



“System-level Design for FPGAs”

Mark Dickinson
FPL 2007



Objective

- Analyse FPGA systems by looking at the common sub-systems
- Define some attributes of “framework” which makes system design more effective
 - Look at SystemC as a way to describe this framework
- Illustrate the ideas with real examples
- Give my view of a score-card on FPGA system design

Agenda

- Part 1: Analysis of the system
 - Anatomy of an FPGA system
 - Why do FPGAs present a unique challenge?
 - Desired attributes of system tools
- Part 2: Analysis of the components of the system
 - A look at each component of the system
 - Rules associated with each component – illustrated by example
 - The role of SystemC for each component
- Summary
 - A system-design score card



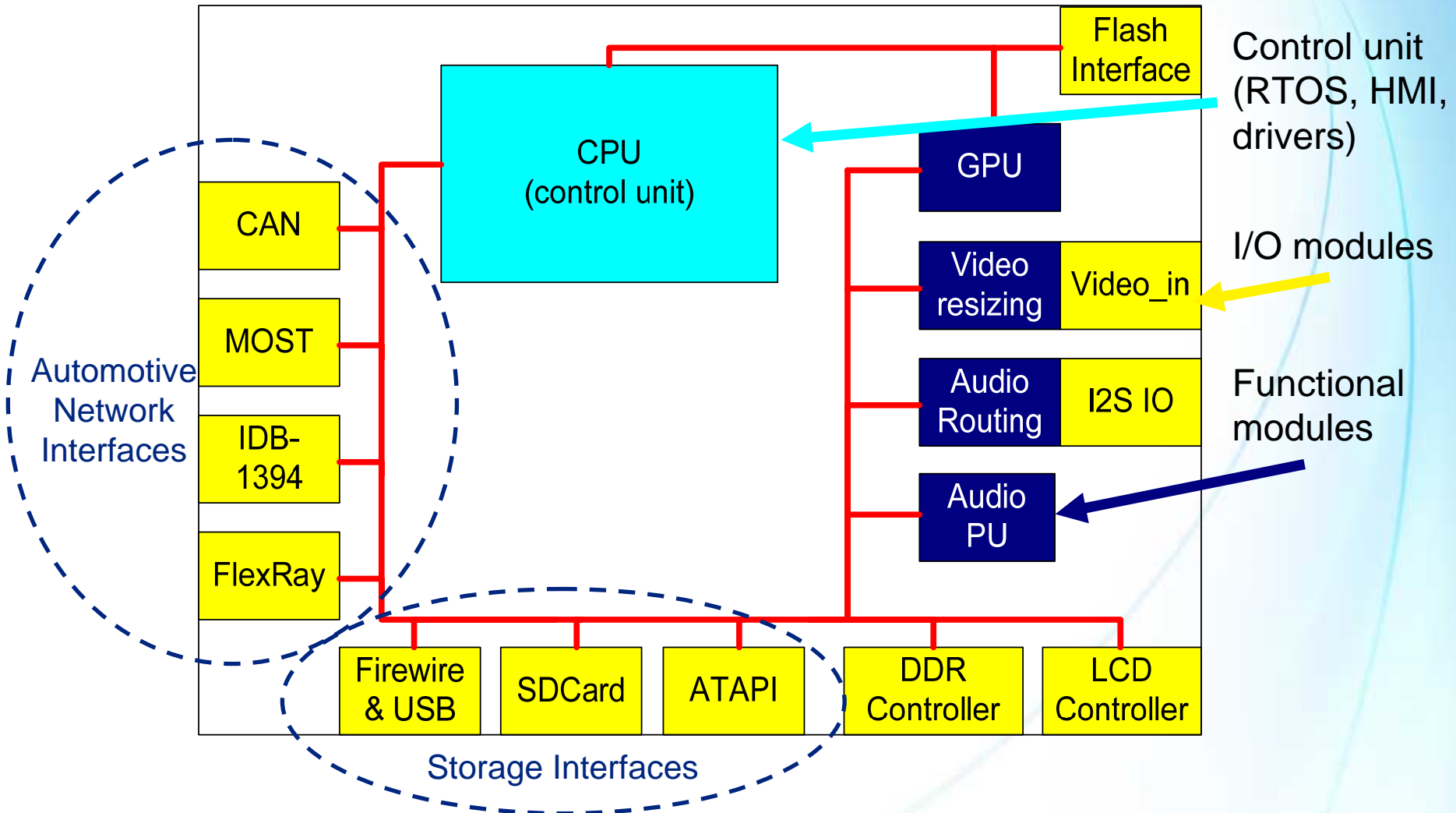
Part 1: Analysis of the System



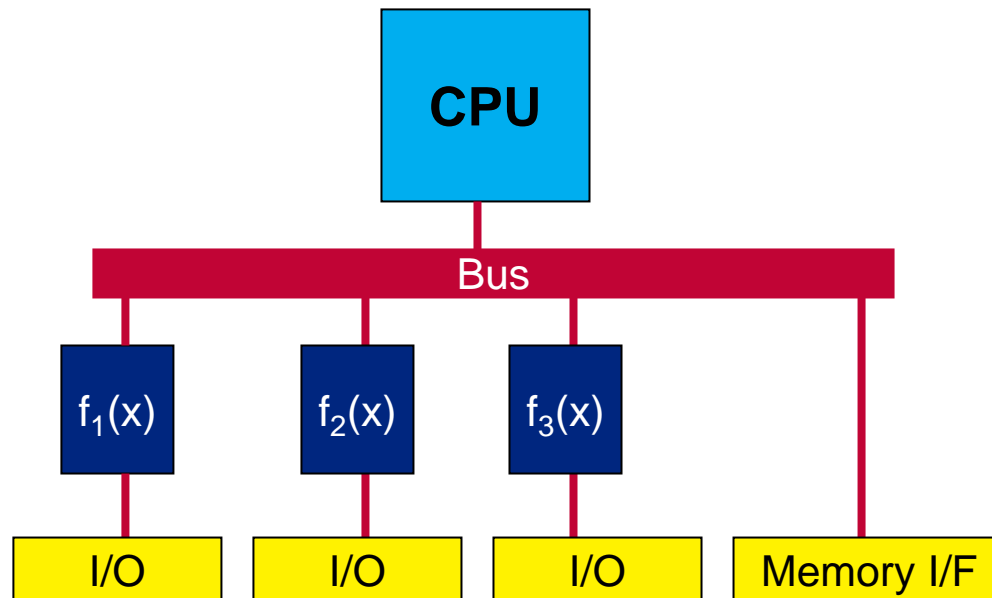
Types of System on an FPGA

- Two types of FPGA system are becoming increasingly common:
 1. Processor-centric system
 - Emphasis on applications running on a “processor”
 - Hardware peripherals and coprocessors surround the processor
 2. Data processing system
 - Emphasis on data-path processing of samples, frames, packets etc of data
 - Hardware modules connected in a complex and potentially dynamic way
- A simplistic view – but useful?

