Bitvis

Developing an FPGA module
- with a strong focus on efficiency, quality and modifiability

Bitvis

- Independent Design Centre for SW and FPGA (& ASIC)
- 6 designers – from March 1, 2012
  - Expect to increase significantly in 2012/2013
- Building on the Digitas legacy
  - Efficiency/Quality → Methodology
  - Customer partnership relation
- Concept, Specification, Architecture, Implementation, Verification, Test
- Methodology, Reviews, Sparring partner
- Lots of experience on the critical issues
  - Architecture, HW/SW interface, RTOS, Linux, Drivers, Test, …
  - Design structure, Timing & Clocking, Verification, …
  - Xilinx, Altera, Actel. ARM, AVR, Nios, MicroBlaze, Leon, ….

Design Services for Customers

- Module design and/or FPGA design
- In house or at customer's site
- Verification / Unit Test / Debugging
- FPGA methodology
- Reviews and/or Sparring partner at all levels

Available resources at Bitvis from March 1st

- Embedded software: 2 Designers
- FPGA: 1,5 designers (incl myself 50%)

(My focus the next two weeks : Sales!!!)

FPGA Best Practices courses

April 2012

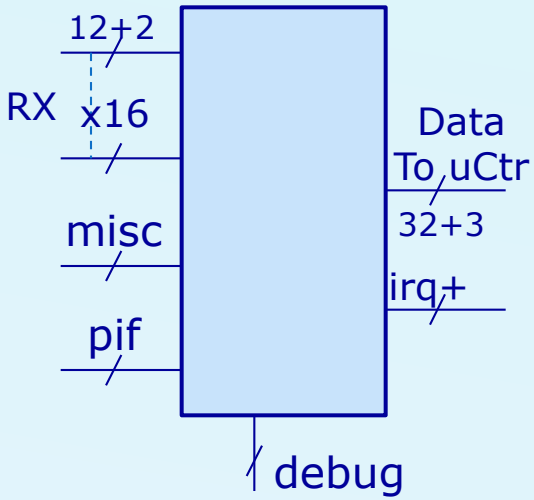
- Bitvis offers the well established course: "FPGA Development Best Practices"
  - Based on the Digitas course previously presented in DK, SE, NO
  - Two full days of learning
  - Major focus on how to improve your FPGA-based projects
    - Design Structure and Methodology
    - Verification Structure and Methodology
    - Focus on Efficiency and Quality improvement
    - Focus on a Best Practices approach that you can apply today
  - Very good feedback from participants in all countries
- Other courses available on request
  - D&V Reuse, Advanced Verification, Timing and clock domain crossing
  - Basics – to get started in a structured manner.

Agenda

- Specification and challenges
- Attacking the challenges
  - Potential pitfall wrt. Quality and Efficiency
- Design / Architecture / Coding
- Module verification
- FPGA verification
- Spec. changes
- Lessons learned
- Conclusions

Module (SBC) functionality

- Data collection: 16 ch * 12-bit
  - All channels @ 5M Sps (= 60M bps) each → 120M Bps (= 960M bps)
  - Any 3 or less channels @ 40M Sps each
  - Any combination with a maximum total sample rate of 130MS/s
- Generate data packets for active channels
  - Data: 4 - 87.366 samples + extended mode
  - 22B packet header information
- Forward packages to uCtr
  - Selectable burst sizes at 50M Bps
- Various status and termination control
- System test/debug support (pattern & RTW)



Implementation challenges

- Lots of scenarios and corner cases
  - Channel configurations (number, combinations, order, data rate)
  - Packet sizes, Extended mode,
  - uCtr burst sizes
  - Rejection mechanism
  - Graceful completion
  - Combination of channel configuration, packet sizes and burst sizes
- Internal 256kB RAM to be used efficiently in all scenarios
- 100 MHz → 10 ns per logic stage
- By far the customer's initial worst worry
- There will be spec. changes

