





# EmLogic

## The Good, The Bad, and the Ugly

UiO, 7 April 2022

- Design Centre for Embedded Systems and FPGA
- 1<sup>st</sup> of January 2021. **Extreme ramp up**
  - January 2021: 1 person
  - March 2022 : → 21 persons (SW:6, HW:3, FPGA:10, DSP:1) - **And still growing fast...**
  - Located in Asker, Trondheim (August 2022), Oslo (2022/23), ...
- Continues the legacy from  **bitvis**
  - All previous Bitvis technical managers are now in EmLogic
- Verification IP and Methodology provider 
- Course provider within FPGA Design and Verification
  - Accelerating FPGA Design (Architecture, Clocking, Timing, Coding, Quality, Design for Reuse, ...)
  - Advanced VHDL Verification – Made simple (Modern efficient verification using UVVM)
- Part of TechSeed
  - Sister company TechSeed Edge for IoT established in Jan. 2022, - More to come...

# We are recruiting...

- A Design Centre yields a major competence and experience kick-start
- We are 10 FPGA designers – (5 Principals and 2 Seniors)
  - The most experienced Design centre in Norway on FPGA
  - Major growth and experience also on ESW, HW and DSP
- We have experienced designers who want to help novice designers
  - Mentors and Sparring partners
- We run all fresh designers through
  - our FPGA design course & our FPGA verification course
  - real cases on how to specify, architect, code, synthesize and verify an FPGA (Who else does this...)
- EmLogic offers significant ownership to all employees
  - More and at a lower cost than anybody else in our business

# Main design problem areas

## Bad & Ugly code:

- Micro architecture
- HDL coding style
- Naming

## Bad & Ugly design

- Architecture
- Digital design issues
- Clock domain crossing
- Timing closure

## → Seriously affects:

Quality, Schedule and Cost,  
Frequency, Power and Area,  
Readability, Modifiability and Risk

# Bad names - Abbreviations

- Non-standard abbreviations
- Extremely common

Abbreviations only ok when:

- clearly defined or
- obvious to anyone

```
-- Address FIFO is almost empty
```

```
afae
```

```
af_ae
```

```
afifo_ae
```

```
addr_fifo_ae
```

```
addr_fifo_almost_empty
```

```
-- Block enable
```

```
blen
```

```
bl_ena
```

```
block_ena
```

# Why is showing bad naming important?

- Awareness, Awareness, Awareness
- Hopefully helps to show that naming is important
  - Or in fact **VERY** important
- To really understand that people think differently
  - And thus YOU should code for that
- To understand that seemingly good names are not always that good
- To get some hints on improvements

**NOTE: These are real examples from the industry**

# Bad names - Variants of a signal

- Signals that have been slightly modified
- Extremely common problem
- Readability drastically reduced

```
frame_bit_counter <= ..... (actual counter)
frm_bit_counter   <= a snapshot when address
                   field completed
frame_bit_cnt     <= at end of previous frame
frm_bit_cnt_tmp   <= value held for bit period
                   - in case of jitter
frame_bit_count   <= expected number of bits
```

Extreme case,  
- but lots of cases with 2-3 variants.  
They get mixed up all the time....

# Bad names - N dimensions

- Signal array with N dimensions  
e.g.
  - line number (A/B)
  - channel number (1-6)
  - bit number (N)
  - delay number (0-3)
  
- Extremely confusing

Typically lots of signal and variable variants  
 - Special naming - hopefully structured  
 - Names refer to different dimensions  
 They get mixed up all the time....

```
data(1,3,2)  -- Line A, Ch 3, Bit 2
data_a(3,2) -- Line A, Ch 3, Bit 2
dout(1,3)   -- Line A, Ch 3,
din_a_3(2)  -- Line A, Ch 3, Delay 2
```

**How would you reference the following signals:**

1. **<my\_sig>: line a, channel 3, bit 4?**
2. **The same signal vector two clock cycles later?**



# Bad names - N dimensions

- Signal array with N dimensions

e.g.