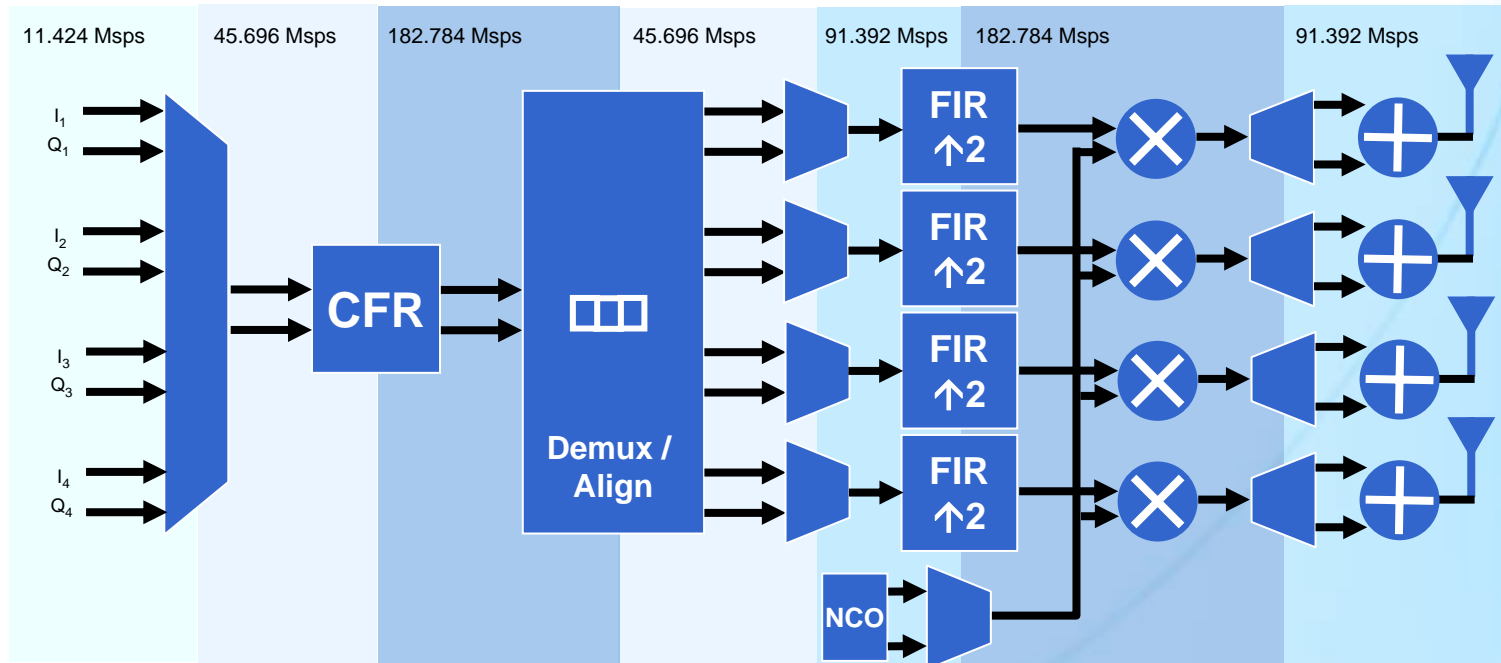
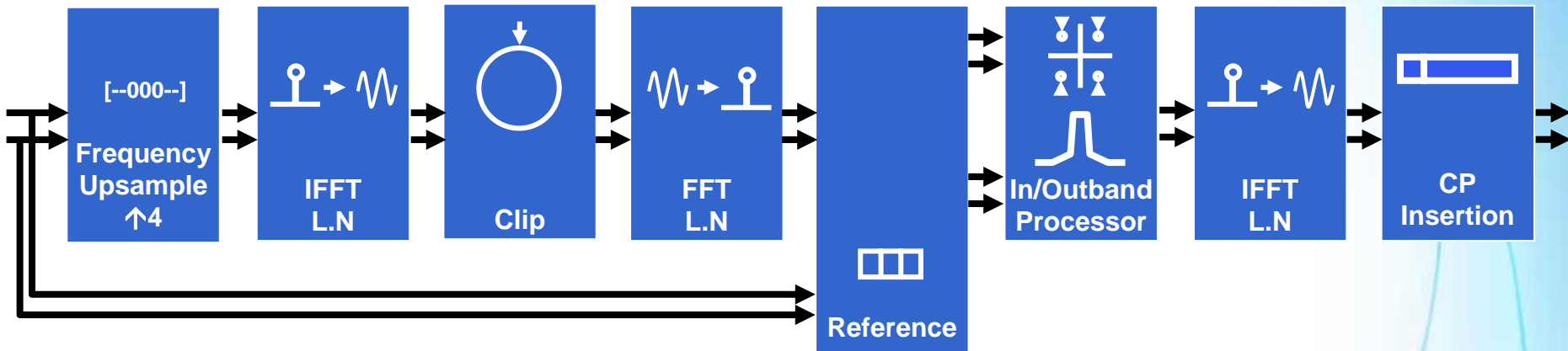
Processor Centric: Auto Infotainment System



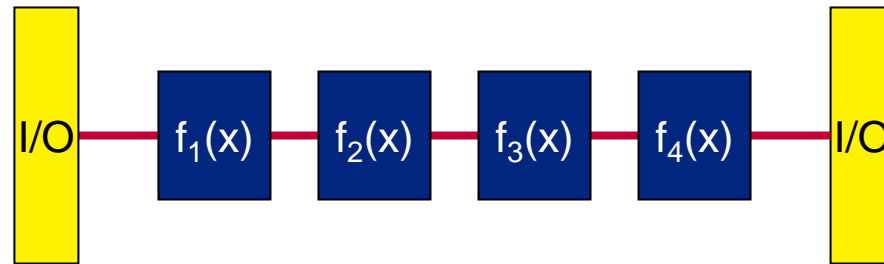
Simplified Processor-Centric Architecture



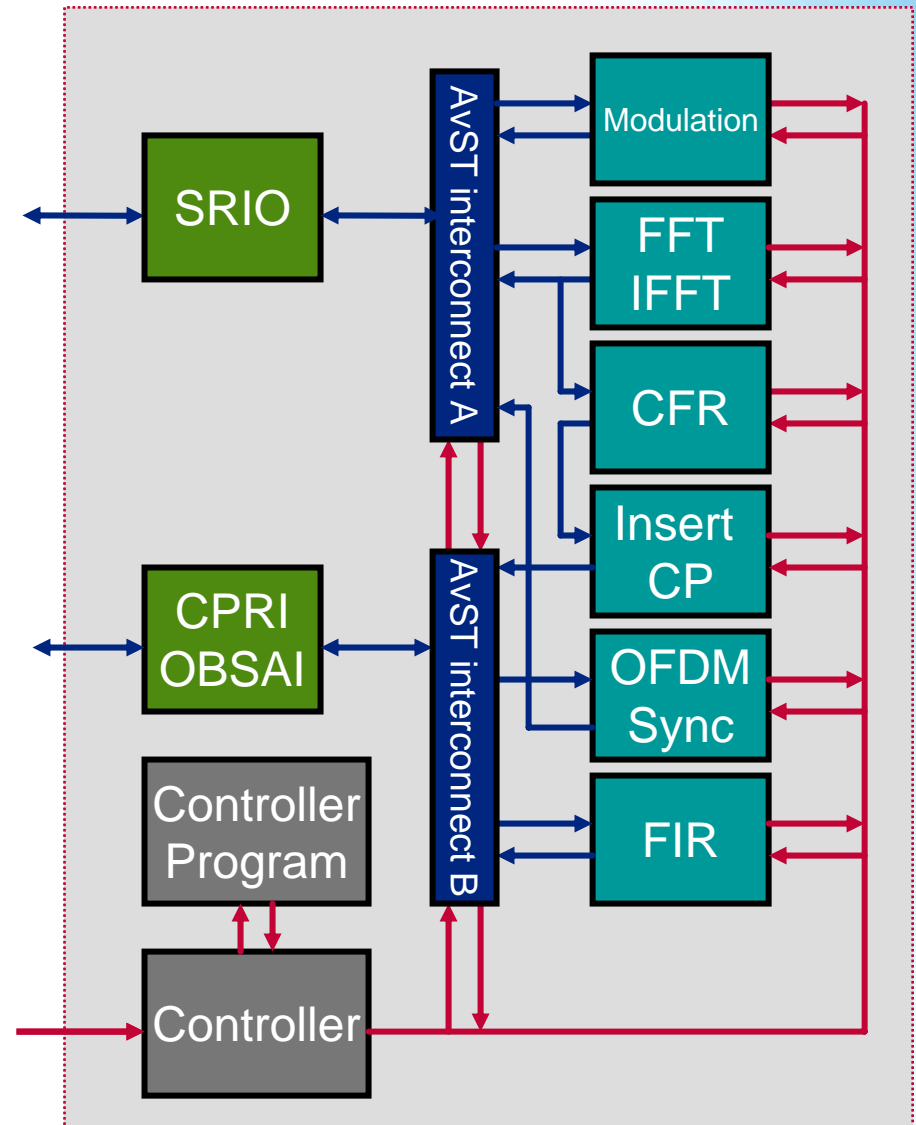
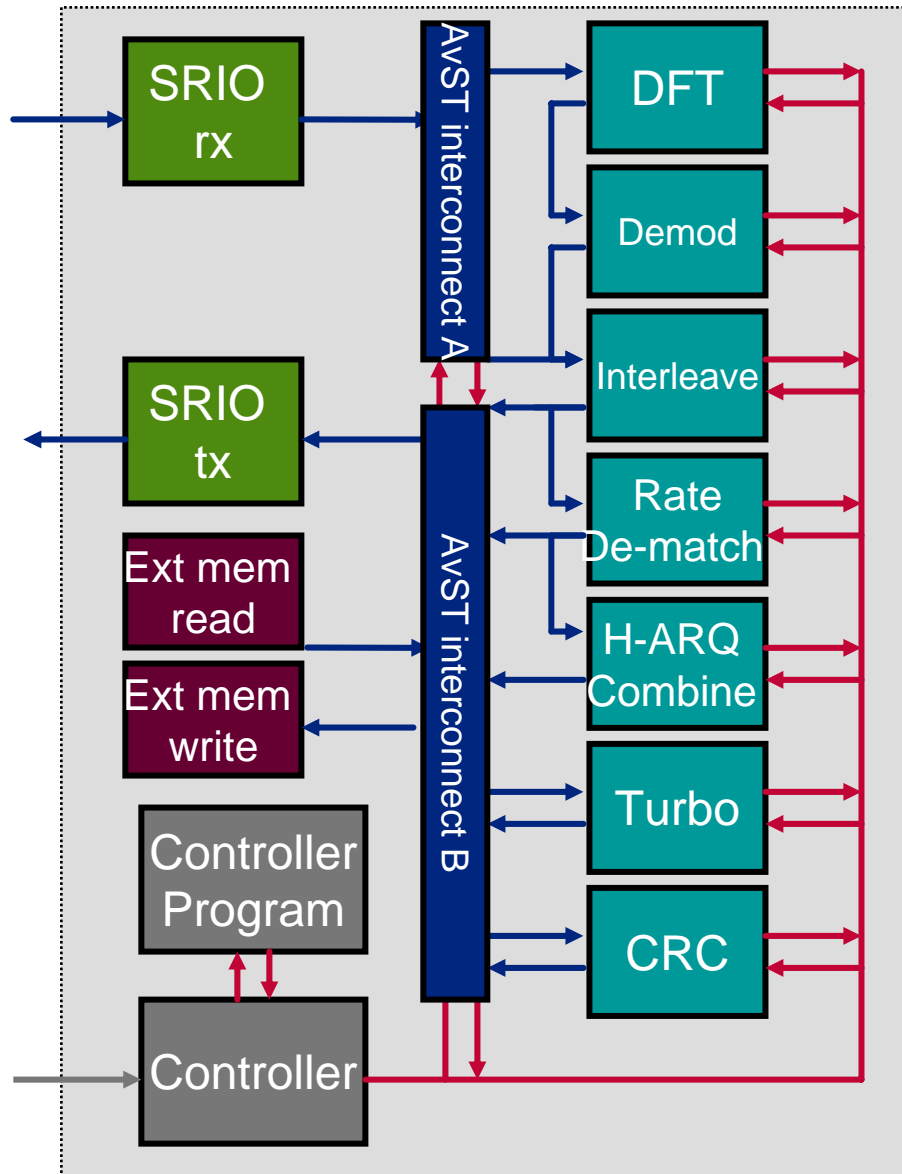
Data Path System: IF Modem for OFDM