Focus

- Minimise design effort
  Less consultancy hours ➔ lower cost
- Minimize risk
  Complex design with lots of corner cases ➔ Quality
- Optimize for changes/modifications
  Spec. changes expected
- Optimize for customer understanding and hand-over
  Quality assurance and future modifications

Potential Quality & Efficiency pitfalls

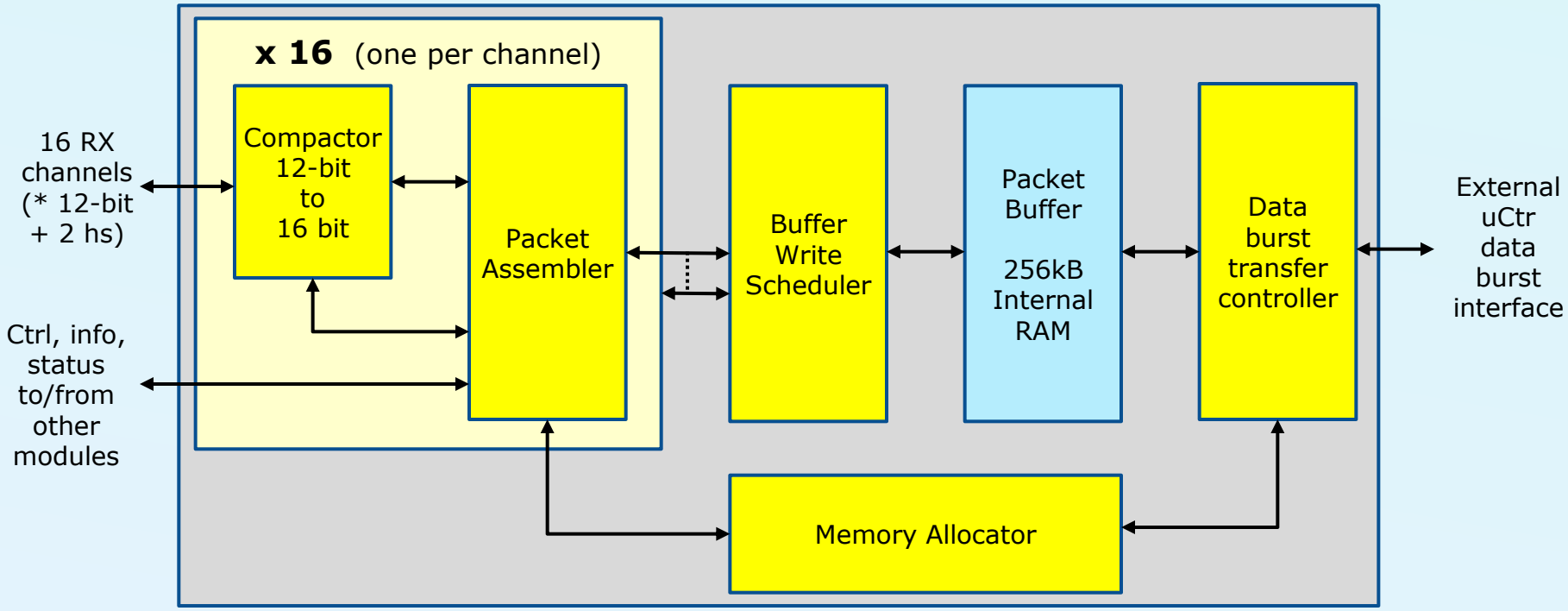
Pot. Quality & Efficiency Pitfalls	Mitigation / Avoidance
Clocking issues (CDC)	Single signal dual sync
Specification-related corner cases	
Implementation-related corner cases	
Timing problems	
Module Verification complexity	
FPGA verification complexity	
Module interface mismatches	
Software interface mismatches	
Lab-test complexity and efficiency	

