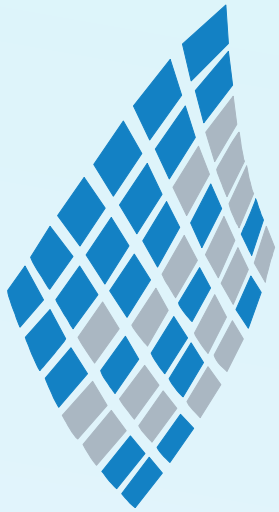
bitvis

1100010110100111101001110110110100111110011

Backgrounder for: 'FPGA Development Best Practices'



Your partner for SW and FPGA (& ASIC)

Today's FPGA opportunities

- One device does "everything" (or lots…)
  - Exploding functionality
    - Functionality conglomerate in a single chip
    - Lots of very complex functionality
- Extreme Flexibility
  - Use of complex external IP (soft or hard)
    - Often very complex parameterization
    - Often very complex interfaces
    - Sometimes need to modify
    - Often "black box"
  - "Easy" to make dedicated modules
  - Easy to add more functionality, features, flexibility
- Massively Parallel

Secondary effects

- Interfaces and functionality are buried
- More than just design for functionality
  - For simulation, lab-test, power,
- Timing is far more complex
  - Interactivity
  - Lots of complex clock/data relations
- Flexibility leads to very complex designs
- Parallelism is hard to coordinate

How do we handle these challenges?

- We structure our design, but not sufficiently
- We go wild with our flexibility
- We exploit the extreme parallelism
  - and make a “mega multi thread” system
- We integrate, test and debug a lot
  - Wasting a lot of time
  - Patching and leaving un unstable product
- We do analyse timing
  - But far from good enough
- We know about clocking problems
  - But very few know how to handle them
- We do document, but very far from sufficient
- We do simulate, but we don’t like it - and it shows…
- **We are not really structured at all!** (…with one single exception…)

Resulting **product delays** - examples from Scandinavian companies

- 2 months delay due to sporadic errors in the system test (Debug only. Fix took 1 day)
- 3-4 months delay due to unstable complex interface (debug, total redesign, detailed test)
- 1 year delay after product was "actually ready" (patching, testing, patching, testing, patching, testing, …)
- 2-4 months delay due to lack of implementation coordination
- 3-6 months delay due to bug after bug after bug
- 6 months delay due to terrible state machines (HW sequencer)
- Standard problems
  - 1-3 months delay due to HW/SW/FPGA integration problems
  - 1-6 months delay due to quality of design and documentation

Resulting **product deficiencies** - examples from Scandinavian companies

- Communication switch with lots of bit errors
  - Did not appear in the lab (or was ignored)

  ➔ Lower throughput. Unhappy customer.
- Industrial system increasingly failing after a few years
  - Multiple functional errors and timing and clocking errors

  ➔ More than a year to debug

  ➔ Expensive down time. Unhappy customer.
- Customer's application SW not working
  - Starting fine, but problems appearing with new SW
  - Did not happen in the lab. Not tested + New sequences.

  ➔Delayed customer's product
- (Telecom volume product with sporadic errors)
  - Deadlock in hardware

  ➔ Had to reset device. Lots of unhappy customers.

Some FPGA key issues

- There are many excellent FPGA designers, but far too many…
  - … have insufficient FPGA experience
  - … have insufficient FPGA or design knowledge
- Most FPGA-projects …
  - … have insufficient development flow structure
  - … have insufficient team experience
  - … make terrible testbenches – wasting a lot of time
- Proper FPGA development
  - Requires proper knowledge of **HW, FPGA-technology and Digital design**
  - Requires **Experience** and **Structure** for **Design and Verification**
  - Requires proper development **support**
    - Conventions, Templates, Checklists, Cookbooks, etc
  - Requires **Quality** Assurance **at the right level**
  - Requires **sufficient time** to structure Design and Verification
  - Requires a proper **Methodology** and someone to **follow up**

