Presented at IPC Printed Circuits Expo®
SMEMA Council APEX®
Designers Summit 04

FPGA on Board

Rick Stroot
Mentor Graphics
The Netherlands

Abstract
Whilst the number of new ASIC designs has decreased over the last couple of years, there has been a dramatic increase in the number of FPGA designs implemented. Not only have the number of designs increased rapidly, the complexity and also the size of these devices have grown over this period. In the early 1980s, the first PLD devices had around 300 gates, while today's FPGAs exceed two million gates. Along with the increasing FPGA gate count there has been a corresponding increase in the number of available I/O pins such that there are over 2000 pins available on the largest BGA packaged FPGA today. As FPGAs continue to grow larger and more complex, it seems that the design tools used by the design engineers become increasingly unsophisticated. Which begs the question: How are designers going to place these large components on to a PCB in an automated and consistent way?

Since the problem spans the two processes of FPGA and PCB design, it is difficult to decide where a solution should be created. Central to this discussion are the problems of symbol creation and I/O assignment, and given the fact that it concerns the two processes, how to keep the information consistent between them. This paper discusses the problems and possible solutions to integrate today's large FPGAs on a PCB, where subjects like scalability to larger/smaller devices, corporate library structures and the origin of the I/O constraints will be discussed. This paper also addresses some ways to help overcome these FPGA integration problems by using the right tools.

Introduction
With the Non Recurring Engineering (NRE) costs of an ASIC design becoming increasingly expensive and the advances in FPGA technology, the tendency for more and more ASIC designs to be implemented using FPGA technology is seen. Since it takes at least a couple of months to design and verify an ASIC, the board designers usually have enough time to integrate the ASIC device onto the PCB. With the trend of moving to FPGA design technology, we see that it takes a quarter of the time to create a prototype of the FPGA ready for testing on the board. This means that the PCB designer now finds himself on the critical path of the design trying to integrate this increasingly complex FPGA device onto the board.

The ASIC NRE costs and long design times cause the pin-out to be fixed early in the ASIC design cycle. In the case of an FPGA there is more flexibility to change the I/O assignment during any part of the design cycle. The side effect of this flexibility is that the I/O design may be changed during both the FPGA and PCB design cycles, causing additional iterations of both the FPGA and PCB. This situation demands a tight interaction between the FPGA and PCB design teams and the tools that are being used.

An aspect that is inherent to the FPGA design flow is the level of automation. An ASIC designer is typically more used to tools that require some form of scripting or setup in order to do the job for them. However, today's FPGA designers demand a high level of automation from the tools. This automation should offer the FPGA designer a solution that enables him to concentrate on the design, verification and debug of the FPGA. As detailed in the remainder of this paper, this high level of automation is difficult to achieve between the tools used for FPGA design and PCB design.

Two worlds, Two Flows, Two Designers
Traditionally, the design of an FPGA and a PCB has been viewed as two different worlds using two different designers or teams of designers. Integrating the FPGA on the PCB has fallen to the PCB designer or team. In the early days of small gate count devices, both design processes were based on schematic capture tools often provided by the same vendor. With the advent of multi-million gate devices, the use of HDLs now dominates the FPGA design process, whilst schematic capture is still predominant for PCB design. This causes a situation where PCB and FPGA designers are using different toolsets for their part of the design. Integrating a large complex 2000 pin FPGA on a dense PCB, while managing high speed and other PCB layout constraints, is a difficult job on its own. Therefore, bringing these two worlds together is the key to getting the integration job done.

The first problem to solve is to bring the two teams of designers together. There are situations in which one designer does both the FPGA and board design, and this could be considered as the ideal situation for integrating the FPGA on a board. However, the reality is that in the majority of cases there are multiple teams and management levels between the FPGA and board designer, making the joint design task even more complex. Therefore, company management has to acknowledge the

integration of an FPGA on a board as being a high level design problem that can only be solved by making sure both (FPGA and PCB) teams have this issue as a common focal point.

Besides these personal aspects, there is also the problem of two tool sets that do not communicate. Having common tools from one EDA vendor does help, but in the majority of organizations this is not the case. Therefore, it is important to introduce a new technology that can bridge the gap between the FPGA and PCB design environments. This technology should enable the designers to jointly work on the integration of an FPGA on the board facilitated by built-in data management capabilities to keep track of any changes made by either one of them. This data management functionality would also assist in situations where the two design teams are in different locations, as is often found in large organizations. Therefore, the key to the problem of getting these two tool sets to work together is a tight interfacing between the tools in both flows. This can be achieved by Mentor Graphics' BoardLink Pro product (Figure 1), which enables the designer to exchange the I/O design (and other properties related to integrating the FPGA on board) between the two tool sets.

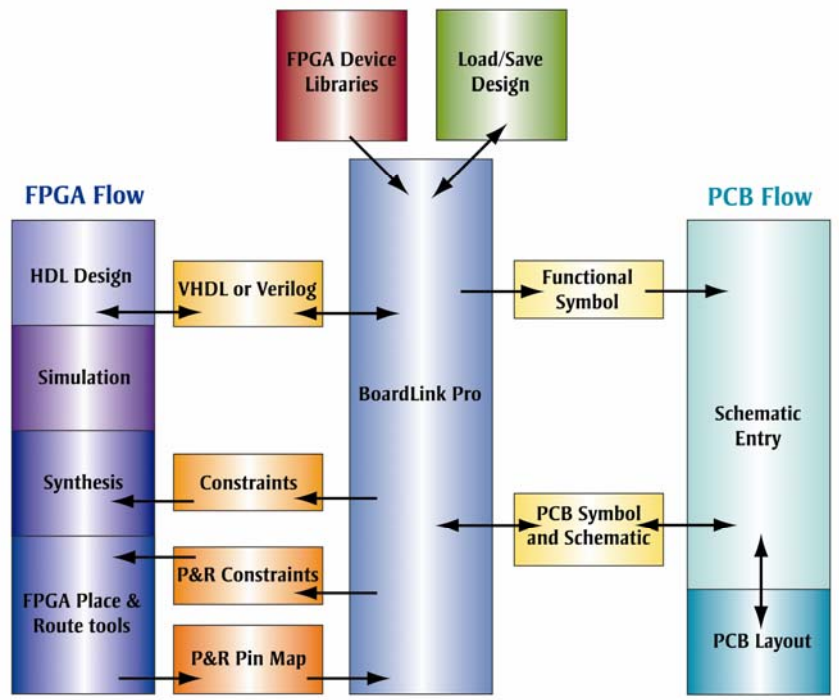


Figure 1 - Forming a Bridge between Two Worlds

Today's solution offers a basic integration between the FPGA and PCB flows, in the future, users will see tighter real-time integration allowing both designers to work on the I/O assignment concurrently. The backplane of this integration is an environment where FPGA, PCB layout and schematic entry tools are all working on the same set of data.

I/O Design

The term "I/O design" means assigning the pin-out of an FPGA. The result of this process is that the I/O ports of the top level FPGA design are assigned to dedicated pins on the selected FPGA device. Since there are several constraints that influence this I/O assignment process, we should first spend some time explaining them before discussing the I/O design process itself. The constraints can be divided into two groups, the FPGA constraints and the PCB layout constraints.

The Constraints

The FPGA is constrained by the timing requirements of the design (timing constraints), the capacity and architecture of the device (routing constraints) and the I/O standards applied to the I/O buffers (I/O constraints). The introduction of configurable I/O buffers has meant greater flexibility within each device to support a wide range of low-voltage signaling standards, but support of these standards imposes constraints on which standards may be used in close proximity to each other. To maximize this flexibility, the devices group signals in to I/O Banks further complicating the assignment rules. Each of these types of constraint influences the I/O assignment.

The other constraints that influence the I/O assignment process are the PCB constraints. Similar to the FPGA constraints, the PCB influences the optimum I/O assignment depending on the number of routing layers available and the orientation of the device on the board (routing constraints). In addition to the routing constraints, the PCB layout may have to meet Signal

Integrity (SI) and timing constraints for the overall system design (SI and timing). Since these SI and timing constraints limit the length, clearance and other physical aspects of the traces on the board, they also influence the location of the pins that these I/O ports are assigned to.

The following list is the total set of constraints that influence the I/O design:

- FPGA timing
- FPGA routability
- FPGA I/O
- PCB routability
- PCB SI and timing

The I/O Design Process

Since these constraints are typically managed by different designers (e.g. FPGA designer, PCB designer, and SI designer) and influence the same I/O assignment process, it is a difficult task to co-ordinate. The situation is complicated further by the priority given to each set of constraints during the design process. For example, if the goal is to have a prototype of the board as soon as possible, then the pin-out must be fixed early in the design process.

Ideally, the PCB layout designer should determine the pin assignment during the PCB layout process so that the PCB constraints are met and the PCB optimized, while all FPGA constraints are automatically applied. In the past, the I/O assignment was done automatically by the FPGA vendor place & route tools with little regard for the PCB requirements. However, with the increasing complexity of the PCB this process needs to be managed by the design team. Therefore, the typical process today is to define these constraints up-front before going into the synthesis and place & route process. Often these constraints are defined in a tool specific constraint file that passes directly into the place & route tools or into the synthesis tool and then forwarded to the place & route tool. Defining the constraints through the ASCII constraint file requires the designer to understand the FPGA I/O pin details and assignment rules (FPGA I/O constraints) before being able to assign an I/O port. The FPGA data book can help in this situation. However, it is still a manual and error prone task where the designer not only has to concentrate on the FPGA I/O assignment but at the same time, has to ensure they do not violate any of the constraints discussed in section 0. Since it is typically the FPGA designer who does this job, they probably are not aware of the PCB layout details and so will not optimize this part of the design. Therefore, the assignment process is dependant upon the knowledge of both the FPGA and PCB designer calling for flexibility in whom and when this task is done.

Maintaining Consistency

So, in order to successfully integrate an FPGA on a board, the I/O assignment has to be possible at any time during the design cycle and by any member of the design team. It is therefore difficult to keep the I/O design consistent amongst the design team and design processes they operate. Keeping the I/O design consistent means that whenever one designer changes the pin-out, the changes should be automatically propagated to the rest of the design tools involved in FPGA integration. For example, if the FPGA designer decides to change the pin-out, this influences the way the traces are connected to the FPGA on the PCB layout. Also, if the PCB designer decides to swap two pins, it influences the internal routing of the FPGA. Having one integrated design environment for FPGA and PCB design would be the ideal solution. It is important to find a vendor who can provide tools in both areas and currently offers the capability of exchanging crucial data with each of the design processes. This is a first step in the integration, since the final goal is to offer a system design environment that seamlessly works as one environment while offering dedicated technology for each of the design areas.

Automating the Process

We have seen that the process of I/O design is complex and may be done in multiple places by many designers while trying to cope with all the surrounding constraints. Since this is often a manual and error prone task, it would reduce the risk and costs to automate this task as much as possible. As mentioned earlier, one integrated design environment for both FPGA and PCB design would be ideal but not practical. Due to the fixed architecture of the FPGA device, it is relatively easy to automate the place and route process, and manage the timing and I/O constraints. The PCB layout process on the other hand is still difficult to totally automate since there are simply too many variables to take into consideration while creating the layout. During PCB layout, the designer typically routes the board while trying to avoid violating any of the timing, signal integrity and routability constraints. Therefore, it is obvious that any tool designed to integrate an FPGA on a board should focus on automating the FPGA constraints management, all while allowing the designer to concentrate on difficult to automate tasks within the PCB layout. During this process the I/O design has to be tested constantly against all the FPGA I/O, timing and routability constraints. The tool should have a built-in library containing all the necessary device information to allow the constraints to be applied, as well as a good integration to all the tools in the FPGA and PCB design flows.

Scalability

Today's electronic designs must meet the conflicting needs of high complexity, lowest possible cost and the shortest time to market. As FPGAs give designers flexibility in implementing their design with reasonably low cost, they increasingly become a central component in electronic designs. Since FPGAs come in different die and package sizes, one can pick the best FPGA for each design. However, while the electronic product evolves the design inside the FPGA may grow until it will not fit into the selected FPGA anymore. Similarly, a larger FPGA than necessary may be selected since the complexity and size of the design is not known yet. Using a larger device than necessary does increase the cost of the total design. Therefore, it is common practice to reduce the size of the selected FPGA once the design is completed. This means that scaling up or down to a larger or smaller device during the design phase is a very common step. Since each of the pins on an FPGA device may have special properties (as we have discussed earlier and defined as FPGA I/O constraints), a designer needs to take the migration to a larger or smaller device into account when assigning the I/O pins. When the designer decides to switch to a larger or smaller FPGA device he does not have to re-assign the I/O. Since changing the I/O design always causes an additional re-spin of the board design, it is crucial to the designer to avoid this step, if it is not really necessary. See Figure 2.

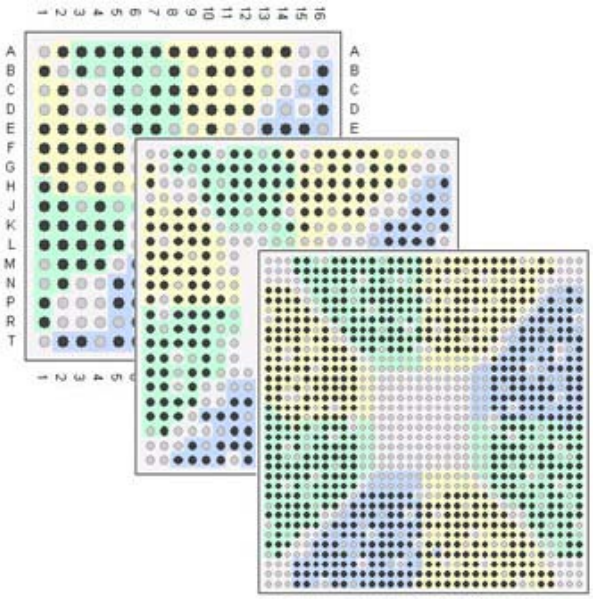


Figure 2 - Scalability of devices

Symbol Creation

As seen in the previous section, one problem in integrating the FPGA on a board is completing the I/O design such that both the FPGA and PCB are optimized. The second problem is the creation of the FPGA data necessary to complete the board layout process. In order to start the board layout process, one has to have a netlist of the board connectivity created using a schematic capture tool. The schematic capture tool contains a symbol representation for each of the components on the board along with the circuit connectivity. This means that in order to integrate the FPGA device onto a board, it is necessary to have a schematic symbol created and connected to the appropriate components on the board.

Design Dependency

For conventional components with fixed pin-outs, the symbol is typically created once by the librarian or designer and stored in a corporate library. Each pin on the schematic symbol of such a conventional component has a dedicated function or signal that is fixed. In the case of an FPGA, the designer has the flexibility to assign a different function to each of the pins, depending on how the FPGA has been designed. This means that although the company librarian can create a generic symbol for each FPGA device, it is still unclear which signal is going to be assigned to each of the pins of this component. In this case, each of the pins needs to have a generic name like IO1, IO2 or PAD1, PAD2 and it is the PCB and FPGA designer's task to know which signal to assign internally and externally to a specific pin.

The ability to create a symbol with the FPGA I/O port function or signal names attached to the pins of the symbol would simplify the task of creating the schematic for the PCB designer. However, the problem with such a symbol is that it is no longer generic and therefore becomes design dependent. It is because of this design dependency that the task of creating such a symbol shifts from the librarian to the designer. Assuming that an organization has ten FPGA designers doing four designs a year using four different types of FPGAs, this means that the amount of work creating these symbols increases from four

generic symbols per year by the librarian to 4x10=40 symbols a year by the designers. This would not be a problem if the process of creating such symbols were automated. However, all too often these symbols are created manually.

A couple of years ago the FPGA pin count was around 300 pins per device. However, today using Ball Grid Array (BGA) packages, the devices go up to 2000 pins. Creating a single symbol to represent a 2000 pin device would not be practical as the symbol would be too big to fit on even the largest schematic sheet. This forces the designer to fracture his symbol into multiple smaller parts. These smaller parts must then all link back to the single FPGA component on the board. Although there are ways to fracture the symbol based upon the empty FPGA device pin data, the FPGA designer may want to divide the symbols based upon the separate functions created inside the design. This means that the fracturing scheme will also become design dependant and increases the demand to use design dependant FPGA symbols.

One thing that increases the demand for having a local FPGA symbol but decreases the amount of work involved is the fact that often not all the I/O pins of an FPGA are being used in a design. This means that designers could create smaller design dependant symbols having only those physical pins that do have a signal assigned to them. The other unassigned pins can be left unconnected or be automatically connected to power or ground without having to add them physically to the schematic symbol.

Considering that it typically takes an engineer a day to create a 200 pin schematic symbol manually, this goes up to the approximately ten days for today's 2000 pin symbols. Add the extra demand of having a design specific set of fractured symbols, then the amount of time taken to create the symbols increases, and as it is a manual task, so does the probability of there being errors in the symbols. Therefore, it becomes clear that something has to be done to automate this process while supporting all of these demands.

Maintaining Consistency

Consistency is a key aspect in integrating an FPGA on a board. The way this consistency is maintained in the PCB environment is by creating a schematic on which the FPGA symbol is connected to the appropriate external signals. Automating the process of creating such a schematic would not only help to reduce the error prone manual task of drawing the schematic, it would also make sure the consistency is maintained between the FPGA and PCB design flow. New design tools use the I/O design information to automatically generate the necessary symbols and schematics and maintains these schematics if the I/O assignment changes. If the changes come from the FPGA side, then the schematic is simply updated and the changes propagated to the PCB layout tools. However, tools exist that also create pin-swap information for the PCB layout tools. The PCB designer is therefore able to change the I/O design by swapping pins of the same type during the PCB layout process. (Figure 3) To maintain the consistency, the information about the swapped pins is then propagated back to the FPGA flow in order to update and rerun the FPGA vendor place & route process. This consistency can be automatically maintained if the tools keeps track of any changes to the I/O design from both the FPGA design tools and the PCB layout tools.

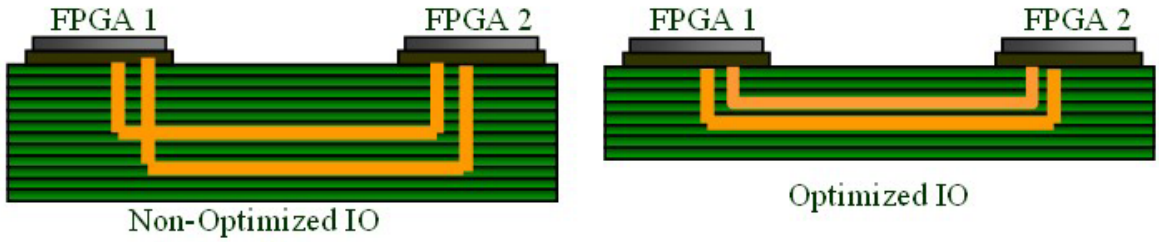


Figure 3 - Reducing PCB Layers using the Ability to Swap FPGA Pins

Today's FPGAs enable the designer to configure the I/O buffer using pre-defined I/O standards, such as LVTTL, LVCMOS, LVDS, etc. The reference voltage levels of these I/O standards can be controlled by external voltage signals, where the group of I/O pins sharing one common reference voltage is called an I/O bank. Once a certain I/O standard and its reference voltage level have been selected inside an I/O bank, then there are constraints on which other I/O standards can be used within the same I/O bank. This is what we referred to in the previous section as I/O banking rules. In the case of an FPGA where some I/O ports demand different I/O standards that will violate the I/O banking rules, then the only way to implement these I/O ports is to assign them in two different I/O banks. This does not only limit the I/O design process, it also limits the pin swap capability since a swap of these two pins should not be possible during PCB layout. The limitation of this swap capability based upon the I/O banking rules is called swap rules. In the PCB world, these swap rules are implemented by grouping together the pins that can be swapped with each other, into a swap group. Since the selection of the I/O standards and reference voltage levels is based on the design, these swap groups are also design dependant.

Corporate Library

In the previous section we discussed the demand for creating design specific symbols. These symbols are typically stored in the local project directory of the PCB design since they are created by the designer who usually does not have permission to store them in the corporate library. The majority of companies have such a corporate library structure in which their component information is stored. This corporate library is typically managed by a librarian, who creates the symbol and maintains all the properties associated with it. These properties form the relationship between the component and the rest of the systems surrounding the electronic design environment. The surrounding systems ensure the board can be produced and the components are in stock when the production process starts. Because of these relationships, companies attach great value to such a corporate component library and very often do not allow designers to use a component that is created locally. This means that quite often the requirements for design dependent local symbols is in contradiction with corporate library policy.

A component is formed by two parts: the graphical schematic symbol and the properties. As discussed these properties are the key elements that need to be stored in the corporate library since they form the link to the surrounding systems. The graphical symbol on the other hand does not have such a critical role. If the benefits of using local FPGA symbols outweigh the drawbacks of using a corporate library symbol to integrate the FPGA on the board, then companies might consider a local symbol to be used in combination with the properties stored in the corporate library.

This would allow the designer to meet the need of creating (or preferably generating) a local, design dependant FPGA symbol that could be fractured based upon design criteria. In addition to the local fracturing such a split between symbol and properties would also allow the use of design specific swap groups stored locally with the FPGA symbol. With the library integration a designer can reuse the corporate library data such as the component properties, while using a local design specific fractured symbol with design specific pin swap criteria. (See Figure 4.)

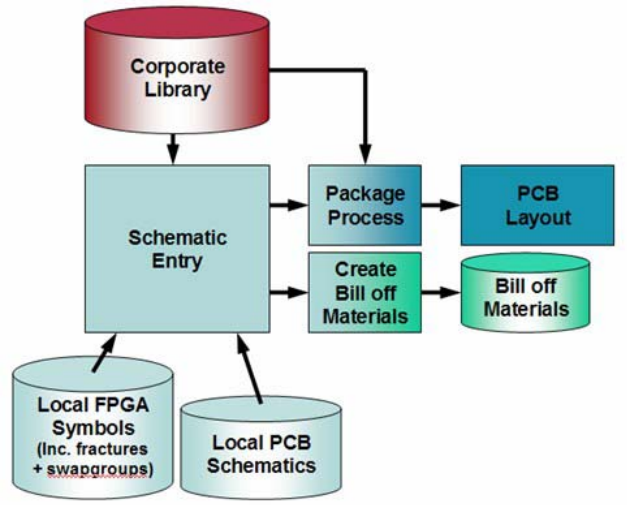


Figure 4 - Using Local Symbols and Fractures in a Corporate Library Environment

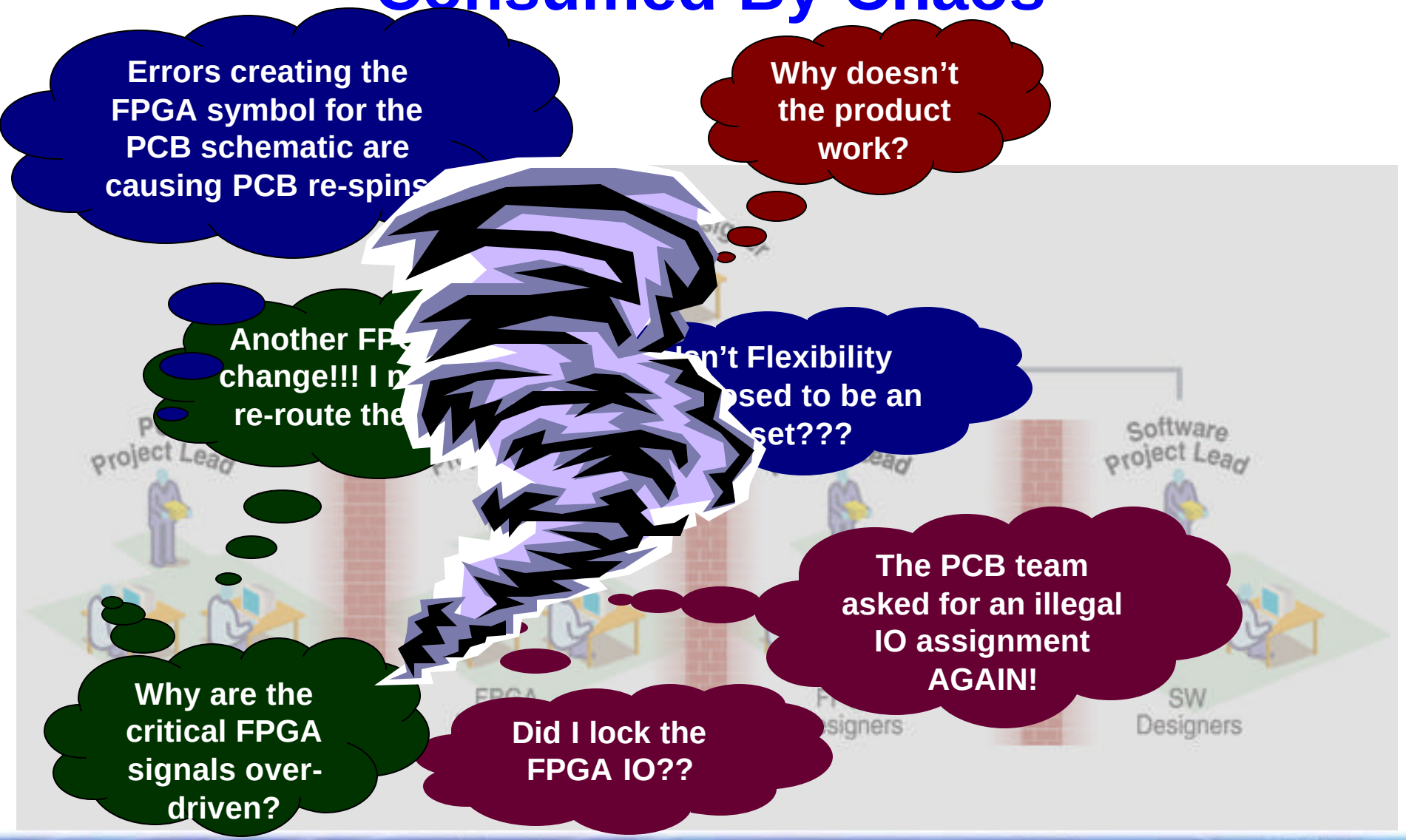
Conclusion

Integrating an FPGA onto a board is not an easy task, as we have seen in this paper. Bringing together the two worlds of FPGA and PCB design is a difficult task, since they both contain dedicated design teams and tools focused on their part of the design. The FPGA on board integration also brings together the set of constraints coming from both worlds to one task: the I/O design. We have seen that it involves creation of symbols and schematics, as well as generating and maintaining the correct constraint files in order to keep the I/O design consistent throughout the entire FPGA and PCB design. The new generation of design tools can form the bridge between FPGA and PCB design flows to overcome the problem of maintaining the consistency between them. It also generates all necessary design dependent symbols and schematics and makes sure that any changes that are being made during PCB layout make their way back into the FPGA flow. Integrating an FPGA onto a board now becomes an automated process, that obsoletes the error prone manual tasks that designers where facing up till now.

