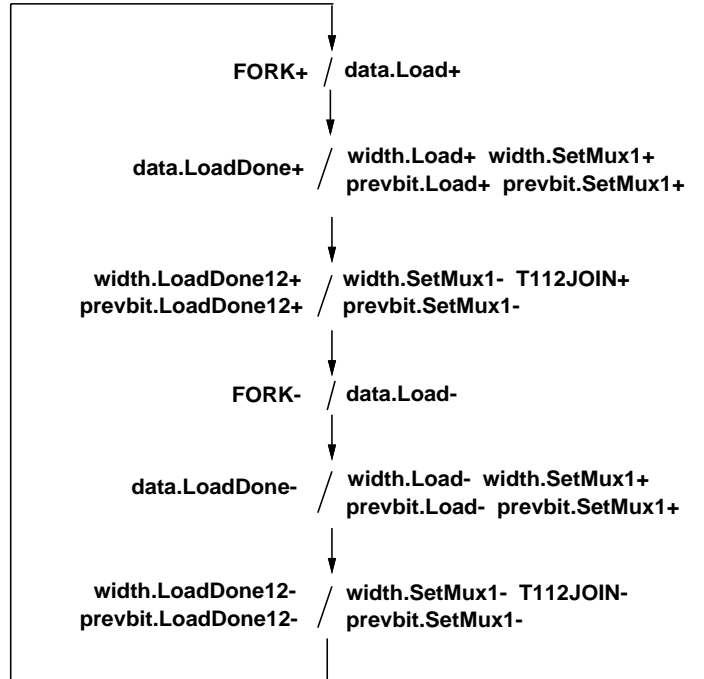
Example: Barcode Reader (HLS95)



