Synthesis of control circuits from STG specifications. Problems

J. Cortadella M. Kishinevsky A. Kondratyev L. Lavagno A. Yakovlev

July 22, 1997

1 Handshake communication

1.1 Half-handshake

Figure 1 shows a data processing structure consisting of two computation blocks, A and B, and a control circuit. The goal is to design the speed-independent/ control circuit. Different control disciplines are possible. Let us assume that we have chosen a discipline, called half-handshake. The STG specifying the half-handshake control circuit is given in Figure 2. Signals Ri and Ai are inputs, and Ao and Ro are outputs.

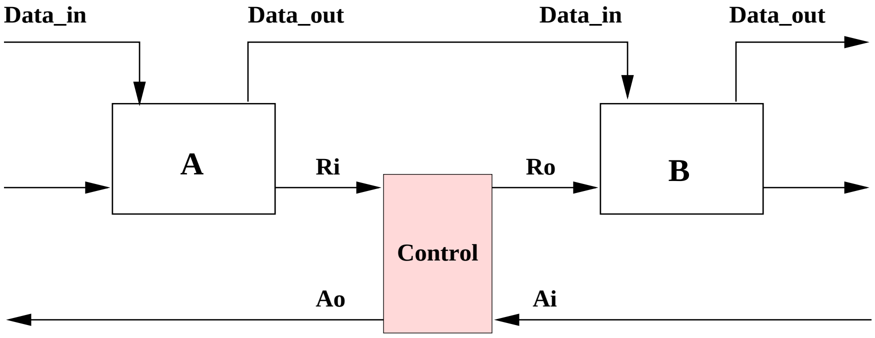

Figure 1: A data processing structure.

1. Specify half-handshake control following steps presented in Figure 2.
2. Check that the second version is unbounded. Check that the third version has CSC-violations. Check that the final version is implementable with a speed-independent/ circuit.
3. Derive netlists of logic equations for output signals using `-cg` option of `petrify`.
4. Derive monotonic cover implementation using `-mc` option.

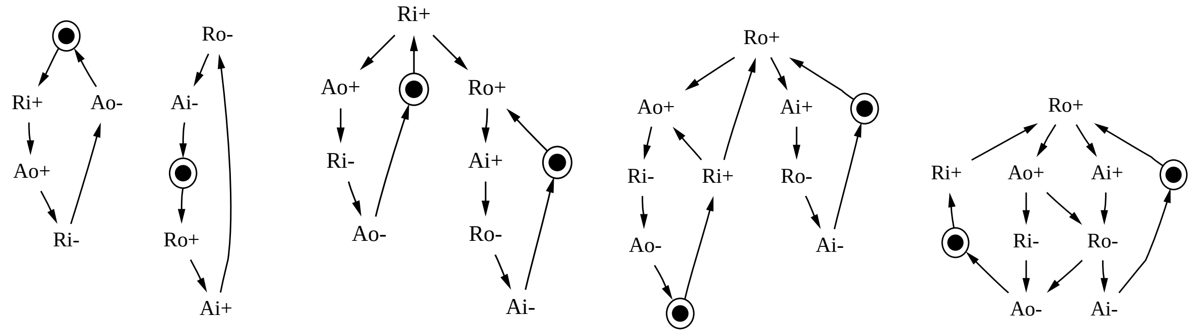

Figure 2: An STG for the half-handshake: (a) independent handshakes at the left and right ports (b) unbounded version with $Ri+ \rightarrow Ro+$, (c) version with CSC-violation, (d) final version with CSC and a redundant arc removed

5. Derive implementation for the library of three-input gates using `-lit3` option and then for the library of two-input gates using `-lit2` option. Compare the implementation for signal Ao. Explain why it is different.

6. Find implementation using only C-elements as asynchronous latches. Use `-latch C` option.

7. Find implementation without C-elements using `-latch SRD` option.

8. Find implementation without latches at all, using combinational gates with possible feedbacks using `-nolatch` option. Observe the result of signal insertion into the STG using `xastg` drawing utility.

9. Explain why the half-handshake control circuit allows only half-utilization of the pipeline. **Hint: There are two arcs in the STG corresponding to computation:** $Ao- \rightarrow Ri+$ **and** $Ro+ \rightarrow Ai+$

1.2 Full-handshake

Produce another specification of the control circuit which would allow to achieve full utilization of the computational modules in the pipeline.

Hint: for synchronization between the left and the right handshake cycles use arc $Ao+ \rightarrow Ro+$ **as shown in Figure 3**. As before signals Ri and Ai are inputs, and Ao and Ro are outputs.

1. Complete the specification of the full-hanshake control circuit following steps similar to presented in Figure 2, removing unbounded behavior and CSC-violations.