Chaos to Opportunity:
FPGA ↔ PCB Integration

Dave Brady
FPGA Advantage
Product Marketing Manager

Consumed By Chaos



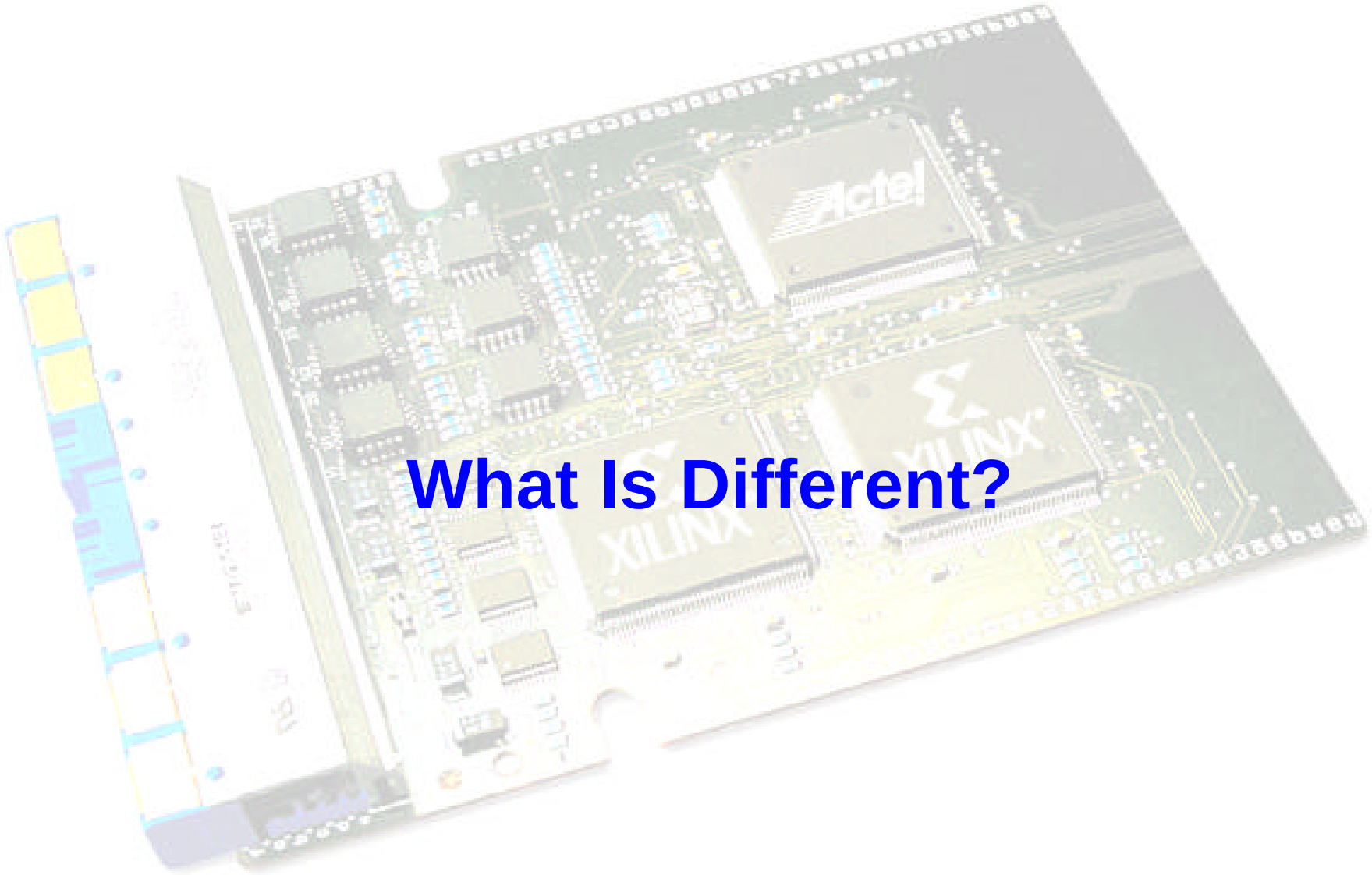
Harnessing Chaos

Attacking FPGA « PCB integration barriers creates competitive opportunities

- **Faster time to market**
- **High-speed FPGAs on high-speed PCB**
- **New manufacturing optimizations**

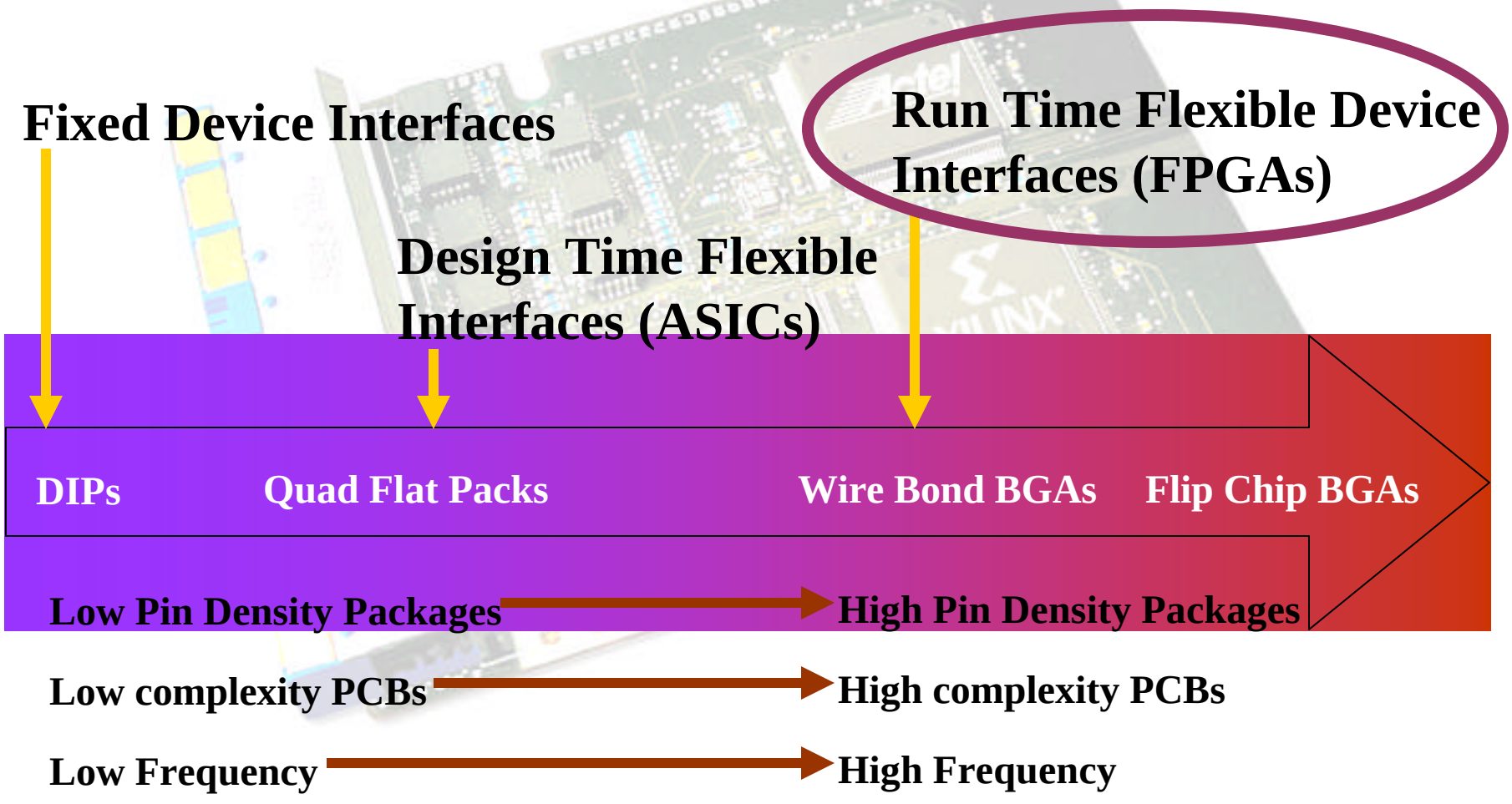
Agenda

- **What is Different?**
- **Business advantages of FPGAs**
- **FPGA « PCB process**
- **FPGA « PCB barriers**
- **FPGA « PCB opportunities**
- **Integration cost model**
- **MGC Solutions**
- **Summary**

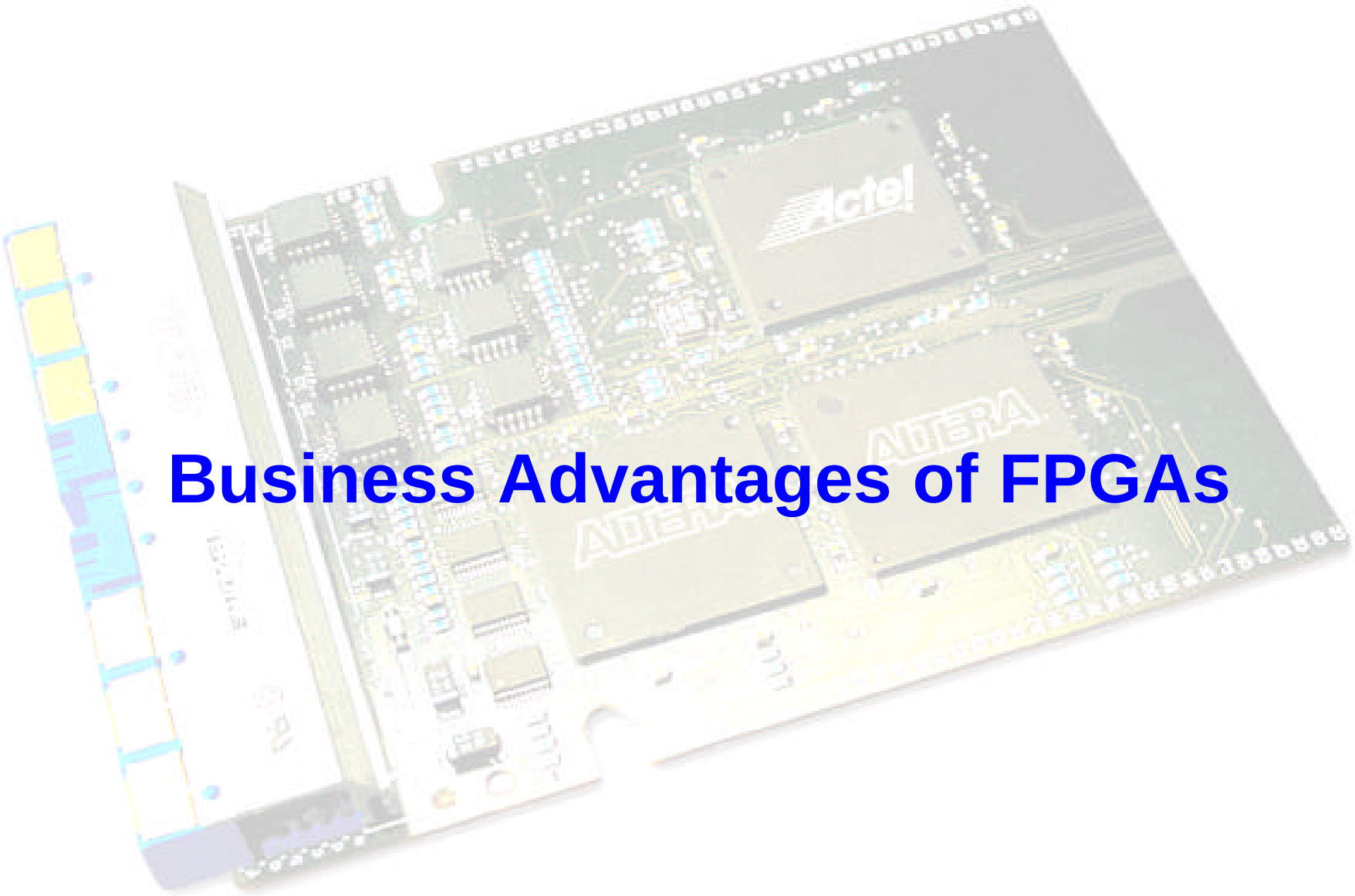


What Is Different?

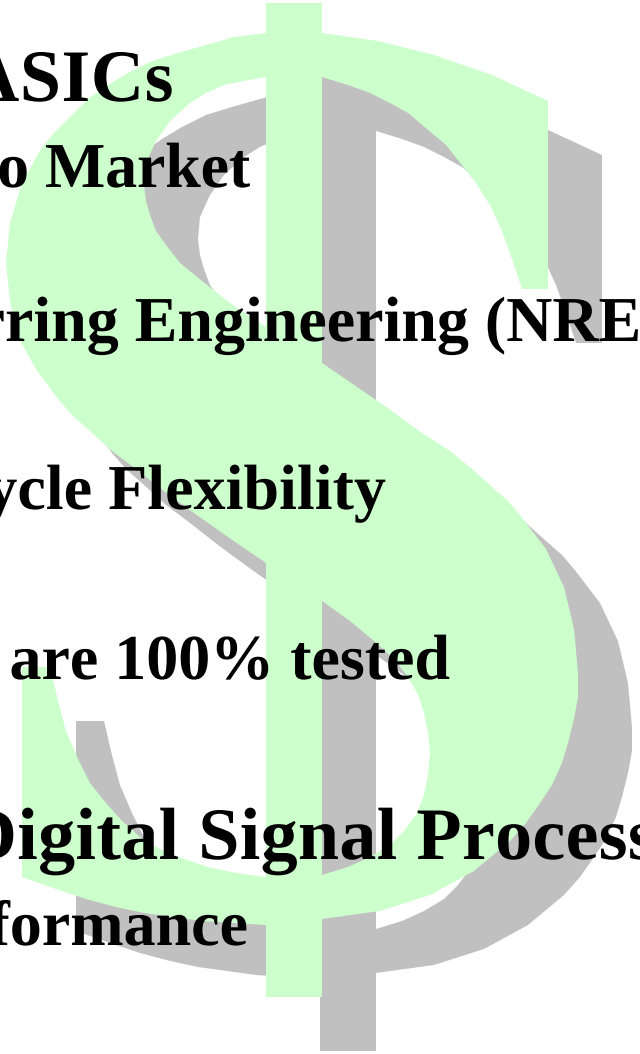
How Is FPGA IO Different?



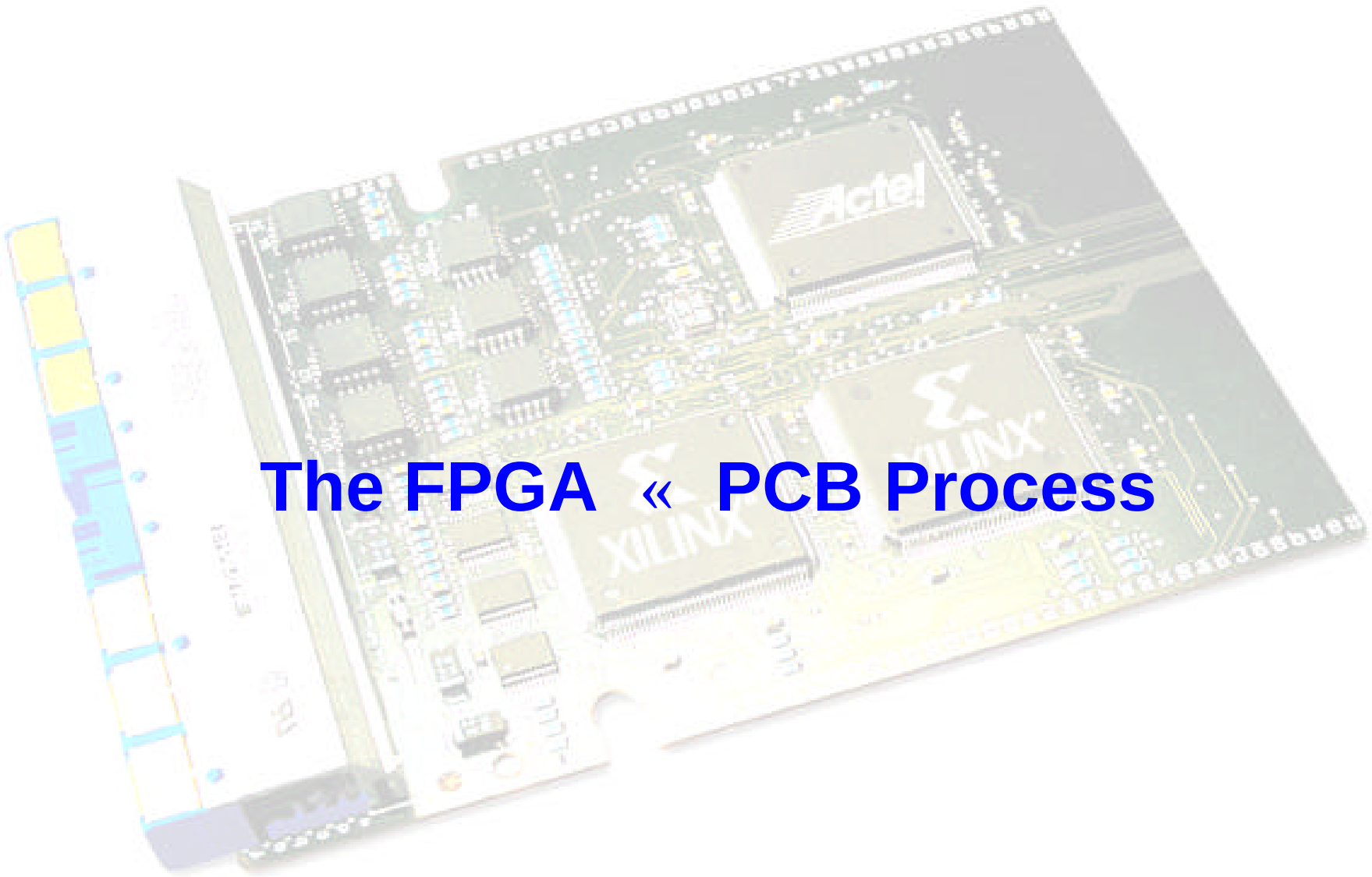
Business Advantages of FPGAs



Why Use FPGAs?

