Quantifying the Cost and Benefit of Latency Insensitive Communication on FPGAs

Kevin E. Murray
Dept. of Electrical and Computer Engineering
University of Toronto, Ontario, Canada
kmurray@eecg.utoronto.ca

Vaughn Betz
Dept. of Electrical and Computer Engineering
University of Toronto, Ontario, Canada
vaughn@eecg.utoronto.ca

ABSTRACT

Latency insensitive communication offers many potential benefits for FPGA designs, including easier timing closure by enabling automatic pipelining, and easier interfacing with embedded NoCs. However, it is important to understand the costs and trade-offs associated with any new design style. This paper presents optimized implementations of latency insensitive communication building blocks, quantifies their overheads in terms of area and frequency, and provides guidance to designers on how to generate high-speed and area-efficient latency insensitive systems.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communications**]: Interconnections (Subsystems); B.5.1 [**Register-Transfer-Level Implementation**]: Design—*Styles*

General Terms

Design, Performance

Keywords

FPGA; Latency Insensitive; Pipelining

1. INTRODUCTION

Modern process technology scaling has introduced many challenges related to the design and implementation of FPGA systems. In particular, the different scaling characteristics of devices, local interconnect, and global interconnect [10] are making it more difficult to achieve timing closure in a predictable and timely manner.

The difference in scaling between local and global interconnect[1] is illustrated for FPGA devices in Figure 1. This shows that the speed of local communication within a relatively small amount of logic (i.e. 40K LEs) has more than doubled over five generations. In contrast, the speed of global communication across the full device (i.e. Max LEs) has not improved. This growing mismatch between local and global communication speed makes it difficult to close timing on large designs.

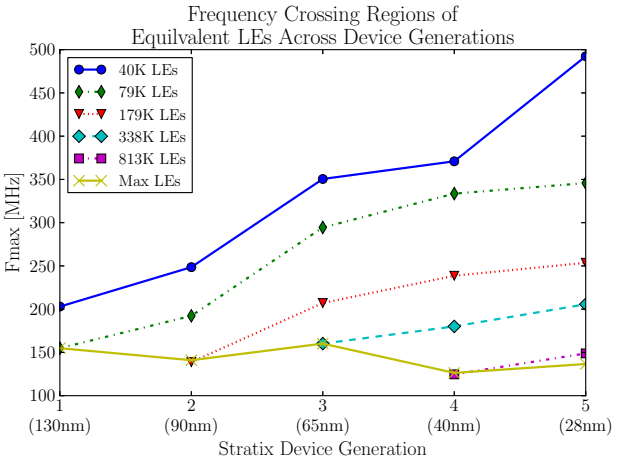


Figure 1: Achievable register to register operating frequency across regions containing an equivalent number of Logic Elements (LEs) for Stratix devices; measured with Altera's Quartus II. Max LEs corresponds to the largest device available each generation.

One solution to the interconnect scaling problem is to insert pipeline registers on communication links that traverse large portions of the chip. This breaks the link into shorter segments which can operate at higher speed, and allows multiple clock cycles for the signal to propagate.

The problem with this solution is that it modifies the latency of the communication link. This changes the RTL behaviour of the system, requiring the re-design and re-verification of the system's control logic. Furthermore, the impact of these RTL changes are not known until after the time consuming physical design flow (which may take multiple days [16]) has been completed, making this a slow and iterative process.

Latency Insensitive Design (LID) [4] has been proposed as a design methodology to avoid these issues by making design components insensitive to the latency of the communication between them. This enables the pipelining of communication links while ensuring that the correctness of the design does not change. As a result the insertion of

[1]This is particularly important for FPGAs where interconnect already contributes significantly to overall delay.

FPGA'14, February 26–28, 2014, Monterey, CA, USA.

ACM 978-1-4503-2671-1/14/02 ...$15.00.
http://dx.doi.org/10.1145/2554688.2554786.

pipeline registers could potentially be pushed to later stages in the design flow, since they no longer require the designer to manually change the system's RTL description. This further abstracts the design from the implementation details of the FPGA, potentially enhancing the timing portability of designs when re-targeting larger or newer FPGAs. Additionally, this also makes the process of inserting pipeline registers much more amenable to design automation. Potential CAD optimizations could include automatic pipeline register insertion during early floorplanning, or pipeline register insertion during routing. These capabilities could be beneficial for architectures featuring pipeline registers embedded in the routing fabric [18, 7].

However, the extra flexibility and abstraction provided by LID will come at some cost. This has not been well characterized, particularly for FPGAs. This paper aims to quantify the costs of latency insensitive communication on FPGAs and present design recommendations to help minimize overhead. Our contributions include:

- Quantification of the area and frequency overhead of LID on FPGAs
- Identification of potential frequency limitations in LI systems and optimizations to improve operating frequency
- A comparison of the efficiency of LI and non-LI pipelining
- Design guidelines for determining the LI communication granularity appropriate to produce area-efficient systems

2. LATENCY INSENSITIVE MOTIVATION

Several different design methodologies have been proposed to address the design issues outlined in the introduction. This section compares these and further explains why this paper focuses on latency insensitive design.