Simplified Data Processing Architecture

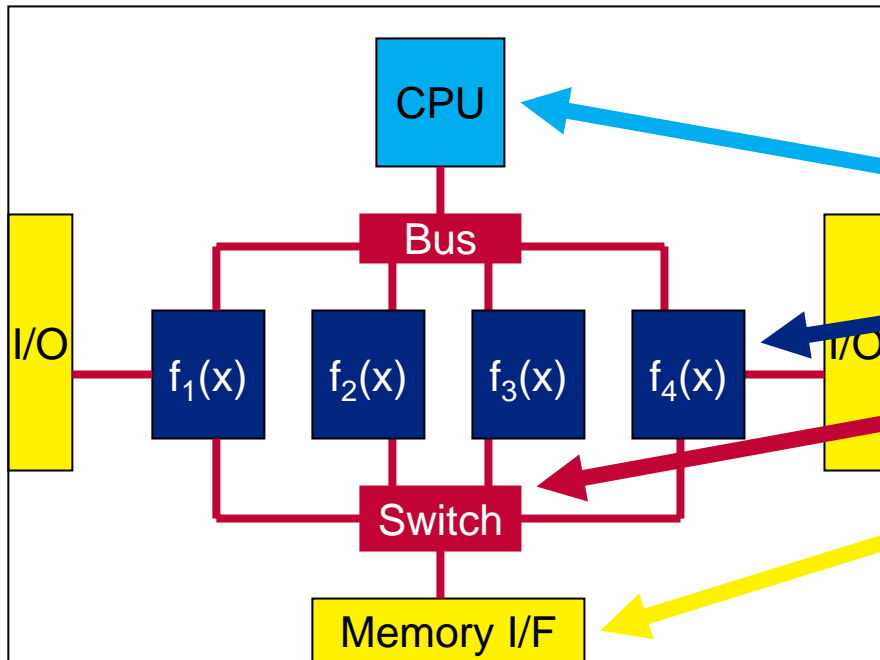


Complex Data Path: Baseband Modem



A Generalised Data Processing Architecture

Four types of entity

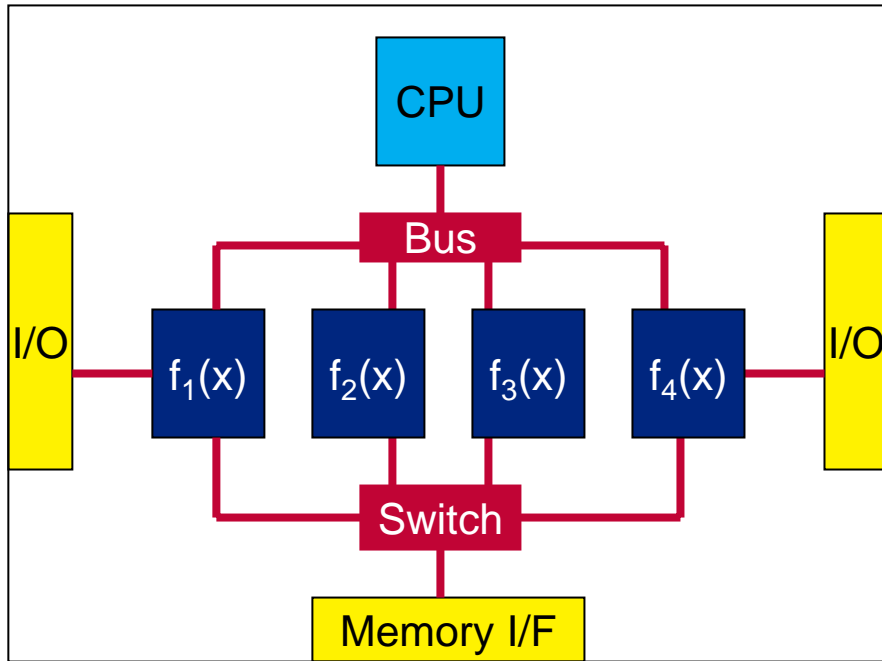


1. Control module
2. Compute module
3. Interconnect module
4. I/O module

What is Different About FPGAs?

- FPGAs must take a different approach to system design to ASICs
- FPGAs add value by doing the heavy lifting (the compute blocks) in hardware
 - ASICs can afford to instantiate a processor or DSP
- However, run-time flexibility and fast development are very important
 - Characteristics normally associated with a software methodology
- The “framework” idea addresses these apparently mutually exclusive requirements

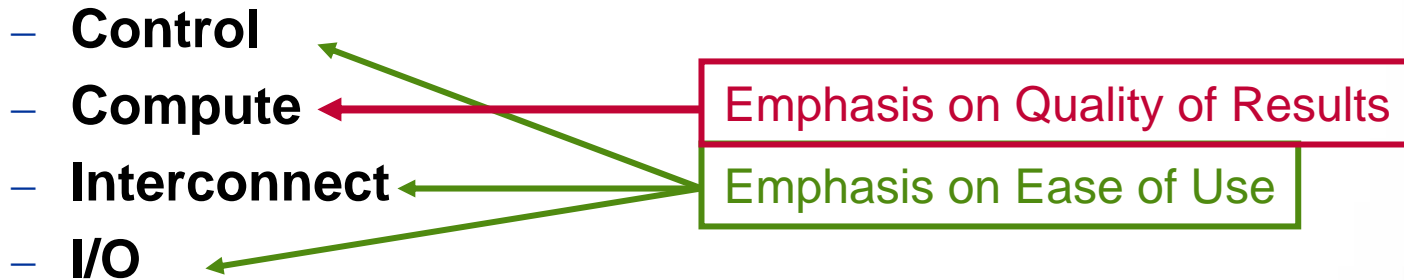
A Generalised Data Processing Architecture



- Predominantly hardware implementation
- Software-centric design methodology?