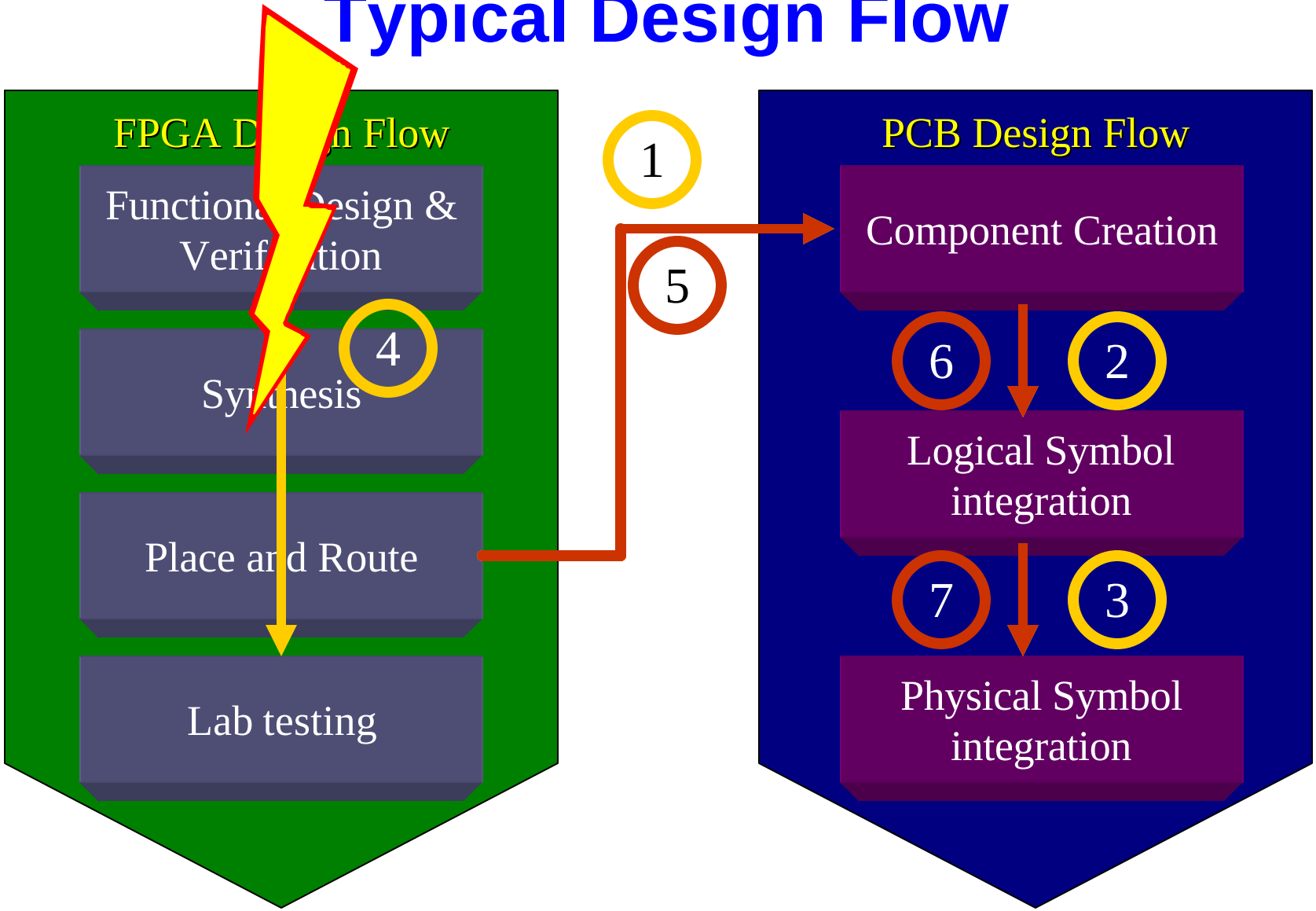


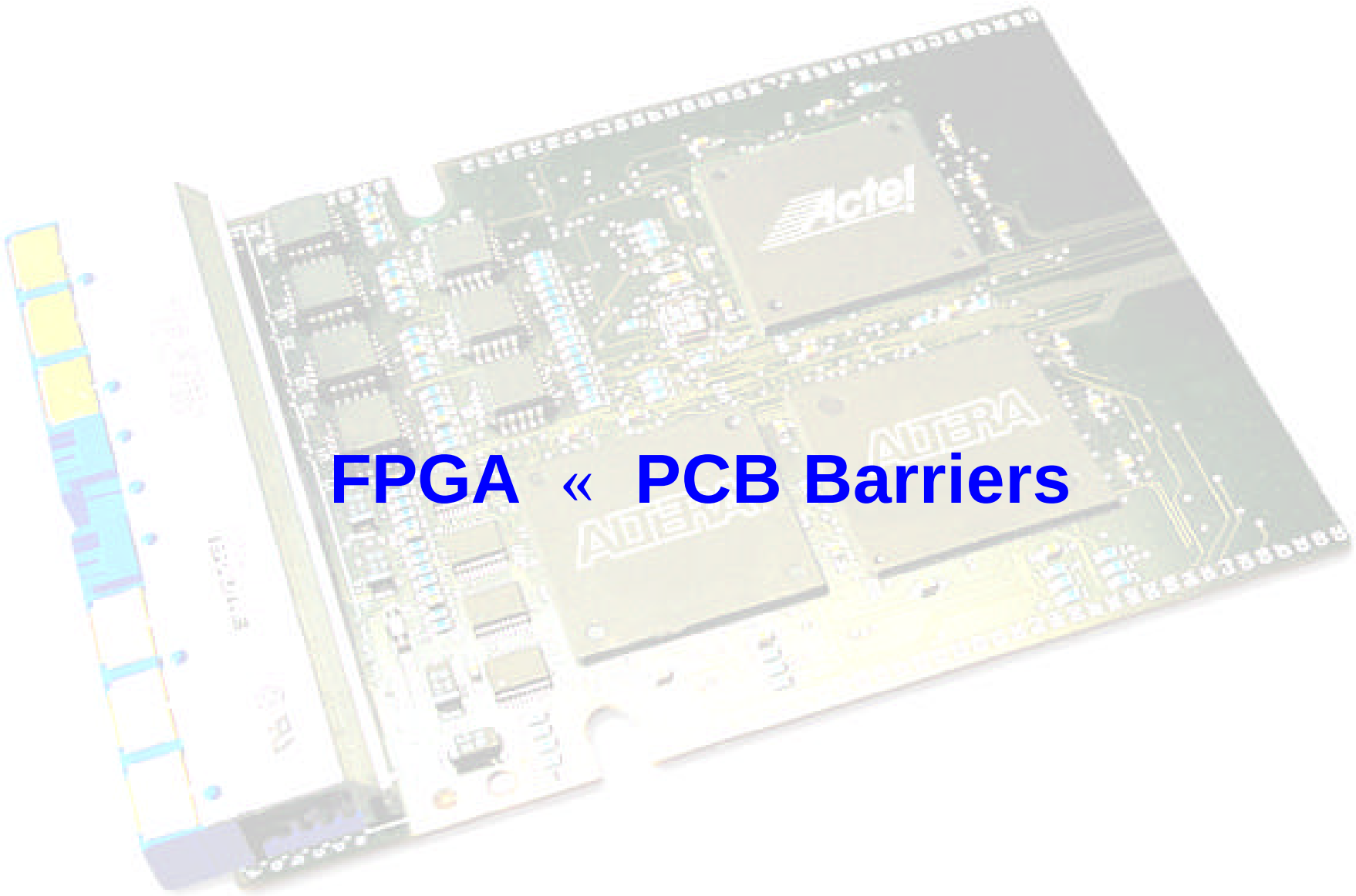
- **Compared to ASICs**
  - **Faster Time To Market**
  - **No Non-Recurring Engineering (NRE) costs**
  - **Late design Cycle Flexibility**
  - **FPGA devices are 100% tested**
- **Compared to Digital Signal Processors (DSP)**
  - **Very high performance**



The FPGA « PCB Process

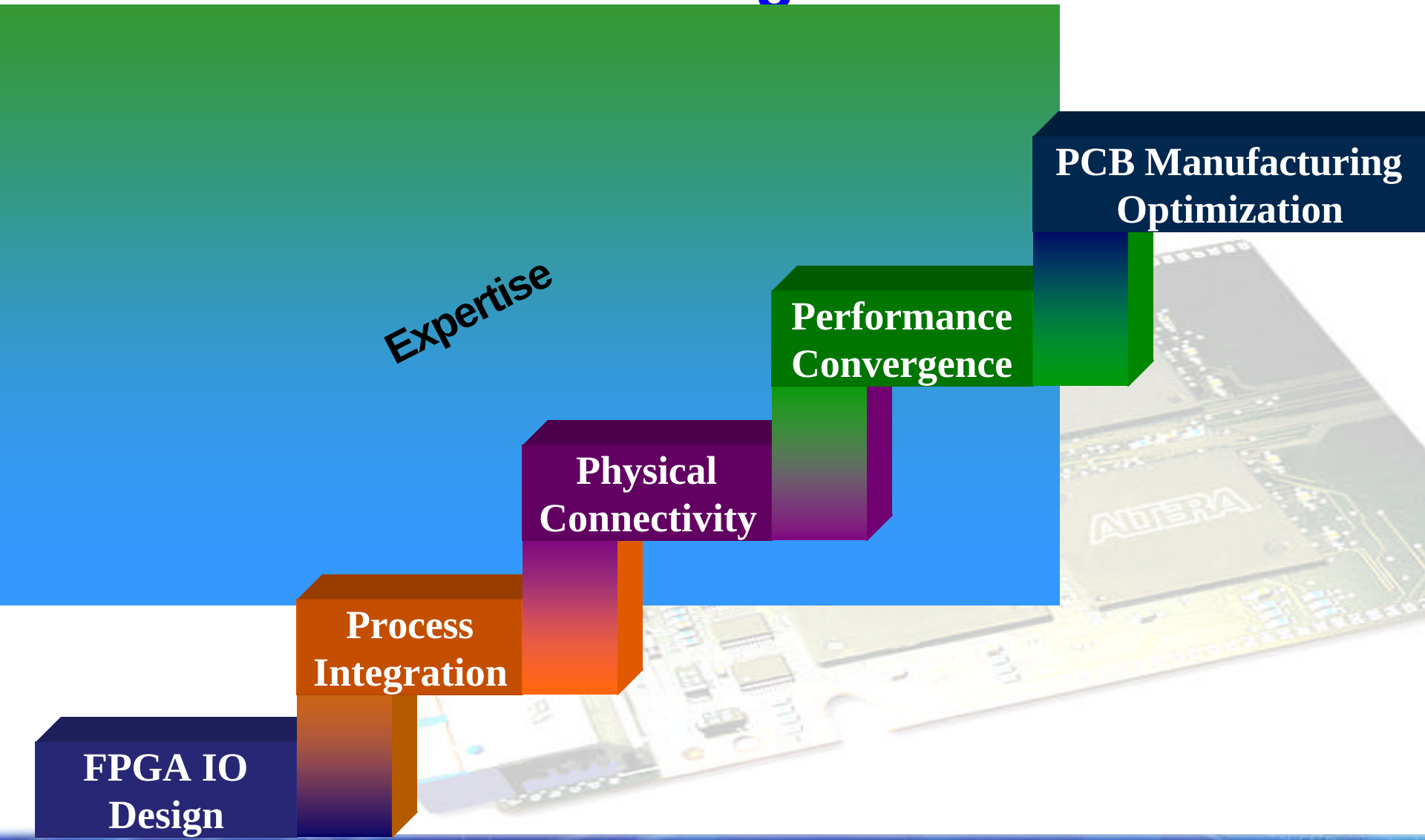
Typical Design Flow

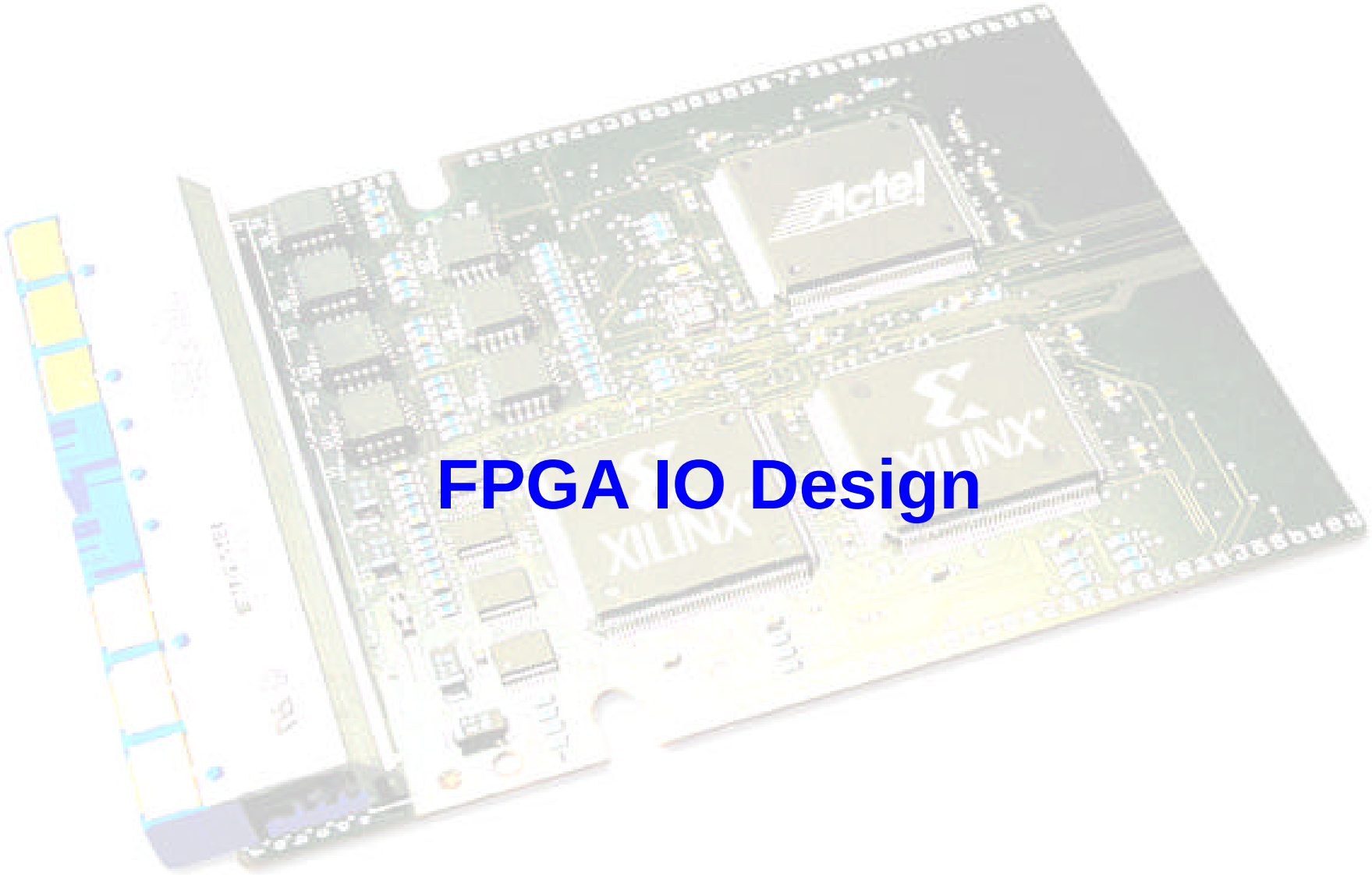




FPGA « PCB Barriers

FPGA « PCB Integration Barriers





FPGA IO Design

Example IO Standards

Single Ended IO Standards

Table 2-30: **Supported Single-Ended I/O Standards**

I/O Standard	Output Source Voltage (V_{CCO})	Input Source Voltage (V_{CCO})	Input Reference Voltage (V_{REF})	Board Termination Voltage (V_{TT})
LVTTL	3.3	3.3	N/A	N/A
LVCMOS33	3.3	3.3	N/A	N/A
LVCMOS25	2.5	2.5	N/A	N/A
LVCMOS18	1.8	1.8	N/A	N/A
LVCMOS15	1.5	1.5	N/A	N/A
PCI33_3	Note (1)	Note (1)	N/A	N/A
PCI66_3	Note (1)	Note (1)	N/A	N/A
PCI-X	Note (1)	Note (1)	N/A	N/A
GTL	Note (2)	Note (2)	0.8	1.2
GTLP	Note (2)	Note (2)	1.0	1.5
HSTL_I	1.5	N/A	0.75	0.75
HSTL_II	1.5	N/A	0.75	0.75
HSTL_III	1.5	N/A	0.9	1.5
HSTL_IV	1.5	N/A	0.9	1.5
HSTL_I_18	1.8	N/A	0.9	0.9
HSTL_II_18	1.8	N/A	0.9	0.9
HSTL_III _18	1.8	N/A	1.08	1.8
HSTL_IV_18	1.8	N/A	1.08	1.8
SSTL2_I	2.5	N/A	1.25	1.25
SSTL2_II	2.5	N/A	1.25	1.25
SSTL18_I (3)	1.8	N/A	0.9	0.9
SSTL18_II	1.8	N/A	0.9	0.9

Notes:
1. For PCI and PCI-X standards, refer to XAPP653.
2. VCCO of GTL or GTLP should not be lower than the termination voltage or the voltage seen at the I/O pad.

Double Ended IO Standards

Table 4: **Supported Differential Signal I/O Standards**

I/O Standard	Output V_{CCO}	Input V_{CCO}	Input V_{REF}	Output V_{OD}
LDT_25	2.5	N/A	N/A	0.500 – 0.740
LVDS_25(1)	2.5	N/A	N/A	0.250 – 0.400
LVDSEXT_25(1)	2.5	N/A	N/A	0.330 – 0.700
BLVDS_25	2.5	N/A	N/A	0.250 – 0.450
ULVDS_25	2.5	N/A	N/A	0.500 – 0.740
LVPECL_25	2.5	N/A	N/A	0.345 – 1.185

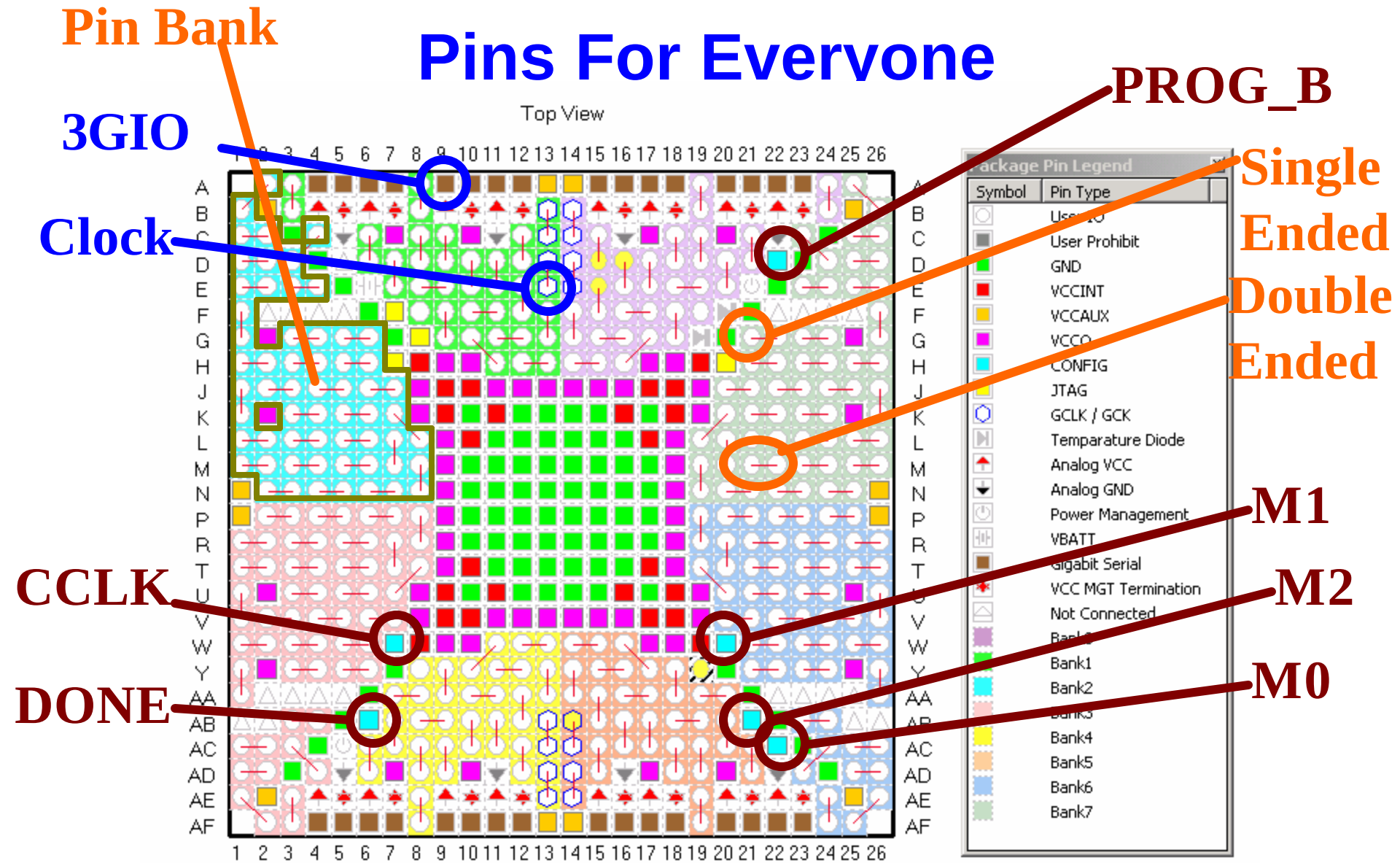
Notes:
1. LVDCI_XX is LVCMOS output controlled impedance buffers, matching all or half of the reference resistors.

DCI IO Standards

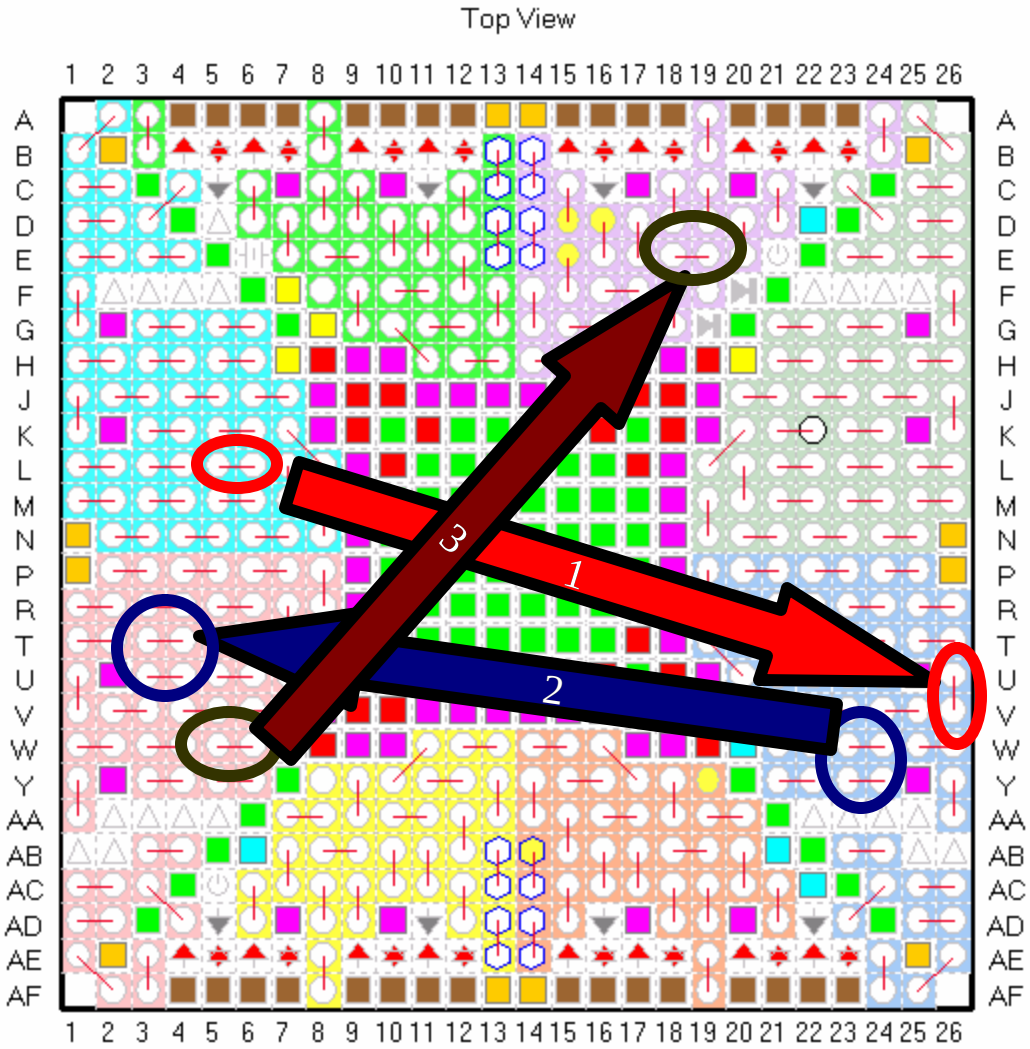
Table 5: **Supported DCI I/O Standards**

I/O Standard	Output V_{CCO}	Input V_{CCO}	Input V_{REF}	Termination Type
LVDCI_33(1), (2)	3.3	3.3	N/A	Series
LVDCI_25	2.5	2.5	N/A	Series
LVDCI_DV2_25	2.5	2.5	N/A	Series
LVDCI_18	1.8	1.8	N/A	Series
LVDCI_DV2_18	1.8	1.8	N/A	Series
LVDCI_15	1.5	1.5	N/A	Series
LVDCI_DV2_15	1.5	1.5	N/A	Series
GTL_DCI	1.2	1.2	0.8	Single
GTLP_DCI	1.5	1.5	1.0	Single
HSTL_I_DCI	1.5	1.5	0.75	Split
HSTL_II_DCI	1.5	1.5	0.75	Split
HSTL_III_DCI	1.5	1.5	0.9	Single
HSTL_IV_DCI	1.5	1.5	0.9	Single
HSTL_I_DCI_18	1.8	1.8	0.9	Split
HSTL_II_DCI_18	1.8	1.8	0.9	Split
HSTL_III_DCI_18	1.8	1.8	1.08	Single
HSTL_IV_DCI_18	1.8	1.8	1.08	Single
SSTL2_I_DCI(3)	2.5	2.5	1.25	Split
SSTL2_II_DCI(3)	2.5	2.5	1.25	Split
SSTL18_I (4)	1.8	1.8	0.9	Split
SSTL18_II	1.8	1.8	0.9	Split