- line "number" (A/B)
- channel number (1-6)
- bit number (N)
- delay number (0-3)

- Extremely confusing

- Use Conventions for delay

- Use Enumerated

- And arrays of enumerated

Typically lots of signal and variable variants

- Special naming - hopefully structured
- Names refer to different dimensions

They get mixed up all the time....

```
data(1,3,2)    -- Line A, Ch 3, Bit 2
data_a(3,2)   -- Line A, Ch 3, Bit 2
dout(1,3)     -- Line A, Ch 3,
din_a_3(2)    -- Line A, Ch 3, Delay 2
```

**How would you reference the following signals:**

1. **<my\_sig>: line a, channel 3, bit 4?**
2. **The same signal vector two clock cycles later?**

```
data(A,CH3,4)  -- Line A, Ch 3, Bit 4
data_d2(A,CH3) -- Line A, Ch 3, Delay 2
```

# Bad names - Logical mismatch

- Name clearly indicates a function, but does something slightly different.
  - 'read'                    when trigger read is intended.  
                              E.g. one FSM triggering another
  - 'crc'                     when 'crc\_error' is intended
  - '\*\_mask'                when enable is intended (e.g. for interrupts)
  
- Functionality changed, but name is kept
  - 'clk32'                when frequency actually changed to 16 MHz

# Bad names - Unknown number unit

- Name clearly indicates a function or number, but not specific enough  
E.g. number of bits in a frame
  - `frame_size`            Bits, bytes, words, ....?  
Often varying unit in same design.

```
-- Frame size in number of bits
```

```
frame_size
```

```
frame_size_bits
```

```
frame_bits
```

```
num_bits_frame
```

# Bad names - Different understanding

- Name is obvious to the designer, but...

```
-- Number of data bytes in payload
```

```
data_in_payload
```

```
data_bytes_in_payload
```

```
bytes_in_payload
```

```
num_bytes_in_payload
```

What else could it mean?

# Bad name relations

```
if frame_bit_num = num_bit_frame then
```

```
    if frame_bit_num = num_bits_frame then
```

```
        if frame_bit_cnt = C_NUM_BITS_FRAME then
```

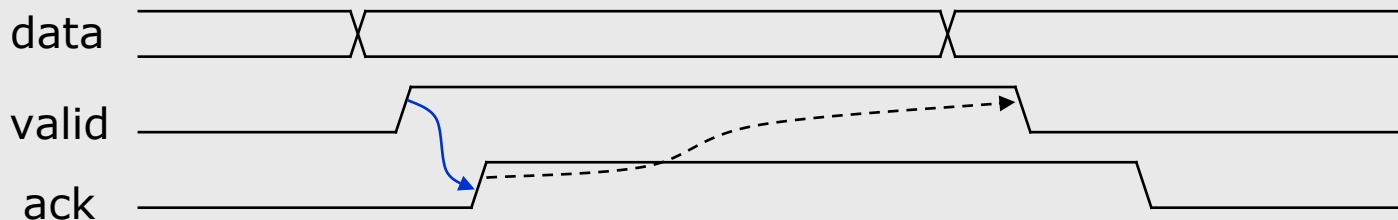
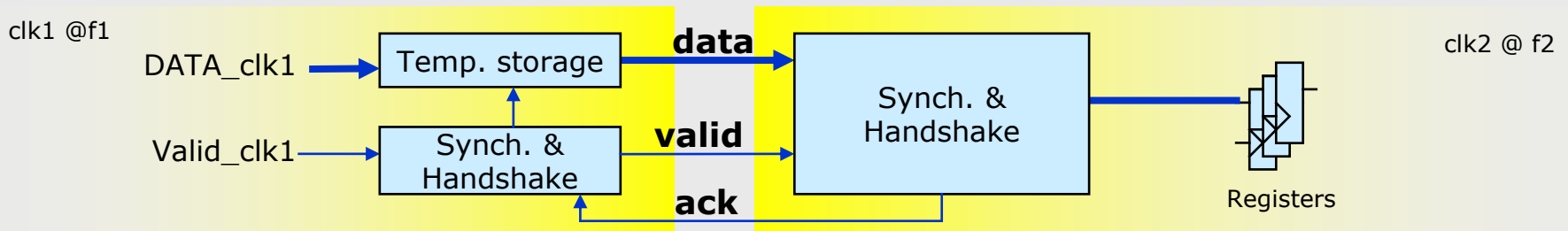
```
            if frame_bit_cnt = num_bits_frame then
```

# Bad names - Toggle-signals

- Signal is "valid" on toggle
- Must find alternative name

**t\_valid**  
**toggle\_valid**

**valid\_on\_toggle**



# Bad names - Number suffix

- Differentiate between variants of number suffixes

```
variables:
```

```
status
```

```
status1 or status_1
```

```
status2 or status_2
```