System Design Components – Summary

- The FPGA system can be broken down into 4 sub-systems:



Characteristics of Good System Design Tools

- Productivity & Accessibility
- Fast functional verification
- Visibility for Debug
- Portability & Flexibility
- Automatic timing closure

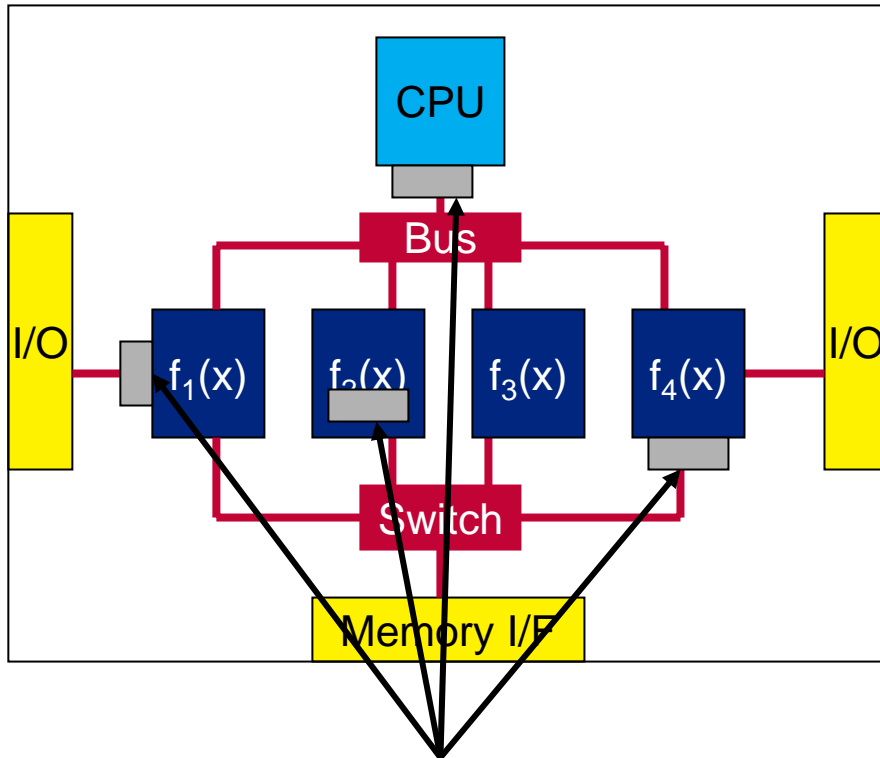
Productivity & Accessibility

- We are being called to provide significantly better productivity than RTL HDL design
- However, the design entry mechanism must be accessible to the target audience
 - Hardware engineers are the FPGA's traditional audience
 - Software/DSP/System engineers are the new expanded audience
 - SystemC is complex, we need to hide the complexity

Fast Verification

- For many systems, functional verification is a significant part of the development effort
 - Image processing: Subjective quality analysis on large sample sets
 - Wireless systems: Rigorous performance analysis within complex simulation environments
- Hardware-based verification is often impractical
 - Impossible to have a repeatable, controlled environment
 - Requires the rest of the system to be available and perfect!
- Support for fast system simulation is highly desirable
 - Using the same source code

Debug Visibility

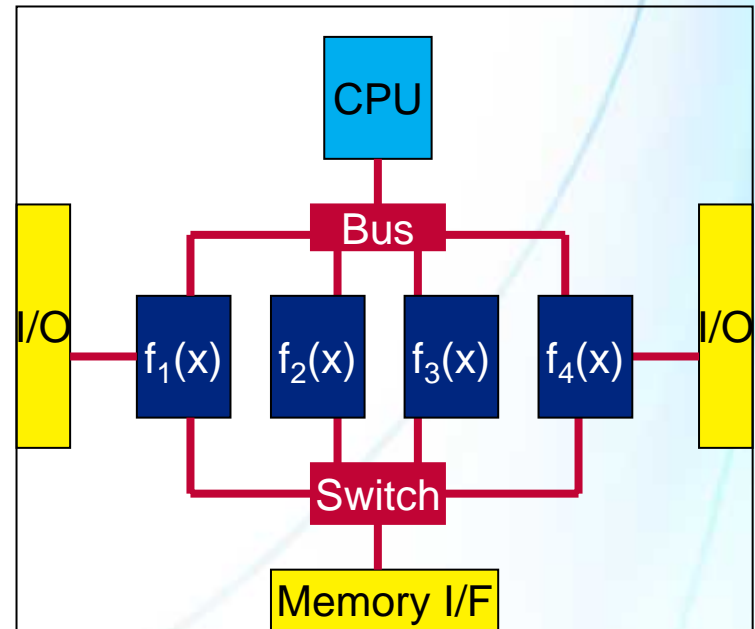


System Instrumentation

- Debug tools that understand system context
 - E.g. Average packet queue depth
- Consistency between the simulation and H/W environments

Portability & Flexibility

- Migration of the design across devices generations
- Easy scaling of the design for different markets
 - Femto-cell versus macro BTS for wireless systems
 - High-end and low-end feature sets called for in automotive applications
- Increasingly applications call for run-time flexibility
- But also “fast update”
 - E.g. modify $f_n(x)$ but not the rest of the system
 - This also allows incrementally constructed systems



Automatic Timing Closure

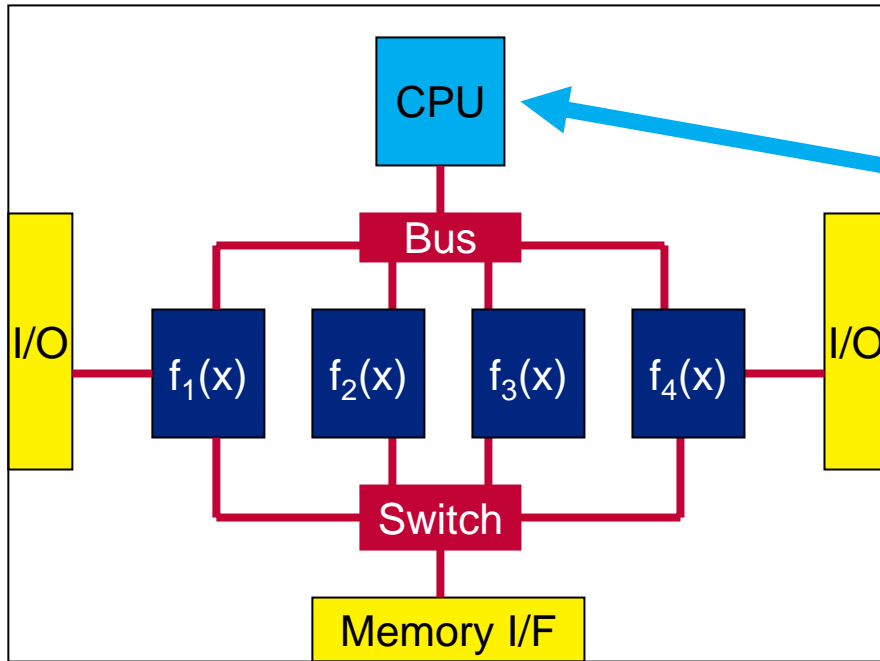
- System constraint-driven design is the goal
- Inputs should be
 - Throughput (e.g. frames per second)
 - Latency (e.g. milliseconds per frame)
- The tool flow should then automatically achieve timing closure
- Not using each resource at its F_{\max} is a waste
 - Tools need to manage this