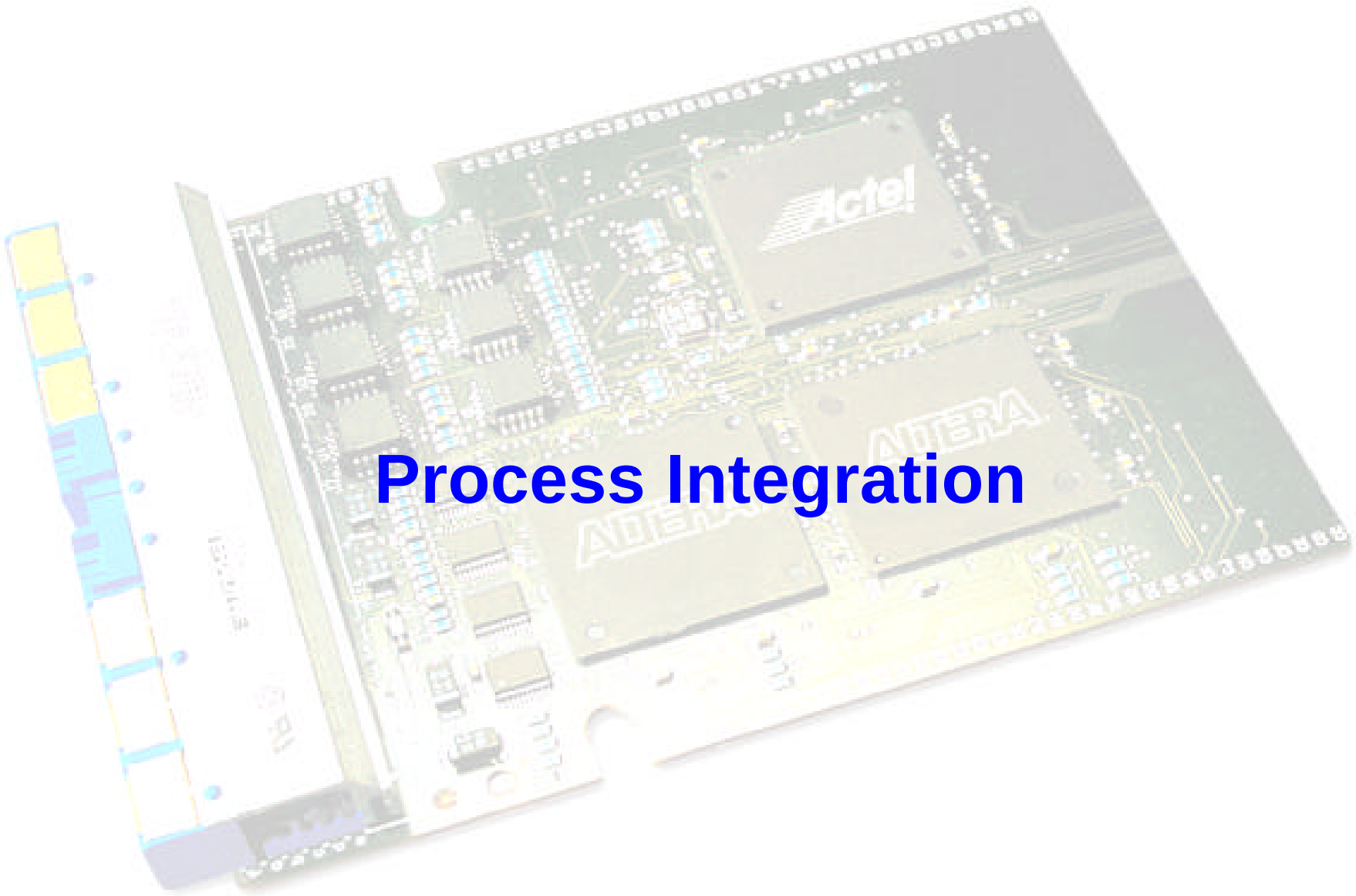
Pins For Evervone



The Domino Effect of Pin Swapping



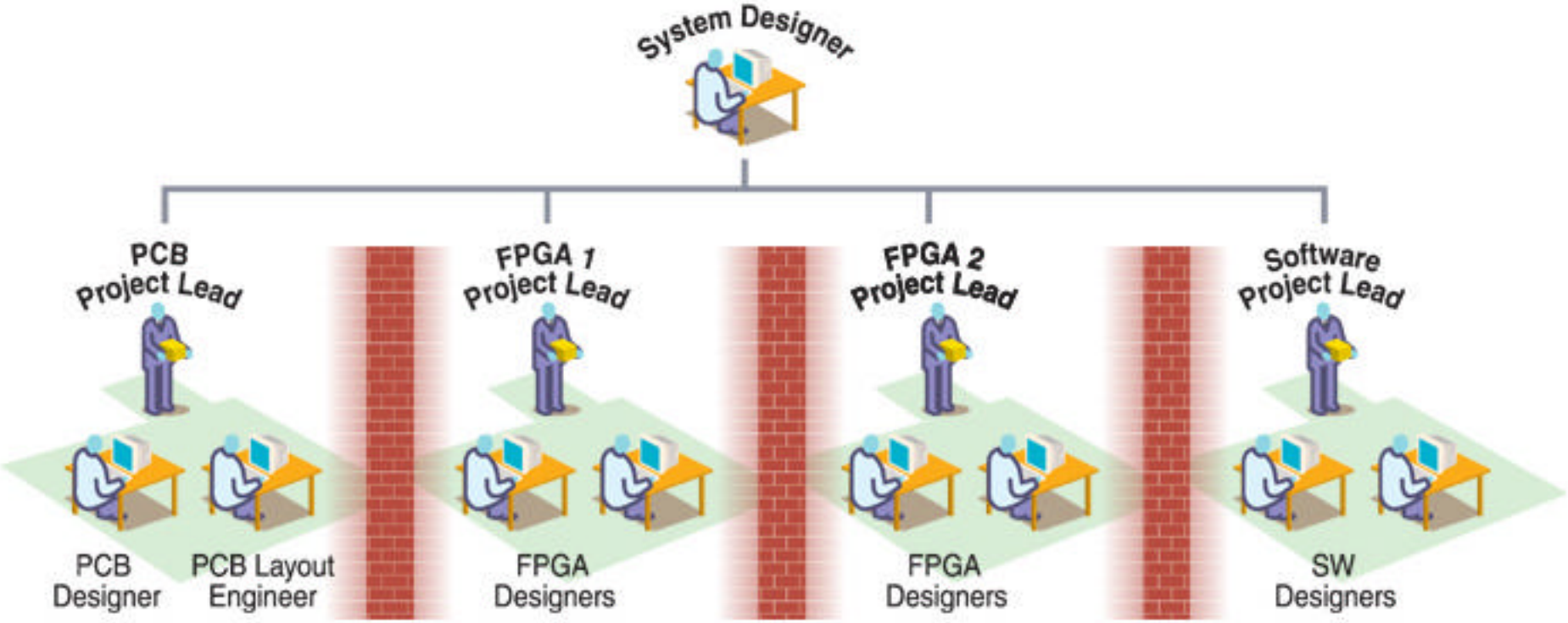
Source: Xilinx



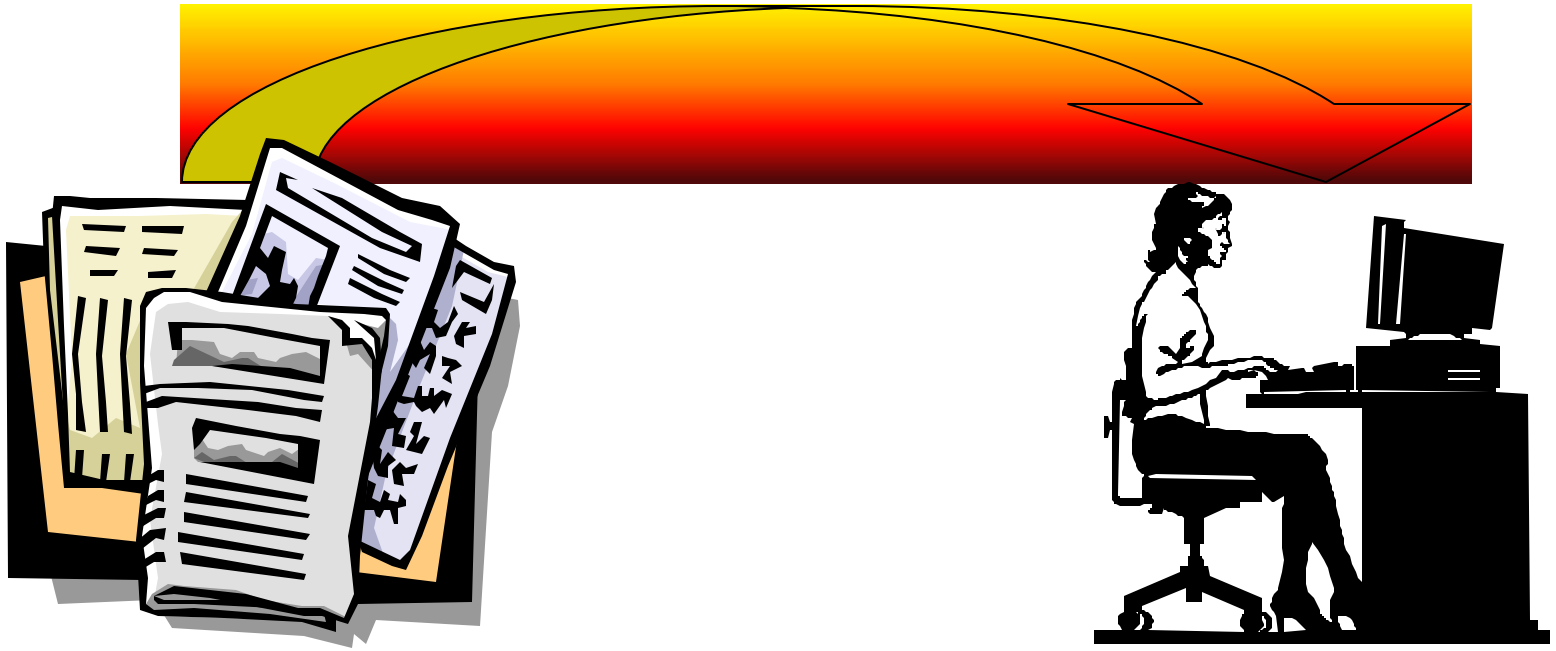
Process Integration

Two Worlds: Two Design Flows

FPGA and PCB design teams typically do not communicate

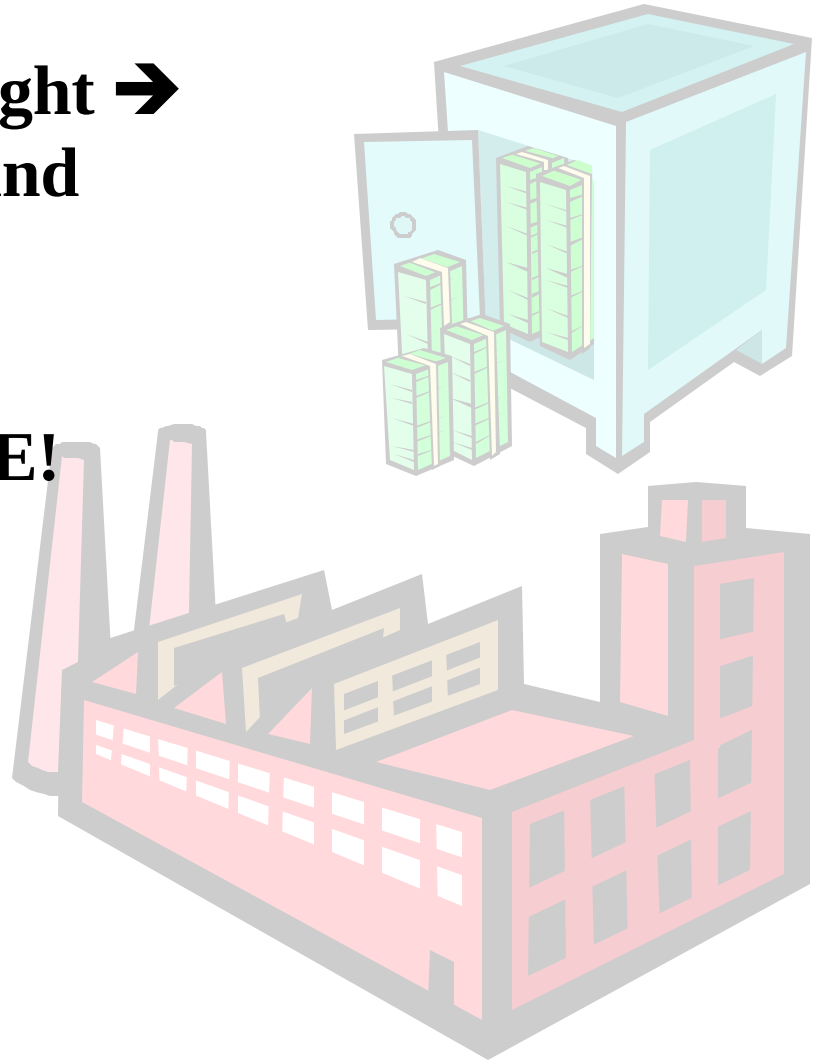


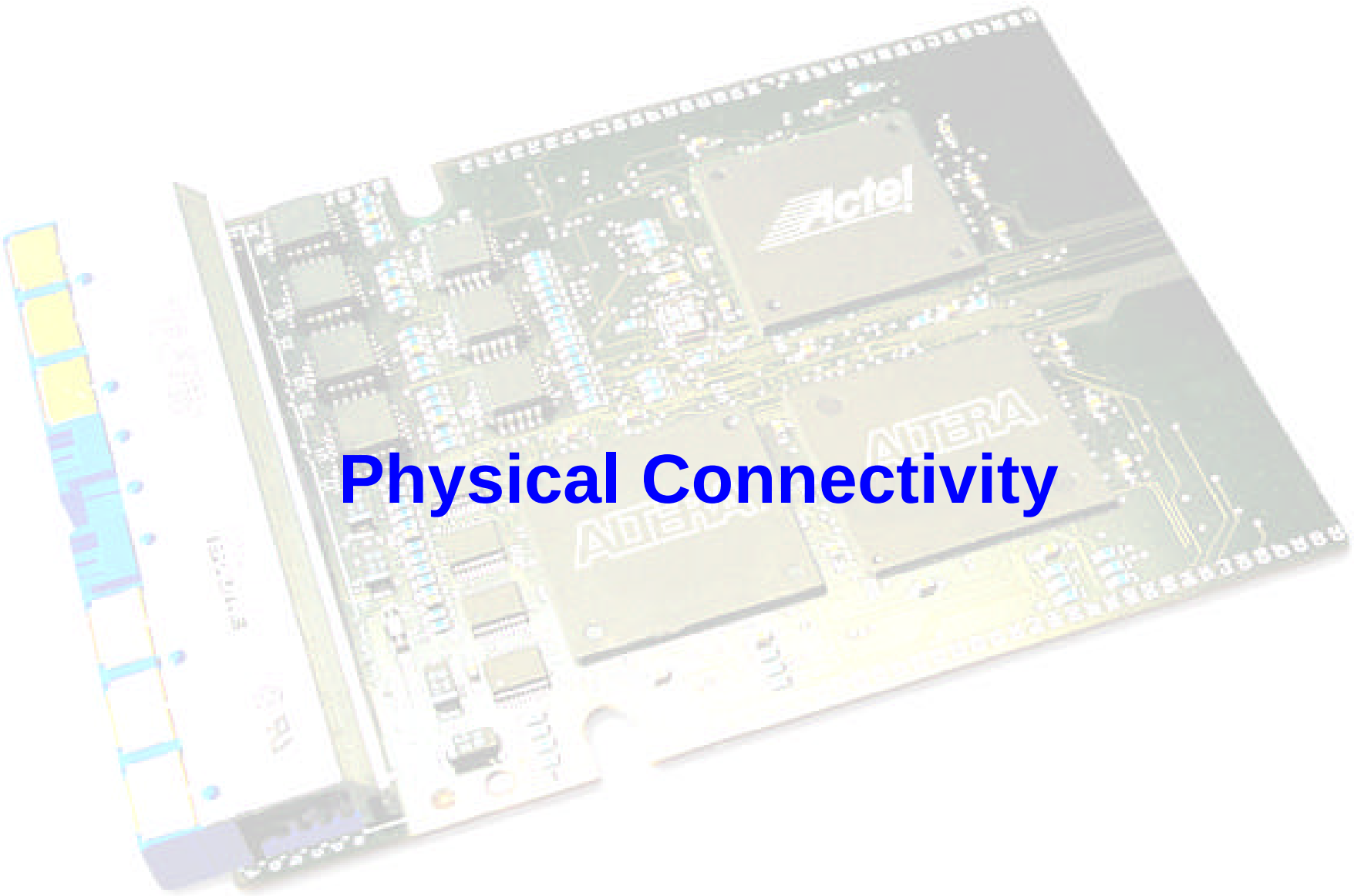
Manual Data Re-Entry Process = ERRORS!



PCB Library Integration Barriers

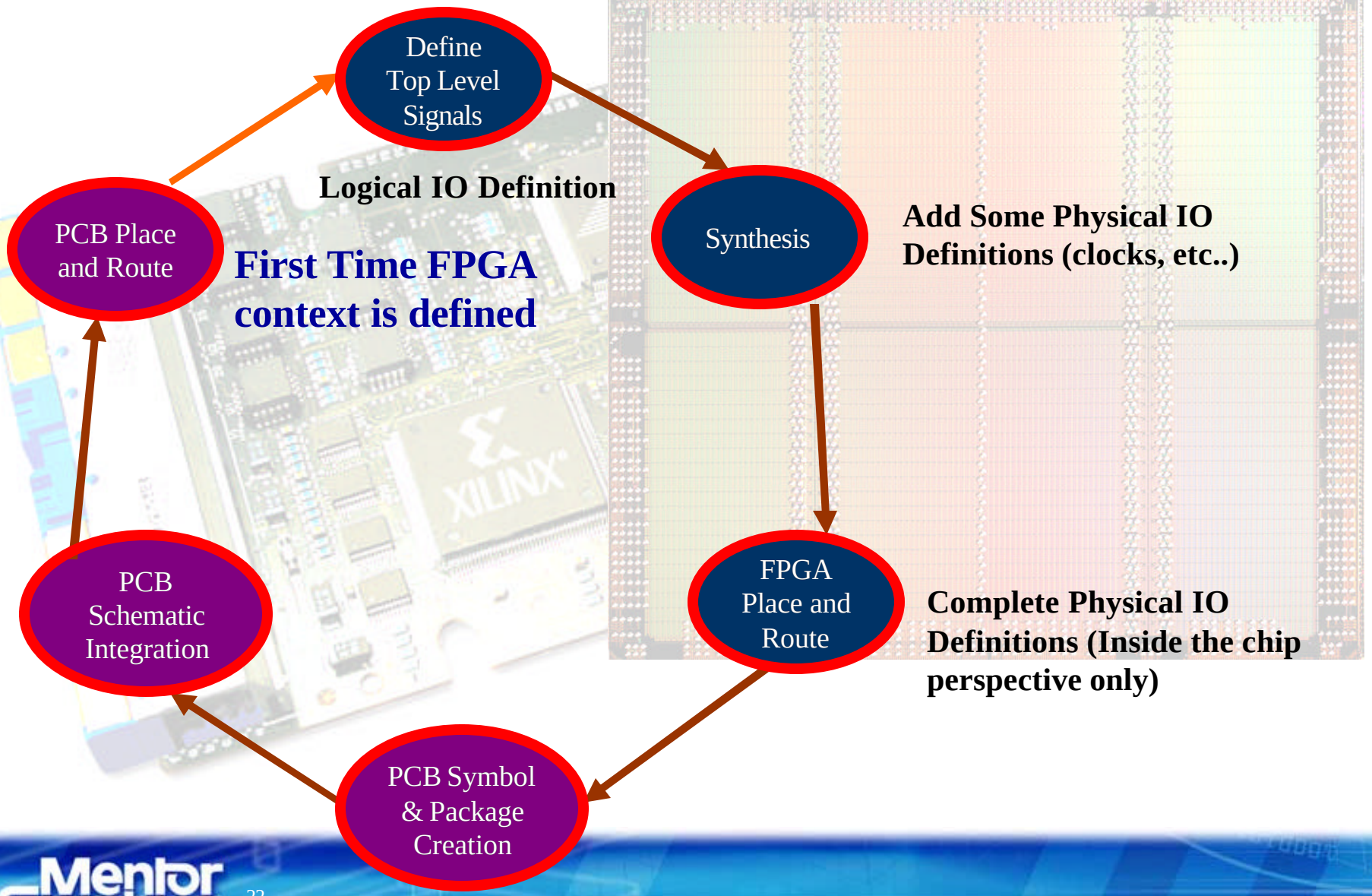
- **PCB libraries must be right ➔ Link to manufacturing and assembly**
- **FPGA symbols CHANGE!**
- **PCB libraries are NOT designed to support dynamic symbols**