2. Explain why the full-handshake control circuit allows naighbor modules to compute simultaneously. **Hint: There are two arcs in the STG corresponding to computation:** $Ao- \rightarrow Ri+$ **and** $Ro+ \rightarrow Ai+$

3. Repeat all synthesis steps, which have been proposed for the half-handshake circuit.

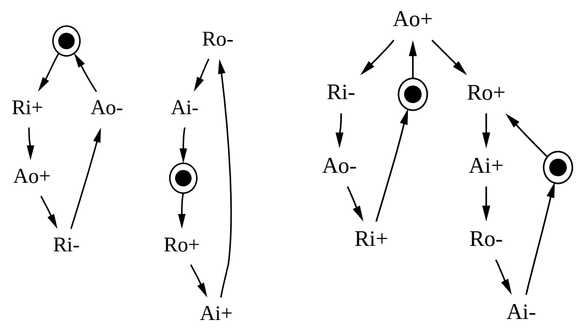

Figure 3: An STG for the full-handshake: (a) independent handshakes at the left and right ports (b) unbounded version with $Ao+ \rightarrow Ro+$

1.3 Solution for the full-handshake specification

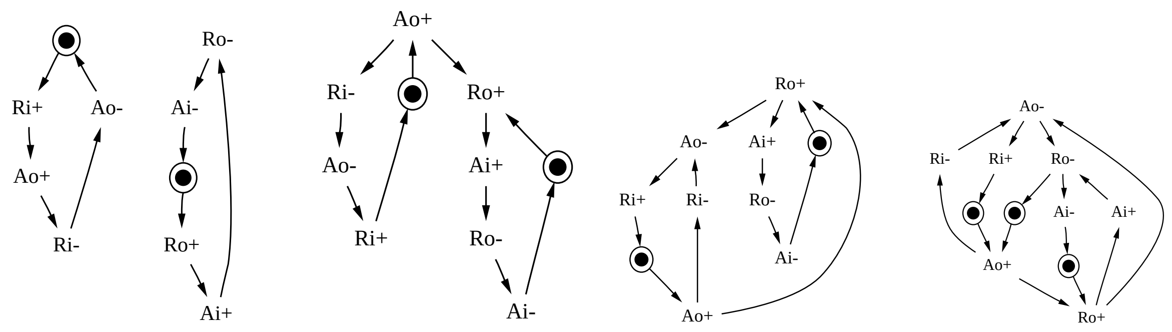

Figure 4: An STG for the full-handshake: (a) independent handshakes at the left and right ports (b) unbounded version with $Ao+ \rightarrow Ro+$, (c) version with a CSC-violation, which cannot be solved without changing interface of the circuit. Arc $Ro+ \rightarrow Ao-$ is inserted to remove non-persistency for signal Ao. (d) final version with CSC and a redundant arc removed

2 1-bit variable

Figure 5 depicts the specification and an implementation of an RS-latch with completion detection.

1. Specify and synthesize a circuit implementing a 1-bit variable with an interface as shown in Figure 6 and using the SR latch with completion also shown in the figure.

   The handshake to write the value i into the variable is as follows:

   $$\text{wr\_}i\text{+} \rightarrow \text{ack\_wr+} \rightarrow \text{wr\_}i\text{-} \rightarrow \text{ack\_wr-}$$

   To read a value:

   $$\text{req\_rd+} \rightarrow \text{rd\_}i\text{+} \rightarrow \text{req\_rd-} \rightarrow \text{rd\_}i\text{-}$$

   Assume that *write* and *read* operations are mutually exclusive.

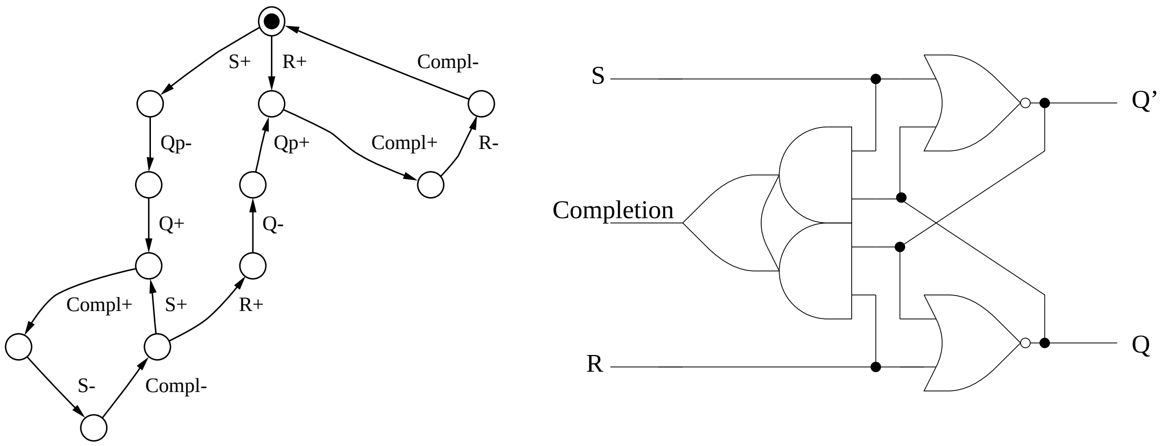

Figure 5: RS-latch with completion detection (specification and implementation).