Main Problem Areas

- Most serious "self inflicted" quality and efficiency problems arise from four main problem areas
  - Timing (mainly on external I/O)
    - Some problems also due to difficult "timing closure"
  - Clocking (mainly internal clock domain crossing)
  - Design complexity – with lack of structure
  - Design complexity – with lack of simulation
    - or Testbenches with unstructured simulation

Examples on FPGA-bugs in final product
1. Timing on external I/O

- Timing diagrams should be made for all I/O
  - Or just comment on why it is not needed.
    → This may be time consuming, but if so….?
  - FPGA I/O must be constrained accordingly
    - But according to what…?
  - FPGAs are seldom properly constrained
- I/O timing paths often have negative slack
  - If you are lucky : Your design will fail sufficiently to alarm you
  - If you are undeservedly lucky: Your design will always work
  - Most of the time however - you will notice too late
    - The "right" mix of environment, SW and signals never occur in the lab
      - … or just very seldom
    - It might hit you in all different ways… (spurious, batch, SW, env., …)
- Problem is seen in most companies
- End product will have strange, non repeatable errors
  - Critical?
  - Difficult to explain for customer. Even more difficult to debug.

Examples on FPGA-bugs in final product 2. Internal Clocking

- Same requirements as for timing
  - but no semi-automation available for checks
- Same symptoms as for timing problems
  - You are lucky if it is detected…
  - It might hit you hard
- Internal clocking must be “correct by construction”
- Problem is seen in most companies
  - Lots of designers do not know how to handle this correctly
- End product will have strange, non repeatable errors
  - Critical?
  - Difficult to explain for customer. Even more difficult to debug.

Clocking and Timing
The "interesting" cases…

- Complete product comes to a halt
  - C1: Caused by a state machine deadlock (undefined state)
  - C2: Caused by two state machines waiting for each other
- Data transfer
  - C3: Pure occasional loss of a word (or one too many)
  - C4: Mismatch between FIFO counter and FIFO contents
- The lost pulse
  - Signal of width T not sampled on any clock edge (with a clock period of T)
- The really interesting one
  - Sporadic loss of data through external FIFO, but…
    - only for some boards, for some sequences of software accesses, and only within a junction temperature range of approx. 5 degrees

Motivation for proper timing analysis

- An FPGA with a timing problem
  - Will not be detected by an RTL simulation
  - May fail in test once in a while – for no obvious reason
    - May however also pass all lab tests
  - May occasionally fail in system test – for no obvious reason
    - And uncertain whether SW, HW, FPGA or test is to blame
    - May however also pass all system tests
  - May occasionally fail in field test – for no obvious reason
    - May however also pass all field tests
  - May fail in field operation – for no obvious reason
    - May happen for new SW, HW, FW
    - May happen for different temperature, voltage, power
    - May happen for FPGA #7, or for FPGAs #7 to #5000

Examples on FPGA-bugs in final product 3. Functional – due to complexity

- Lots of examples characterized by:
  - Unstructured design, lots of parallelism, lack of layering…
  - Designers have very little focus on understandable code
- Very difficult to get correct – or modify – or extend
  → Very time consuming
- End product may seem stable for a while, but then…
  - The unstructured or bad design will typically show errors …
    - For new versions of the software (or untested parts of SW)
      - Thus assuming a software problem?
    - For application scenarios outside that tested in the lab.
  - And what if the customer is writing part of the software?
- This problem is quite common
  - In particular in companies with insufficient simulation

Examples on FPGA-bugs in final product 4. Functional – due to lack of simulation

- Typically:
  - Designs with various options or modes
  - Control-oriented and protocols
- Testing partial and end product might seem sufficient, but only detects bugs within the scope of the test…
  - Difficult to reach corner cases
  - Parts of the FPGA functionality might not yet be used
  - Software might change – or just the sequence of events
- End product may seem stable for a while, but then…
  - Untested functionality will start to fail…
    - For new versions of the software (or untested parts of SW)
      - Thus assuming a software problem?
    - For application scenarios outside that tested in the lab.
  - And what if the customer is writing part of the software?
- A very common problem

Are things getting worse or better?