Typically used for lots of ad-hoc "conventions"

- a) For 3 different types of status?
- b) For status + pipeline stage 1 and 2
- c) For status + synchronized once and twice
- d) For status and slightly modified versions (e.g. masked, enabled, snapshot, etc....)

a) creativity : Zero points  
Find better names

b) Use fixed conventions  
e.g. **status\_p?**

c) Use fixed conventions  
e.g. **status\_s?**

d) Terrible practice  
Find better names

# Numbers - from 0 or 1

- Often confusing whether number N is the N<sup>th</sup> or (N+1)<sup>th</sup> occurrence.  
E.g. whether 13 is the 13<sup>th</sup> or 14<sup>th</sup> occurrence.

E.g.

- bit\_cnt
- bit\_number
- bit\_index
- bit\_pointer

E.g.

- char\_cnt
- char\_number
- char\_index

Do you **know** for your code?

Always?

Other designers' code?

Bits vs char vs anything?

**Sometimes obvious - Often not**

channels?

events?

strings?

node?

- Conventions (e.g. bit\_0idx)?
- Special names (e.g. idx vs cnt)
- Comment on non-obvious



# Constants for obvious values

```
constant C_ENABLE   : std_logic := '1';
constant C_DISABLE : std_logic := '0';
.....

if (.....) then
    bit_cnt_ena <= C_DISABLE;
```

What's the point?

```
constant C_ENABLE   : std_logic := '1';
constant C_DISABLE : std_logic := '0';
.....

my_function(param1, C_DISABLE, param3);
```

Sometimes it improves readability

```
type *** is (ENABLE, DISABLE); :.....

my_function(param1, DISABLE, param3);
```

or use enumerated

# Numeric constants for non-numeric objects

```
-- (0:Cyclone, 1:Spartan, 2: Igloo  
constant C_DEVICE : natural := 2;  
.....  
if C_DEVICE = 2 then....
```

```
type t_device is (cyclone, spartan, igloo);  
constant C_DEVICE : t_device := igloo;  
.....  
if C_DEVICE = igloo then....
```

# Simplify complex expressions

Example: Clear an interrupt on writing '1'  
(Inside a clocked process)

```
if 'CPU writes to irq-reg' then
  irq := NOT data_in AND irq;
end if;
```

**Extremely simple.  
Still -  
need to stop and think**

```
if 'CPU writes to irq-reg' then
  if (data_in = '1') then
    irq := '0';
  end if;
end if;
```

**More readable even  
when more lines**

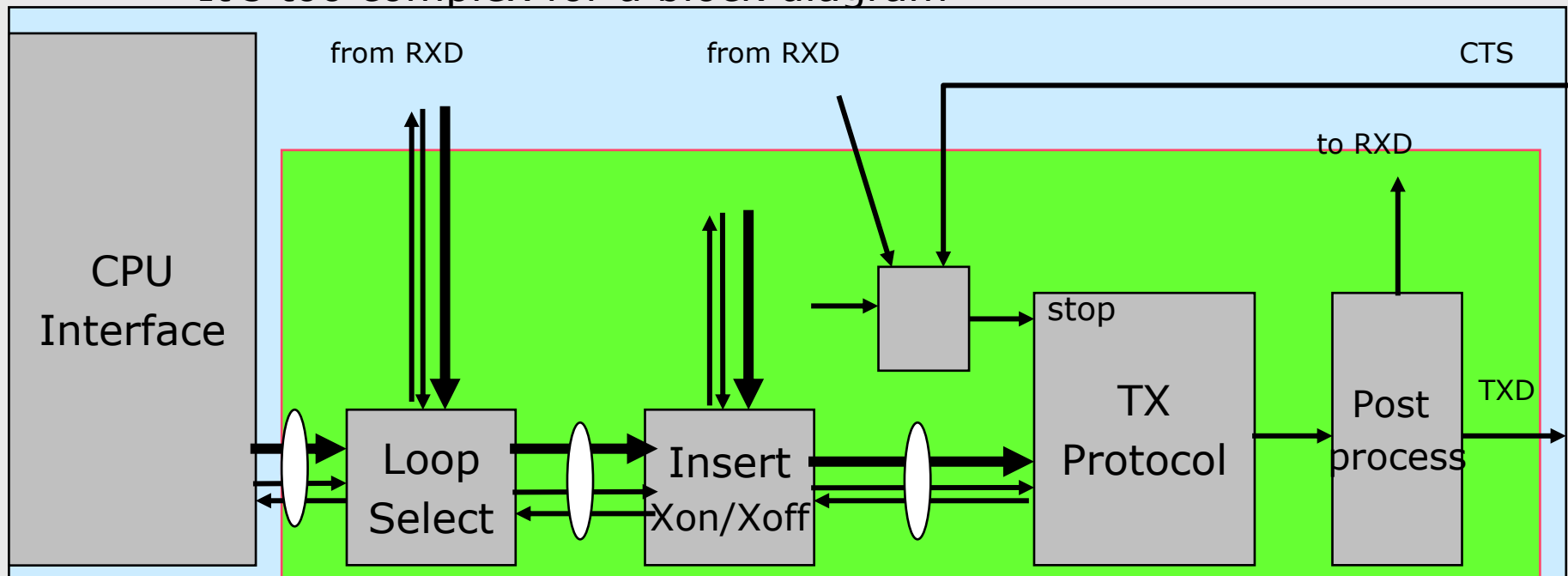
```
if 'CPU writes to irq-reg' and (data_in = '1') then
  irq := '0';
end if;
```