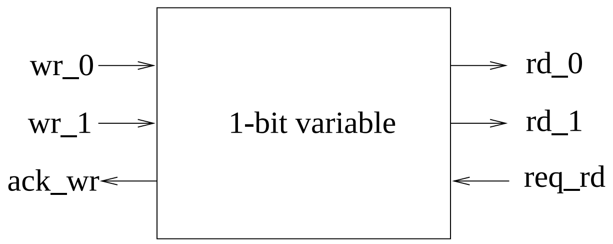

Figure 6:

2. Specify and synthesize the same circuit without using the SR latch.

3 Synchronization: Join and Fork modules

3.1 Join module

Figure 7 shows a data processing structure synchronizing two data sources and consisting of three computation blocks, A, B, and C and a control circuit. The goal is to design the speed-independent/ control circuit. Two versions of STGs partially specifying behavior of the Join are given in Figure 8. Signals Ri and Ai are inputs, and Ao and Ro are outputs.

1. Complete the first specifications for the Join.
2. Synthesize a netlist with `-cg` option.
3. Perform decomposition and technology mapping with `-lit2 -tm` options.
4. Do the same for the second specification.

3.2 Fork module. Four-phase – Two-phase transformations

Figure 9 shows a data processing structure perform inverse with respect to Join operation. This operation is called Fork.

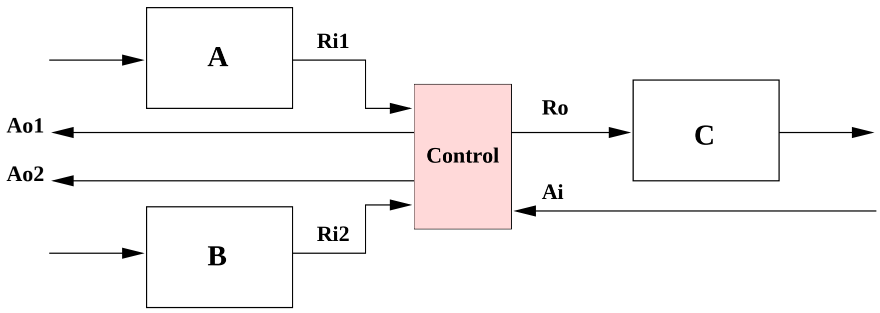

Figure 7: A Join structure.

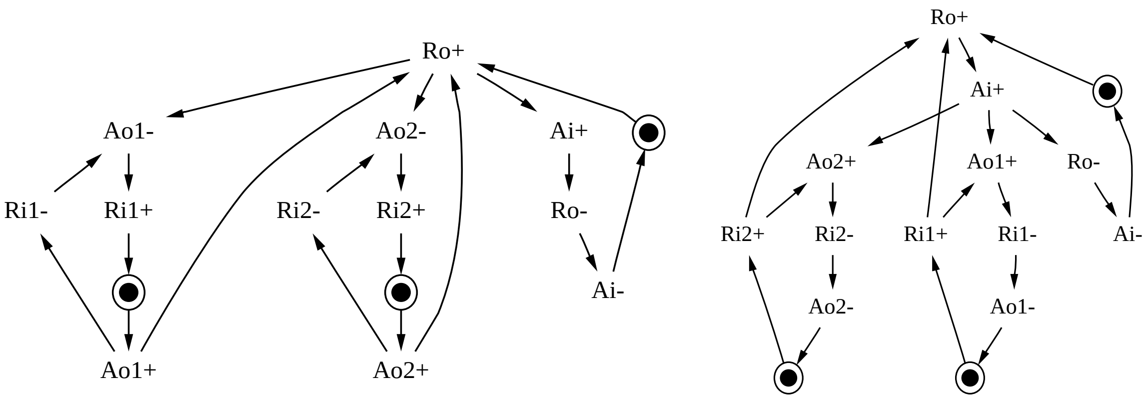

Figure 8: Two versions of the partial STG specification for the Join.

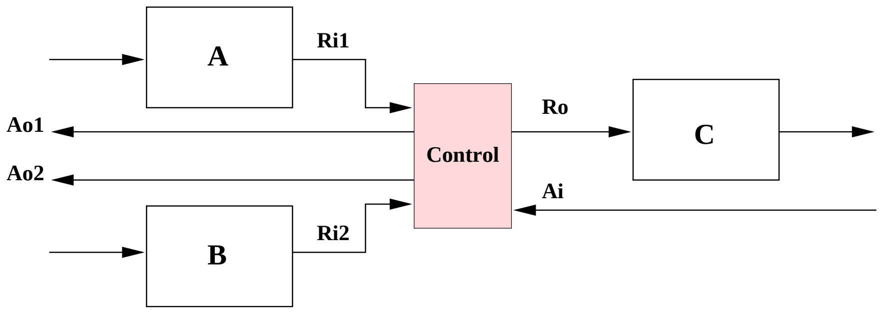

Figure 9: A Fork structure (inverse to the Join).