2.1 Limitations of Synchronous Design

Synchronous design is the dominant paradigm for digital design. This is largely due to its amenability to design automation, simple conceptual model and flexibility. However, synchronous design is also restrictive, enforcing the *synchronous assumption* – that all communication must occur within a single clock cycle. On modern devices where it may take multiple clock cycles to traverse the chip this can be too restrictive.

The work-around, adding pipeline registers, is time consuming and error prone. After compiling their system and identifying timing issues, designers must manually insert pipeline registers, modify their system's control logic and re-verify the overall system. They must then re-compile their system (which could take days) before being able to evaluate the impact of their changes. However, after this process, the problem may not be solved. Timing paths may have moved or new critical paths could have appeared, requiring the whole process to be repeated with no guarantee of convergence.

2.2 Why Not Wave-Pipelining?

In a conventional synchronous system each data bit transmitted along a wire must be latched by a clocked storage element before the following bit is launched. With wave-pipelining, multiple data bits are allowed to be in flight along the same wire. This allows the interconnect to behave as if pipelined – with the wire itself storing the multiple data bits in flight rather than registers. This saves the area, power and timing overhead of using registers. It was shown in [20] that wave-pipelined interconnect could be used in an FPGA.

Wire-pipelining however, does not avoid the problem of re-designing a system's control logic to account for the additional communication latency, and also introduces further design issues. Since no stable storage element is used to separate the multiple bits transmitted along a wire, wave-pipelining systems must be meticulously designed to ensure correct operation and avoid interference between subsequent bits. One challenge for these systems is that they can not be run at lower speeds, which makes debugging difficult. This undesirable behaviour is caused by tying the latency of a wave-pipelined link to the (constant) delay of a wire, rather than to the number of registers. As a result, the effective latency of a wave-pipelined link changes with clock frequency. Additionally, wave-pipelining systems must operate robustly in the presence of die-to-die and on-chip variation, as well as in the presence of crosstalk and power supply noise [20]. These non-idealities are expected to become more significant in future process technologies, and the flexibility of FPGAs would make verifying such systems difficult.

Wave-pipelining does not resolve the problem of re-designing control logic, introduces additional limitations to system behaviour, and increases design complexity. As a result, wave-pipelining fails to be a practical solution.

2.3 Why Not Asynchronous Design?

Asynchronous design has long been touted as an alternative to synchronous design. Under this design methodology no clock is used to enforce globally synchronized communication. Instead components of the design detect when their inputs are valid and only then compute their results.

However, despite decades of research, asynchronous design methodologies have seen limited adoption. The reasons for this include a lack of CAD flows and tools to implement and verify designs, the difficulty designers have reasoning about the correctness of their systems, and the challenges of testing asynchronous devices [9].

2.4 Why Not GALS?

Another alternative design methodology is Globally Asynchronous Locally Synchronous (GALS). In this methodology small sub-modules are designed synchronously, but global communication between modules occurs asynchronously, typically through a wrapper module. This allows timing paths to be isolated within each sub-module easing timing closure. Furthermore, since smaller more localized clocks with lower skew are used, this may help to improve performance and power.

One of the key challenges in any GALS design methodology is avoiding metastability when transferring data between sub-modules, since their clocks are no longer synchronous. Several different GALS design styles have been proposed to address this issue [19, 12]. One approach is based on pausable clocks, where each sub-module has a locally generated clock which is paused before data arrives to ensure that metastability is avoided. Alternately, GALS can be implemented using asynchronous FIFOs to handle communication

between sub-modules. Additionally in some cases, where the relationships between sub-module clocks are known, conventional flip-flop based synchronizers can be used.

On current FPGAs, it is not possible to locally generate clocks for sub-modules as would be done on an ASIC. As a result these clocks would have to be centrally generated (with a PLL/DLL) and distributed to the local sub-modules. FPGAs typically contain a relatively small number of fixed clock networks, consisting primarily of global, and large regional/quadrant clock networks. Since these clock networks are pre-fabricated, there is not much to gain (in terms of skew and power) by using them to distribute small clocks. This is different from an ASIC where custom smaller clock trees can be designed. While FPGAs do also support some smaller fixed clock networks, these are typically quite small (limiting the size of sub-modules), restrict placement flexibility, and may be difficult to reach from clock generators. While it is possible to distribute clocks with the regular inter-block routing, it is undesirable. The inter-block routing network is not designed for clock distribution, lacking shielding (increasing jitter), and having unbalanced rise-fall times which may distort the clock waveform. Such a clock network would also consume more power and typically have more skew than an equivalent fixed clock network.

GALS also faces problems similar to fully asynchronous design for the asynchronous portions of the system, including difficulty implementing, verifying and testing such systems. While CAD flows for GALS design are perhaps better developed than for fully asynchronous design, they still require substantial design knowledge and manual intervention [23]. These challenges make adopting a GALS design methodology for FPGAs quite disruptive.