Part 2: Analysis of the Components of the System



Control Blocks



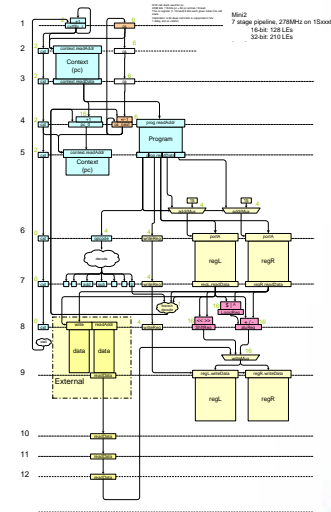
1. Control module

Control Blocks

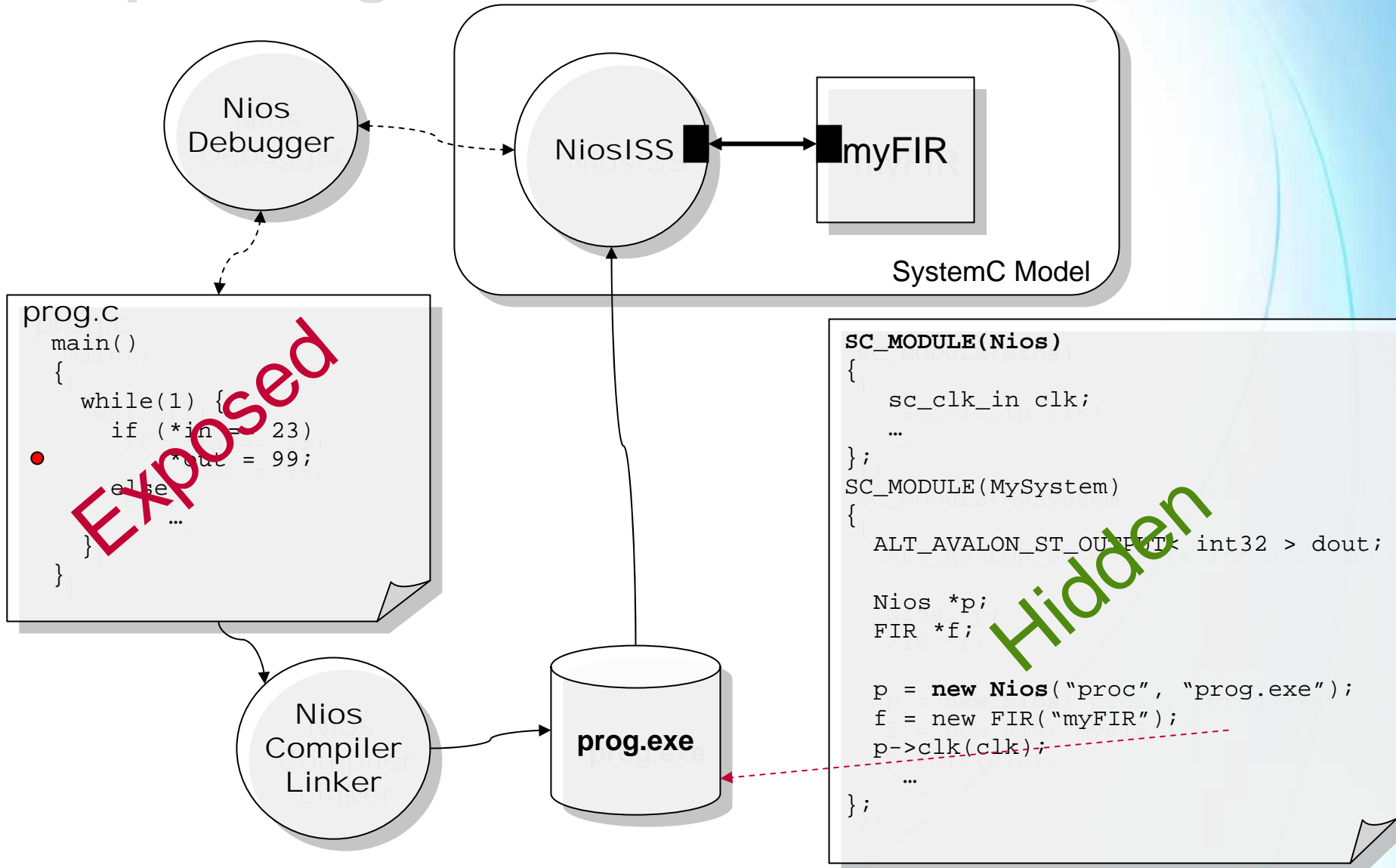
- Simple State Machine
 - E.g. Determination of a scaling factor dependent on input and output resolutions
- Complex protocol
 - Exception handling for an unrecognised packet
- Widely differing performance requirements
 - Time to make a single decision (ns to ms)
 - # decisions per second (tens to millions)
- This is the part of the system which requires greatest flexibility
- Natural language for expressing design intent is C
- S/W design methodology allows best productivity
 - Time to make a change
 - Readability
 - Visibility (debug)

Example: Packet Processing Controller

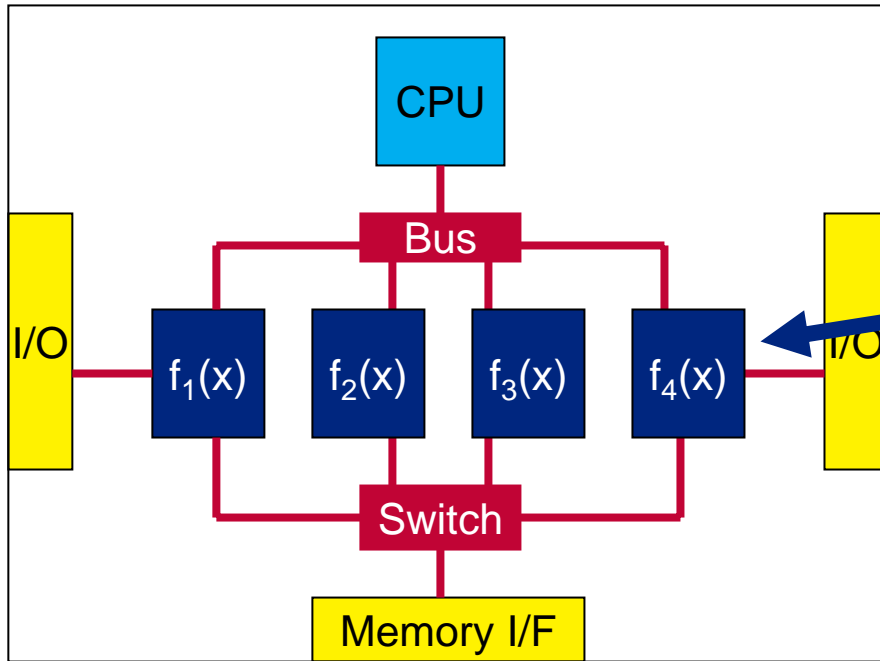
- Multi-threaded packet header processing engine
- Many threads (and many engines) allow very high “packets-per-second”
 - To deal with up to 40Gbit streams
- Latency not an issue
 - Design optimised for maximum hardware re-use
- Programmable in C



Expressing the Control Block in SystemC



Computation Blocks: “Heavy Lifting”



2. Compute module

Compute Blocks

- A major area of value add for FPGAs
- Functions for which FPGAs are the most efficient “programmable” platform, e.g.
 - DSP functions such as FFT
 - Packet processing such as packet classification
- “Most efficient” means:
 - Lowest cost
 - Lowest energy
- 3 levels of flexibility required
 1. None
 2. Run-time configuration amongst known parameters
 3. Embrace new requirements (but similar functionality)

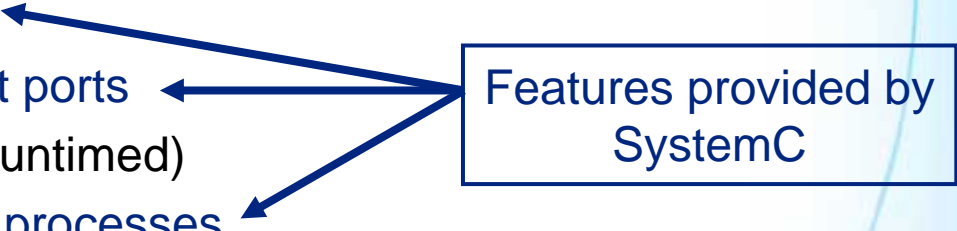
Synthesis of Compute Blocks

- Major focus is QoR
 - These blocks are often the biggest and most performance critical parts of the system
 - They are often the parts that define the system “cost”
- Much work has been/is being done on high-level synthesis for such blocks
- Increasingly RTL is not seen as the best language
 - Too hard to do design space exploration (especially for DSP)
 - FPGAs are unique in that their resources are a product of the actual resources and the Fmax of that resource