Pitfall mitigations

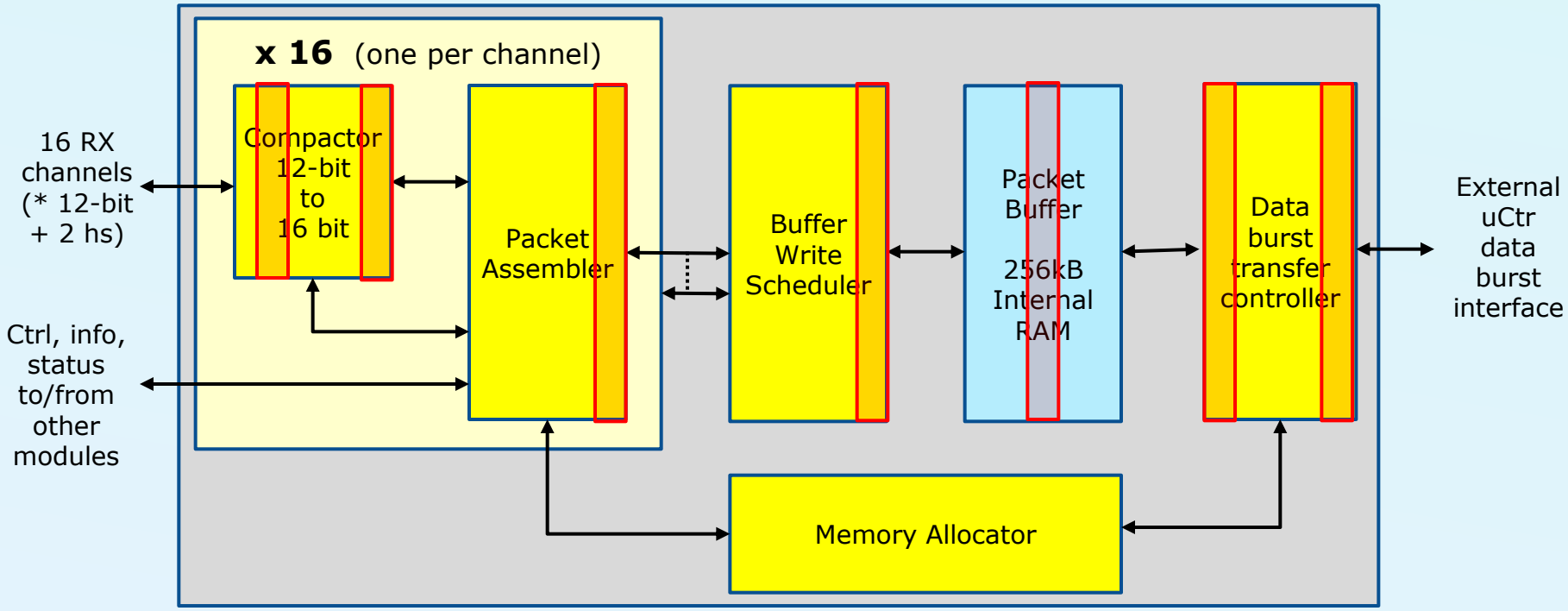
- Specification-related corner cases
- Implementation-related corner cases
- Timing
- Verification complexity: Module and FPGA
- Interface mismatches: FPGA, External HW, SW
- Lab-test complexity

- Spend sufficient time on design architecture
  - Strive for high cohesion and low coupling
  - Proper design structure and layering
- Define sufficiently deep pipeline as part of the architecture
- Avoid complex solutions as far as possible
- Provide mechanisms to support SW development
- Provide mechanisms to support lab-test
- Spend sufficient time on verification architecture
  - Strive for high cohesion and low coupling
  - Proper testbench structure and layering
  - Allow re-use from module to FPGA level testbenches
  - Provide assertions
- Document at the right level

