

Input Trigger lists allowed by Splitting and Oring
Compiled by Jim Linnemann
December 30, 2005

Definition of Splitting and Oring Functionality

Let me try to enumerate the trigger inputs which would be allowed by the XML we have chosen for combined splitting and oring. At each level of triggering, we require a non-null script, even if the script expresses ignoring this level of triggering for a particular L1 bit. This note is about what the entry and report interfaces of the TDB need to do, rather than what xmlgen and COOR need to do.

I define a notation more compact than XML:

+ followed by
[x] one or more
{y} 0 or more
a | b a or b is allowed

In this notation, the first example in the Appendix is:

L1 + [L2 + Split[L3]] + Or[L3]

Where

L1 is a L1 definition

L2 is a L2 definition

Split(L3) is a set of split L3 definitions, associated with (=tested upon passage of) a specific L2 definition, whether the number in the set of definitions is nonzero or not: either Split[L3] or Split{L3}

Or(L3) is a set of or'd L3 definitions, associated with (tested upon passage of) a specific L1 definition confirmed by passage of any of the L2 definitions associated with that L1 definition.

To be still more compact, we could just write S(L3) or O(L3) instead of writing out Split or Or:

L1 + [L2 + S[L3]] + O[L3]

This xml example was written this way for concreteness, to exhibit all possible cases, but I am trying to also represent what can be omitted without producing illegal (null) definitions. The xml was intended to represent something like

L1 + [L2 + S[L3]] + O{L3} | L1 + [L2 + S{L3}] + O[L3]
Splitting with optional Oring Oring with optional Splitting

The reason for the two separate forms in my description of Scott's xml is the **constraint preventing null L2 or L3 scripts**. A L3 definition must exist for each L2 definition, and a L2 definition for each L1 definition: an Or L3 block is optional if there is a L3 splitting block for each L2 definition, but an Or L3 block is required if L2 definitions with no L3 splitting block are allowed. This existence constraint applies even if the actual definition means "pass 100% of the time", and even if having all the L2 definitions in this form actually implies that the L2 hardware does not participate in the trigger decision. This constraint could be implemented in the entry interface either by selecting which form is being entered (call them Split or Or), or by imposing constraints before storing the L1 block of the form: if any L2 block has no L3 split definition, then the L1 block must have a L3 Or block. *As long as both Splitting and Oring are allowed (even as restrictively as making the choice once for an entire trigger list), both of the fundamental input forms, and the corresponding constraint, are required.*

I use "term" and "definition" interchangeably here, and maybe "bit" as well. Each refers to a set of conditions which must be fulfilled for a given level of triggering to be passed. In complex triggers, there may be many such at a given level of triggering.

Any of the L2 or L3 definitions could be reused. In particular, **a given L3 definition could be used in one context as a one of a Split(L3) and in another as one of a Or(L3)**. It is an implementation choice whether to allow this in full generality, or to force copying on entry, but my own prejudice is that any tagging is better associated with the entire block of definitions, rather than the specific definition, so I've shown it that way.

Next I show in this notation the escalating cases of possible trigger lists, starting from where we are now.

1. The present, No L2branching:

$L1 + L2 + S[L3]$ producing lists of the form of $L1$
 $L2$
 $m L3$

With $m > 0$

Here I introduced also the notation

$n X$ for $n \geq 0$ instances of X (n an integer)

The indentation among $L1$ $L2$ and $mL3$ above is vague: it is a compromise between a representation of xml nesting, and what might be presented in the trigger report where $L1$, $L2$, and $L3$ are arranged in columns with blanks in previous columns where the same lower-level trigger conditions obtain.

Note that the present $L3$ definitions are implicitly carrying a Split tag.

Presently it is sufficient to count $L1$ and $L3$ bits in the report interface, since the $L2$ bits are 1:1 with $L1$ bits; this definition would consume 1 $L1$ bit, 1 $L2$ bit, and m $L3$ bits.

2. Pure splitting: (escalation beyond where we are now, **using L2 branching**)

$L1 + [L2 + Split[L3]]$ producing lists of the form $L1$
 $n (L2$
 $m_i L3)$

With $n > 0$ and $m_i > 0$

That is: n $L2$ definitions associated with the $L1$ definition, each associated with m_i $L3$ definitions

Here $L2$ bits must also be counted.

This example would consume 1 $L1$ bit, n $L2$ bits and $\sum(m_i)$ $L3$ bits.

Here the entry interface does not need to recognize a new $L3$ definition type: it's still a split $L3$ definition.

But it does have to release the constraint $n=1$. **No L2Branching** is just pure splitting with the additional constraint of $n=1$.

3. Pure Oring:

$L1 + [L2] + Or[L3]$ producing lists of the form $L1$
 $n L2$
 $p L3$

Where $n > 0$ and $p > 0$

Again, all 3 bit types must be counted. This example would consume 1 $L1$ bit, n $L2$ bits, and p $L3$ bits.

A new feature for Oring is that the p $L3$ terms need some designation in a trigger report to indicate that they are associated with the Or of the n $L2$ terms listed above them.