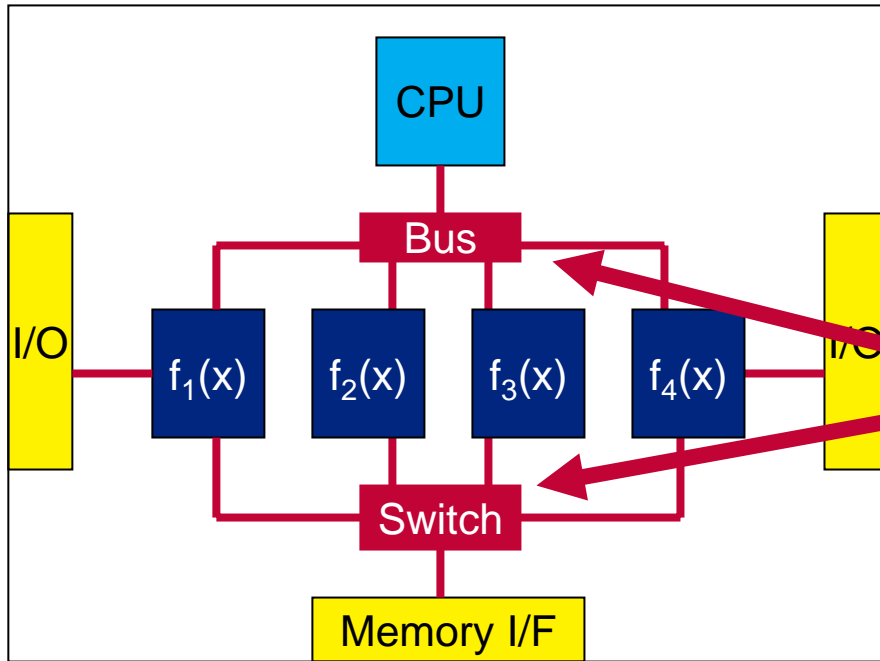
Example of Developing a Compute Block

- Some Altera IP uses SystemC as the source code
- SystemC is used to:
 - Express fixed point types
 - Define the input and output ports
 - Describe the functionality (untimed)
 - Identify any asynchronous processes
- This is converted into a scheduled design (RTL)
 - Optimised for the system constraints on the specific target platform
- Allows fast simulation
 - But describing the system in SystemC is not enough on its own
 - I will expand on this once we have talked about interconnect

Features provided by SystemC



Interconnect



3. Interconnect module

System Interconnect Abstraction

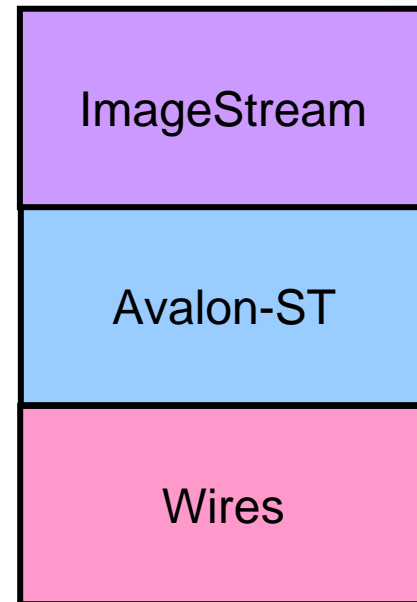
- It is well understood that standardisation and abstraction of interconnect is useful
- Standardisation of the interface defines:
 - Electrical compatibility
 - Logical conventions
- Altera extensively uses standardised interfaces for IP and system tools
 - Memory mapped interfaces (Avalon Memory Mapped Interface)
 - Streaming interfaces (Avalon Streaming Interface)
- This allows automated wiring
 - SOPC Builder is extensively used for system design
 - But we can do more...

System Interconnect Abstraction

- There are benefits to further standardisation
- Define an interface standard that “defines semantics as well as syntax”
- This brings two major additional benefits
 1. IP modules will work together (rather than just synthesise without error!)
 2. New simulation possibilities open up
- This is best illustrated by example:

ImageStream and Avalon-ST

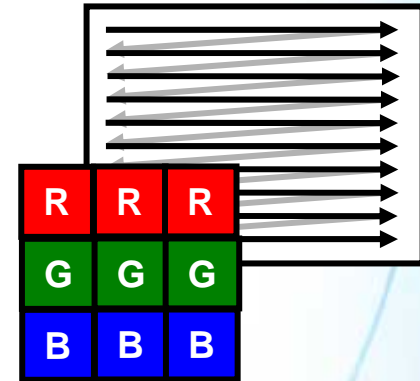
- ImageStream: A protocol defined over Avalon-ST
- Layers, like TCP stack
 - Wires provide simple clocked connections
 - Avalon-ST provides point-to-point flow-controlled transmission of an ordered sequence of N -bit “symbols”
 - ImageStream transmits a sequence of frames of video



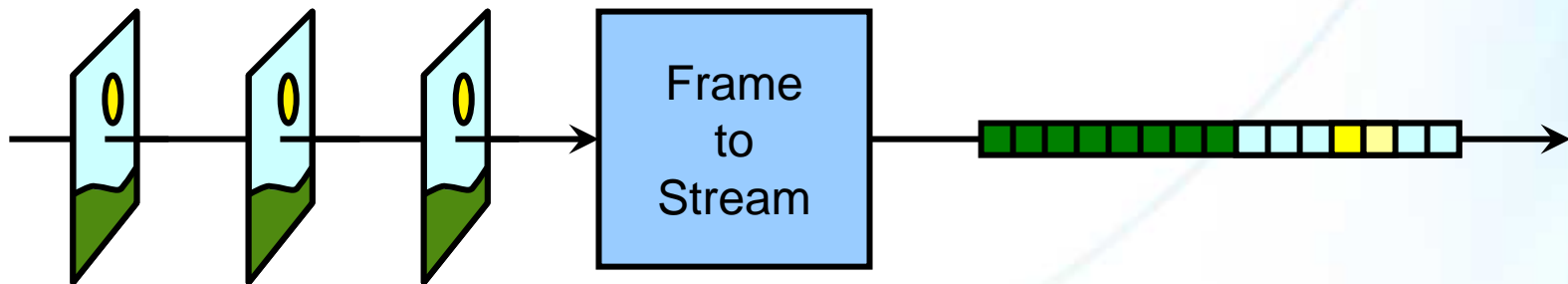
What Does ImageStream Define?

■ How to send video over Avalon-ST

- Order to send pixels
- How to send different types of video data
 - Colour formats: RGB, YCbCr 4:4:4, 4:2:2, ...
 - Interlaced or progressive video
 - Colours in parallel or sequence
- How to mark start and end of frames