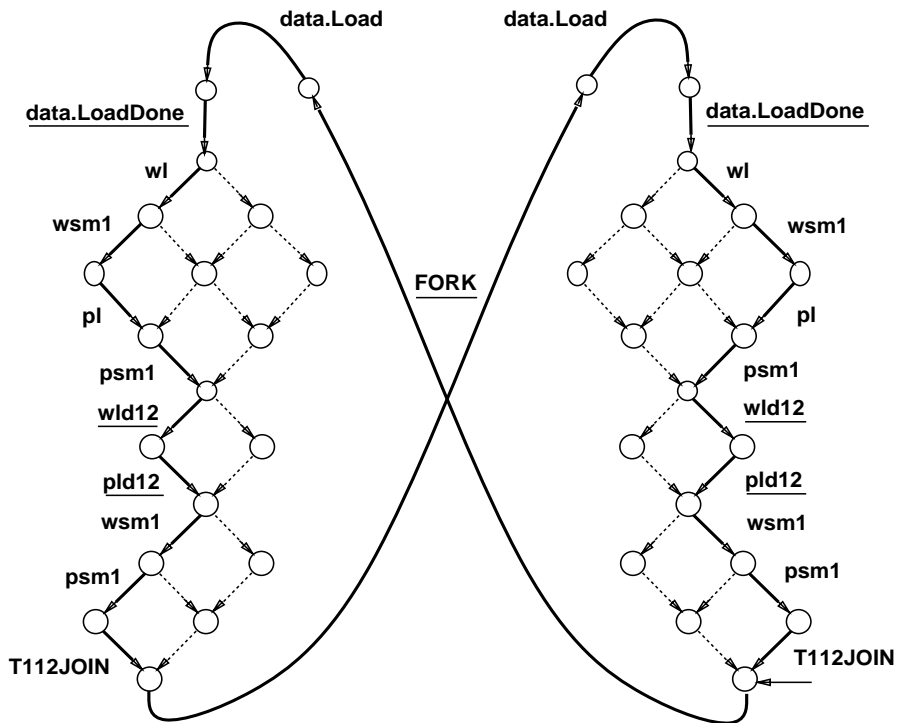
TWO PHASE



(a) Refined Petri Net controller

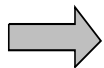
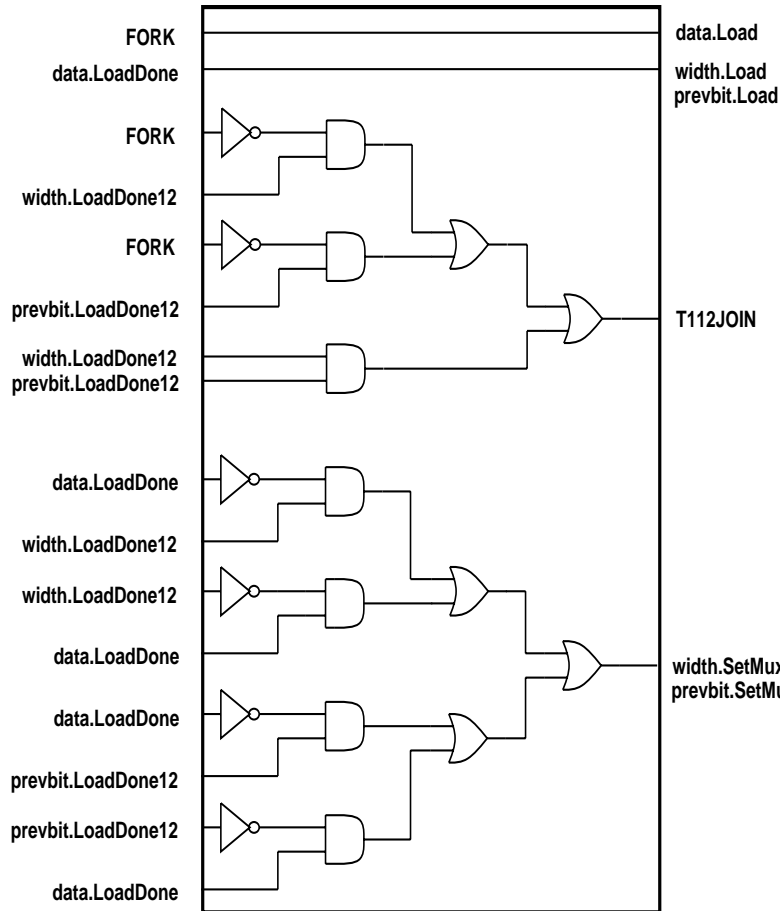


