# Harmonizing ALF and DCL modeling for function and timing in OLA

**Wolfgang Roethig**

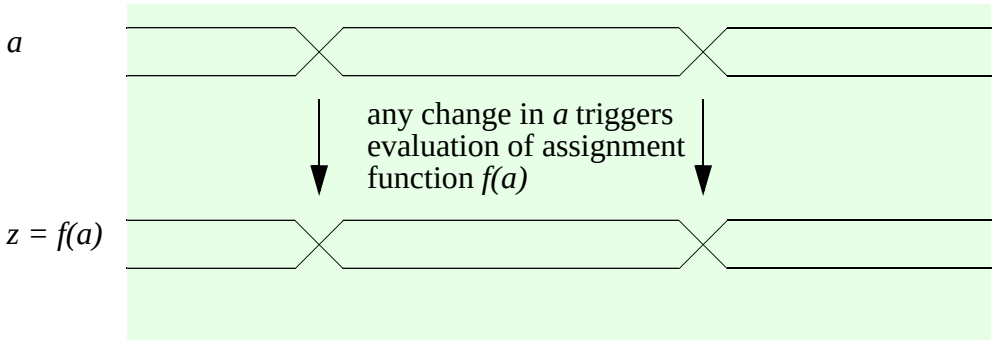
This note addresses timing and functional modeling issues, where harmonization between ALF and DCL is needed. It presents a conceptional view rather than a detailed specification, because we feel that agreement on the modeling principles is needed first.

## 1.0 Data-Control Flow Graph Modeling

### 1.1 Review of ALF functional modeling concept

#### 1.1.1 Combinational Logic

Combinational logic is modeled with continuous assignments, like in Verilog.




Examples of continuous assignments are shown below.

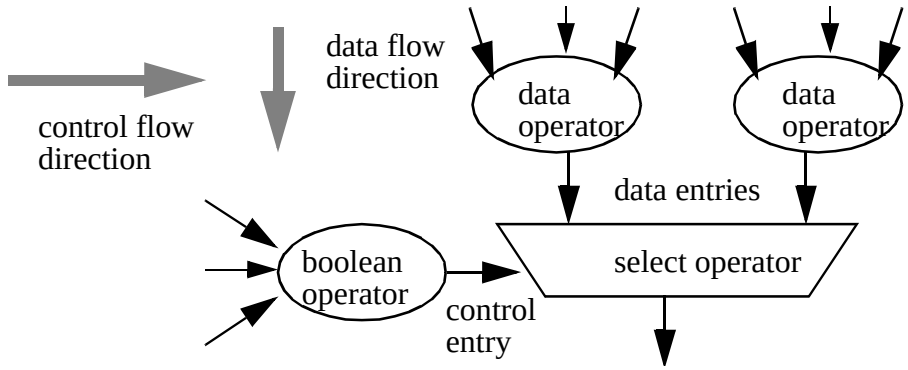
```
z = a && b || c;
//and-or cell

q = (!s0 && !s1) ? d0 : (s0 && !s1) ? d1 : s1 ? d2 : ‘bx ;
//3-way mux

p[15:0] = a[7:0] * b[7:0];
// 8x8 multiplier
```

The operations and operands can be represented in a pure data flow graph, with exception of the if-else operation (example of the 3-way mux), which has its control entry and its data entry.

The combinational if-then-else operation can be graphically modeled using a select operator, which has orthogonal entries for data and control, as illustrated below.



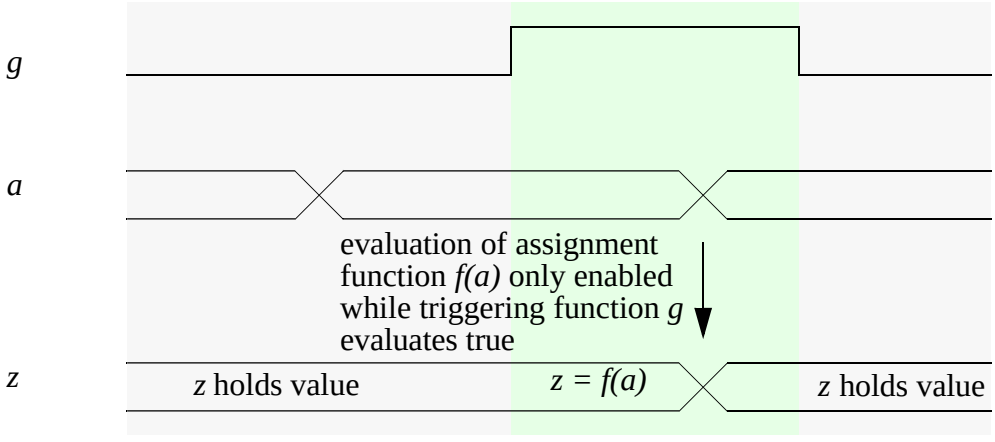

Cascaded if-elsif-elsif-elsif operations can be represented by cascaded select operators.