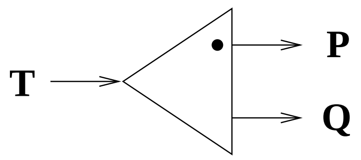

Figure 10: Toggle

1. Specify behavior of the Fork using *two-phase* specification
2. Transform specification to a four-phase by applying `-untog` option
3. Use inverse transformation with `-tog` option to check that you can get back the initial specification.
4. Synthesize implementations using different options of `petrify`
5. Hide $Ro2$ and $Ai2$ and perform projection.

4 Counters

This section is devoted to the design and implementation of modulo-n counters. A counter is a circuit with one input signal, T, and two output signals, P and Q. The behavior of a modulo-n counter can be modeled by the following regular expression:

$$((T;P)^{n-1};T;Q)^*$$

where the symbols T, P and Q denote events on their respective signals (either positive or negative transitions).

A similar behavior can be derived by using a 4-phase handshake protocol:

$$((T+;P+;T-;P-)^{n-1};T+;Q+;T-;Q-)^*$$

4.1 Toggle

A toggle circuit is a modulo-2 counter, and is represented by the symbol shown in Figure 10.

1. Specify the behavior of a toggle using a 2-phase protocol. Check that the STG fulfils the CSC condition.
2. Derive monotonic covers for the output signals (with `petrify -mc`).
3. Derive a netlist by mapping the circuit onto a library (`-tm`) using only C elements (`-latch C`).
4. Derive a netlist using any type of latch in the library and compare with the solution obtained previously.

5. Specify the behavior of a toggle using a 4-phase protocol. Solve CSC manually and compare the solution with the one given by petrify.

6. Derive a netlist by mapping the circuit onto a library. Compare with the circuit obtained when using a 2-phase protocol.

4.2 Counter

1. Specify the behavior of a modulo-5 counter using a 2-phase protocol and derive a speed-independent implementation.

2. Specify and synthesize a similar behavior using a 4-phase protocol and compare the solution with the previous one.

5 2-to-4 and 4-to-2 phase converters

Figure 11 depicts the interface of 2-to-4 and 4-to-2 phase converters.

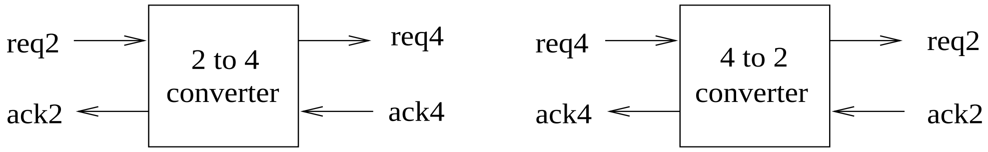

Figure 11: 2-to-4 and 4-to-2 phase converters

5.1 2-to-4 phase converter

1. Specify the behavior of a 2-to-4 phase converter. Solve CSC manually and compare the solution with the one given by `petrify`.

2. Derive monotonic covers for the output signals (`petrify -mc`).

3. Obtain a netlist of the circuit by decomposing it into 2-input gates and mapping it onto a gate library (`petrify -lit2 -tm`) and compare with the previous solution.

5.2 4-to-2 phase converter

Figure 12 shows three different specifications for a 4-to-2 phase converter.

1. Obtain STG specifications with only rising (+) and falling (−) transitions (use the `-untog` option in `petrify`).

2. Derive a netlist for each one of the specifications and compare the solutions. Evaluate the area/delay trade off for each circuit.

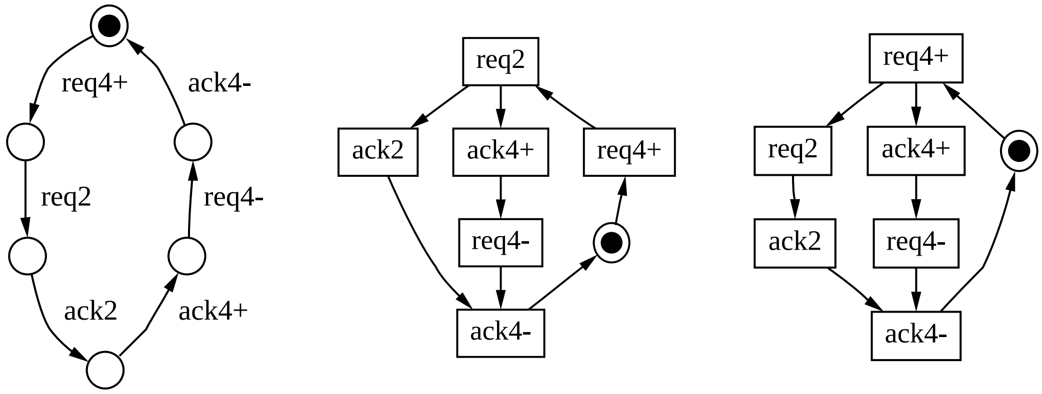

Figure 12: Specifications of 4-to-2 phase converters.