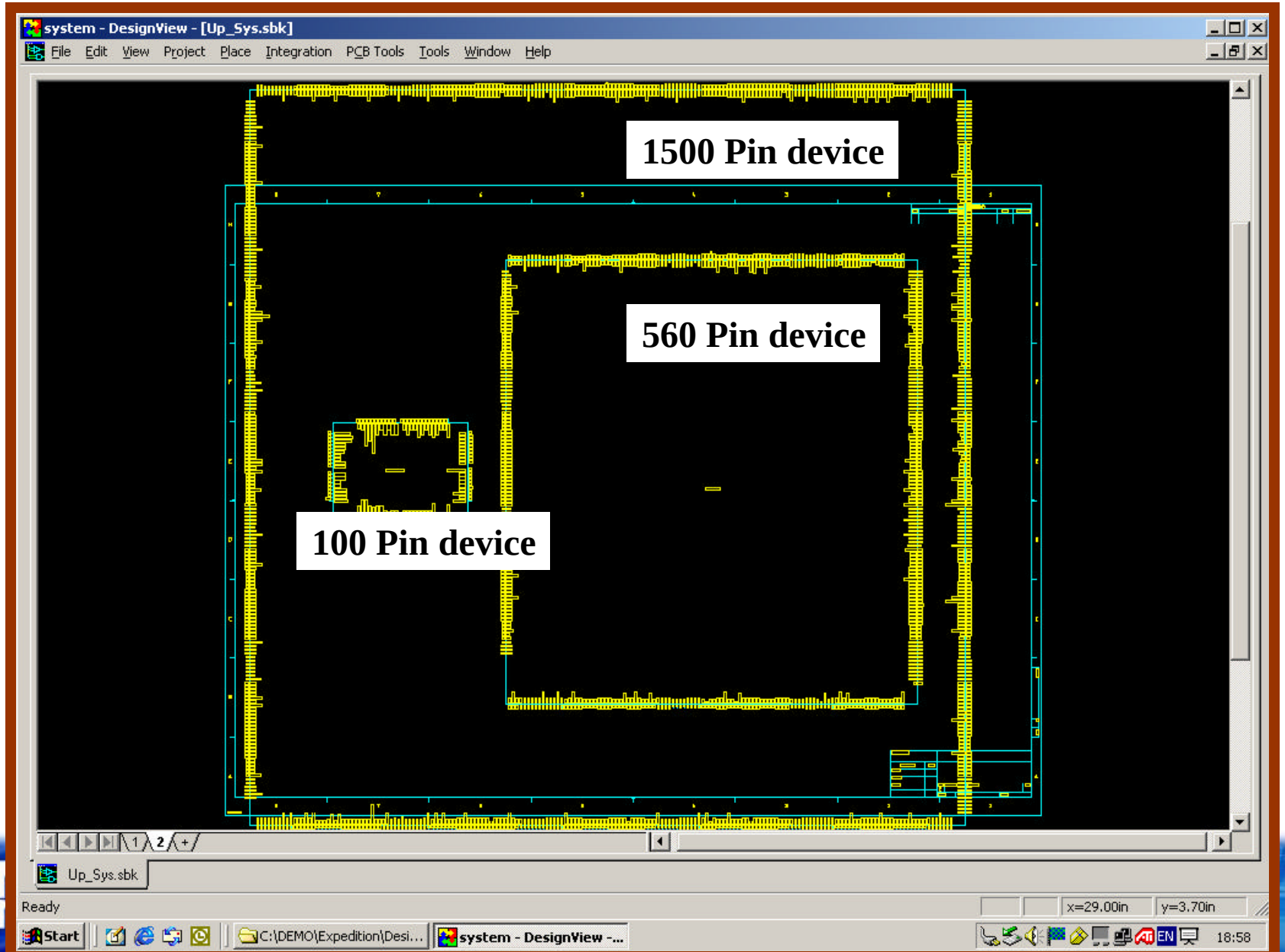


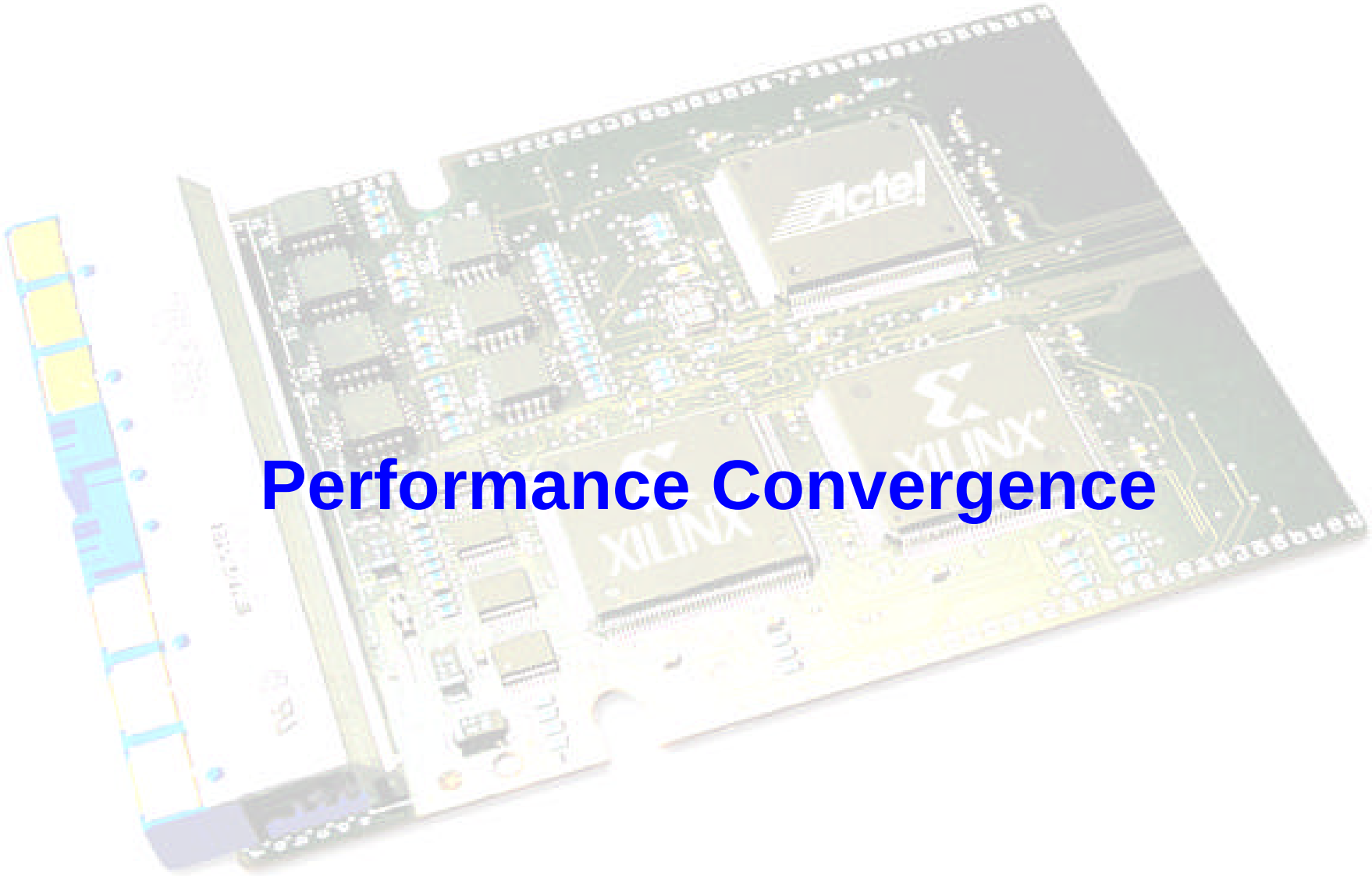
Physical Connectivity

FPGA IO Design Barriers



Traditional FPGA Symbol Barriers

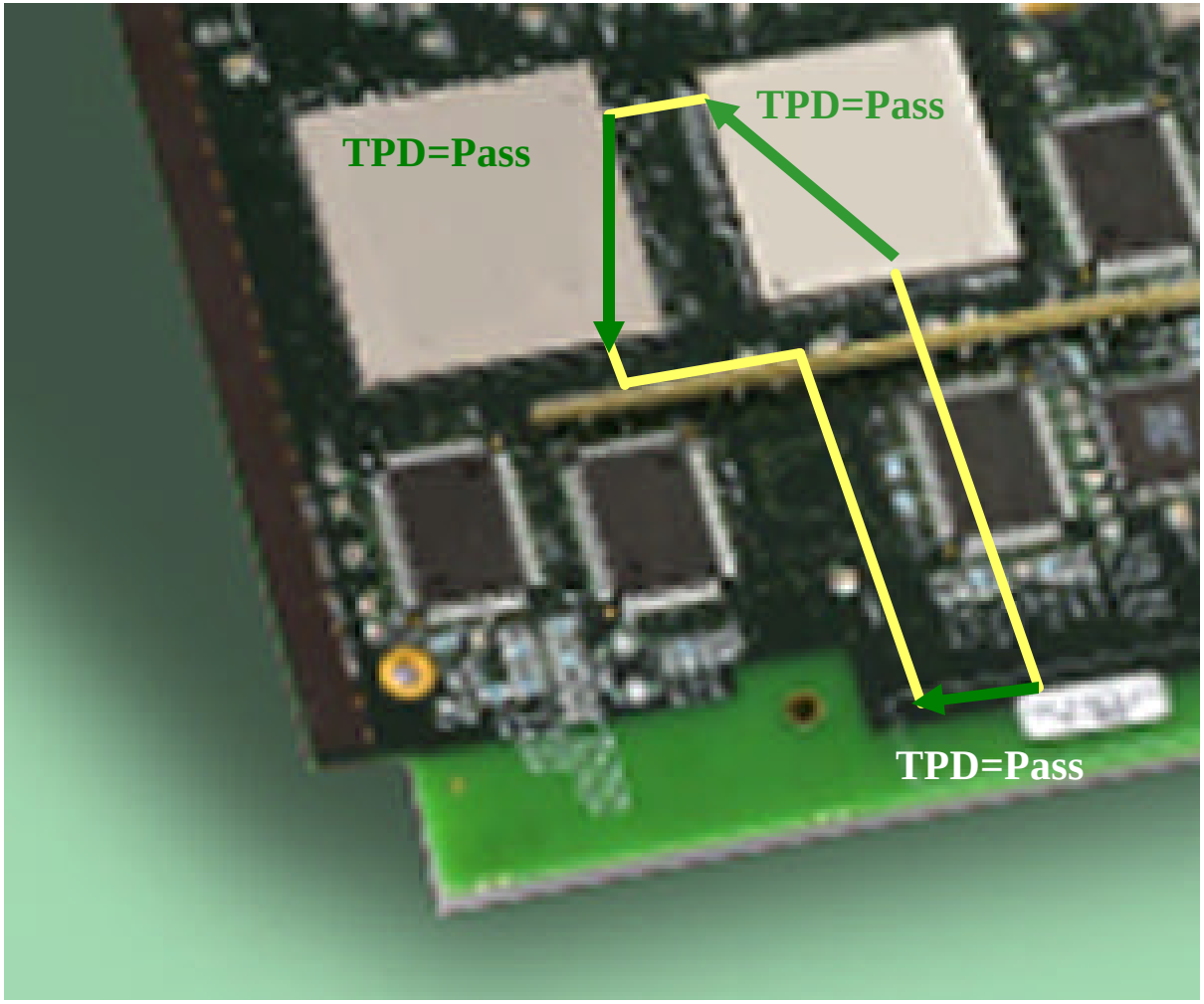


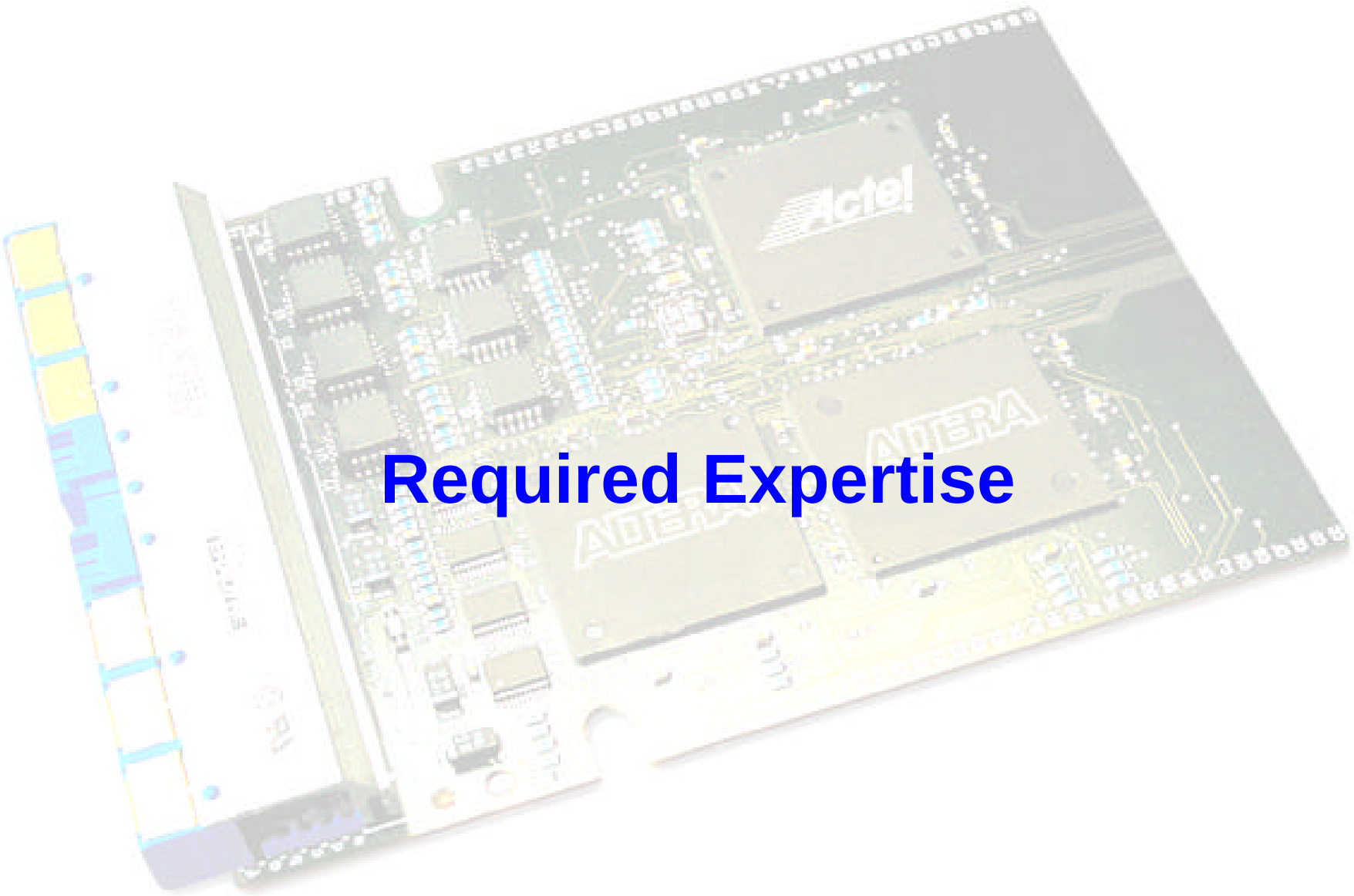


Performance Convergence

System Performance Barriers

- Changing the FPGA IO is easy
- Freezing the FPGA IO requires effort
- Changing the FPGA IO will impact the PCB **AND** the FPGA





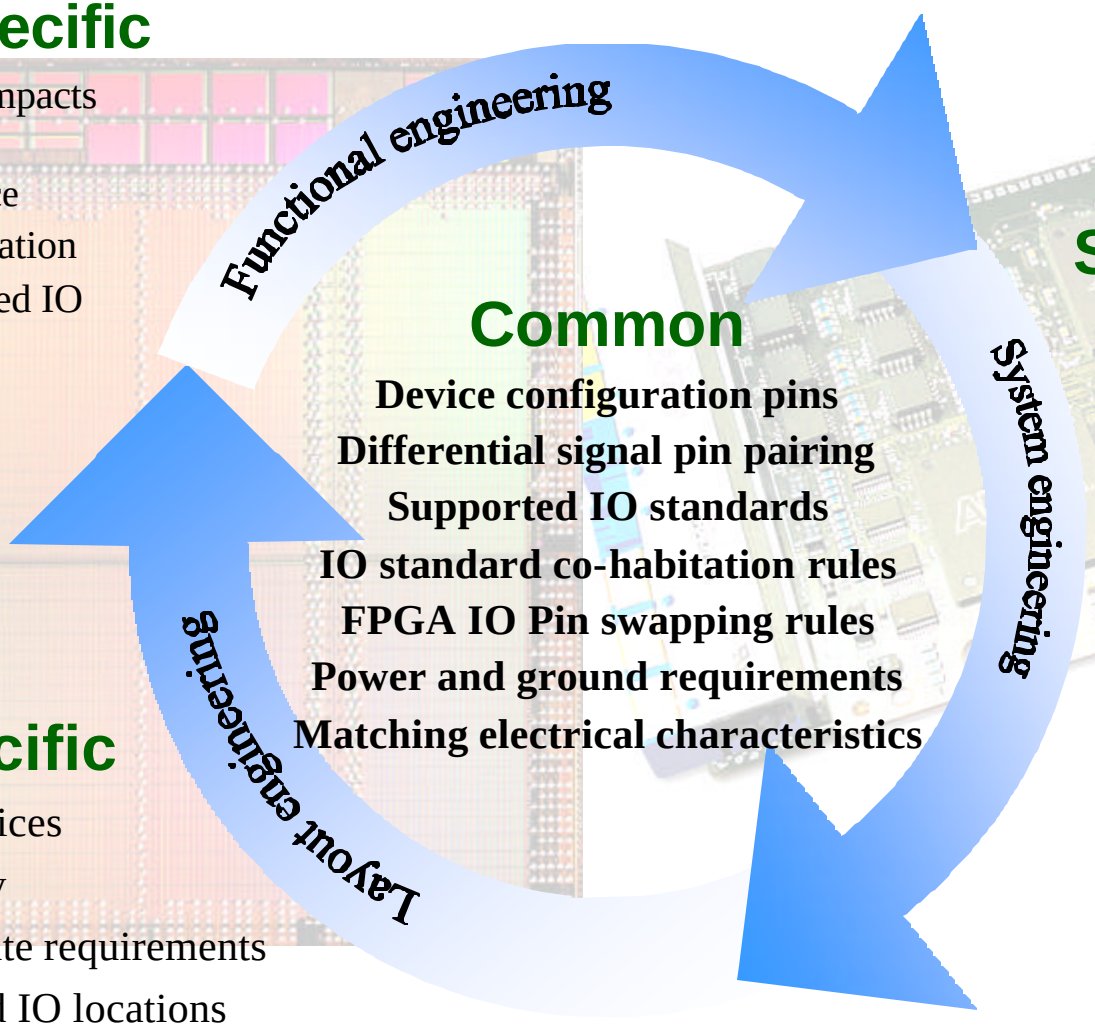
Required Expertise

IO Knowledge Barriers

FPGA Specific

Internal FPGA impacts
- Routability
- Performance

Device Configuration
Special high-speed IO
Clock IO



Common

Device configuration pins
Differential signal pin pairing
Supported IO standards
IO standard co-habitation rules
FPGA IO Pin swapping rules
Power and ground requirements
Matching electrical characteristics

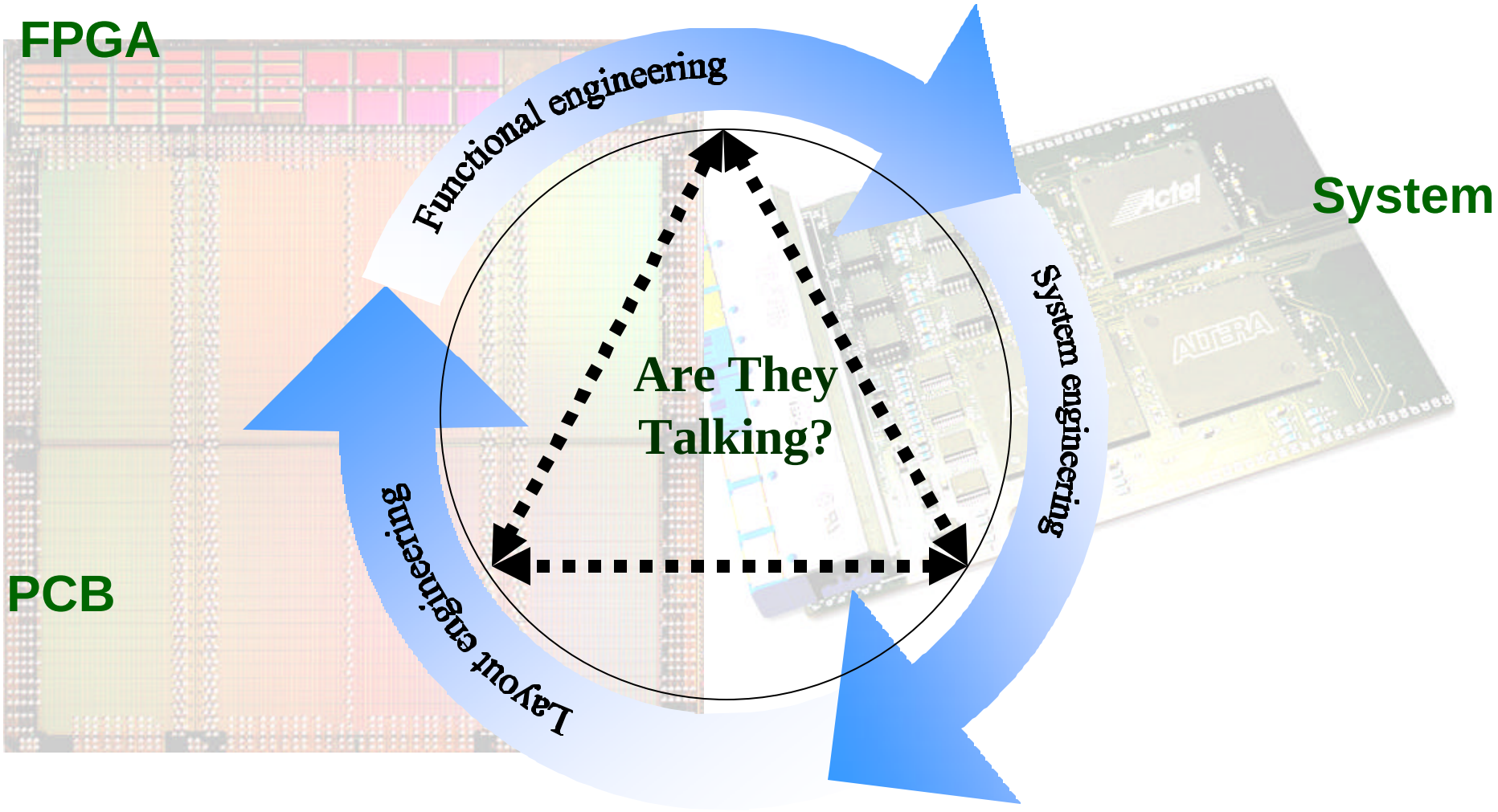
System Specific

Passive/Active devices
PCB impacts
Performance
LVDS requirements
Turn high-speed IO on/off
High Speed IO locations

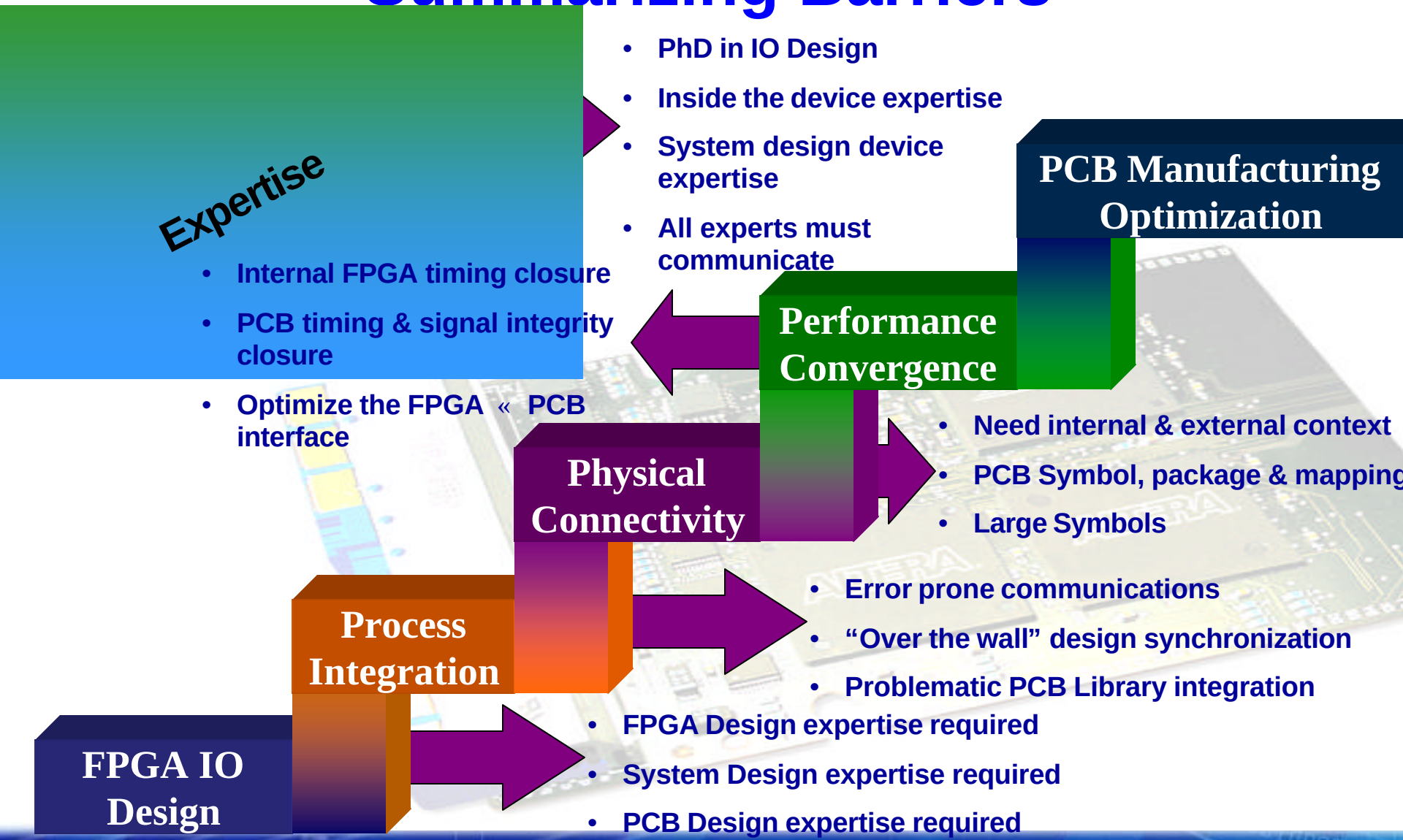
PCB Specific

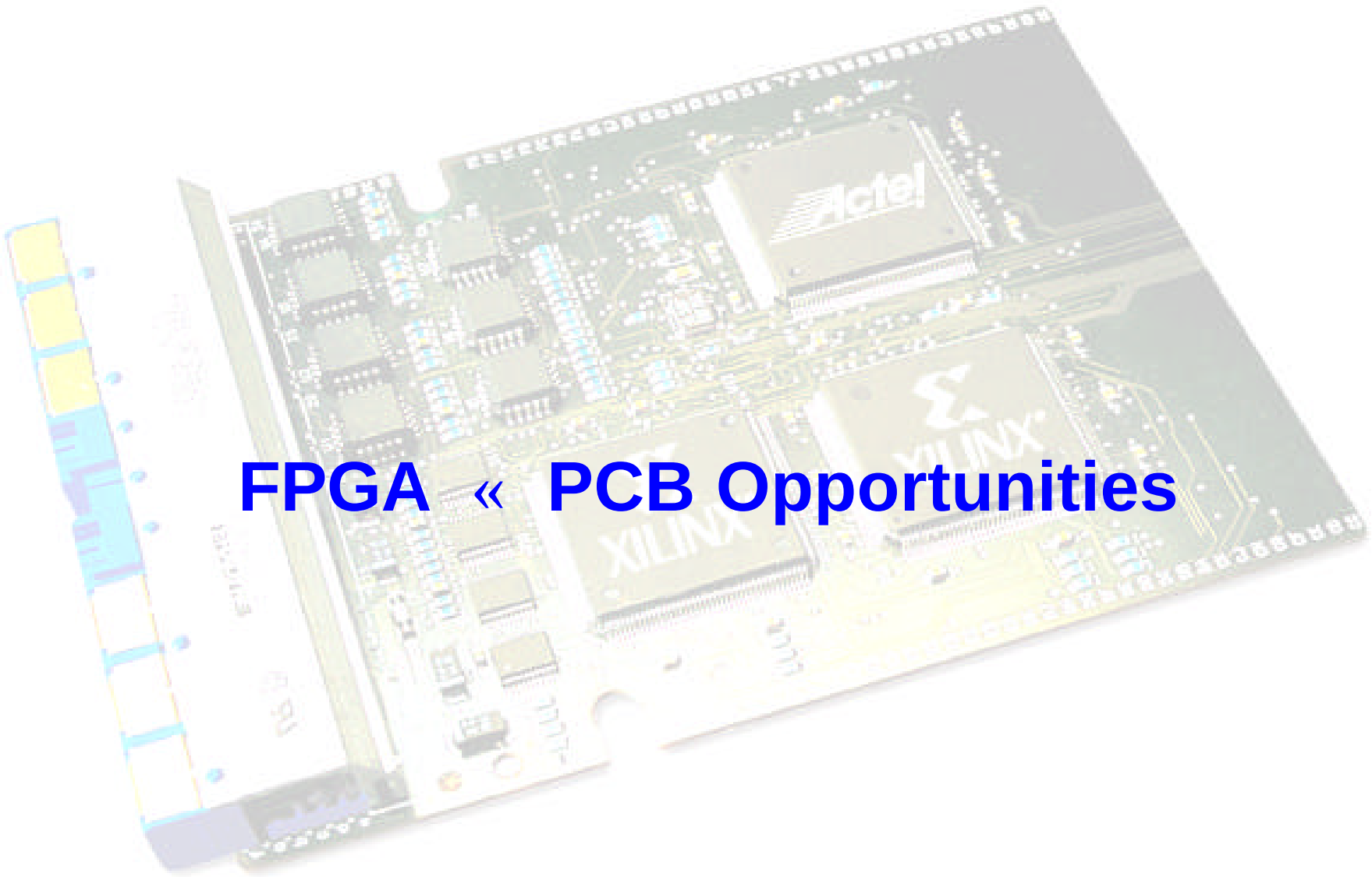
Supporting Devices
- Routability
- LVDS Route requirements
- High Speed IO locations