At this point, we encounter a new type of L3 definition, which somehow the entry interface must tag as Oring, to distinguish splitting, which has been the implicit default for all list entries to date.

4. Mixed:

$L1 + [L2 + S\{L3\}] + O[L3]$ producing lists of the form $L1$

$$\begin{matrix} n(L2 \\ m_i L3) \\ p L3 \end{matrix}$$

Where $n > 0, p > 0$, but $m_i \geq 0$

Once again, the trigger report must distinguish the p L3 terms as being run on passage of any of the n L2 terms, independent of whether their directly-associated (Split) L3 terms passed. This example would consume 1 L1 bit, n L2 bits, and $\sum(m_i) + p$ L3 bits.

If $p=0$, I'd have required $m_i > 0$ and called this pure splitting. If $p > 0$ and $m_i > 0$, I'd still call this a mixed case.

Expressive power of Splitting and Oring

Now I make some further remarks on actual **representation of split/or triggers**:

1) There are things the present syntax forbids: it can't apply an Or group of L3 terms to only some L2 terms; you have to burn a L1 bit and separate the L2 terms into Or'd and not Or'd:

Not allowed: $L1 + [L2 + S[L3]] + [[L2' + S[L3']] + O[L3]]$

2) Any trigger can be written even with the present no-branch setup. It's just less compact, and chews up more resources, particularly precious L1 bits.

$$[L1+L2+S[L3]] = L1 + [L2 + S[L3]]$$

If pure splitting is written with no-branching, it uses n L1 and n L2 bits instead of 1 L1 and n L2 bits.

3) Pure Oring can be written with Splitting:

$$L1 + [L2] + O[L3X] = L1 + [L2 + S[L3X]]$$

Splitting consumes more L3 bits: $1 + n + p$ becomes $1 + n*p$

4) The reverse, expressing splitting as pure Oring, consumes L1 (but not L3) bits:

$$L1 + [L2 + S[L3i]] = [L1 + L2 + O[L3i]]$$

If splitting is written with Oring, $1 + n + \sum m_i$ bits increases to $n + n + \sum m_i$, where m_i is the number of bits in the i th L3 splitting group.

5) Mixed cases can also be written with pure splitting:

$$L1 + [L2 + S[L3i]] + O[L3y] = L1 + [L2 + S[L3i + L3y]]$$

Again, more L3 bits are consumed: $1 + n + \sum m_i + p$ becomes $1 + n + \sum m_i + n*p$ bits.

Appendix: Scott's notes on XML for Splitting and Oring

This is an example of the xml to be produced by xmlgen for splitting and oring:

```
<l1trigger name="l1bit"/>
  <l1termlist ... />
  <l2trigger name="l2first">
    <l2script ... />
    <l3trigger ... /> # runs on real l2 bit l2first
  </l2trigger>
  <l2trigger name="l2second">
    <l2script ... />
    <l3trigger ... /> # runs on real l2 bit l2second
  </l2trigger>
  <l3trigger ... /> # runs on pseudo l2 bit l1bit; i.e., l2first or l2second
</l1trigger>
```

The above, with extra L3 triggers inserted:

```
<l1trigger name="l1bit"/>
  <l1termlist ... />
  <l2trigger name="l2first">
    <l2script ... />
    <l3trigger ... /> # runs on real l2 bit l2first
    <l3trigger ... /> # runs on real l2 bit l2first
  </l2trigger>
  <l2trigger name="l2second">
    <l2script ... />
    <l3trigger ... /> # runs on real l2 bit l2second
    <l3trigger ... /> # runs on real l2 bit l2second
    <l3trigger ... /> # runs on real l2 bit l2second
  </l2trigger>
  <l3trigger ... /> # runs on pseudo l2 bit l1bit; i.e., l2first or l2second
  <l3trigger ... /> # runs on pseudo l2 bit l1bit; i.e., l2first or l2second
  <l3trigger ... /> # runs on pseudo l2 bit l1bit; i.e., l2first or l2second
  <l3trigger ... /> # runs on pseudo l2 bit l1bit; i.e., l2first or l2second
</l1trigger>
```

Commentary:

This xml indicates an 'or' by moving the associated l3trigger elements outside of the l2trigger element, to the end of the l1trigger element (rather than indicating an 'or' by putting multiple l2script elements inside one l2trigger element.).

Such an alternate structure does not properly handle cases of mixing splitting and or'ing on a single l1 bit.

For example, you have 'l1bit1' with two l2 bits defined: 'l2bit1', 'l2bit2'.

You want to run the l3 triggers 'l3bit1' if 'l2bit1' fires, and run 'l3bit2' if either 'l2bit1' or 'l2bit2' fires.

```
<l1trigger name="l1bit1">
```

```
</11termlist ... />  
<12trigger name="12bit1">  
  <12script ... />  
  <13trigger name="13bit1"/>  
</12trigger>  
<12trigger name="12bit2">  
  <12script ... />  
</12trigger>  
<13trigger name="13bit2"/>  
</11trigger>
```

I don't think it's possible to write this in the alternate's structure, at least not without duplicating 12 bits.