(c) Burst Mode controller



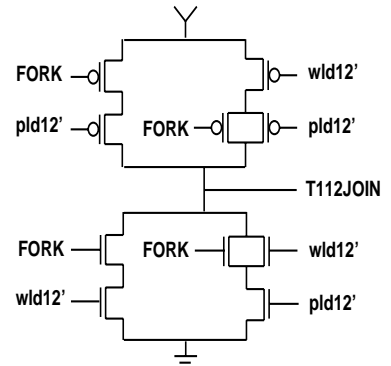
(b) State Graph

transistors:
 T112JOIN = 22
 Width.SetMux1 = 32

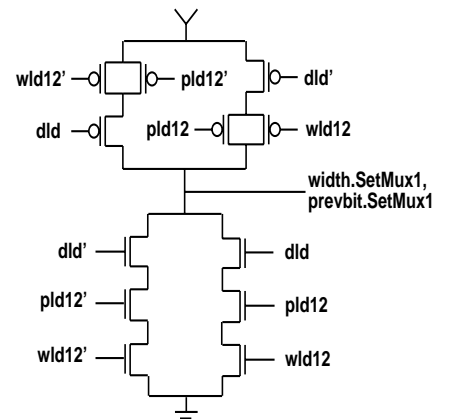


Two level gate implementation

transistors:
 T112JOIN = 14
 Width.SetMux1 = 18

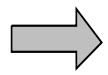
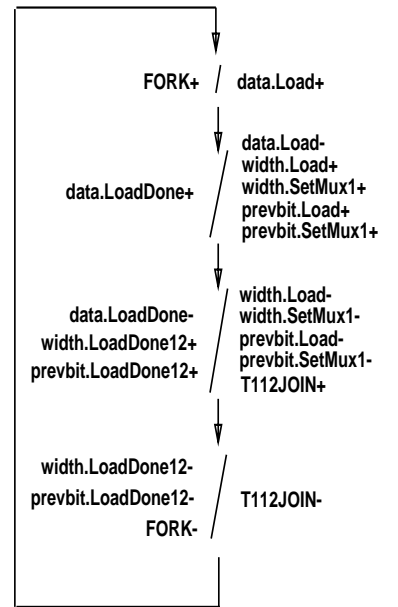
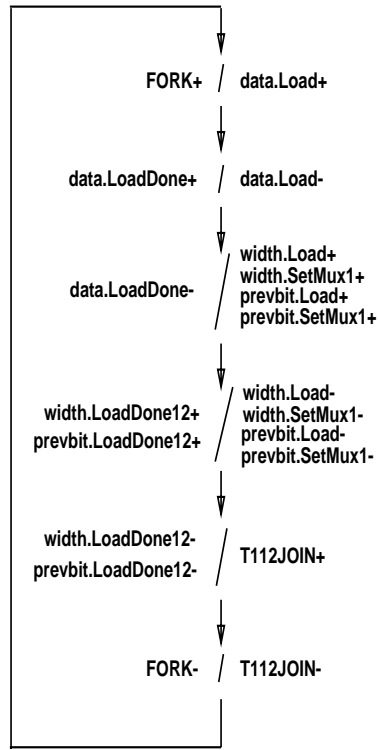
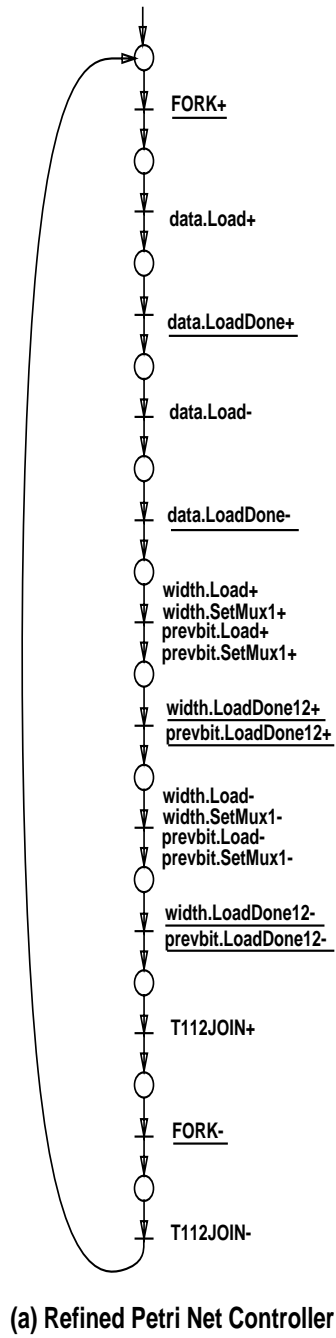