The values in data flow direction can be all values allowed by the standard, i.e. boolean, unsigned, Z, X, L, H, W... Therefore any operators which produce such values are allowed. AND, OR, Multiplication etc. are all allowed. Let us call them “data operators” for now. On the other hand, in the control flow only the operators which produce boolean values are allowed. Let us call them “boolean operators” for now. AND, OR are boolean operators. They are a subset of data operators.

“Boolean operators” can have general data input, but boolean output. Examples are unary reduction of unsigned numbers, equality comparison of non-boolean data.

### 1.1.2 Level-sensitive sequential logic

Level-sensitive sequential logic is modeled by conditional continuous assignments.




An example of conditional continuous assignments are shown below.

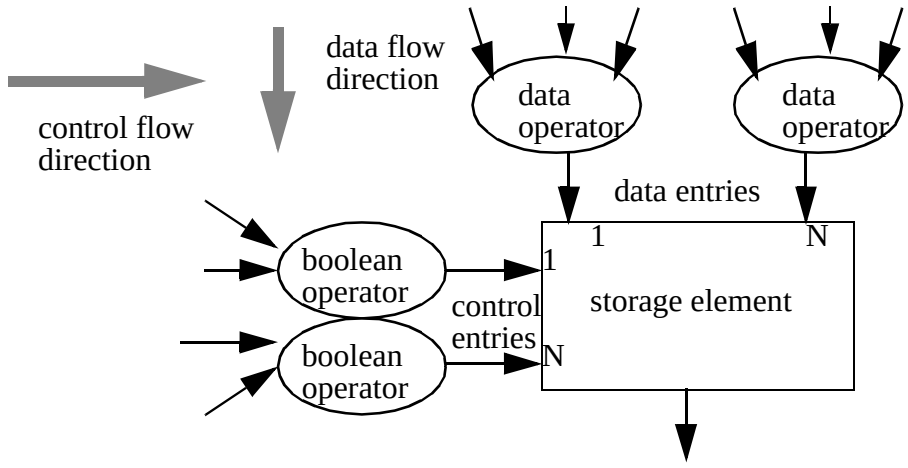
```
//latch with input scan mux and asynchronous clear

@ ( enb && !clear) { q = scan_enb ? test_in : data_in ;}
@ (clear) { q = 0 ; }
// modeled with concurrent “if”

@ (clear) { q = 0 ; }
: (enb) { q = scan_enb ? test_in : data_in ;}
// modeled with priority “if”, “else”
```

The sequential “if” is represented by “@”, the sequential “elsif/else” by “:” in ALF. If each of the “@” and “:” clauses are proven false, the latch holds its value. If concurrent non-exclusive “@” clauses are used, the latch will go to “X” in the case of simultaneously true “@” clauses.

Sequential logic can be represented in a graph using a storage element. The storage element has the same number of data entries as control entries, and each control entry is associated with exactly one data entry.




The control entries need to be ordered in priority. A flow with concurrently activated control entries can always be transformed in a flow with prioritized control entries by defining a conditional assignment to “X” for the “true” intersection of the concurrent clauses. An example is shown below.

```
@ (a) { q = q1; }
@ (b) { q = q2; }

//is equivalent to
@ (a & !b) { q = q1; }
: (!a & b) { q = q2; }
: (a & b) { q = ‘bX; }
```

It is debatable whether implicit conflict resolution (i.e. if the values of `q1` and `q2` are the same while `a&b` is true) should be provided as default or not. With implicit conflict resolution, the equivalence would be as follows.