Design architecture



Pipelining – of data path



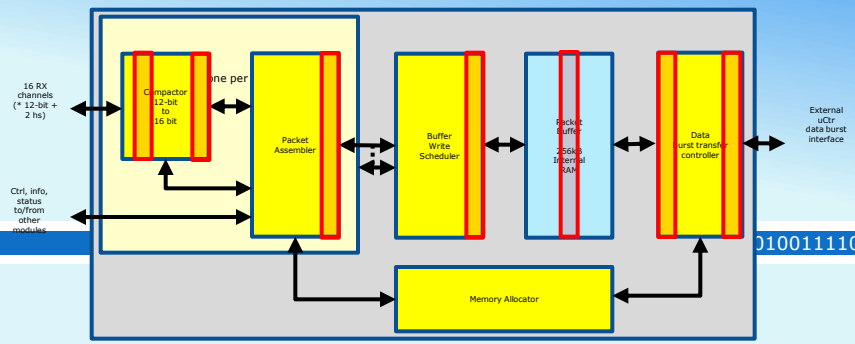
Also pipelining pure control paths

Submodules only connected to single input and single output submodules
Minimum coupling and acceptable cohesion

Coding issues

- Sort out interfaces early
  - Agree on signals, types and organisation
    - Records are good, but don't overdo it
    - Good names are very important
      - Verbose names when required
- Use common coding conventions
- Single source for registers
- Comment and Document properly

Status so far



Pot. Quality & Efficiency Pitfalls	Mitigation / Avoidance
Specification-related corner cases	Good mitigation
Implementation-related corner cases	Minimised
Timing problems	Minimised given good STA
Module Verification complexity	Minimised from design
FPGA verification complexity	(Minimised from design)
Module interface mismatches	Reduced risk at this stage
Software interface mismatches	Reduced risk at this stage
Lab-test complexity and efficiency	(Minimised from design)

Module Verification

- Protocol oriented design
- Significant configuration space
  - Lots of corner cases
- Need flexible data input mechanism
  - Selectable channel input
  - Selectable set of data inputs (start, increment, number)
  - Directed-random/selectable period between samples
  - Directed-random time from common ch. sync to ch. start
- Need multiple active processes
  - 16 * RX-ch stimuli + Data burst checker, (+timer, clk, etc..)
  - All to be controlled from a single testbench sequencer
  - Need model for conversion from stimuli to output bursts
  - Registers accessed by BFMs in sequencer

FPGA verification of module

- All input now provided by FPGA-internal modules
  - Register access as before
  - 16 ch data:
    - Selectable channels, data inputs, rates, etc…
    - Central setup (records + generics)
  - Timers, strobes, data information, etc. directly input
  - Events and interrupts via register access to other modules
- Output transformation
  - Dedicated procedure to transfer setup init. to SBC TB format
  - Dedicated procedure to transfer data stimuli to SBC TB format
  - Same complex checker procedure as for module TB
  - Directed-random burst enabling
- Various system checks

Special Verification Issues

- All required TB infrastructure support – (Digitas Utility Lib)
- Dedicated procedures or processes for all interfaces
- Non blocking (non-time-consuming) procedures called from sequencer trigger parallel processes
  - Not using verification components
  - Using a very structured procedure approach (TLM/BFM)
- Dedicated "SW drivers"
  - Configure_sbc(parameters), with sanity check
  - As given to SW developers
- Test-configuration sanity checker
  - Legal setup and Capacity issues – for early warning

Design for verification and test

- Assertions will trigger in simulation at module and FPGA TB
  - FSM case: When unexpected others
  - Unexpected else-branch
  - Wrong set of generics in design (or TB)
  - Interface requirements (protocols, or sequences)
    (e.g. new frame start when previous still active
  - SW registers "overlap" – and must match

