Trigger module

This module is built to collect all the triggers coming in (physics-, calibration- or cpu-trigger) and to combine them, measure their duration and determine their type in order to calculate the proper time to be read out.

```
15  ENTITY trigger_handling IS
16     GENERIC( n_triggers : INTEGER := 3);
17     PORT( clk : IN STD_LOGIC;
18           reset : IN STD_LOGIC;
19           physics_trigger_in : IN STD_LOGIC_VECTOR (n_triggers-2 DOWNTO 0);
20           cal_trigger_in : IN STD_LOGIC;
21           cpu_trigger_in : IN STD_LOGIC;
22           pre_trigger_length : IN VECTOR_ARRAY(0 TO n_triggers);
23           readout_length : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
24           trigger_processed : OUT STD_LOGIC;
25           full_triggers : BUFFER STD_LOGIC_VECTOR(n_triggers DOWNTO 0)
26     );
27  END trigger_handling;
```

The input signals are:

- *clk* - a 100 MHz clock.
- *Reset* - the connected reset.
- *physics_trigger_in* - a vector of physics triggers, variable in length (number of triggers). This will be converted into single inputs (if your proposal is right this will be three), which are combined to this vector.
- *cal_trigger_in* - a trigger sent for calibration measurements.
- *cpu_trigger_in* - a periodic trigger, coming from the cpu can be connected here.
- *Pre_trigger_length* - The information about the requested pre-trigger-samples for the readout of each trigger. This will be included in the configuration.

The output signals are:

- *Readout_length* - this is a 9 bit vector, displaying the length of all combined triggers, arriving in one trigger block plus the number of pretrigger samples for the trigger with the highest priority.
- *Trigger_processed* - this is a one cycle event. It appears, when the trigger block ends. On its rising edge the other outputs will definitely hold its final state determined for the last trigger block.
- *full_triggers* - A vector, saving all triggers which have been high coincidently in the last trigger block

Inside the module, the incoming trigger lines are all combined to a trigger vector (*trigger_in*). As long as this vector is not equal zero, one or more triggers are high. A STD_LOGIC SIGNAL *trigger_combined* will be high in this case.

```
46     trigger_in <= physics_trigger_in & cal_trigger_in & cpu_trigger_in;
47     trigger_combined <= '1' WHEN (trigger_in /= zero) ELSE
48                         '0' WHEN (trigger_in = zero);
49
```

As long as this signal is high, a counter will be raised each clock cycle (implemented in the process below). It is converted into a vector (*trigger_length*) which will be added to the chosen *pre_trigger_length*.

```
84    PROCESS(reset, clk, trigger_combined)
85      VARIABLE inhibit : STD_LOGIC;
86    BEGIN
87      IF reset = '1' THEN
88         trigger_length <= (OTHERS => '0' );
89         count <= 0;
90         inhibit := '1';
91         trigger_processed <='0';
92       ELSIF clk'EVENT AND clk = '1' THEN
93         IF trigger_combined = '1' THEN
94             trigger_processed <='0';
95             inhibit :='0';
96             count <= count + 1;
97        ELSIF trigger_combined = '0' AND inhibit ='0' THEN
98           trigger_length <= conv_std_logic_vector(count, 9);
99           trigger_processed <='1';
100          inhibit :='1';
101          count <= 0;
102       ELSIF inhibit = '1' THEN
103           trigger_processed <='0';
104      END IF;
105    END IF;
106   END PROCESS;
```

In the same process the *trigger_processed* SIGNAL is set high at the next rising edge of the clock after the last trigger signal switches low (this is done by means of the *inhibit* VARIABLE).

```
57    PROCESS(reset, clk, trigger_combined, trigger_in)
58     VARIABLE inhibit1 : STD_LOGIC;
59     VARIABLE full_vector : STD_LOGIC_VECTOR(n_triggers DOWNTO 0);
60    BEGIN
61     IF reset = '1' THEN
62       full_triggers <= (OTHERS => '0');
63       full_vector := (OTHERS => '0');
64       inhibit1 := '1';
65     ELSIF (clk'EVENT AND clk = '1') THEN
66       IF(trigger_combined = '1') THEN
67         IF(inhibit1 = '1') THEN
68           full_vector := (OTHERS => '0');
69         END IF;
70         inhibit1 := '0';
71         FOR i IN trigger_in'RANGE LOOP
72           CASE trigger_in(i) IS
73               WHEN '0' => full_vector(i) := full_vector(i);
74               WHEN OTHERS => full_vector(i) := '1';
75           END CASE;
76         END LOOP;
77       ELSIF (trigger_combined = '0') THEN
78         full_triggers <= full_vector;
79         inhibit1 :='1';
80       END IF;
81     END IF;
82    END PROCESS;
```

In parallel to this all triggers within one trigger block are saved in the vector *full_triggers* (see above). The trigger with the highest priority will occupy the highest significant bit. This bit is detected by a priority encoder (see below; the result is saved as the INTEGER SIGNAL *priority_trigger*).

```
110     PROCESS(full_triggers)  --Priority encoder
111        VARIABLE counter : INTEGER RANGE -1 TO n_triggers;
112     BEGIN
113  --    IF reset = '1' THEN
114  --       priority_trigger <= 0;
115  --    ELSE
116        counter := n_triggers;
117        FOR i IN full_triggers'RANGE LOOP
118           CASE full_triggers(i) IS
119              WHEN '0' => counter := counter - 1;
120              WHEN OTHERS => EXIT;
121           END CASE;
122           CASE counter IS
123              WHEN -1 => counter := counter + 1;
124              WHEN OTHERS => counter := counter;
125           END CASE;
126        END LOOP;
127           priority_trigger <= counter;
128  --    END IF;
129     END PROCESS;
130
```

To the *pre_trigger_length* for the trigger of highest priority a 3 bit vector “100” is attached behind. This vector is then added to the counted *trigger_length* (for security it is made sure that the SIGNAL *priority_trigger* has a positive value).

```
54    readout_length <= (pre_trigger_length(priority_trigger) & "10000") + trigger_length WHEN priority_trigger > -1 ELSE
55                      "000000000";
```

The utilization summary on a SPARTAN 3 FPGA:

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	29	1,536	1%	
Number of 4 input LUTs	38	1,536	2%	
Number of occupied Slices	33	768	4%	
Number of Slices containing only related logic	33	33	100%	
Number of Slices containing unrelated logic	0	33	0%	
Total Number of 4 input LUTs	38	1,536	2%	
Number of bonded IOBs	32	124	25%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	2.88			

A 2 µs simulation of the whole module can be seen below. The output signals are marked.