- Some issues are getting better – some very few places
  - Provided there is a strong focus on improvement
    - For companies building up experience and infrastructure
- Most issues are getting worse
  - As complexity is increasing – and fast
  - As this trend requires a defined methodology
    - For a team, but also for a single person
  - As most companies do not take this seriously enough

Addressing the challenges

- First… Get your true internal status
  - Where do you stand today?
  - Where do you waste time?
  - How is your product quality?
  - At what stage do you detect bugs?
- What improvements can be made?
  - Get overview of challenges
  - Get overview of solutions
    - Evaluate how they affect the problems
    - Evaluate what it takes to implement
    - Evaluate what it takes to follow up
  - Prioritize

Remedies for the worst time wasters (1)

- **Verification structure**
  - ➔ Should improve verification competence and experience
  - ➔ Should structure verification environment
  - ➔ Should have "self-checking" simulations for regression
  - ➔ Should coordinate verification at an early stage
  - ➔ Should provide a common testbench infrastructure
  - ➔ Should provide common libraries and templates

A bad verification structure does not only affect schedule. Wasting time here allows less time for risk reduction.

Remedies for the worst time wasters (2)

- **Timing closure**
  - ➔ Remember that timing closure might require a change in architecture; - so resolve as early as possible
  - ➔ Fine tuning code and constraints is very time consuming; - so spend more time early.
- **Interfaces between Modules, FPGAs, PCB and SW**
  - ➔ Very important issue to resolve for implementation responsible
  - ➔ Note that embedded systems need multiple impl. resp.

Note that the "time wasters" normally also waste time for SW designers, HW/PCB designers etc.

Remedies for the worst risk issues (1)

- **Implementation complexity**
  - ➔ Must improve competence and experience
  - ➔ Must assure proper layering and structure
  - ➔ Must avoid "feature creep" (ever increasing functionality)
  - ➔ Need early "review" of architecture proposal
  - ➔ Need early coordination between related modules
  - ➔ Document ALL complex issues and decisions
  - ➔ Proper verification is absolutely required

There are so many bad ways to implement complex designs…

- Allow a slow and structured start
- Follow up during early stages

Remedies for the worst risk issues (2)

- **Timing and Clocking**
  - ➔ Must improve competence and awareness
  - ➔ Must define proper design flow and checklist
  - ➔ Must structure and automate timing analysis
  - ➔ Must document properly
  - ➔ Must review

- When complex – this is definitely time consuming
- There is no shortcut here – other than to a potential disaster

- Analysis, analysis, analysis, …
- Documentation, documentation, documentation, …

Technical management

- Make sure you have the right competence and experience
  - Complex FPGAs are very often underestimated…
- Make active evaluations on efficiency and quality
  - Not only by QA system
- Make sure methodology/implementation is properly coordinated, supported and deployed
  - for FPGA Development flow
    - In particular the critical issues
  - for Product implementation
  - for Verification
  - for Reuse coordination
- Provide and maintain an infrastructure for all important aspects of FPGA development
- Follow up, Follow up, Follow up

FPGA development Best Practices

- A two-day course
- Covers the mentioned subjects
- Unique course - not offered anywhere else
- Focus:
  - Best practices
  - Quality and efficiency
  - Practical and Pragmatic
  - Tool independent
- Open general course or on-site - on demand

End

bitvis

We are a team of very dedicated and experienced designers
who have chosen to specialize in developing embedded SW and FPGA;
because this is what we do best,
- but even more important
– because we enjoy it.

A partner
for SW and FPGA (& ASIC)