2.5 Latency Insensitive Design

LID can be viewed as a middle ground between the synchronous and asynchronous design methodologies. It breaks the synchronous assumption, but does not go so far as to totally remove global synchronization. Instead, LID allows the latency of communication links to vary. This means that while communication is still synchronized to a clock at the physical level, it may take multiple clock cycles for communication to occur in the designer's RTL description.

This yields additional flexibility during the design implementation process compared to synchronous design, but is more tractable than asynchronous design. Keeping communication synchronous at the physical level means conventional synchronous CAD flows and tools can be used to implement designs, and designers can still reason about the correctness of their systems from the perspective of timing constraints. Additionally, emerging FPGA communication styles such as embedded NoCs [6, 1] result in variable latency communication, essentially requiring designs to be latency insensitive. LID also does not require modification of existing FPGA architectures, as would be required to fully support wave-pipelining [20], asynchronous, or GALS [17] design styles.

Furthermore, the formal theory of latency insensitive design [4] shows that any conventional synchronous system, typically called a *pearl*, can be transformed into a latency insensitive system, provided it is stall-able[2]. This is accomplished by placing it in a special (but still synchronous) wrapper module, typically called a *shell*. The theory further shows that such wrapped modules can be composed together, and the latency of communication links between them varied, by inserting relay-stations (analogous to registers), without affecting the correctness of the overall system.

An example system is shown in Figure 2. The logical system, as described by an RTL designer, is shown in Figure 2a. After implementation with a latency insensitive CAD flow the design implementation may appear as in Figure 2b.

The scheme described above (and in additional detail in Section 3) implements dynamically scheduled LID, where the validity of a module's inputs are determined dynamically at run time by the shell logic. Statically scheduled LID schemes have also been proposed [5], which determine when inputs are valid at design time before implementation. As a result, statically scheduled LID severely limits the flexibility of the system implementation and significantly restricts any potential CAD optimizations, such as automated pipelining. It also precludes operation with variable latency interconnect such as an NoC. Accordingly, we evaluate only dynamically scheduled LID in this work.

An interesting question is what level of granularity is appropriate for latency insensitive communication. While it is possible to use latency insensitive communication at a very fine level, this is not necessarily required. As shown in Figure 1 local communication can still occur at high speed, the problem is with long distance (global) communication. As a result it may make sense to implement latency insensitive communication at a coarser level that captures primarily global communication.

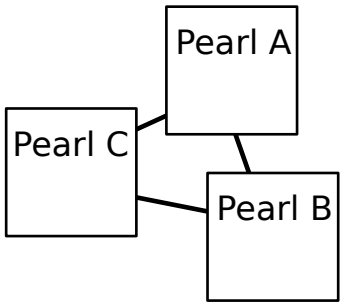
Some previous work has looked at latency insensitive communication in FPGA-like contexts. In [8], explicit latency insensitive communication was used to improve the design and implementation of multi-FPGA prototyping systems. The authors of [11] proposed an elastic CGRA architecture exploiting latency insensitive communication to avoid static scheduling, and to allow simpler translation of high level languages (i.e. C) into circuits. For their system, which implements latency insensitive communication for each ALU element, they identify the area and delay overhead of their elastic CGRA (compared to an inelastic CGRA) as 26% and 8% respectively. The work presented in [3] describes an FPGA overlay architecture that uses latency insensitive communication. The authors report area overheads (compared to a baseline system) of 3.4× and 10.6× for a floating point and integer based overlay respectively. The high overheads can be attributed to the additional routing flexibility required for the overlay and the use of fine-grained latency insensitive communication.

This work differentiates itself from the above by focusing on the overheads of using latency insensitive communication for RTL design targeting conventional FPGAs, rather than as part of an overlay layer or hardened into a device architecture.

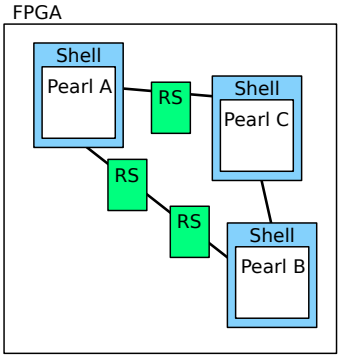
3. LATENCY INSENSITIVE DESIGN IMPLEMENTATION

In order to quantify the costs of a LI design methodology we have created a set of LI wrappers and relay stations based on those presented in [14] and implemented them on Stratix IV FPGAs. Example wrappers are shown in Figure 3.

[2]Informally, capable of maintaining its state independent of its current inputs (i.e. no outputs combinationally connected to inputs). See [4] for a formal definition.

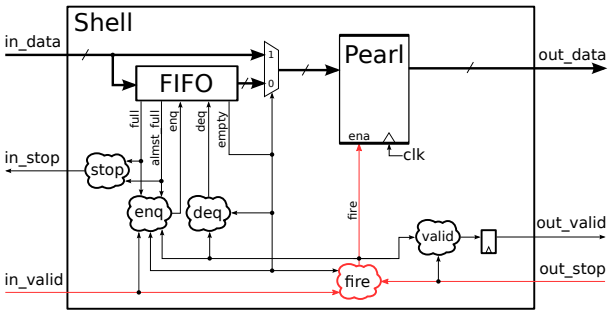


(a) Logical system connectivity.