# Standard names

- Use standard names for repeated naming issues:
  - addr
  - cnt
  - ctrl
  - ack
  - num
  - rd/wr or re/we or rena/wena
  - rst
  - idx
  - ptr
  - etc....

# Main Micro architecture issues

- Block diagrams drawn only down to the module level
  - "More is a waste of time"
  - "It's too complex for a block diagram"



Strive for **Maximum cohesion & Minimum coupling**

# Code commenting

- Comment, comment, comment ! (why/what)
  - On complex code lines
  - On branches/blocks/processes
  - On any special solutions
  - On required pragmas (or synthesis constraints)
- Comment even more. (It is “never” too much) BUT...
  - Make relevant comments (why/what, behaviour)
  - Comment while coding – not afterwards

# Compact code is often not efficient

- Avoid complex concurrent expressions
- Do not combine operations to save code lines
- A sequential process with structured multiple if-statements is normally better than multiple related concurrent statements.
  - Even if 10 lines rather than 3.
- Typing your code is the least time consuming task in the FPGA development.
  - A minor part of the design/implementation time.

# Compact code is often not efficient

Is this correct ?

$$12/18 - 1/4 = 11/24$$

No !!!  $\rightarrow 5/12$

Which is the faster to write ?

$$12/18 - 1/4 = 5/12$$

or

$$12/18 - 1/4 = 8/12 - 3/12 = 5/12$$

Which is the faster to verify and debug ?

How many times do you write it ?

And how many times does someone read it ?



# The code writing paradox

There seems to be a significant focus on fast code writing.  
Yet - code readability and understanding  
is **far** more important...

Concepts, Design, Architecture, Development, Deployment, Maintenance, Support



# The code writing paradox

There seems to be a significant focus on fast code writing.  
Yet - code readability and understanding  
is **far** more important...

Now - what could we do?

More structure



Writing



Better partitioning/structuring

Writing for readability

Reading &  
Understanding



**TOTAL:**



# Conclusions

- Readability and modifiability/maintenance is important
  - Speeding up the code writing it self is NOT important
    - Abbreviate only when always immediately understood
    - Invest time in finding good object names
    - Long names are ok. Enumeration is fine
    - Never use a name that doesn't reflect functionality
    - Make User defined types, but do not overdo hierarchical types
    - Divide and conquer at all stages
    - Assure a good structure all the way down
    - **Prioritise the reader at all times**



# EmLogic

## **The Good, The Bad and the Ugly**

Guest lecture

et@emlogic.no

# Feel free to connect

I publish quite a bit on LinkedIn, so feel free to connect:

<https://www.linkedin.com/in/espentallaksen/>

Check out articles on us in Magasinet Elektronikk (Norwegian only):

<http://viewer.zmags.com/publication/8458c4ab#/8458c4ab/10>

<http://viewer.zmags.com/publication/8db6a978#/8db6a978/24>

<http://viewer.zmags.com/publication/b503f5af#/b503f5af/16>

<https://emlogic.no/>

<https://emlogic.no/prehistory-short/>