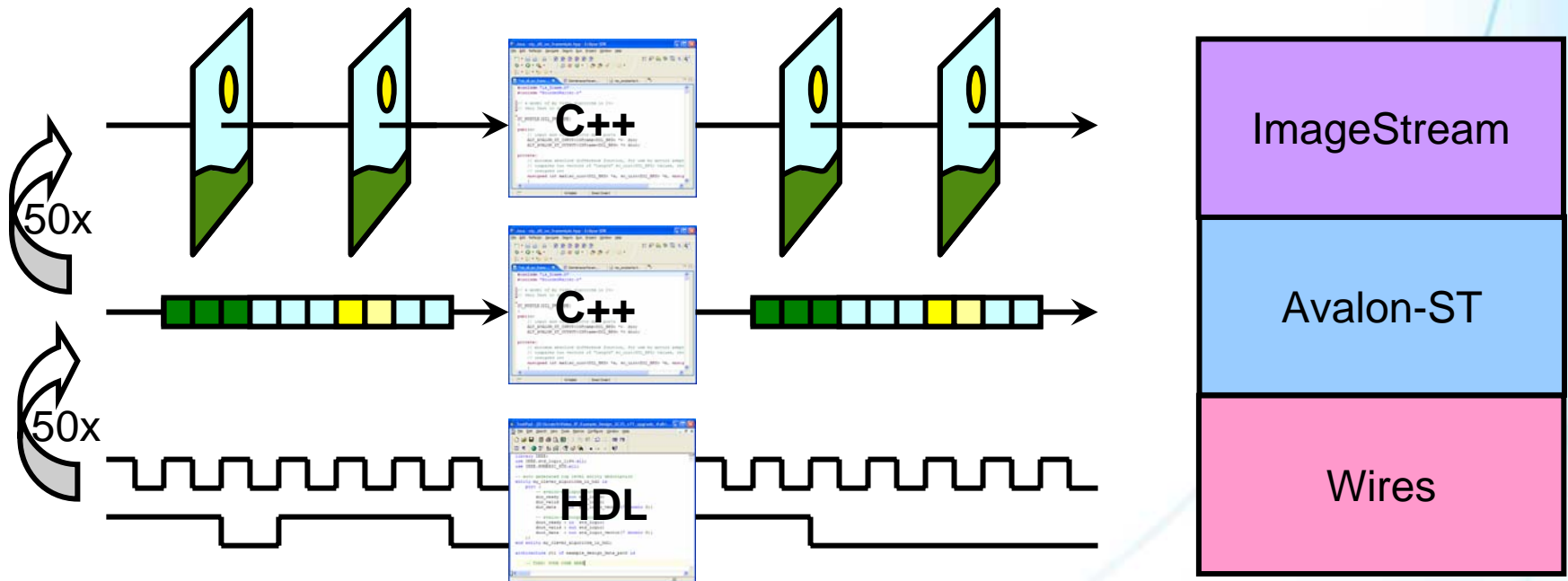


■ Or, a relationship between a stream of frames and a stream of samples

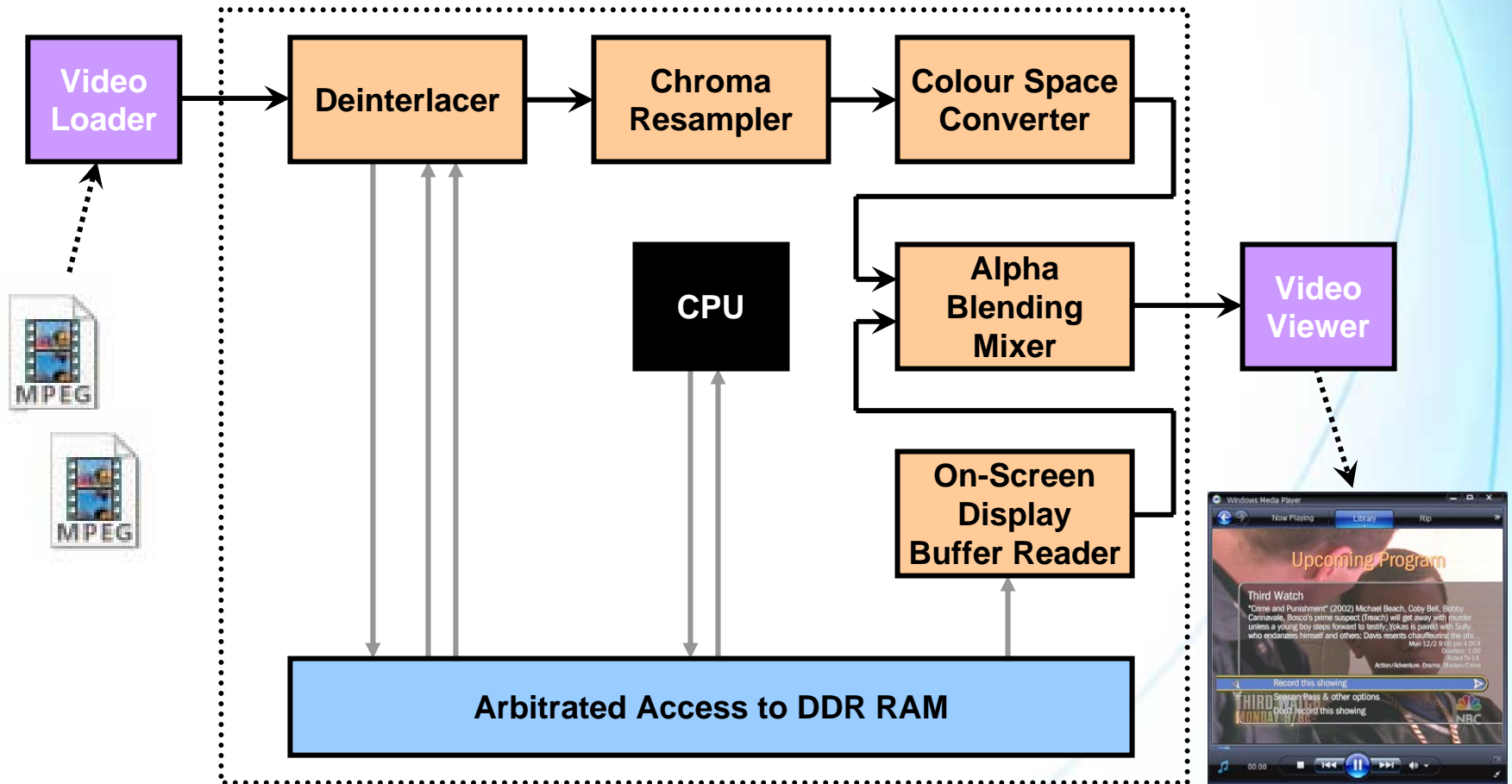


Higher Level Simulation is Faster

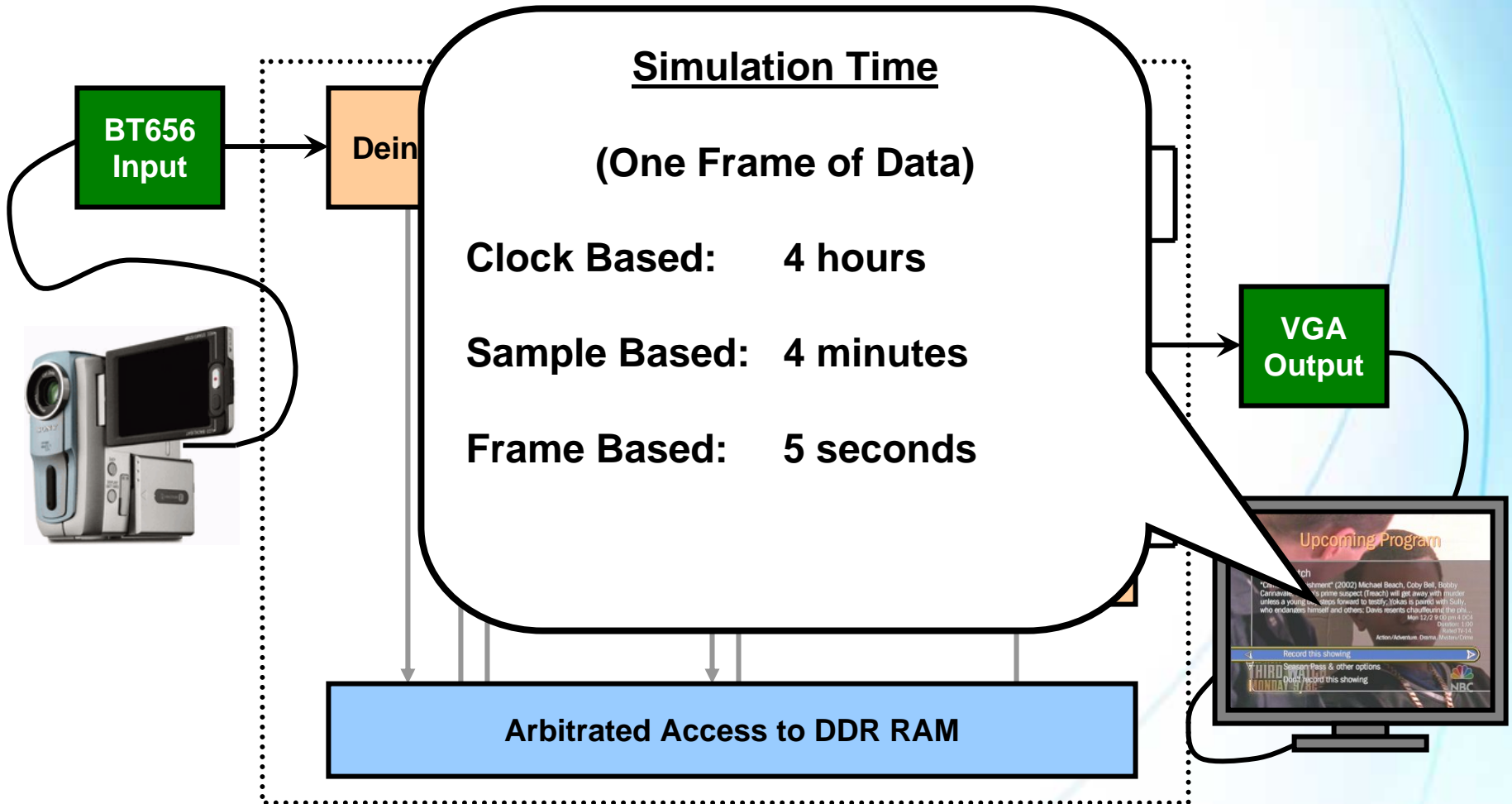
- Current simulation at level of wires and clocks
- Transaction simulation 50x faster
- Frame based simulated 50x faster again!