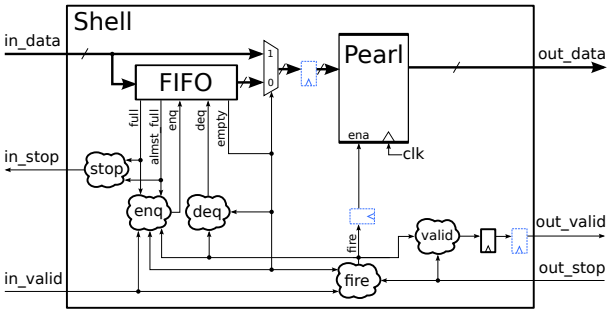


(b) Latency insensitive system implementation, showing shells and inserted relay stations (RS).

Figure 2: Latency insensitive system example.



(a) Baseline latency insensitive wrapper (one input, one output). Critical paths highlighted in red.



(b) Optimized latency insensitive wrapper (one input, one output). Additional registers added in the optimized version shown in dashed-blue.

Figure 3: Latency insensitive wrapper implementations.

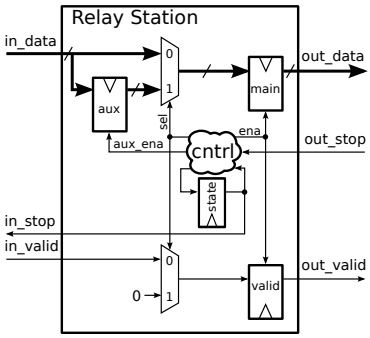


Figure 4: Latency insensitive relay station

One of the key differences between a LI and a traditional synchronous system is the addition of *stop* and *valid* signals on communication channels, forming a 'bundled data' protocol. The *valid* signal allows for data to be marked as invalid and ignored by downstream modules. The wrapper is responsible for stalling the pearl (typically by clock gating) if all of its inputs are not valid. To ensure that no information is lost if valid inputs arrive at a stalled module, they are queued in FIFOs. The *stop* signal provides back-pressure to ensure the FIFOs do not overflow.

Relay stations (Figure 4) are used in place of conventional registers to perform pipelining. Relay stations include additional logic to handle the *valid* and *stop* signals and must be capable of storing two data words to account for the latency of back-pressure communication.

3.1 Baseline Wrapper

The LI wrapper shown in Figure 3a consists of several components. The *pearl* is the original synchronously designed module which is to be made latency insensitive. This is surrounded by a wrapper *shell* which stalls the pearl if one or more inputs are not available, and queues incoming valid data in FIFOs. In [14] stalling was performed by gating the pearl's clock. However, the granularity of clock gating available on FPGAs is very coarse. On some FPGAs the clock is only gate-able at the root of the clock tree [2], requiring a separate clock networks to be used for each gated clock. On other FPGAs clock gating is enabled at lower levels of the clock tree [22]. However, there are still a relatively small number of gating points, and their fixed locations may overconstrain the physical design tools. As a result clock gating was not considered. Instead, the clock gating circuitry was inferred as a clock enable signal sent to all flip-flops in the pearl.

One of the limitations observed with the baseline wrapper was that it reduced the achievable operating frequency of the pearl module (see Section 4.1). Since the motivation behind latency insensitive design is to enable high speed long distance communication, this was undesirable. The cause

was identified as long combinational paths leading from the upstream module's valid signal and from the downstream module's stop signal (highlighted in Figure 3a). As a result the 'fire' logic, responsible for generating the pearl's clock enable signal, was subject to two competing timing paths. This was further exacerbated by the high fan-out clock enable signal. For the relatively small modules presented in Section 4.1, the clock enable fanned-out to nearly 1400 registers. This forced the CAD tool to produce a compromise solution which decreased operating frequency.

One of the largest components of LI wrappers are the FIFOs. To avoid unnecessary stalls these FIFOs require single cycle read/write capability, single cycle updates to full and empty signals and 'new data' behaviour (i.e. the read receives the new data being written) when a write and read occur at the same address. The 'new data' behaviour required additional logic to be inferred around the RAM elements since this mode of operation is not natively supported by the Stratix IV RAM blocks. While it was possible to infer the FIFOs into the MLAB/LUTRAM structures on Stratix IV FPGAs, the choice was left to the CAD tool, which usually implemented them as M9K RAM blocks. Adding native support for 'new data' behaviour in future FPGA RAM blocks would help reduce the overhead associated with these FIFOs.

3.2 Optimized Wrapper

To improve the frequency limitations of the baseline wrapper, an improved wrapper was created by inserting an additional register after the fire logic as shown in Figure 3b. This broke the combinational paths before they became high fan-out and greatly improved achievable frequency. However this required several changes to the wrapper architecture. To ensure that all components remained correctly synchronized with the clock enable signal, additional registers also had to be inserted after the FIFO bypass mux and valid signal generation logic. Overall, this introduces one extra cycle of round-trip communication latency between modules. To handle the additional cycle of latency, the FIFO must reserve an additional word to handle the possibility of an additional data word in flight.