Communication Barriers



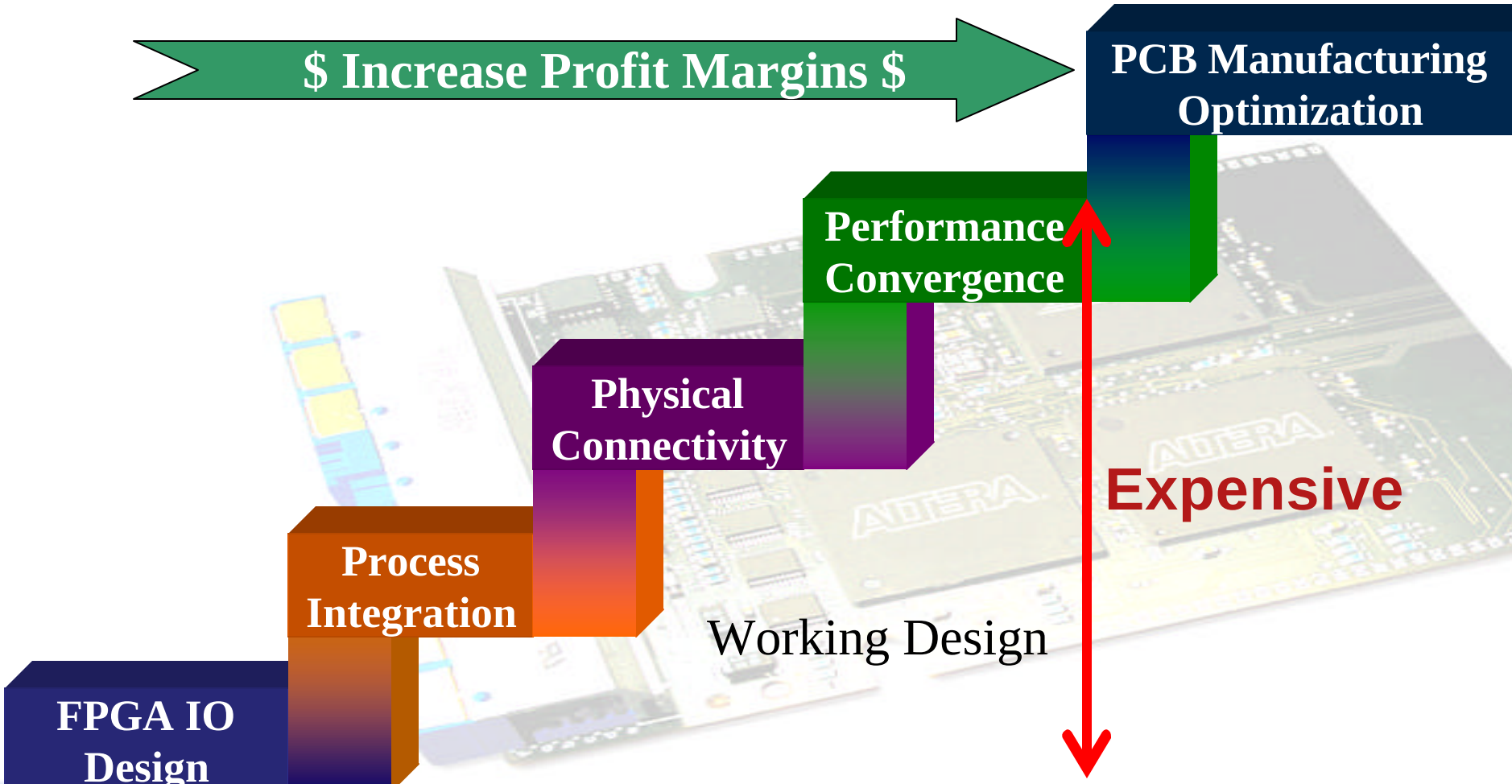
Summarizing Barriers





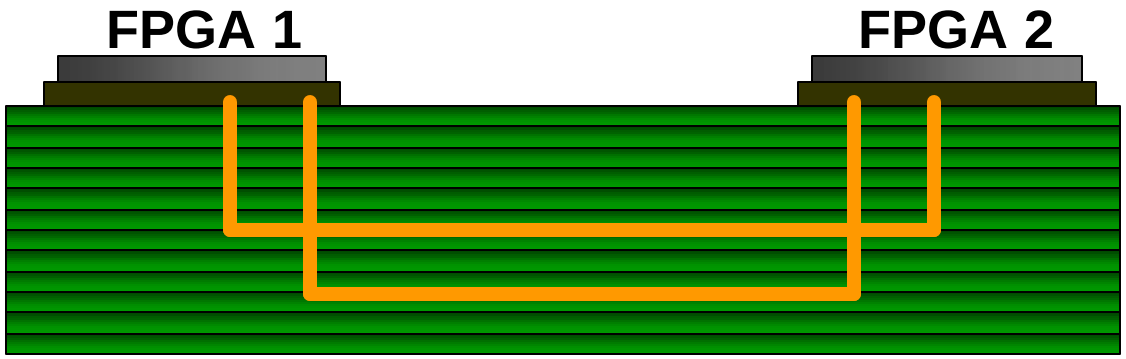
FPGA « PCB Opportunities

Integration Barrier Opportunities



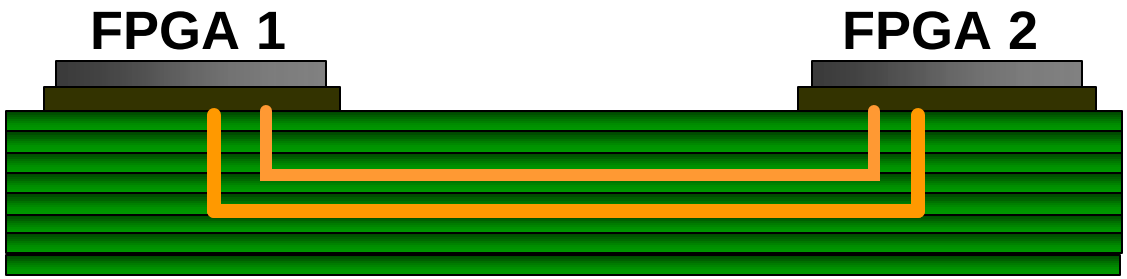
PCB Layer Reduction Through IO Optimization

PCB



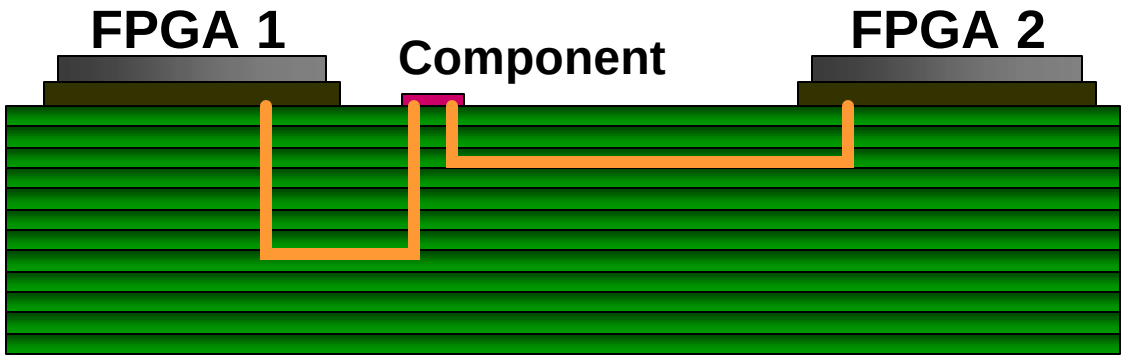
Non-Optimize IO

PCB



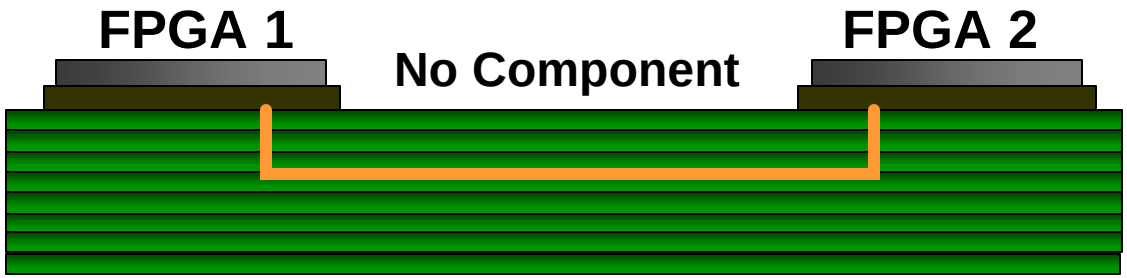
Optimized IO

Component Count Reduction



PCB

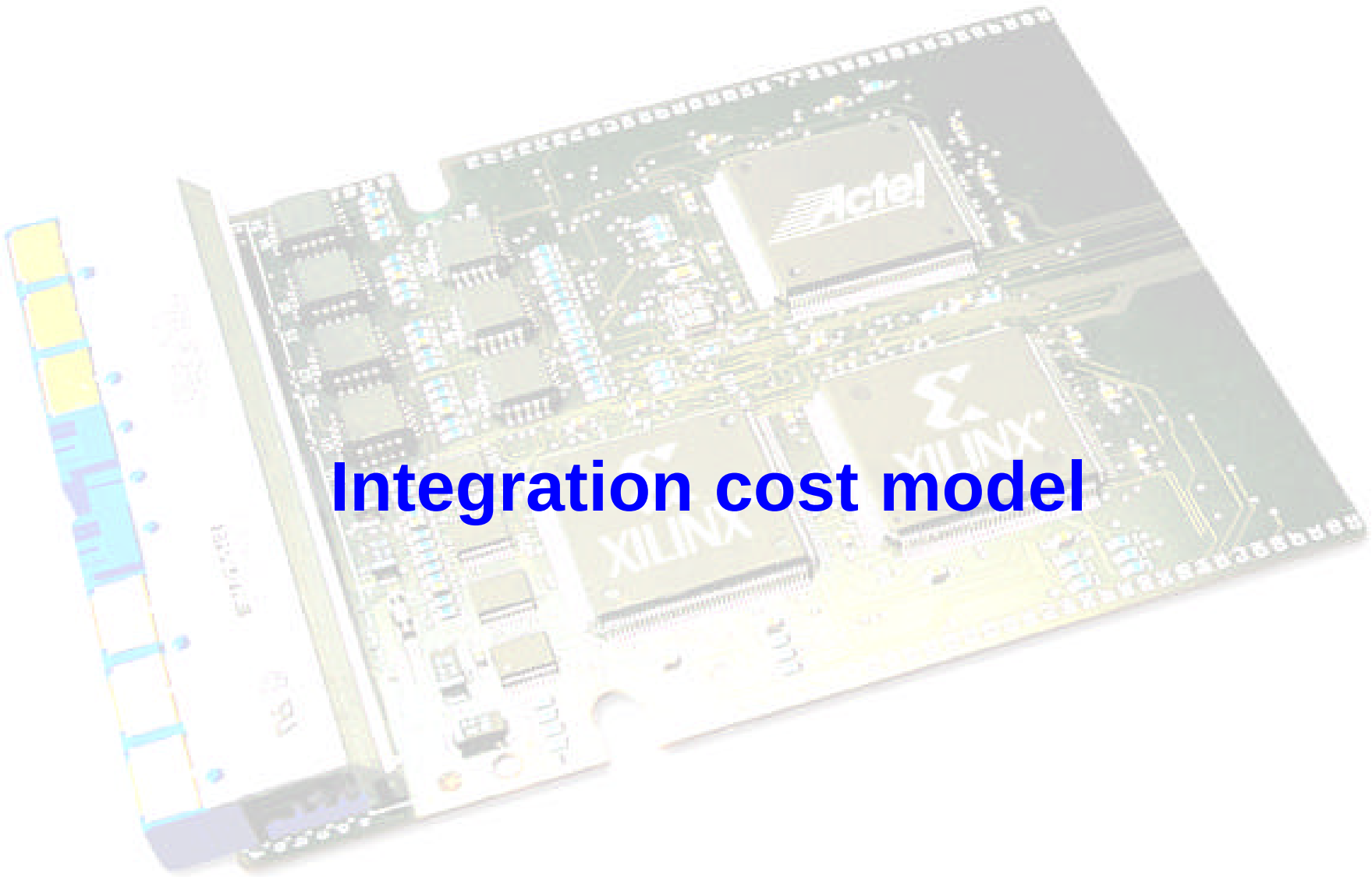
Conventional



PCB

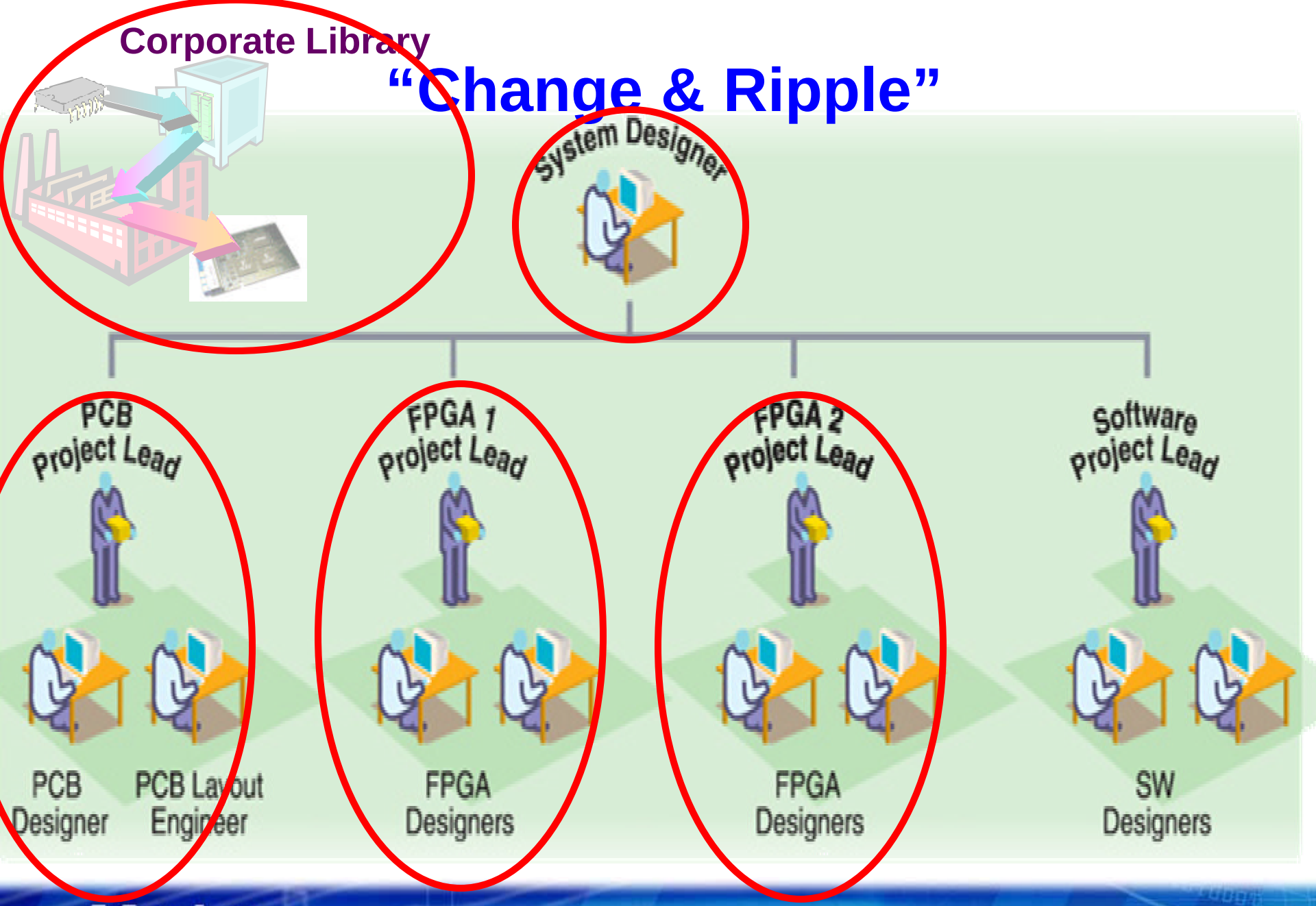
Opportunities

Reduces PCB Layers!!!!

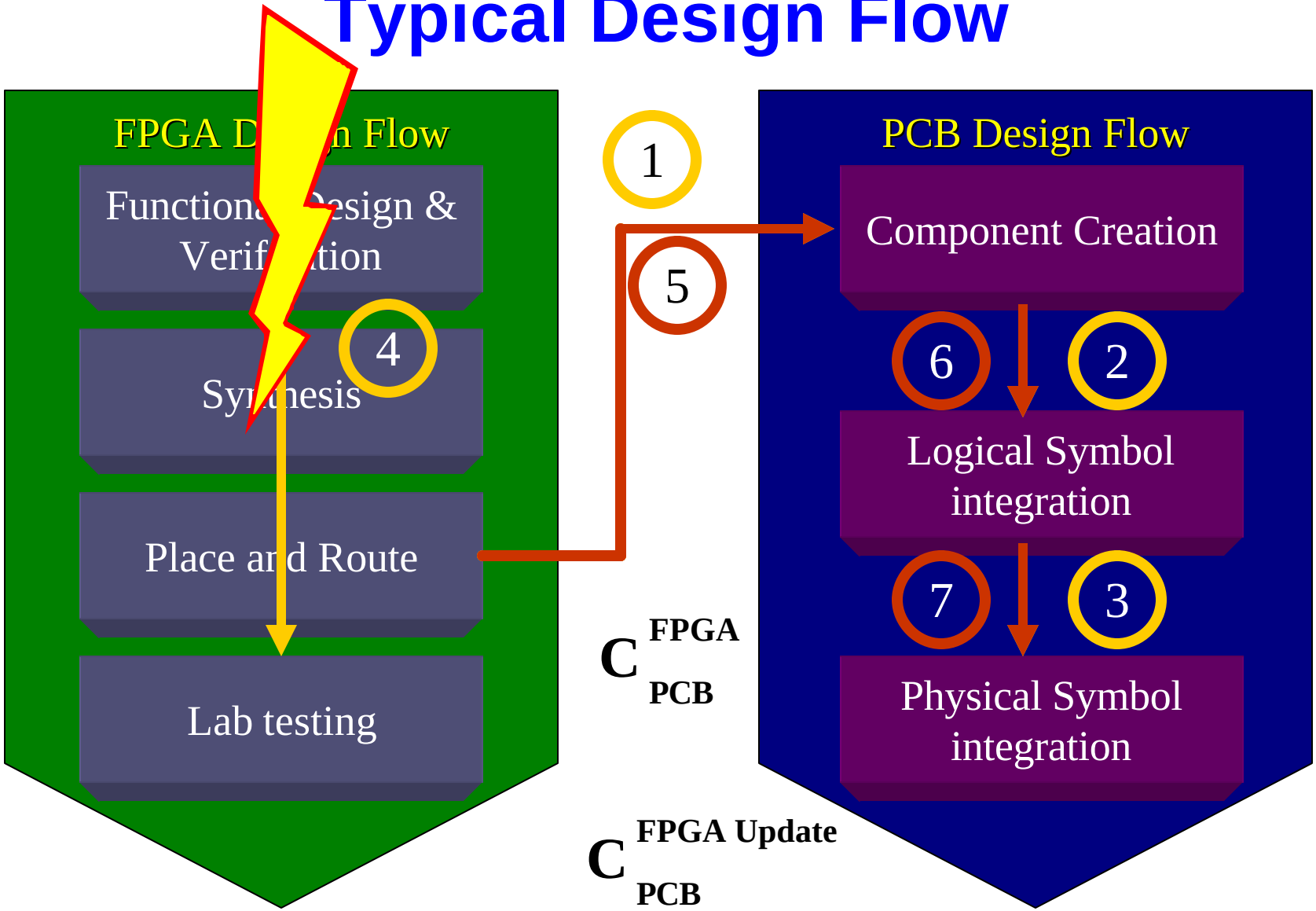


Integration cost model

“Change & Ripple”

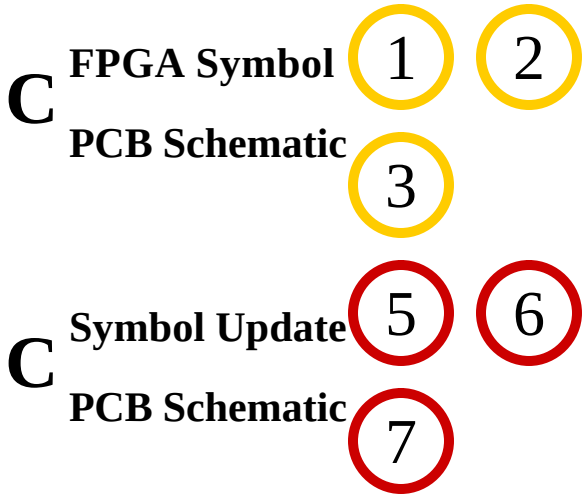


Typical Design Flow



Integration Cost Model

- **Primary costs**
  - **Creation (once)**
    - **Logical symbol of FPGA for PCB schematic**
    - **Physical symbol & associated data**
  - **Update (n times)**
    - **Symbol of FPGA for PCB schematic**
    - **Physical symbol & associated data**
  - **Re-spin PCB due to FPGA integration**
    - **Manual data entry error**
    - **FPGA – PCB interface out of date**
    - **System Performance**
  - **Lost Opportunity**
    - **Lost sales each day product introduction is delayed**