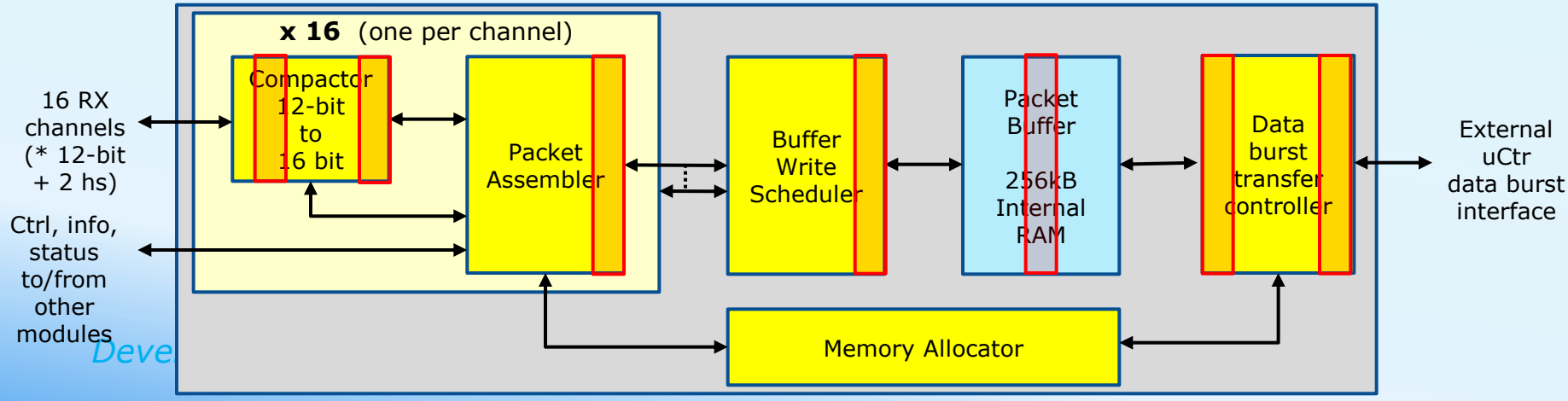
  → Some assume design error inside module or neighbours
  → Others assume bad stimuli, wrong setup, bad "SW"
- SW-readable error regs for selected scenarios
- System test support
- Real Time Window

Status

Pot. Quality & Efficiency Pitfalls	Mitigation / Avoidance
Specification-related corner cases	Good mitigation
Implementation-related corner cases	Minimised
Timing problems	Minimised given good STA
Module Verification complexity	Structured: Efficient and good Coverage
FPGA verification complexity	Structured and reuse Efficient and good coverage
Module interface mismatches	Well checked
Software interface mismatches	Significantly reduced
Lab-test complexity and efficiency	Good support Very little wasted debug

Specification change example

- Initial spec: Any channel max. 5M Sps
- Always focusing on simplest design
  → Buffer write scheduler was fixed round robin, 1ch per cycle
  → Compactor sized to match "slow speed" per ch
  ➔ Far less development time. Safer. Smaller
- New spec: For trial purposes: Up to 3 ch at 40M Sps
  ➔ Simple compactor mod. Increased buffer size. No coupling
  ➔ New complex buffer write scheduler:
  Needed additional step in local control pipeline. No coupling.
  ➔ Near zero effect on TB env. Minimum additional work D&V



Module summary

- "Wasted" time up front on detailed structure for both Design and Verification
- Gained multiple times the "waste" during:
  - Design/coding:
    - Far simpler code and thinking
    - No timing issues – but on the limit
  - Verification:
    - Much easier to make good, understandable test cases
    - Simpler reuse
    - Far better coverage at far less time
  - Test :
    - Burst generations (of "fake data") used a lot
    - Debug mode has simplified and speeded up testing
- Status: 1 simple bug found in review. None since.

Conclusions

Properly structured design and

Properly structured verification yield:

- ➔ Significantly reduced module development time
- ➔ Significantly reduced product development time
- ➔ Significantly increased modifiability
- ➔ Significantly reduced risk and improved quality

MUST spend time up front!

Your partner for SW and FPGA (& ASIC)

Developing an FPGA module
- with a strong focus on efficiency, quality and modifiability

If you are interested in our 2-day course
"FPGA development Best Practices"?
Please get in touch ASAP