While further pipelining of the LI wrapper was attempted, it resulted in only marginal improvement.

4. RESULTS

To evaluate the cost and overhead of LID, we created a program to automatically generate LI wrappers based on a Verilog module description[3]. This program was used to generate wrappers for a design consisting of cascaded FIR filters, and also to more generally investigate the scalability of LI wrappers.

All area and frequency results were determined by implementing the design with Altera's Quartus II CAD tool (version 12.1) targeting the fastest speed grade of Stratix IV devices. To compare area between implementations that make use of hardened blocks (e.g. DSPs and RAM blocks), we calculated 'equivalent Logic Array Blocks (LABs)' based on the normalized block sizes from [21]. Since Quartus II may purposefully spread out the design soft logic and registers for timing purposes (inflating the number of LABs used), we calculated the required number of LABs by dividing the number of required LUT+FF pairs by the number of pairs per LAB.

4.1 FIR Design Overhead

FIR systems are simple to pipeline manually, because of their limited control logic and strictly feed-forward communication. As a result they do not require LID to enable easy pipelining. An FIR system is used here as a high speed[4] design example, which allows us to quantify the impact of LID while varying the level of pipelining in both the LI and Non-LI implementations. A more general investigation of LID overhead is presented in Sections 4.3 and 4.4.

The FIR filter design consists of 49 cascaded FIR filters as shown in Figure 5. Each of the instances is a 51 tap symmetric folded FIR filter with 16-bit data and coefficients, that is deeply pipelined internally (11 stages) to achieve high operating frequency. The structure of each FIR filter is shown in Figure 6. Its characteristics are listed in Table 1.

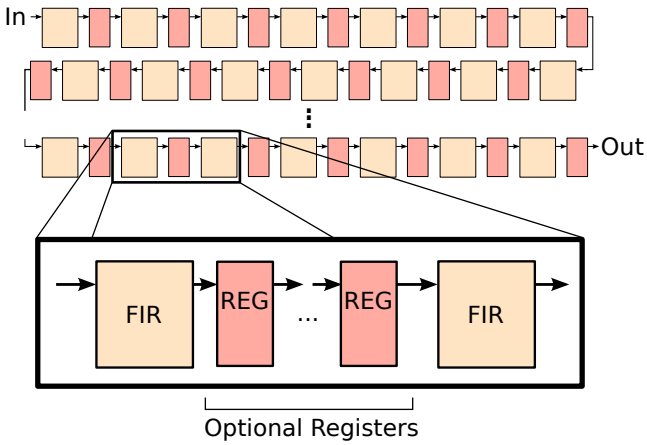


Figure 5: System of 49 cascaded FIR filters with optional registers inserted between instances.

Table 1: Cascaded FIR Design Characteristics

Resource	Number	EP4SGX230 Util.
ALUTs	23,084	13%
Registers	65,256	36%
LABs	4661	51%
M9K Blocks	1	<1%
M144K Blocks	0	0%
DSP Blocks	160	99%

Comparisons of the area and achieved frequency for the LI and non-LI designs are shown in Table 2. In these results each instance of the FIR is made latency insensitive by wrapping it (automatically) using one of the shells from Figures 3a or 3b.

[3]The program, along with the LI wrappers and relay stations are available from: `http://www.eecg.utoronto.ca/~vaughn/software.html`

[4]This is important as it allows us to investigate whether the LI wrappers and relay stations would limit such high speed systems.

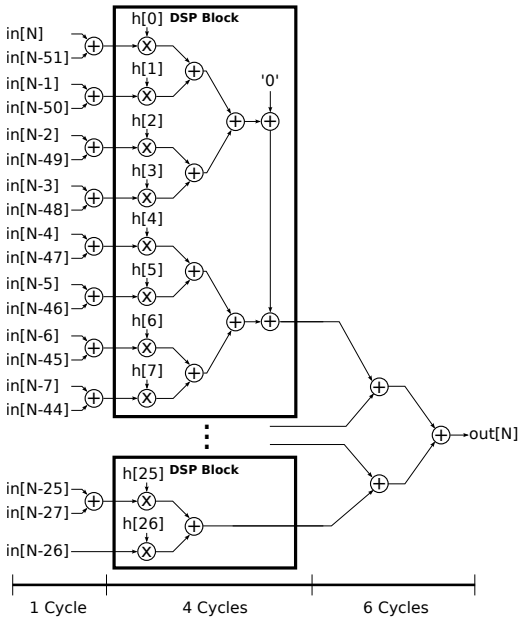


Figure 6: FIR filter architecture. Number of clock cycles required by each portion of the design are annotated.

Table 2: Post fit resource usage and operating frequency for the cascaded FIR design using different communication styles. Values normalized to non-LI system are shown in parenthesis.