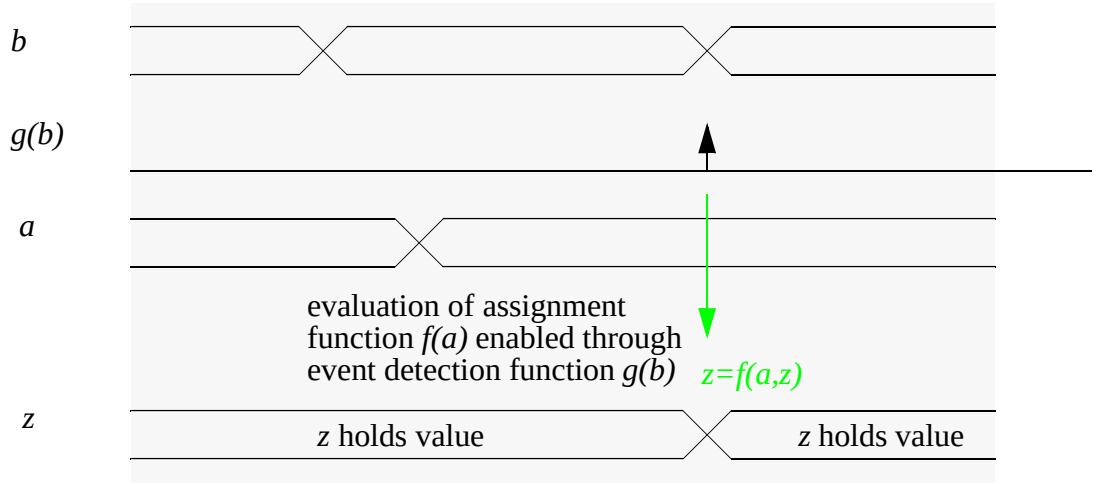
```
@ (a & !b) { q = q1; }
: (!a & b) { q = q2; }
: (a & b) { q = (q1==q2)? q1 : ‘bX; }
```

Note that the equivalence operator “==” is meant for absolute identity, like the Verilog “===”. For boolean equivalence, the exnor operator “~^” should be used. For example “‘bH==’b1“ is false while “‘bH~^’b1“ is true. Nevertheless, both “==” and “~^” are boolean operators, since they produce only true or false or unknown, i.e. “X”.

When the first N clauses with highest priority are proven false and the N+1 clause is unknown, the output will be “X” as well.

### 1.1.3 Edge-sensitive sequential logic

Edge-sensitive sequential logic is modeled in the same way as level-sensitive sequential logic, with the only difference, that “vector-expressions” can be used in the “@” and “:” clauses. A “vector-expression” in ALF describes an event or a sequence of events, which evaluates true at exactly the time when the event or sequence of events is proven detected.




An example is shown below.

```
//positive-edge triggered flipflop with asynchronous clear

@ ( 01 clk && !clear) { q = d ;}
@ (clear) { q = 0 ;}
```

“01” is the rising-edge operator. “10” is the falling edge operator. “?!” is the any-edge operator. Edges are not restricted to boolean values. They could be transitions between unsigned numbers, too.

Level and edge-sensitive clauses can be used in the same model. Assignments triggered by an edge-sensitive clause may contain the LHS variable also in the RHS of the assignment, since the assignment is instantaneous and not continuous. The “<=” operator which is

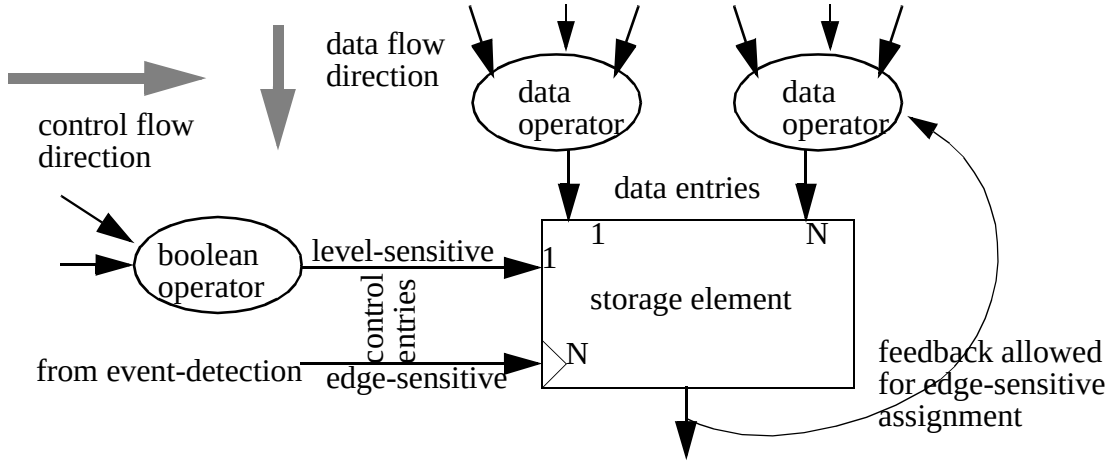
used in Verilog for non-blocking assignments, is not needed in ALF, since all assignments triggered by edge-sensitive clause are non-blocking, i.e. RHS is previous state, LHS is next state.