6 Handshake circuits for VLSI programming

6.1 Sequencer and parallelizer

Figure 13 depicts the interface of a sequencer, a parallelizer and a combined interconnection of both handshake circuits.

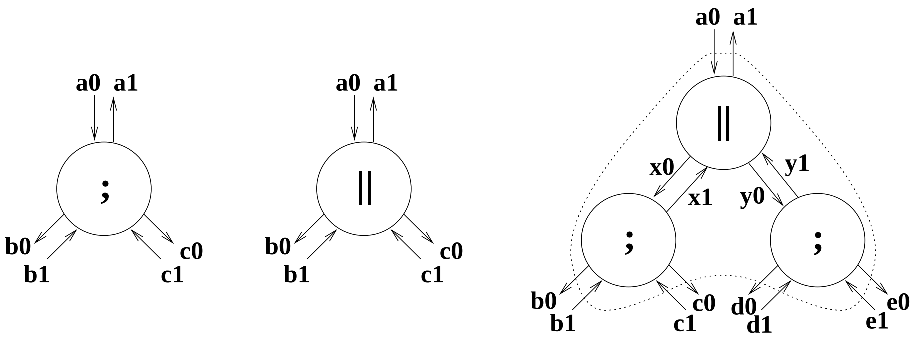

Figure 13: Sequencer and parallelizer

1. Give STG specifications for the sequencer and the parallelizer, synthesize them and obtain a netlist. Compare the solutions with the actual components used by the Tangram's compiler.

2. Give an STG describing the behavior of one parallelizer and two sequencers connected as shown in the figure. The specification must describe the events of the internal signals $x0$, $x1$, $y0$ and $y1$ (explicitly declared as `.internal`).

3. Obtain an STG with the behavior of the external signals only. This can be obtained by using the option `-ni` (hide internal signals) in `petrify`.

4. Derive a netlist from the specification previously obtained (use `-lit2 -tm`). Compare the solution with the one that would be obtained after connecting the circuits implementing the parallelizer and the sequencer.

6.2 If and Do components

The interface of the `If` and `Do` components is identical and shown in Figure 14. The behavior of the `If` component is as follows: a request by a is responded by a request to read a boolean value by channel b. If the value equals *false* the handshake in b and a is completed[1]. If the value equals *true* a handshake is performed in channel c before completing a. The behavior for the `Do` component is similar, but repeatedly executing handshakes on b and c until the value of b equals *false*.

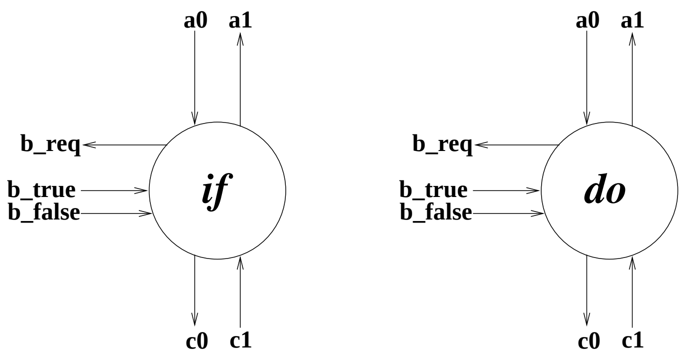

Figure 14: If and Do components.

1. Give STG specifications for the `if` and `do` components. Explore different degrees of parallelism among the handshakes of the channels.

2. Derive netlists for both components using `petrify`. Compare solutions with different degrees of parallelism.

7 Verification of speed independence

1. The 3-input AND gate shown in Figure 15 is a speed-independent implementation of the specifications E1-E4. However, the implementation with 2-input AND gates shown in the same figure is not speed-independent for some of the specifications. Determine which ones produce hazardous behavior and derive traces manifesting such behavior. Assume that all signal values are initially 0.

2. Are the circuits in Figure 16 speed-independent under the environment specified by the STG ? If not, give a trace of events producing hazardous behavior. (Note: Assume the initial value 0 for the internal signal z).