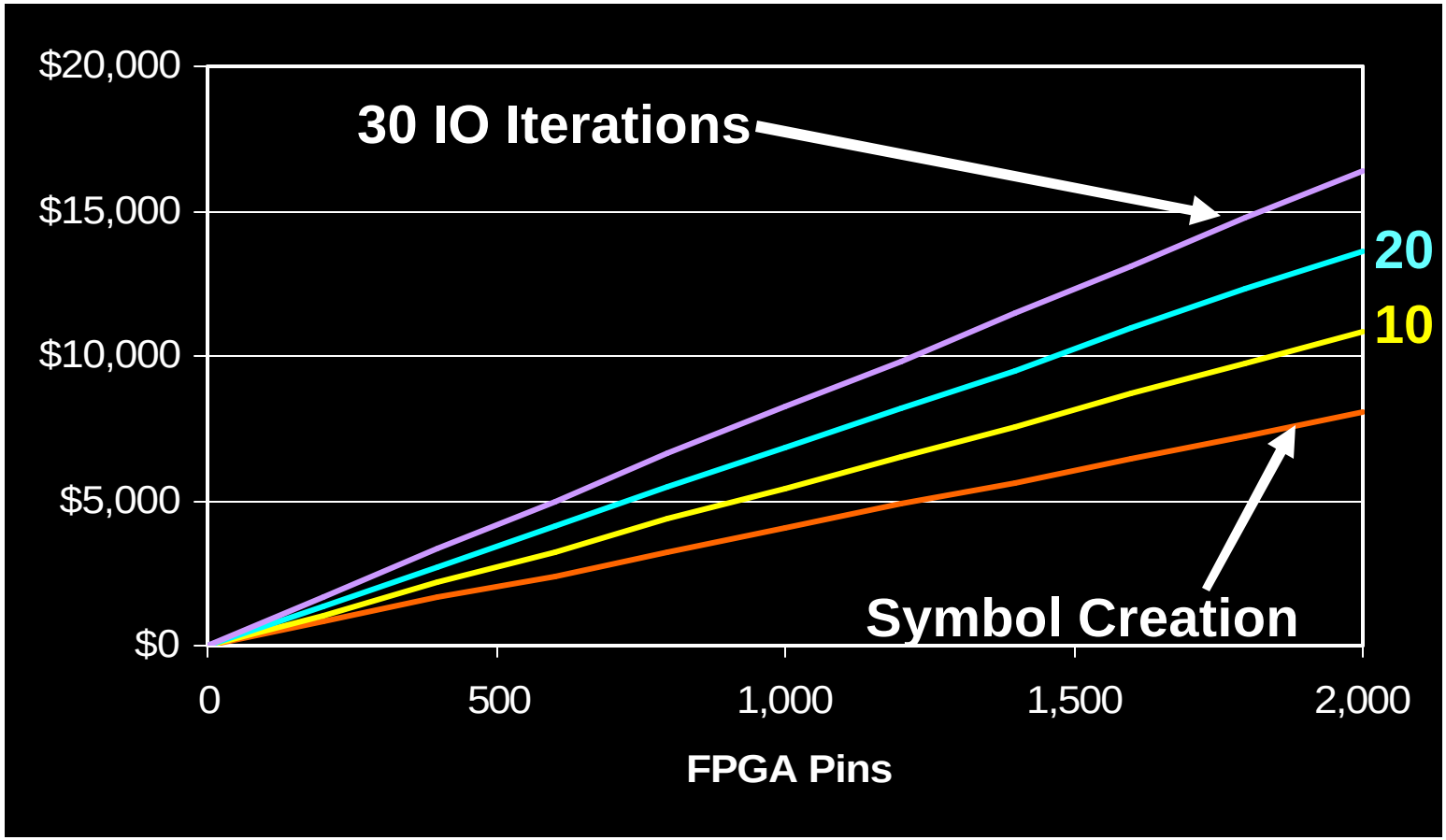


Simple Symbol Creation Model

$$C_{\text{FPGA « PCB Integration}}^{\text{Total}} = C_{\text{PCB Schematic}}^{\text{FPGA Symbol}}$$

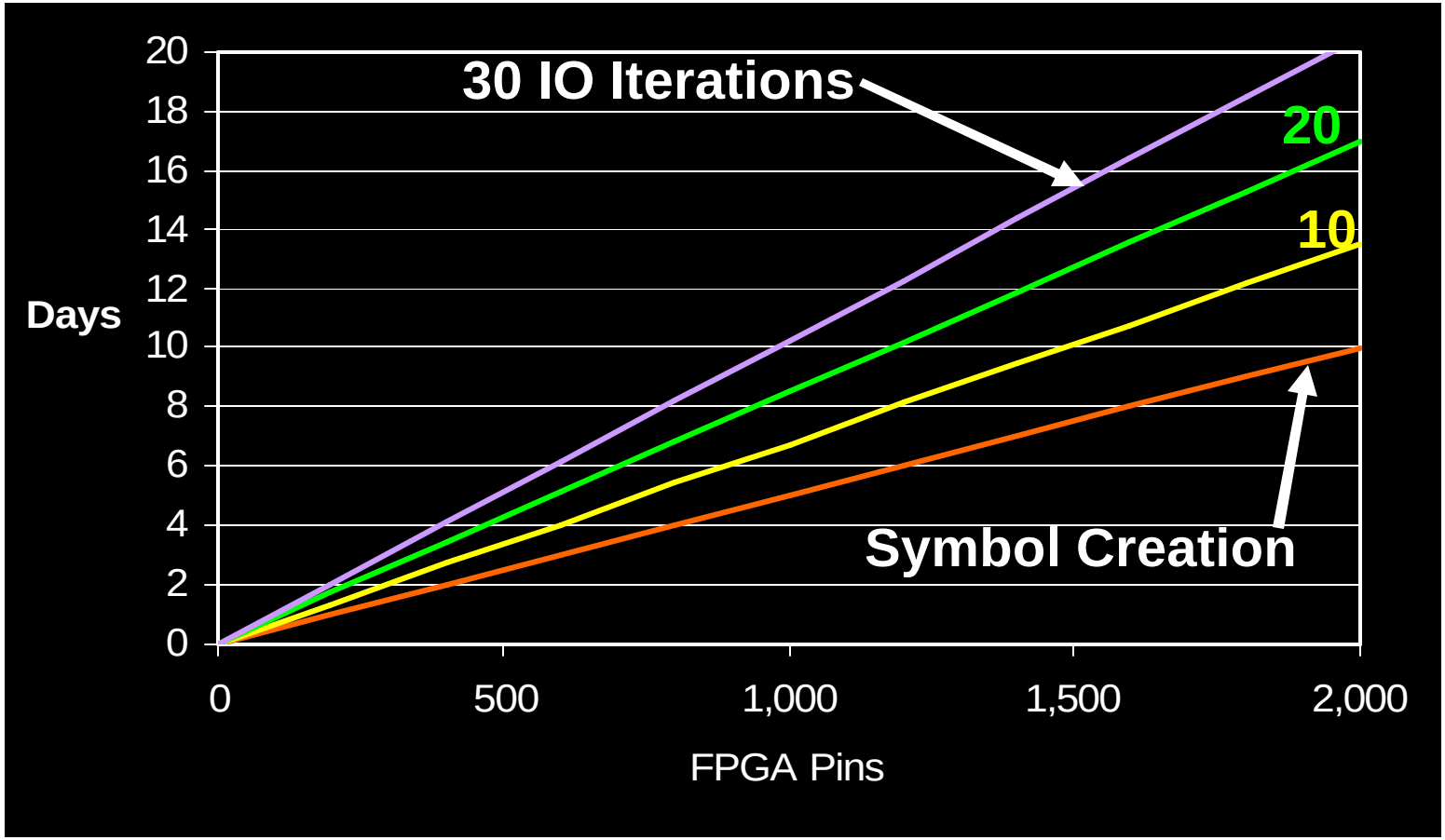
Symbol Update Costs

$$C_{\text{Total FPGA « PCB Integration}} = C_{\text{FPGA Symbol PCB Schematic}} + C_{\text{Symbol Update PCB Schematic}}$$



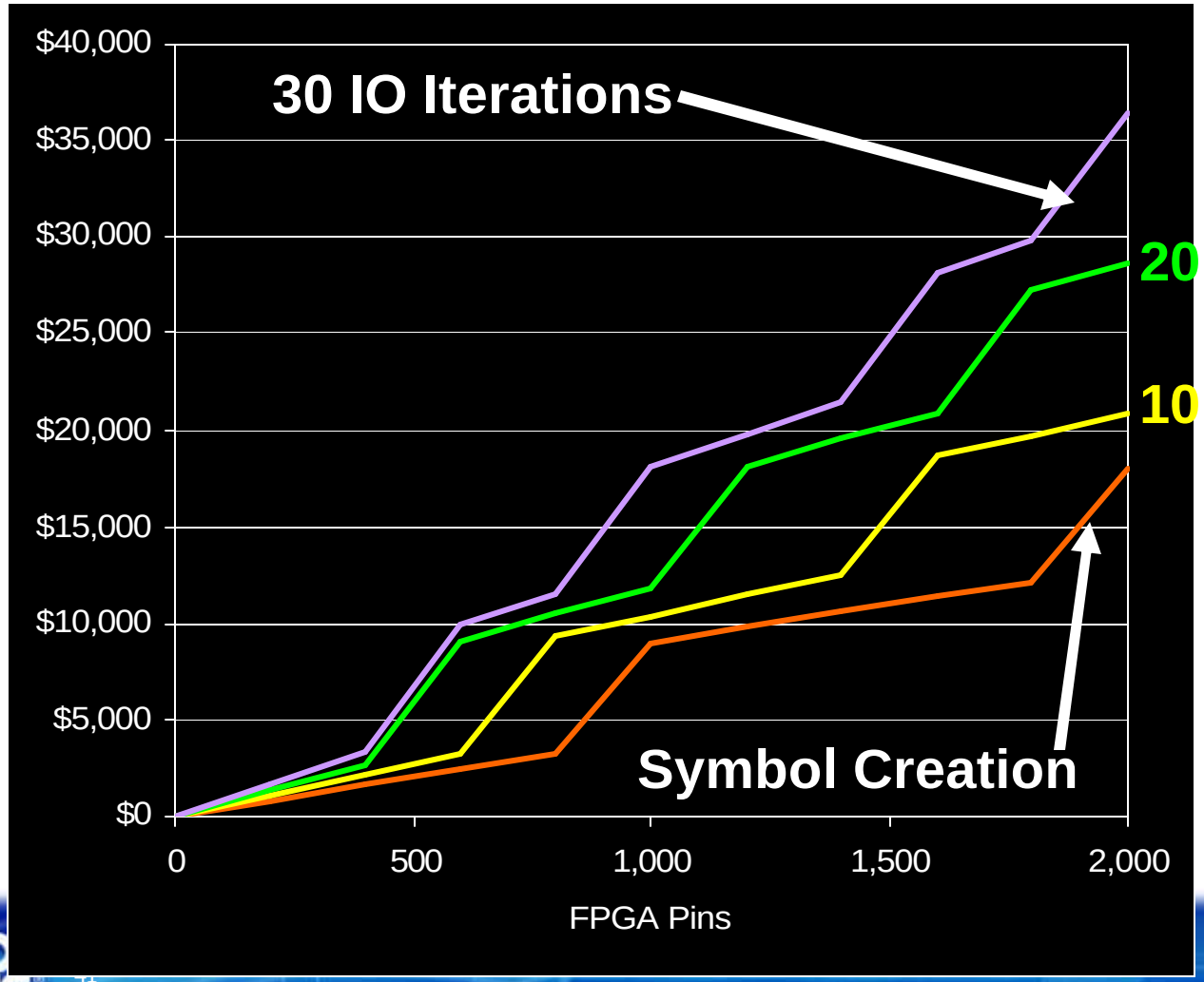
Symbol Update Effort

$$T_{\text{Total FPGA « PCB Integration}} = T_{\text{FPGA Symbol PCB Schematic}} + T_{\text{Symbol Update PCB Schematic}}$$



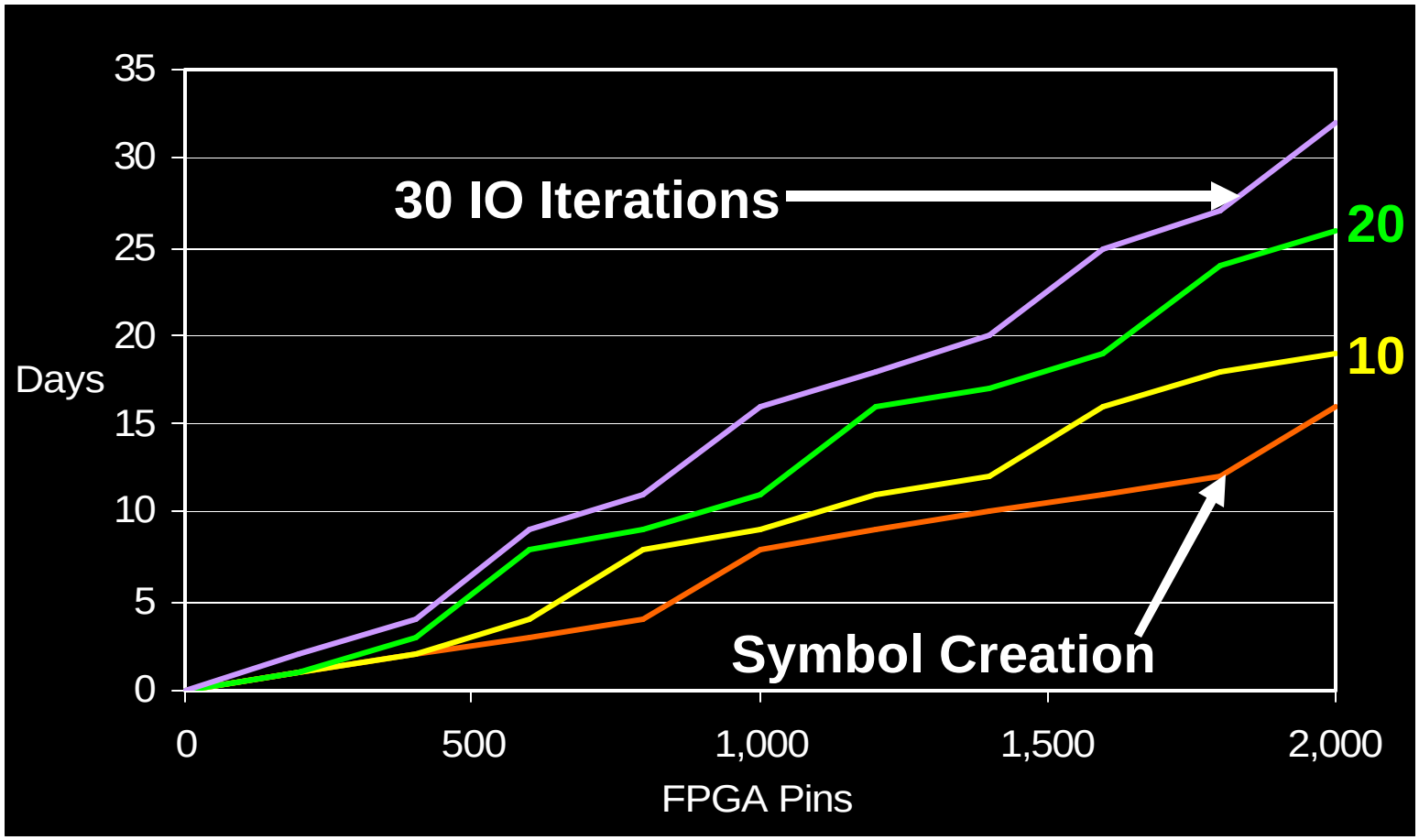
Data Entry Errors ➔ PCB Re-spin Costs

$$C_{\text{FPGA « PCB Integration}}^{\text{Total}} = C_{\text{PCB Schematic}}^{\text{FPGA Symbol}} + C_{\text{PCB Schematic}}^{\text{Symbol Update}} + C_{\text{PCB Re-spin}}^{\text{Re-entry Errors}}$$



Data Entry Errors ➔ More Delays

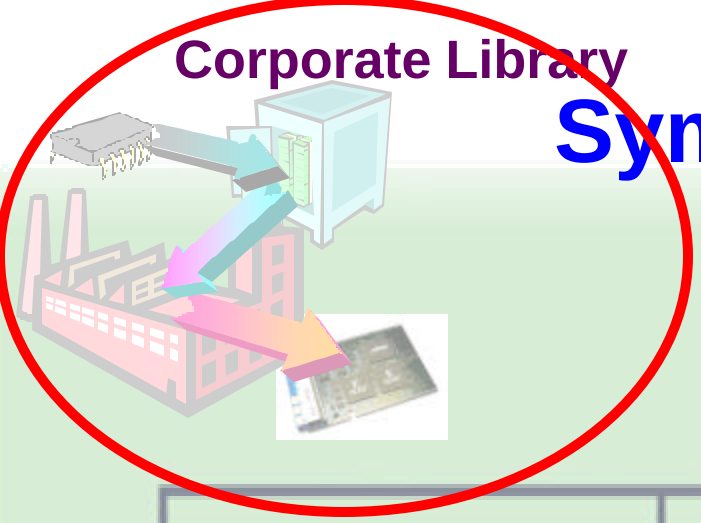
$$T_{\text{FPGA « PCB Integration}}^{\text{Total}} = T_{\text{PCB Schematic}}^{\text{FPGA Symbol}} + T_{\text{PCB Schematic}}^{\text{Symbol Update}} + T_{\text{PCB Re-spin}}^{\text{Re-entry Errors}}$$



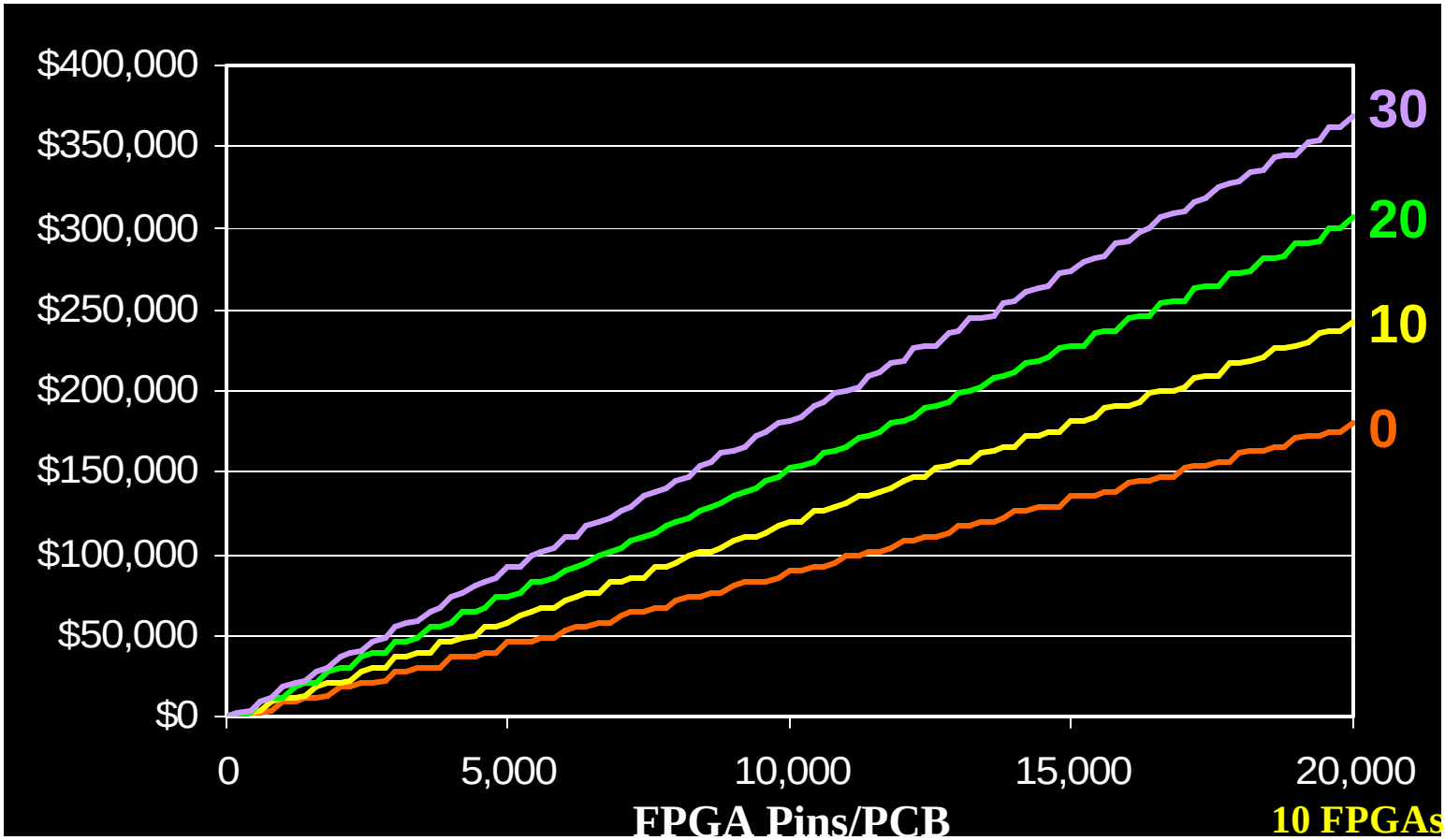
Other Costs To Consider

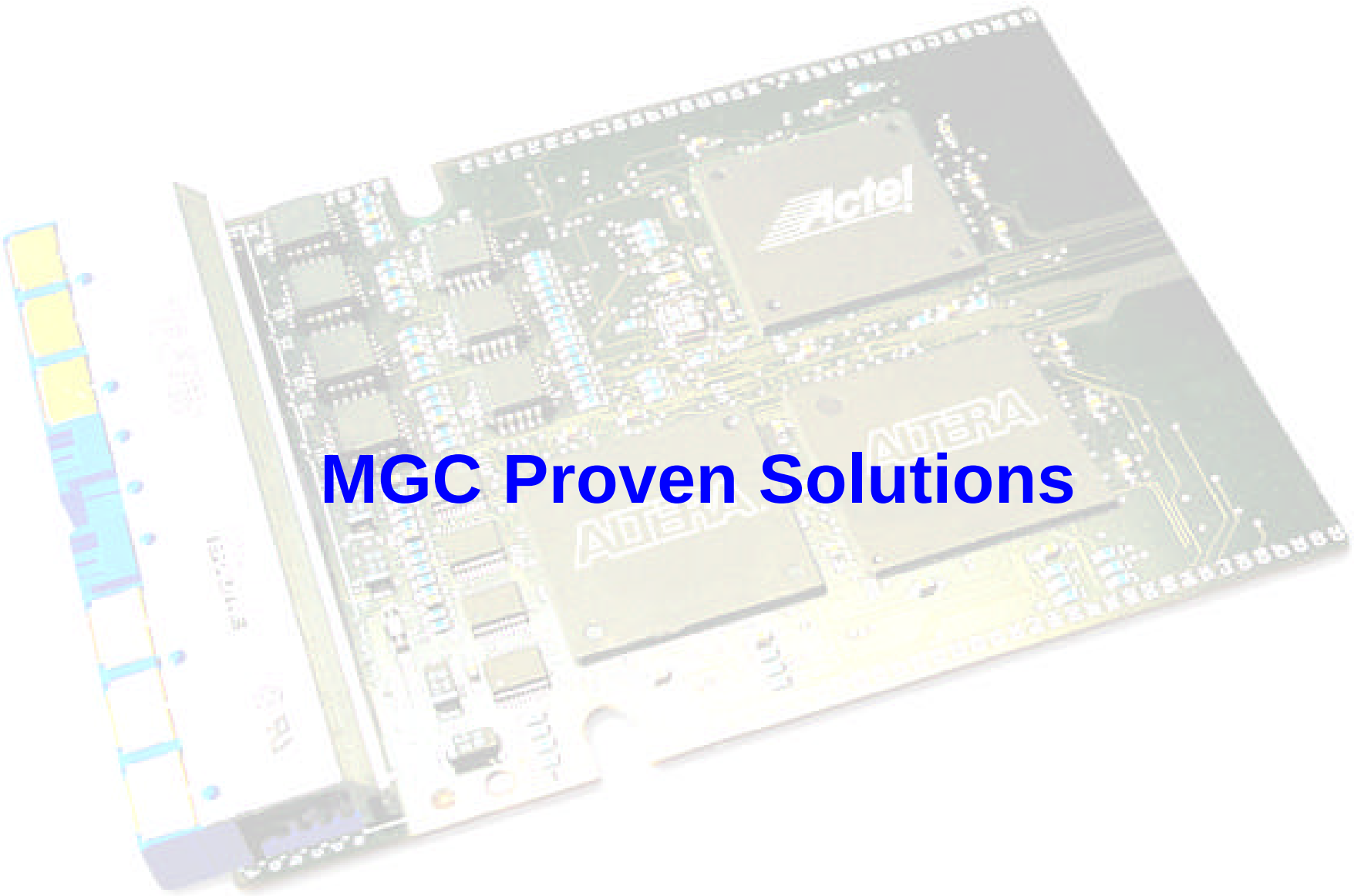
- Opportunity → $$
- Package Creation
- Symbol to Package Mapping Creation
- Symbol to Package Mapping Update
- Connecting/Updating the symbol in the schematic
- Routing & re-routing the package in the physical layout
- Maintaining Fractured Symbols
- PCB Library Integration
- PCB re-spins to achieve system performance
- PCB re-spin due to Design Errors
- Non-linear effects → Many high-speed signals