Examples:

```
@ (01 clk) { q = !q; qn = q; } //toggle flipflop
// qn has the opposite value of q, since it takes the value of previous q

@ (01 clk) { q = (j & k)? !q : (j & !k)? 'b1 : (!j & k)? 'b0 : q; }
// JK flipflop
```

In ALF, level-or edge sensitivity is indicated by the contents of the clause itself, which is determined by the parser. The graph representation, however, should have labels on the entries to indicate level-or edge sensitivity.




The behavior for unknown and conflicting clauses should be the same as for purely level-sensitive logic.

The "vector expressions" can be used for edge-sensitive control only, not for data assignment.

For example, the following assignment is meaningless:

```
q = (01 cp); // not valid ALF
```

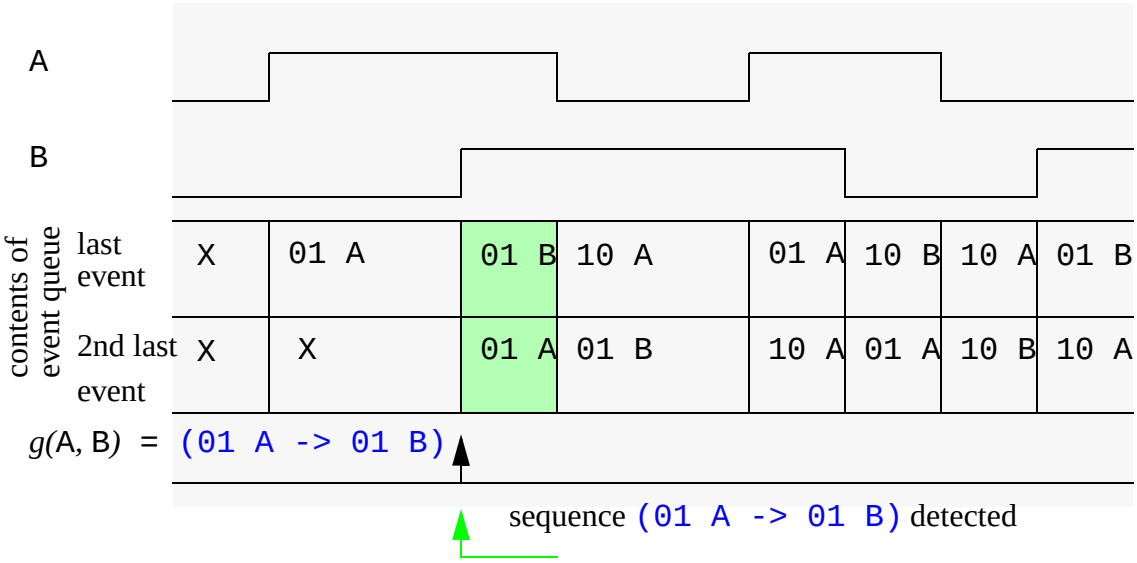
On the other hand, "vector expressions" are also used for ALF vectors, i.e. definition of timing and power arcs.

### 1.1.4 Graph representation of event sequences

A graph representation of a "vector expression" needs yet to be defined in a general way. An event detection mechanism consists of two components:

- an event queue, which can be modeled by a special storage element.
  Inputs to the event queue are all variables in the observation space for event detection.
- a matching logic which compares the contents of the event queue with a predefined event sequence, which can be modeled by a special boolean operator.

An example is shown below.

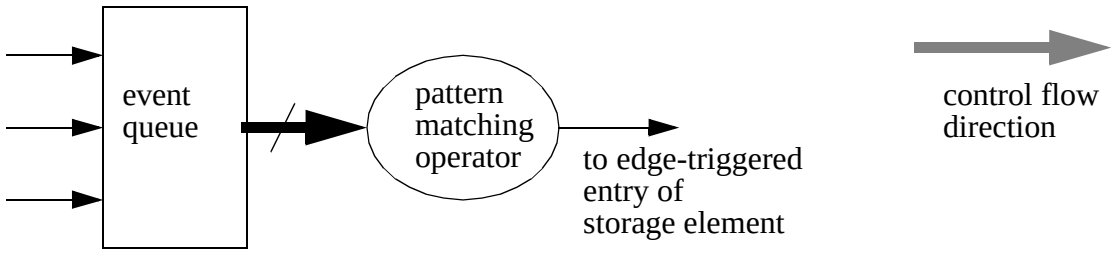



Each stage of the event queue stores both the edge and the variable for which the transition applies. Hence a general event queue needs the following number of bits in storage space:

Number of stages * ($\log_2$(number of variables) + 2*$\log_2$(number of possible values))

Events are ordered by the followed-by operator “->”.