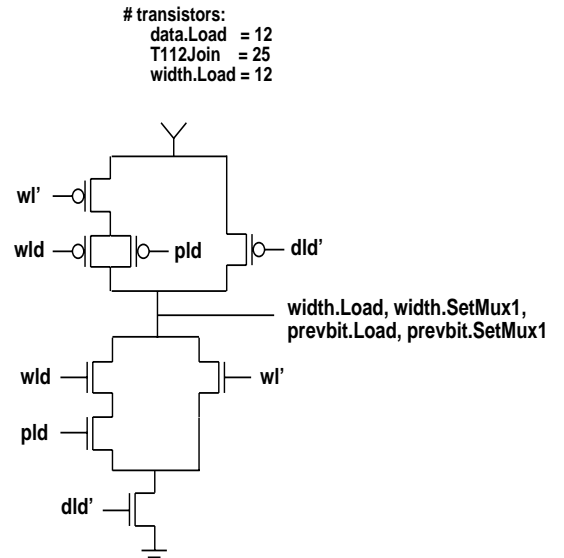
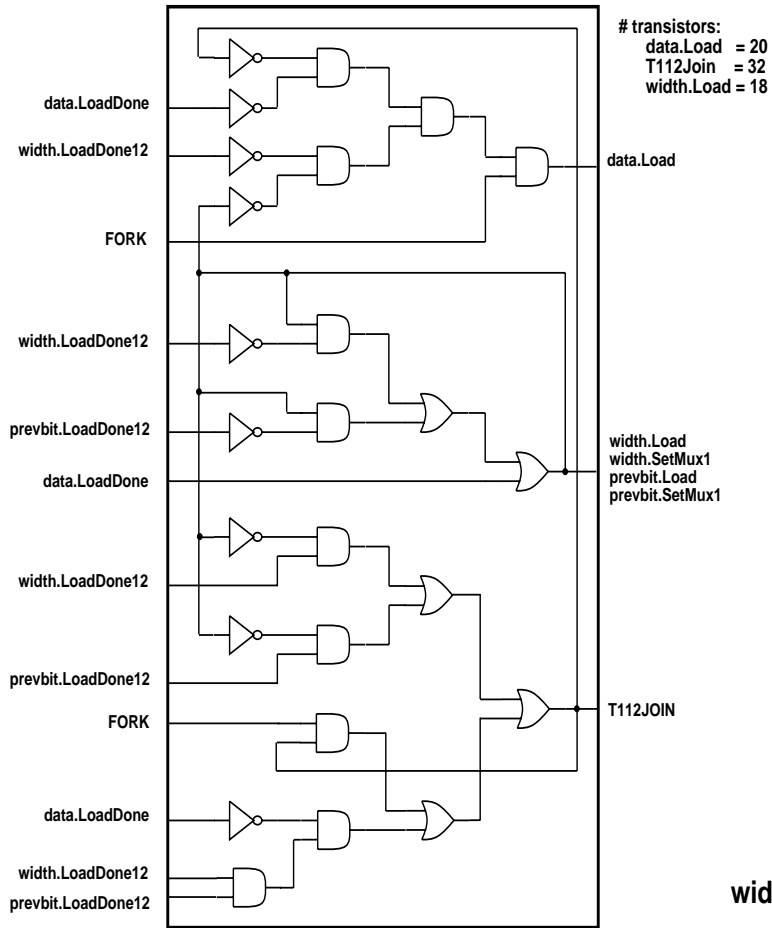
SOP/SOP complex gate for T112JOIN