Resource	Non-LI	Base LI		Opt. LI	
LUT+FF Pairs	54,940	60,086	(1.09×)	60,299	(1.10×)
DSP Blocks	160	160	(1.00×)	160	(1.00×)
M9K	1	49	(49.00×)	49	(49.00×)
M144K	0	0		0	
Equiv. LABs	4,654	5,049	(1.08×)	5,060	(1.09×)
Fmax [MHz]	377	253	(0.67×)	348	(0.92×)

It is interesting that despite implementing a fine grain latency insensitivity system[5], the area overhead is only 8% or 9%. This could be easily decreased further by implementing latency insensitivity at a coarser level. When viewed from the device level (since many FPGA designs do not fully utilize the device resources) the area overhead amounts to less than 3% of the device resources.

The 33% decrease in frequency, from 377 MHz to 253 MHz, observed when implementing the baseline wrapper (Section 3.1) was both surprising and concerning. This motivated the development of the optimized wrapper (Section 3.2) which improved frequency to 348 MHZ, only 8% below the latency-sensitive system. While this is still a notable impact compared to the non-LI system, it is significantly lower than the baseline wrapper, and comes at only a marginal increase in area overhead.

It was also informative to compare what level of pipelining was required between filter instances when using the LI wrappers to achieve an operating frequency comparable to the non-LI system. As shown in Figure 5 additional pipeline registers (or relay stations) are inserted between FIR filter instances. A summary of these results is shown in Figure 7 for various sizes of the cascaded FIR filter design.

[5]Each FIR module is approximately 95 equivalent LABs in area or 0.6% of the EP4SGX230 device.

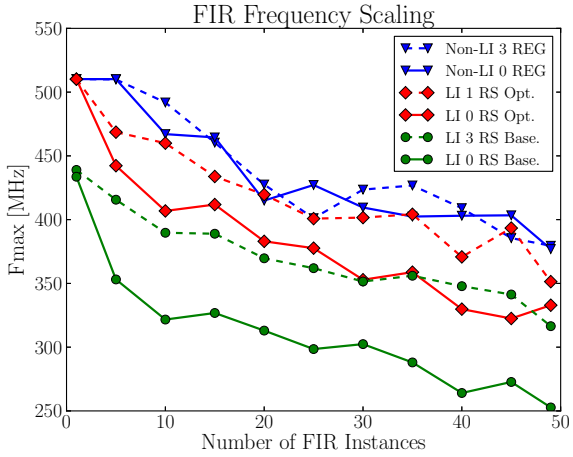


Figure 7: Measured operating frequency versus design size for various communication implementations. The number of registers (REG) or relay stations (RS) inserted between FIR instances are shown in the legend.

The first thing to note is the downward trend in operating frequency associated with increasing design size. This is an artifact of the imperfect nature of the CAD tools used to implement the design. The design is highly pipelined, with no combinational paths between instances. Despite finding a high speed (510 MHz) implementation with one instance in the non-LI system (Non-LI 0 REG) the quality decreases as the design size increases, resulting in a 26% drop in operating frequency when scaling from one to 49 instances. The magnitude of this effect also varies between implementations. For the baseline LI wrapper (LI 0 RS Base.) the frequency dropped 42% across the same range. This disparity is likely a result of the different difficulties these implementations present to the CAD tool, with the baseline LI wrapper containing difficult to optimize timing paths (Section 3.1).

Studying the relative achieved frequency of the different communication implementations, further insights can be drawn. While the baseline wrapper operates at the lowest frequency (LI 0 RS Base.), adding relay stations between filter instances does improve performance (LI 3 RS Base.). However, inserting more than 3 relay stations failed to improve operating frequency. As a result the baseline wrapper fails to match the operating frequency of the non-LI system. The optimized wrapper (LI 0 RS Opt.) performs better than the baseline wrapper, and by inserting only one relay station (LI 1 RS Opt.) performs comparably to the non-LI system. Additional pipelining between filter instances in the non-LI system (Non-LI 3 REG) did not significantly improve operating frequency over the un-pipelined version (Non-LI 0 REG).

4.2 Pipelining Efficiency

One of the interesting questions when comparing different forms of pipelining, whether different latency insensitive implementations or non-LI and LI pipelining, is how much delay overhead is associated with inserting pipeline registers. In the ideal case, on a wire delay dominated path, inserting

a pipeline stage would effectively double the operating frequency. However this is not the reality. The setup and clock-to-q times of registers and, in FPGAs, the cost of entering and exiting a logic block to access registers, all reduce the frequency improvement. In latency insensitive systems there is additional overhead in the form of control logic used to determine data validity and handle back pressure.

To evaluate this, a wire delay limited critical path was created between two instances of the FIR filter from Section 4.1 by constraining the two filters to diagonally opposite corners of the largest Stratix IV device (EP4SE820). The impact of pipelining this long communication link is shown in Figure 8.