We propose to represent the “vector expression” by an “event queue” and an “event matching operator”.




Conceptually, the values of a “vector expression” are binary dirac pulses rather than boolean states. Those dirac pulses can also be propagated through AND and OR operators. OR between vector expressions evaluates true, when either event is detected. AND between vector expressions evaluates true if the events are detected simultaneously. AND is also

defined between vector expression and boolean expression. It evaluates true, if the boolean expression is true at the time of event detection.

Hardware implementation of event queues can be found in asynchronous logic, for which the ALF functional description capability comes in quite handy. An example is the Mueller C-gate, which counts the events on its two inputs and toggles, when the two inputs change subsequently.

```
@ ( ?! A -> ?! B || ?! B -> ?! A) { Z = !Z; }
```

Some shorthand operators are also defined:

`(?! A <-> ?! B)` is equivalent to `(?! A -> ?! B || ?! B -> ?! A)`.

`(?! A <&> ?! B)` is equivalent to `(?! A -> ?! B || ?! B & ?! A || ?! B -> ?! A)`.

### 1.2 Representation of control-data flow graph in DCL

This is currently being addressed by John Beatty's proposal. Operators for continuous assignements (boolean values and general data values) are well-defined therein. Definitions for storage element, sequential control, and event queue need more work.

## 2.0 State-dependent Timing Arcs

### 2.1 Arc existence condition

DCL has a special function which returns the existence condition of a timing arc, ALF has not. However, an ALF vector is not always the same thing as a DCL timing arc. Only vectors with exactly two edges can be mapped directly into DCL timing arcs.

- Trivial case: ALF vector without logic condition, e.g. `(01 A -> 10 B)`
  There is no existence condition, the vector always exists.
- Almost trivial case: ALF vector with logic condition, e.g. `(01 A -> 10 B && C)`
  The existence condition of the vector isthe logic condition, e.g. `C`.
- General case: multiple ALF vectors with multiple logic conditions,
  `(01 A -> 10 B && C)(01 A -> 10 B && D)(01 A -> 10 B && E)`
  The existence condition is the OR of all logic conditions, e.g. `C || D || E`.

This definition goes along with the ALF philosophy: ALF library is the complete characterization database, therefore the existence condition of a timing arc must be the union of all logic conditions of the vectors which map into that timing arc.

### 2.2 Mutual exclusivity of logic conditions

The question of mutual exclusive logic conditions needs to be resolved on the library characterization level rather than on the ALF semantics level. ALF1.0 allows only specifica-

tion of exact delay values, not of a min-max or early-late range of delay values. Therefore, if there are overlapping vectors, the delay values must be the same, otherwise the data was not measured correctly.

If ALF would allow specification of delay data ranges, then the delay range for overlapping vectors would need to be defined as the the non-empty intersection of delay data ranges between the vectors.

## 2.3 Default for state-dependent arcs from ALF library

If the ALF set of vectors with conditions is not complete, how can a default be specified? Specifying a default in ALF means to make the set of vectors complete. The delay value is whatever the characterizer choses as default (again, specification of a range instead of fixed value would be appropriate and could be considered for ALF1.1). The condition is the one which complements the OR of all other conditions to the complete existence condition.

Example:
existence condition is `C`, the vector `(01 A -> 10 B && D)` is characterized.
Since `D` must be a subset of the existence condition `C`, `(D && !C)` is always false.
Default vector needs to be `(01 A -> 10 B && !D && C)`, since
`(!D && C) || D = (!D && C) || (D && C) || (D && !C) = C || (D && !C) = C.`

## 2.4 Arc value condition

Given a set of vectors with conditions, a subset may be proven false, while no single condition can be proven true. DCL currently cannot return the delays of the potentially true vectors, it will return the default. This is because DCL evaluates the conditions in a serial if-then-else manner and will return the delay for the first condition which is proven true or the default as a fall-true value.

The only way in DCL to get delay for a subset of conditions instead of a single condition is to predefine those subsets as modes, e.g. the set of all potentially true vectors in scan modes are those with “scan enable” active.

ALF just contains the set of vectors without defining or restricting the access to the delay values.

The capability of getting conditional delay values would almost eliminate the need for defining modes and would be more general and flexible. However, it would require some chirurgical changes in the DCL architecture. Therefore the cost-benefit tradeoff needs to be considered carefully.

# 3.0 Open issues

## 3.1 Mapping ALF vectors with more than two edges into DCL arcs

to be discussed in the meeting

## 3.2 Mapping ALF vectors with less than two edges into DCL arcs

to be discussed in the meeting