SOP/SOP complex gate for width.SetMux1 and prevbit.SetMux1

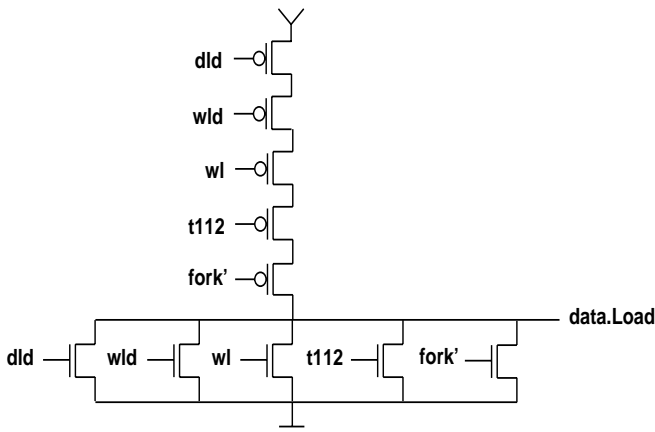
BARCODE: FOUR PHASE



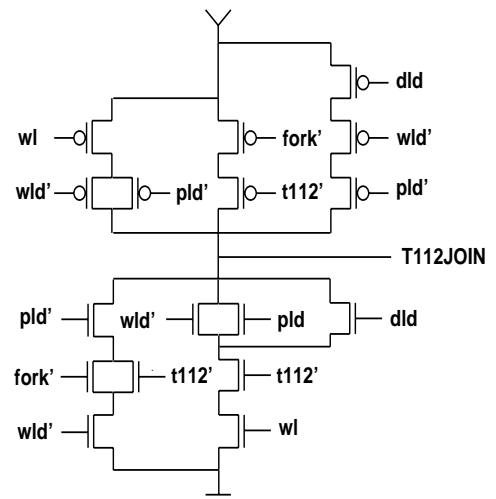


SOP/SOP complex gate for width.Load, width.SetMux1, prevbit.Load and prevbit.SetMux1

Two Level Gate Implementation



SOP/SOP complex gate for data.Load



SOP/SOP complex gate for T112JOIN

Results for two and four phase

| Controller | 2/4 phase protocol | Num of BM transitions | Synth Time | IO size | Literals |
|-----------------------|--------------------|-----------------------|------------|---------|----------|
| <i>Factorial</i> | | | | | |
| shuffle | 2 | 40 | 150 | 19 | 101 |
| shuffle | 4 | 13 | 35 | 19 | 69 |
| <i>Factorial pipe</i> | | | | | |
| Partition 1 | 2 | 32 | 38 | 15 | 50 |
| shuffle | 4 | 11 | 30 | 15 | 42 |
| Partition 2 | 2 | 32 | 63 | 14 | 47 |
| shuffle | 4 | 9 | 26 | 14 | 33 |
| <i>Reg_int</i> | | | | | |
| shuffle | 2 | 28 | 84 | 26 | 36 |
| shuffle | 4 | 15 | 54 | 26 | 65 |
| <i>Reg_sh</i> | | | | | |
| shuffle | 2 | 26 | 160 | 26 | 74 |
| shuffle | 4 | 15 | 67 | 26 | 78 |
| <i>Mixed_rc</i> | | | | | |
| shuffle | 2 | 88 | 3240 | 19 | 212 |
| shuffle | 4 | 17 | 87 | 22 | 63 |
| <i>Diffeq (HLS92)</i> | | | | | |
| Partition 1 | 2 | 26 | 127 | 29 | 109 |
| shuffle | 4 | 13 | 74 | 27 | 51 |
| Partition 2 | 2 | 52 | 110 | 26 | 62 |
| shuffle | 4 | 17 | 88 | 29 | 52 |
| Partition 3 | 2 | 22 | 60 | 25 | 47 |
| shuffle | 4 | 13 | 64 | 27 | 43 |
| ISM 1,2,3 | 2 | 16 | 18 | 5 | 14 |
| noshuffle | 4 | 8 | 18 | 5 | 6 |
| <i>Barcode Reader</i> | | | | | |
| Partition 1 | 2 | 26 | 34 | 14 | 14 |
| shuffle | 4 | 9 | 25 | 17 | 25 |
| Partition 2 | 2 | 40 | 25 | 8 | 3 |
| shuffle | 4 | 9 | 18 | 10 | 15 |
| Partition 3 | 2 | 72 | 501 | 19 | 13 |
| shuffle | 4 | 14 | 33 | 21 | 33 |
| Partition 4 | 2 | 26 | 53 | 23 | 43 |
| shuffle | 4 | 9 | 38 | 25 | 81 |
| ISM 1,2 | 2 | 56 | 22 | 8 | 14 |
| noshuffle | 4 | 14 | 16 | 8 | 8 |
| ISM 3 | 2 | 64 | 28 | 9 | 53 |
| noshuffle | 4 | 16 | 19 | 9 | 17 |

Contributions

- Behavioral description in standard language
- High level synthesis targeting state machines
- Two and four phase implementation
- Partitioning of incompletely specified machines
- Hazardfree technology mapping to complex gates
- Complete asynchronous design framework
- Many realistic examples