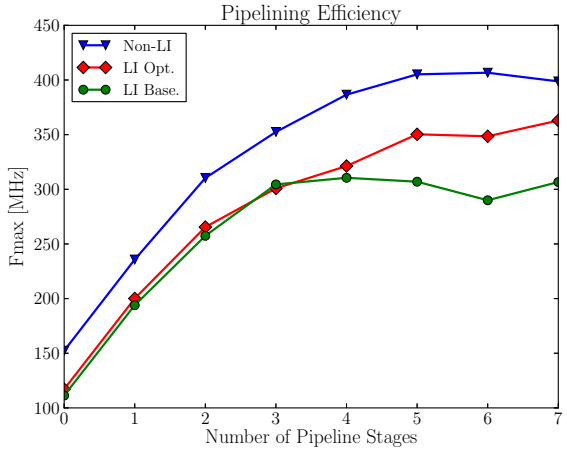


Figure 8: Operating frequency for various numbers of inserted pipeline stages. Results are the average over five placement seeds.

As expected, for an equivalent pipeline depth the non-LI system operates at a higher frequency than the LI systems. The non-LI system ultimately saturates after 5 stages of pipelining. In contrast the baseline LI system saturates after only 3 stages of pipelining and does so at 25% lower frequency. This early saturation is caused by the movement of the critical path from the communication link to the high fan-out clock enable signal internal to the wrappers. The optimized wrapper was not affected by this. While the gap between the optimized LI and non-LI systems grows in absolute terms, the percentage frequency overhead stays fairly constant, ranging from 14-17% for 1 to 5 pipeline stages.

4.3 Generalized Latency Insensitive Wrapper Scaling

While the previous results on the FIR filter design show the potential overheads are manageable, they represent only a limited part of the design space. It is therefore interesting to more generally explore the design space and investigate how LI wrappers scale for different sets of design parameters.

The key design parameters for the LI wrapper are: the number of input ports, the number of output ports, the port widths, and the FIFO depths. While ideally we would investigate all of the interactions between these parameters, this represents a large design space. To decrease the size of this design space, but still gain useful insight into the scaling characteristics of the LI wrappers these parameters were swept individually over a wide range of values.

For the baseline parameters, two input and two output ports were chosen to ensure reasonable control logic was generated, a low port width of 16 was selected to emphasize the scaling impact of ports, and a FIFO depth of 4 (deeper than the typical depth of 1 or 2 words) was used to ensure at least 2 words were available to both the baseline and optimized LI wrappers. While the area results presented do not include the area associated with the pearl used, it is not possible to isolate the pearl's frequency impact. For this reason we chose a very small pearl designed to minimize any impact on the system's critical path. The results are shown in Figure 9. Several useful conclusions can be drawn from the scaling results.

First, as seen in Figure 9a, FIFO depth can be increased with minimal area overhead. This cost is low since the FIFOs are implemented in block RAMs. The large size of these block RAMs means that at shallow depths, the block RAMs are underutilized. As a result, the FIFO depth can be increased at little to no additional cost. This is distinctly different from an ASIC implementation (which would size the FIFO exactly) and highlights the different trade-offs facing FPGA designers. The low incremental cost of increasing FIFO depth may be beneficial for some latency insensitive optimization schemes, which increase FIFO depth to improve system throughput [15]. The frequency overhead of increasing FIFO depth is moderate, staying above 300 MHz until a depth of 16K words.