Symbol Costs Only



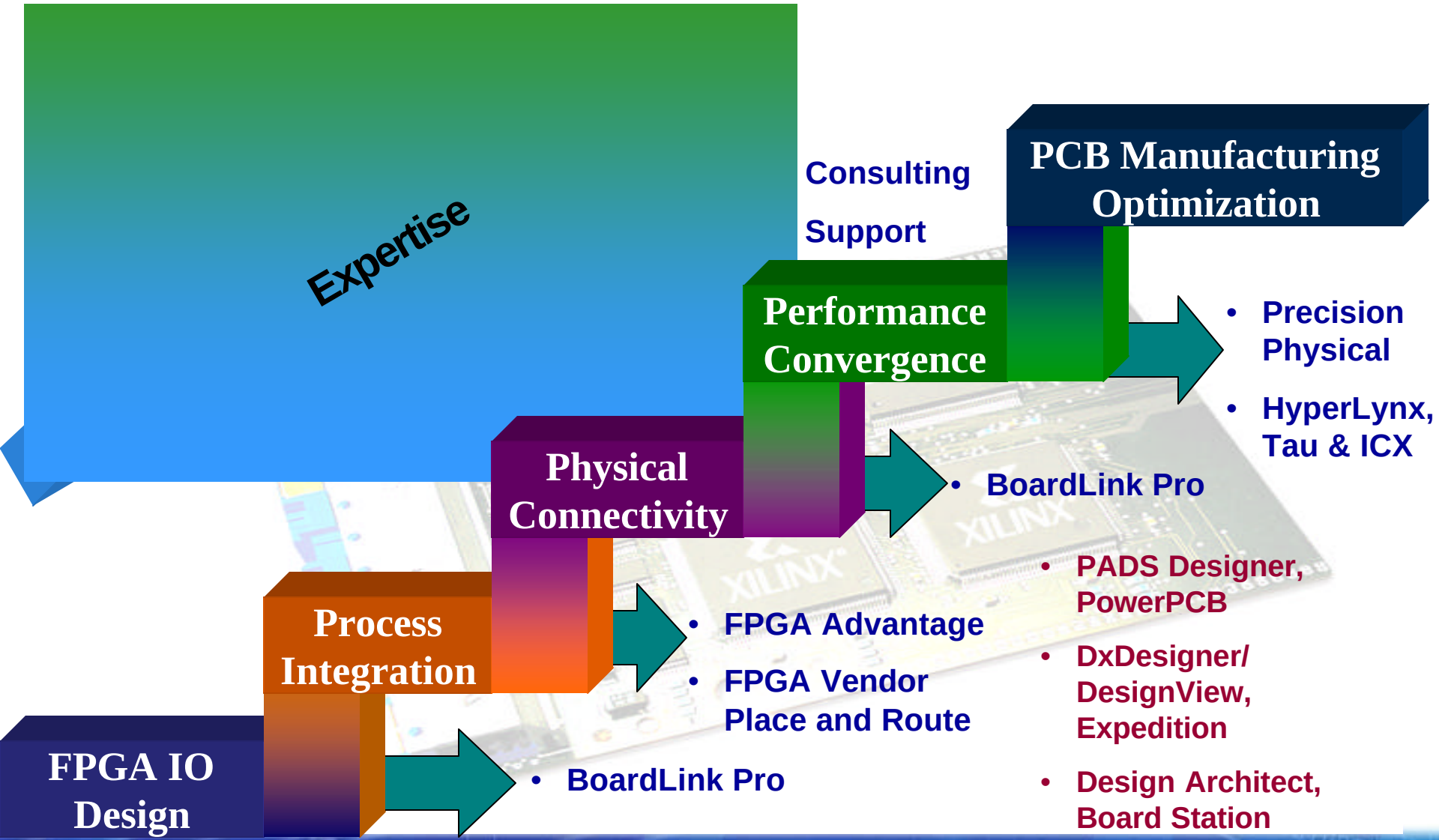
Multiple FPGAs Per PCB

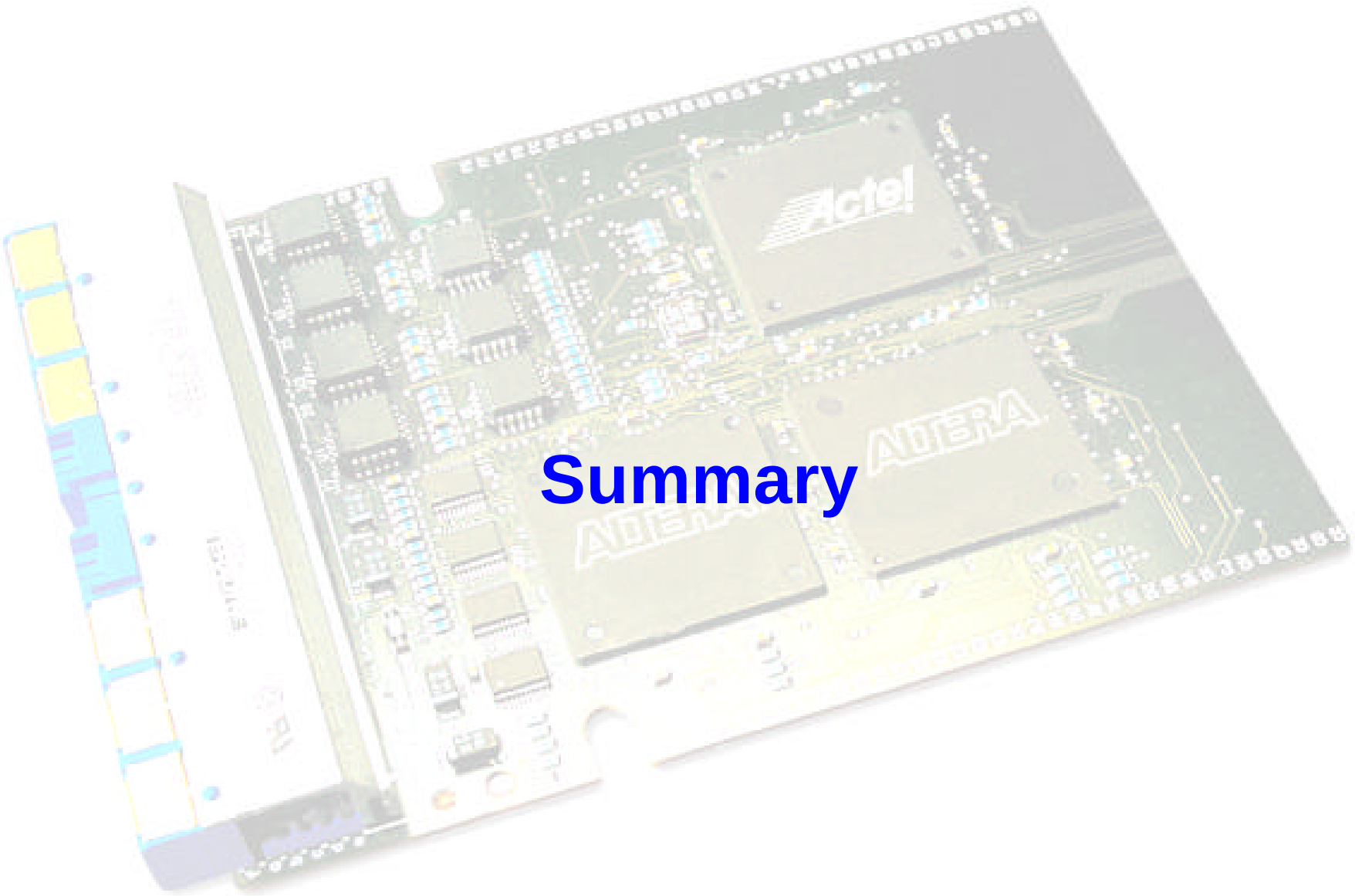




MGC Proven Solutions

MGC FPGA « PCB Concurrent System

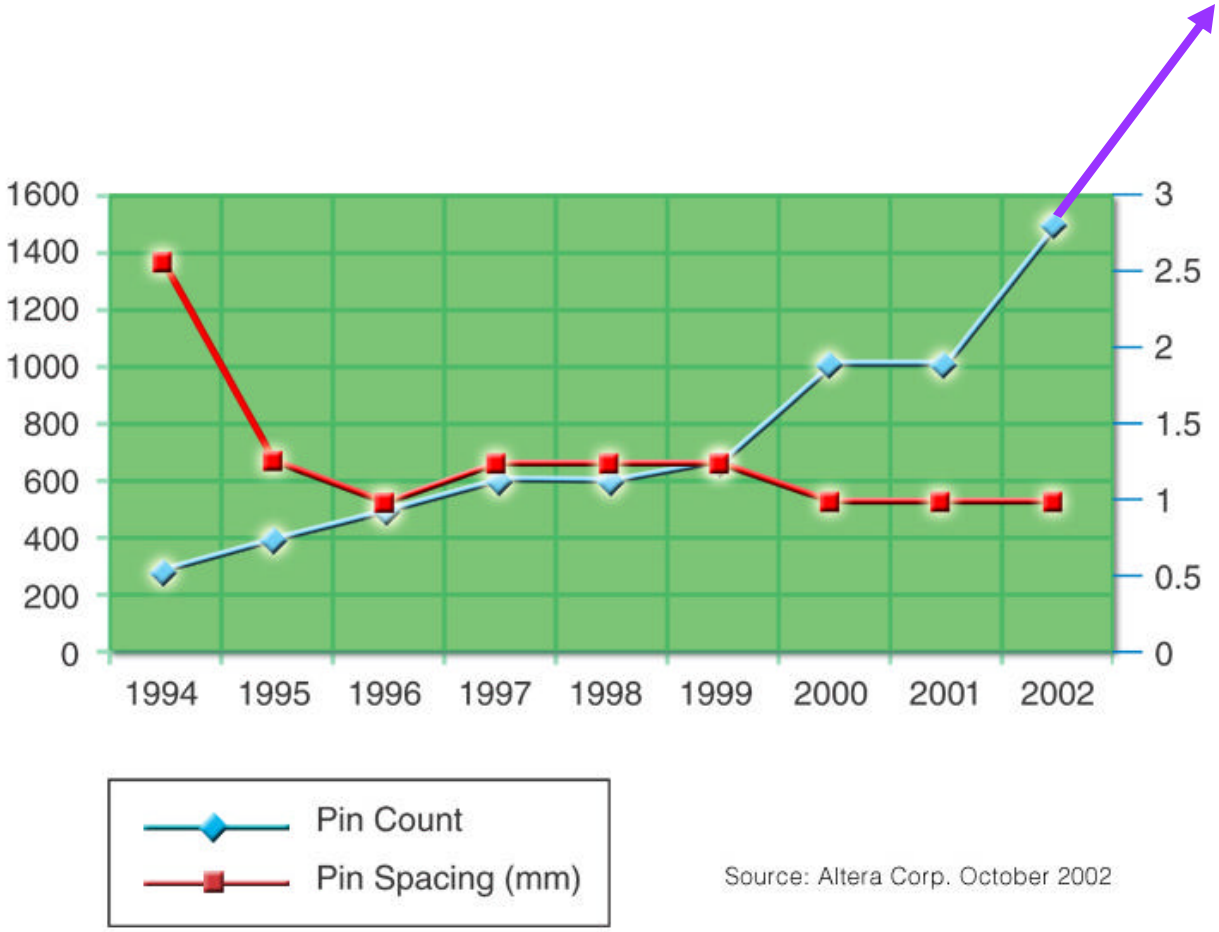




Summary

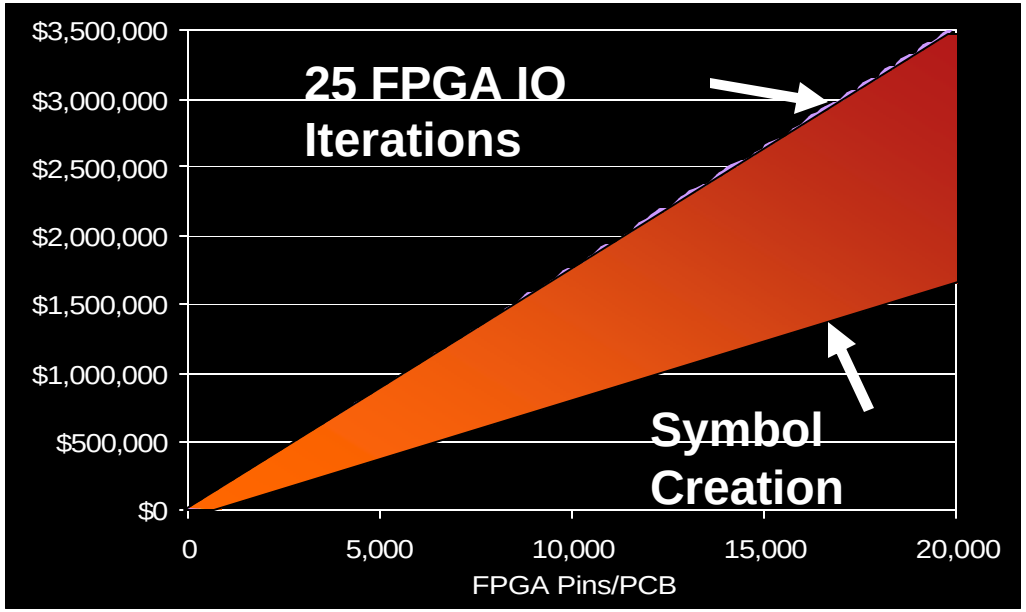
FPGA PCB Barriers Are Increasing

Physical connectivity is needed for *all* FPGAs

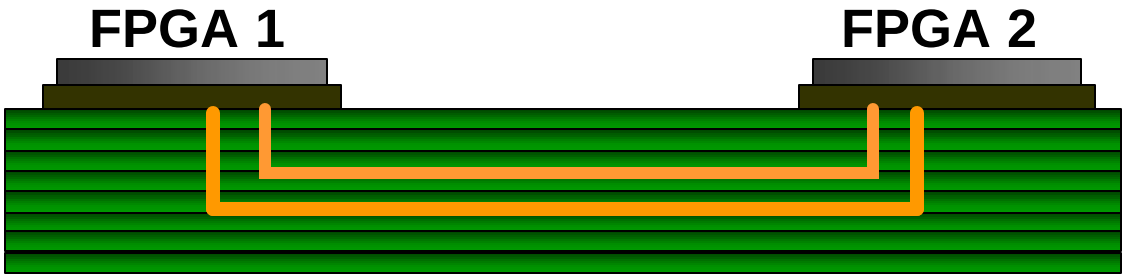


Attacking FPGA « PCB Synchronization pays!!

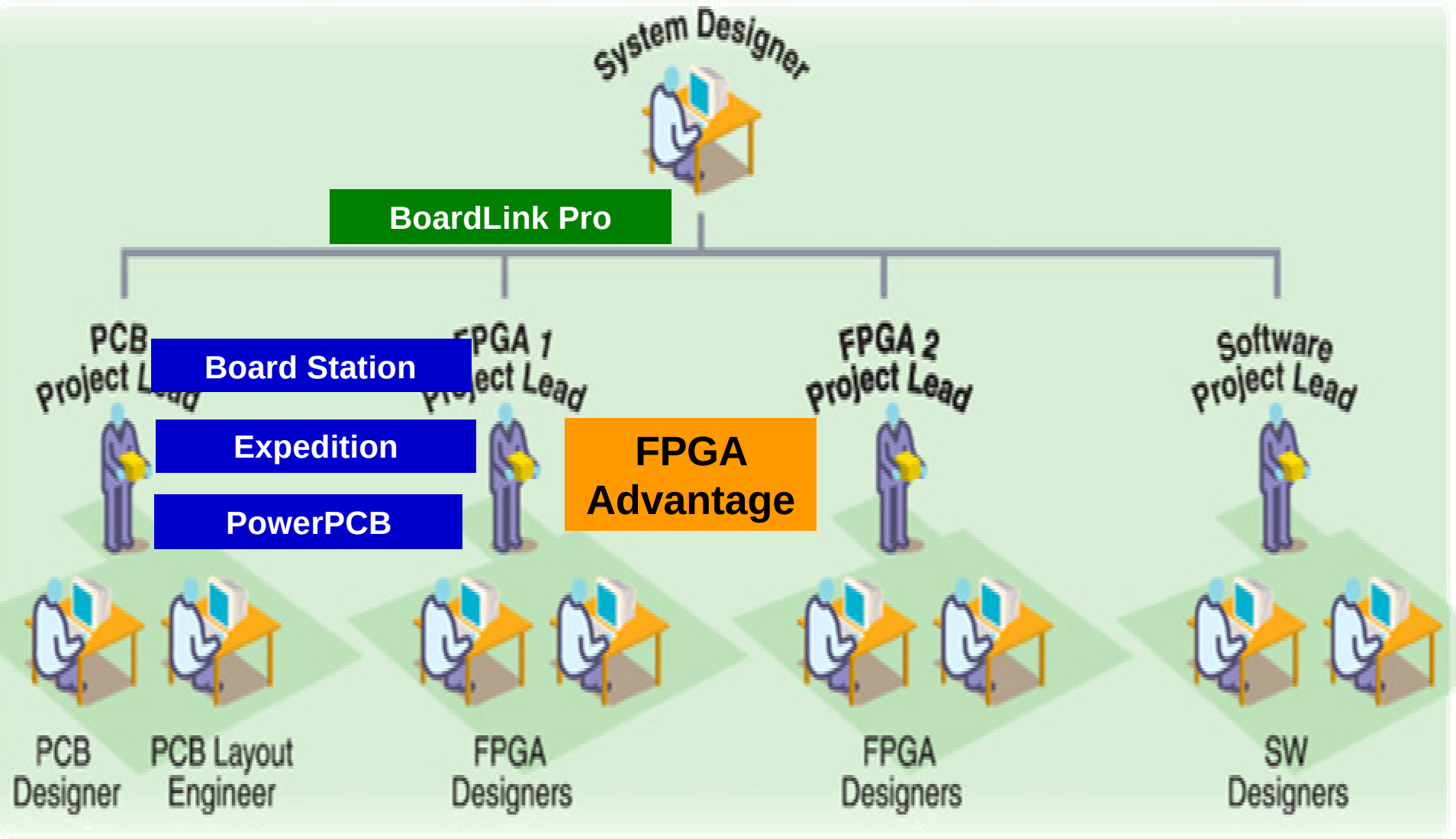
- **Initially ➔ Barrier for design success**

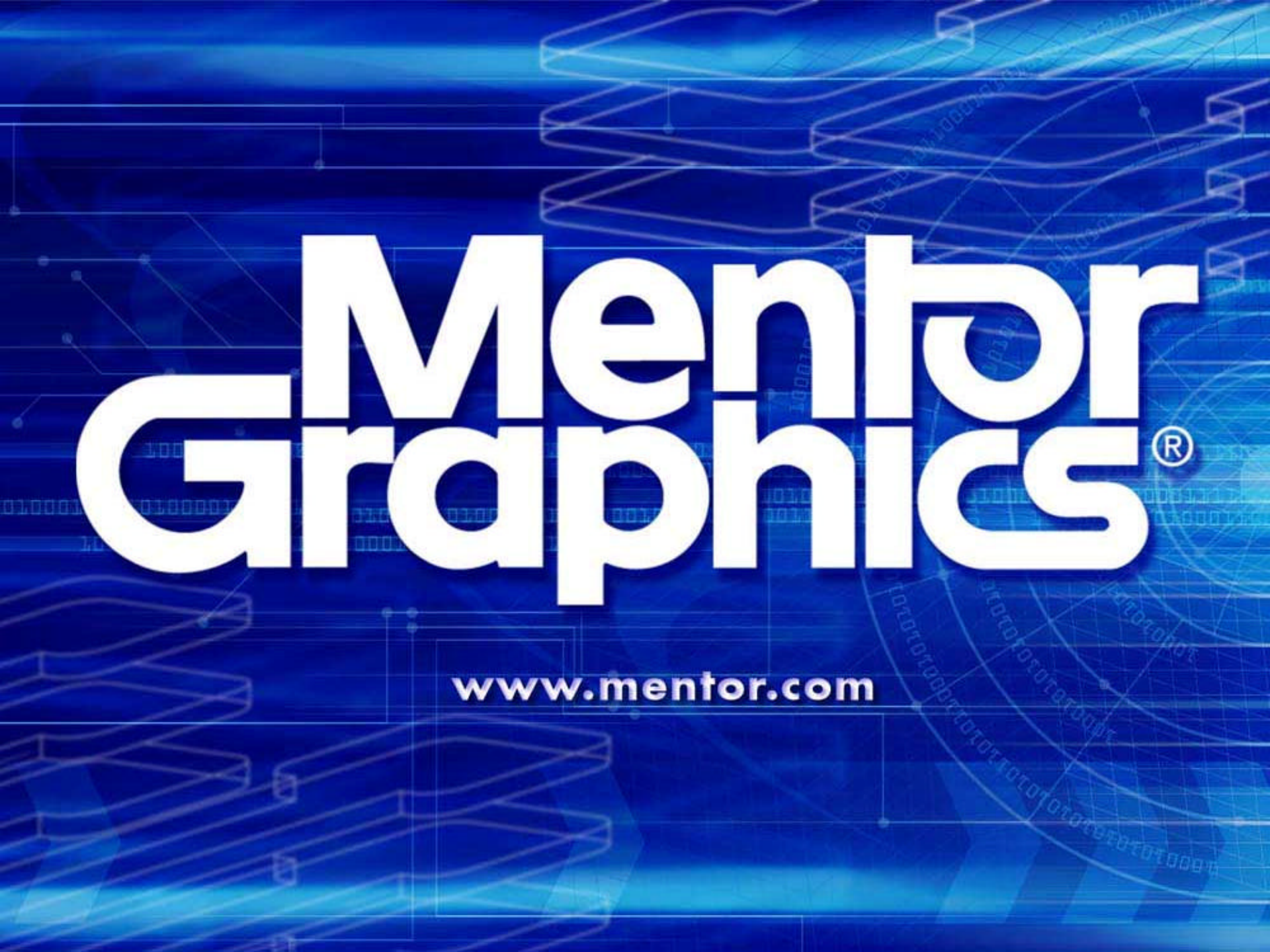


- **Opportunity ➔ Increased Profit Margins**



MGC Has the Complete Solution Today!




Mentor Graphics®
www.mentor.com