8 VME bus controller

Figure 17 depicts the interface of a device with a VME bus. The behavior of the controller is as follows: a request to read from or write into the device is received by one of the signals

[1]Note that this behavior is slightly different from the one used for the Tangram compiler

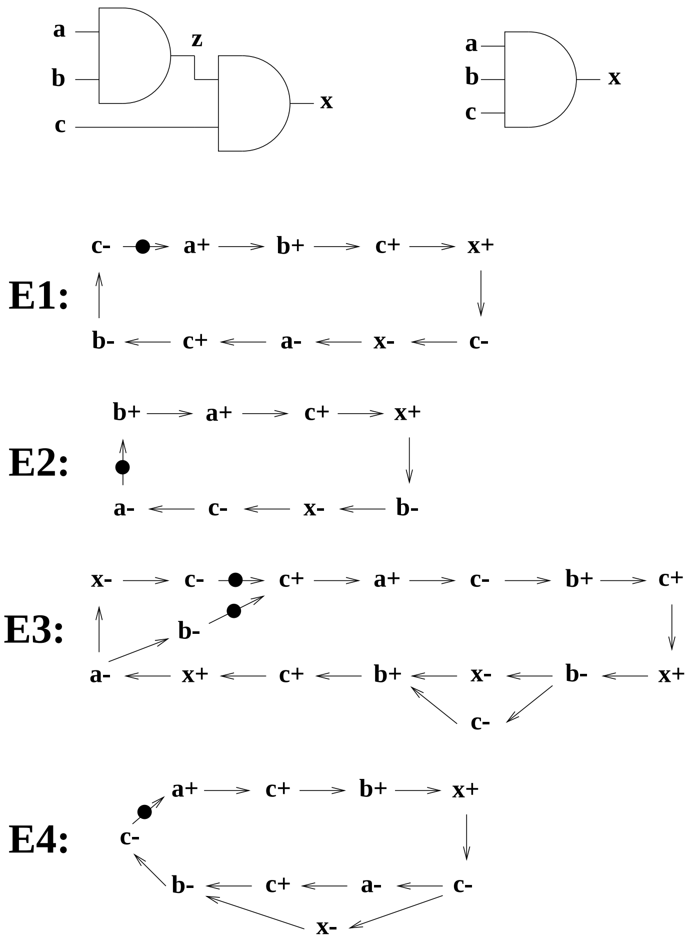

Figure 15:

DSr or DSw respectively. In a read cycle, a request to read is done through signal LDS. When the device has the data ready ($LDTACK$), the controller must open the transceiver to tranfer data to the bus (signal D). In the write cycle, data is first transferred to the device. Next, a request to write is done (LDS). Once the device acknowledges the reception of the data ($LDTACK$) the transceiver must be closed to isolate the device from the bus. Each transaction must be completed by a return-to-zero of all interface signals, seeking for a maximum parallelism between the bus and the device operations.

1. Give an STG describing the behavior of the VME bus controller.
2. Solve CSC using `petrify` (note: the option `-csc2` might be necessary).
3. Obtain a complex gate implementation of the circuit (option `-cg`).
4. Derive a netlist of gates by first decomposing the circuit into 2-input gates and then mapping onto a gate library (`-lit2 -tm`).

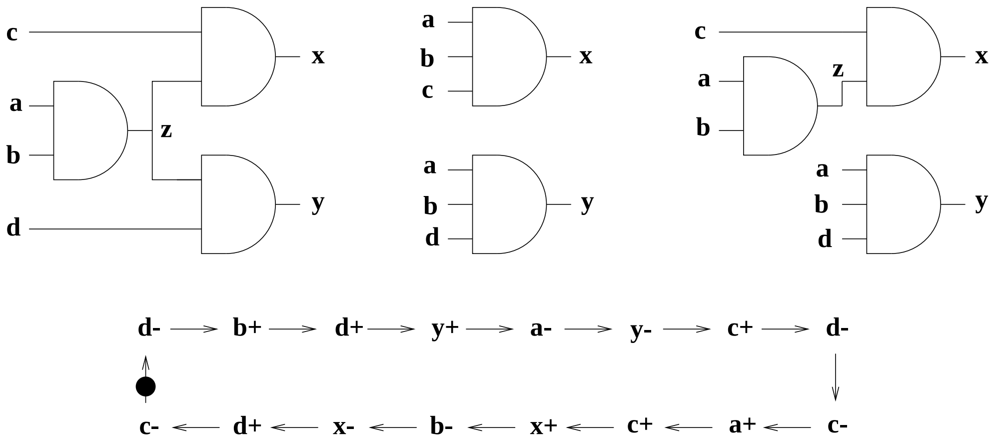

Figure 16:

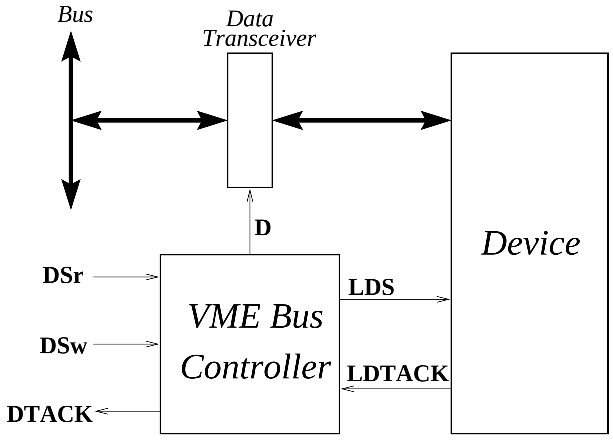

Figure 17: VME bus controller

9 MMU control

9.1 First example

Figure 18 shows a structure of a simple memory management unit taken from [1]. The memory allows single write or read operation at a time (a single port memory). The control circuit has four input signals: Ri, Ci, Ma, RW and three output signals: Mr, Ao, Co. Signal RW defines memory operation as follows: $RW = 1$ - read operation; $RW = 0$ - write operation. Mr is a request signal for a memory operation, Ma is a memory completion signal. A read-write operation is controlled by a program ROM: signal Ci is the request signal from the program ROM indicating that the value of RW is valid. Signal Co – the corresponding acknowledge signal to the program ROM.

The assumptions for the interface protocol are as follows:

For the read operation: the address will be ready ($Ri+$) at the input of the memory when Mr is high and the read data will be at the output of the memory when Ma is high.

For the write operation: If Ri goes high, then address (and data for write operation) are ready at the input of the memory when Mr is high. The operation is completed when Ma is high.

Figure 19,a,b,c show STGs for the read and write operations.

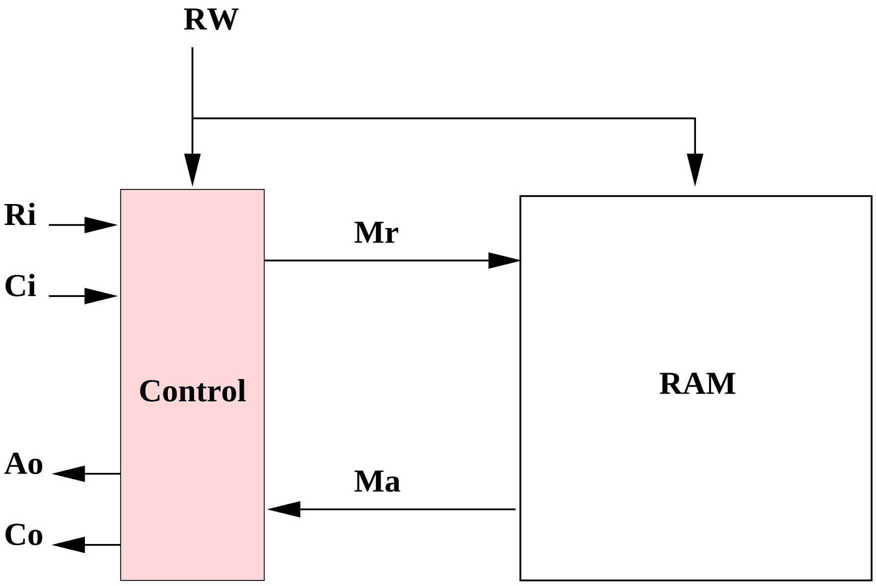

Figure 18: A MMU structure.

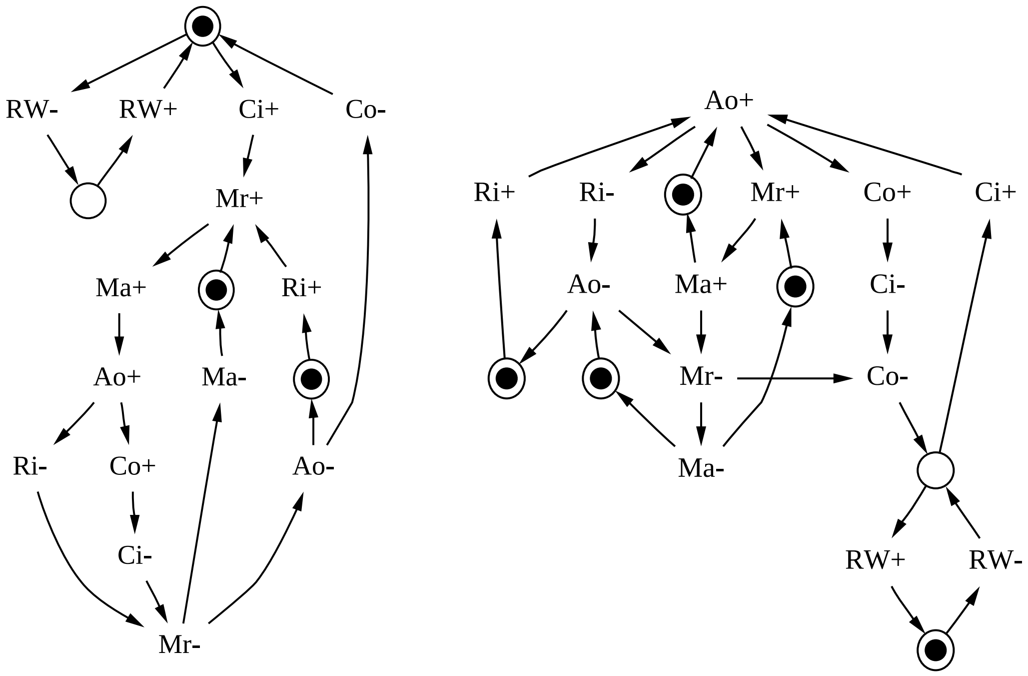

Figure 19: Specifications for the read and write operations for the MMU

1. Create an STG specification for the MMU-control structure. Consider separately Read and Write operations and then join the partial specifications together. **Hint:** To avoid