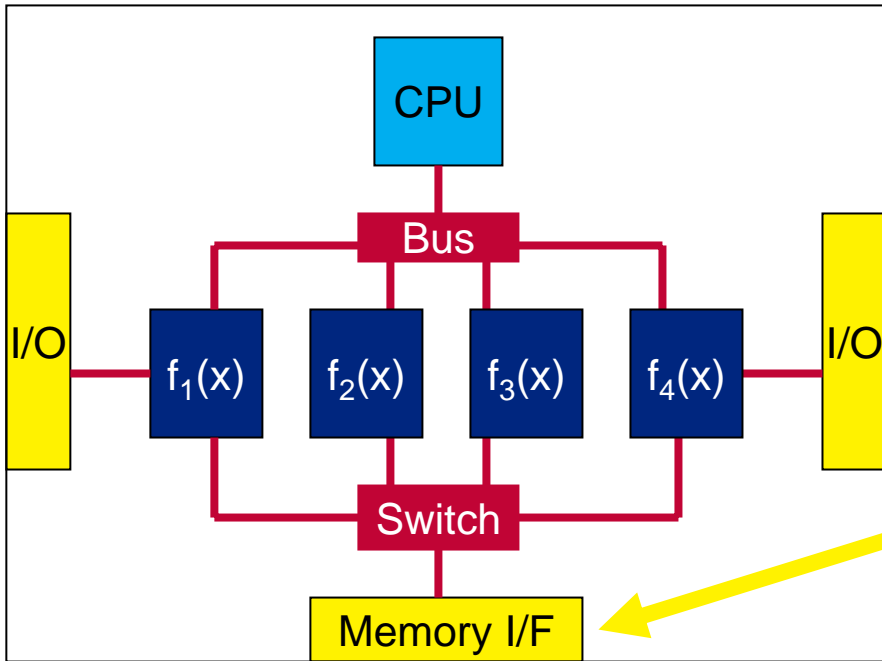
ImageStream in a Video System



ImageStream in a Video System



I/O



4. I/O module

I/O Modules

- I/O Modules are very often standards-based
- It is very common (and sensible) to use pre-verified IP
 - Parameterised to address all the common configurations
- RTL is the correct design language
 - But this poses a problem for system design
 - It forces a cycle-based simulation environment
- A behavioural model is therefore necessary
 - Abstraction of the interface (including mechanisms for synchronisation) allows this
 - SystemC allows the model to be expressed with varying levels of complexity
 - Trading off timing accuracy with ease of development and simulation speed

Levels of Modelling for RapidIO

Transaction Level Modelling

- Natural to write
- MHz simulation speeds
- Good for correctness

```
SC_MODULE(RIOModel)
{
    ALT_AVALON_ST_OUTPUT< int32 > dout;

    void behaviour()
    {
        int32 a;

        while (true) {
            // construct some data for the fft
            for (int i = 0; i < 32; i++) {
                a = rand(100);
                dout.write(a, (i==31), FFT, 0);
            }
            dout.write(0, true, FFT, 1); // EOD
        }
    }
    SC_CTOR(RioModel) { SC_THREAD(behaviour); }
};
```

Cycle Accurate Modelling

- Lower level state machines
- kHz simulation speeds
- Good performance feedback

```
SC_MODULE(RIOModel)
{
    sc_in_clk clock;
    ALT_AVALON_ST_OUTPUT< int32 > dout;
    int i;

    void behaviour()
    {
        int32 a;

        a = rand(100);
        dout.write(a, (i==31), FFT_DESTID, 0);

        if (i == 32)
            dout.write(0, true, FFT_DESTID, 1);
    }
    SC_CTOR(..) { sensitive << clock; }
};
```

ALTERA®

Summary



Summary

- By way of summary lets look at an industry score card

- Note: This is a pragmatic view gained from two sources
 1. Customer interaction
 2. Internal development experience

Industry Score Card

	<i>Productivity</i>	<i>Fast Simulation</i>	<i>Debug</i>	<i>Portability & Flexibility</i>	<i>Timing Closure</i>
Control	Green	Yellow	Green	Green	Green
Compute	Yellow	Red	Red	Yellow	Yellow
Interconnect	Yellow	Red	Yellow	Green	Green
I/O	Green	Red	Yellow	Green	Green

Control Module – Summary

- **Productivity**
 - **Good:** C based already and good tools for instantiation in the system
- **Fast simulation**
 - **Mediocre:** Fast simulation is supported for stand-alone instances
- **Debug**
 - **Good:** Processor-like development environments supported
- **Portability & Flexibility**
 - **Good:** Recompile of software without H/W recompilation supported & controller IP is typically device agnostic
- **Timing Closure**
 - **Good:** Standard IP that has been optimised for different platforms
- **Comment**
 - A new requirement is a SystemC wrapper to connect to rest of system

Compute Module - Summary

- Productivity
 - **Mediocre:** Many designs done in RTL
- Fast simulation
 - **Poor:** requires further adoption of high-level synthesis to give common source for simulation and synthesis
- Debug
 - **Poor:** not much support for in-system debug (in simulation or H/W)
- Portability & Flexibility
 - **Mediocre:** IP providers do a good job but RTL designs often require re-work
- Timing closure
 - **Mediocre:** The re-work required for porting often requires timing work
- Comments
 - Use a methodology that provides a simulation and synthesis view (ideally from the same source)
 - Use abstracted interfaces that include mechanisms for system synchronisation
 - Both the above can be expressed in SystemC, which may only be a wrapper

Interconnect – Summary

- **Productivity**
 - **Mediocre:** Good tools for automatic wiring, including arbitration and other functions but there is little use of higher protocols
- **Fast simulation**
 - **Poor:** Support for full system simulation just emerging for some applications though some use of high-level protocols emerging
- **Debug**
 - **Mediocre:** Good low level visibility and some transaction analysis tools but poor system-level debug support
- **Portability & Flexibility**
 - **Good:** Standards are not device specific, and generally very flexible
- **Timing closure**
 - **Good:** Standards don't compromise performance
- **Comments**
 - Adopt (application specific) protocols over standardised interconnect
 - Complement model-based tools with language-based methodology
 - Network on Chip?

I/O – Summary

- Productivity
 - **Good:** Widespread adoption of pre-verified IP
- Fast simulation
 - **Poor:** Little use of high-level models, most verification is cycle based
- Debug
 - **Mediocre:** Emerging visibility of system-level characteristics and diagnostics, but FPGAs can offer a lot more
- Portability & Flexibility
 - **Good:** IP for I/O is often very flexible and ported to all relevant devices
- Timing closure
 - **Good:** Pre-verified IP includes closing timing on the target devices
- Comments
 - Abstract models allowing fast system verification are essential to increase productivity
 - SystemC can provide a good basis for this, but this can be transparent to the user

Industry Score Card

	<i>Productivity</i>	<i>Fast Simulation</i>	<i>Debug</i>	<i>Portability & Flexibility</i>	<i>Timing Closure</i>
Control	Green	Yellow	Green	Green	Green
Compute	Yellow	Red	Red	Yellow	Yellow
Interconnect	Yellow	Red	Yellow	Green	Green
I/O	Green	Red	Yellow	Green	Green

Conclusion

- Breaking the system into its parts allows us apply the correct “synthesis” technique to each part
- This will have maximum impact if we apply some rules so the parts work together at all levels

- SystemC provides a very good framework
 - At Altera we are seeing benefits
- And with care, the complexities of SystemC can be hidden
 - Either completely by use of system design tools
 - Or from all but the system designer

ALTERA®

Thank you...