inconsistent behavior you can use complementary places corresponding to the zero and one values of signals Ri and Ma.

2. Assume slow environment. For this add `.mode TIMED` line into the header of the specification
3. Synthesize a netlist with `-cg` option.
4. Perform decomposition and technology mapping with `-lit2 -tm` options.
5. Do the same for the second specification.

9.2 Second example

Figure 20 shows another structure of a simple memory management unit taken from [2]. Figure 21 shows an STG specification of the device.

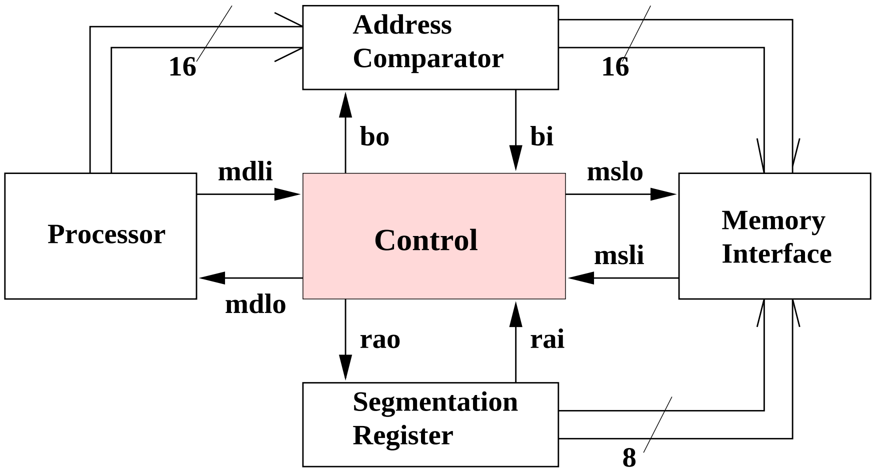

Figure 20: A MMU structure (second example).

1. Project specification on subsets of input-output signals
2. Synthesize a circuit and perform technology mapping assuming two different modes of the environment:
   - Slow environment – `.TIMED` mode in `petrify`
   - Fast environment – a default mode in `petrify`.

References

[1] Teresa H.-Y. Meng. *Synchronization Design for Digital Systems*. Kluwer Academic Publishers, 1991. Contributions by David Messerschmitt, Steven Nowick, and David Dill.

[2] Chris J. Myers and Teresa H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.

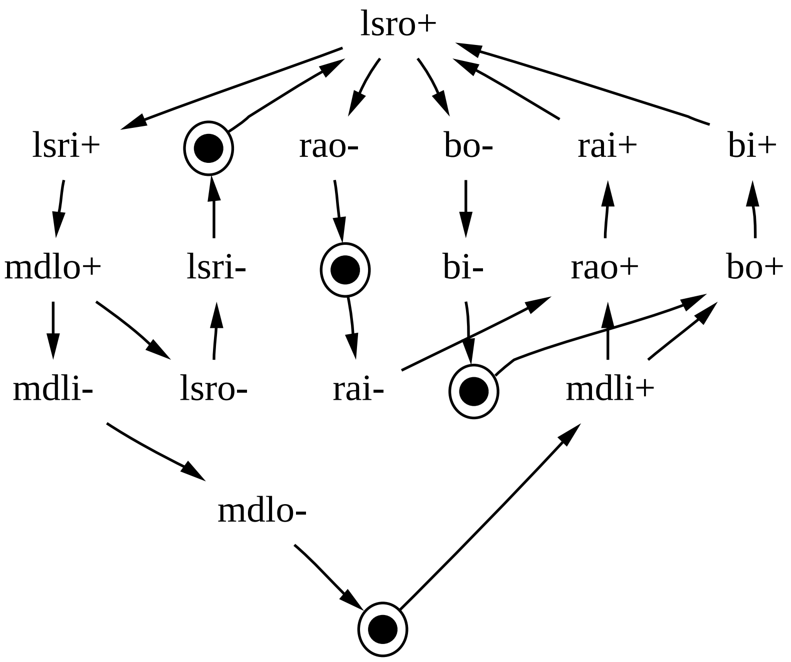

Figure 21: Specifications of a MMU (second version)