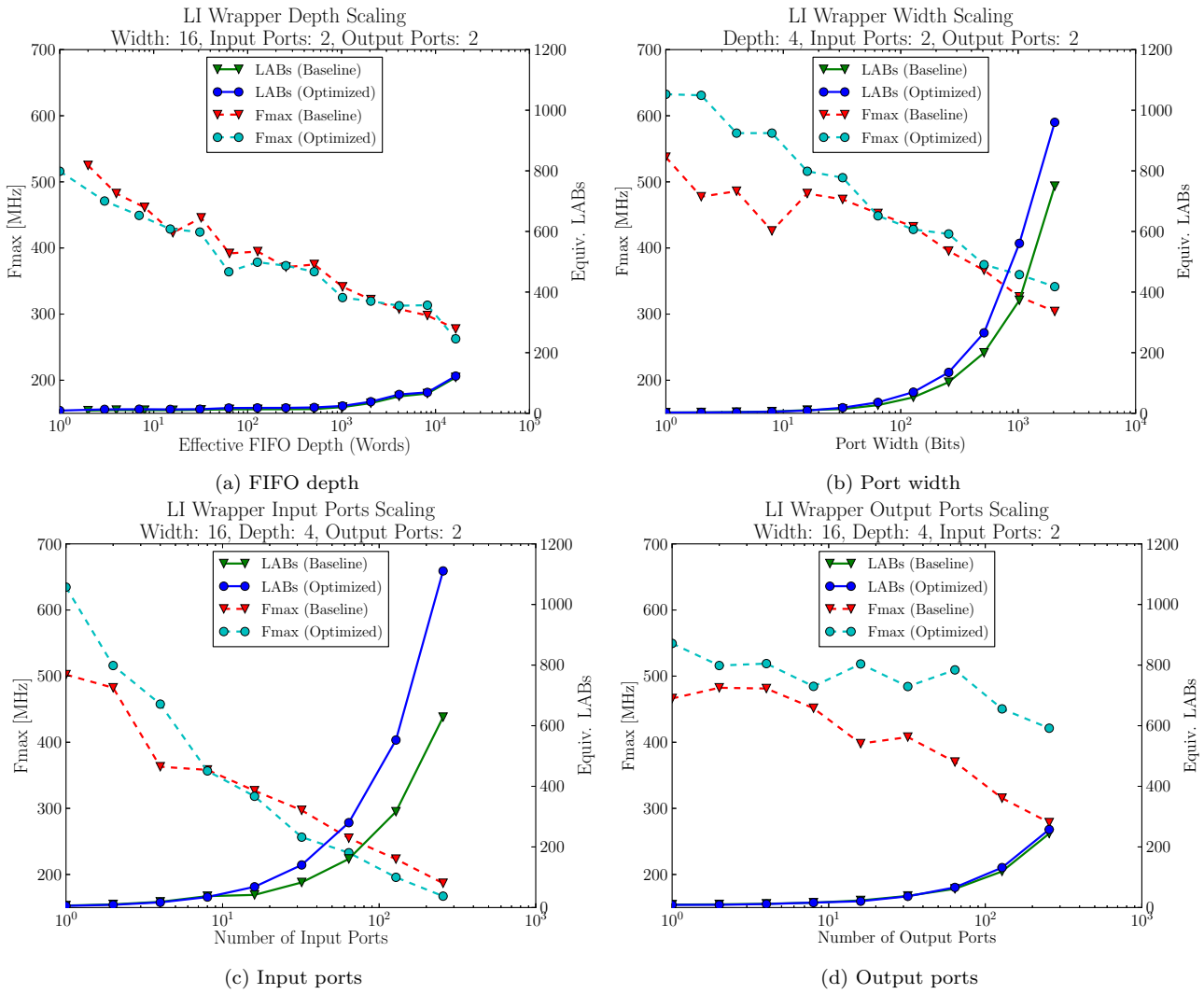
Second, increasing the width of ports (Figure 9b) or increasing the number of input ports (Figure 9c) are fairly expensive, in terms of both area and frequency overhead. However it is interesting to contrast the relative costs of both. Increasing port width results in a lower area overhead than increasing the number of input ports for the same number of overall module input bits. This is perhaps not surprising, since increasing the port width improves the amortization of the FIFO logic, and does not introduce additional control logic (while adding input ports does). The results are similar from a frequency perspective, with scaling input ports more expensive than scaling port widths. The wrappers have no problem operating above 300 MHz (using only two ports) for port widths up to 2048 bits. In contrast, this speed is only possible if fewer than 32 ports are used.

Finally, increasing the number of output ports (Figure 9d) is less costly, since it adds only a small amount of control logic to handle back-pressure and valid signals. It is however, important to note from a system perspective that each output port has an associated FIFO at the downstream input port. Similarly to the area overhead, the frequency overhead of increasing output ports is low, with 300 MHz operation possible with up to 256 output ports.

4.4 Latency Insensitive Design Overhead

One of the challenges when designing a LI system is determining the level of granularity at which to implement latency insensitive communication. To get the most flexibility, a fine level of granularity may be desired, but this could come at an unacceptably large area overhead.

To provide some guidance, a coarse estimate of the area overhead associated with latency insensitive communication for various module sizes was developed by combining the



(a) FIFO depth

(b) Port width

(c) Input ports

(d) Output ports

Figure 9: Latency insensitive wrapper scaling results.

results of Section 4.3 with Rent's rule, which relates I/O requirements to module size.

Rent's rule [13], stated as:

$$P = KN^R \tag{1}$$

is an empirically observed relation between the average number of blocks in a module (N) and its average number of externally connecting pins (P), where K is the average number of pins per block and R is the design dependant Rent parameter. The Rent parameter captures the complexity of the interconnections between modules. A Rent parameter of 0.0 corresponds to a linear chain of modules, such as the FIR design presented in Section 4.1. A Rent parameter of 1.0 corresponds to a clique where all modules communicate with each other. Typical circuits have Rent parameters ranging from 0.57 to 0.75 [13].

It was found for a modern FPGA benchmark set [16] that K was 32.2 for Stratix IV LABs. Assuming the number of pins predicted by Rent's rule split evenly between inputs and outputs, that each port is 64 bits wide, and FIFO depths of 4 are used, it is possible to estimate the area overhead of a module's latency insensitive wrapper based on the data from Section 4.3.

The area overhead of LI communication compared to module size is shown in Figure 10 for various Rent parameter values. It is clear that modules with low to moderate Rent parameters are amenable to the creation of area-efficient latency insensitive systems. Circuits with good communication locality ($0.5 \leq R \leq 0.6$) can achieve low area overhead ($< 10\%$) when wrapping modules ranging in size from 50K to 300K LEs. Circuits with moderate communication locality ($0.6 < R \leq 0.7$) can achieve moderate area overhead ($< 20\%$) when wrapping modules from 160K to 700K LEs in size. Circuits with poor communication locality ($R > 0.7$) are problematic, and will likely result in latency insensitive systems with high area overhead.

5. CONCLUSIONS

In conclusion, a quantitative analysis of the impact of latency insensitive design methodologies on FPGAs has been presented. We have shown that system level interconnect speeds are not scaling, while local interconnect speeds con-

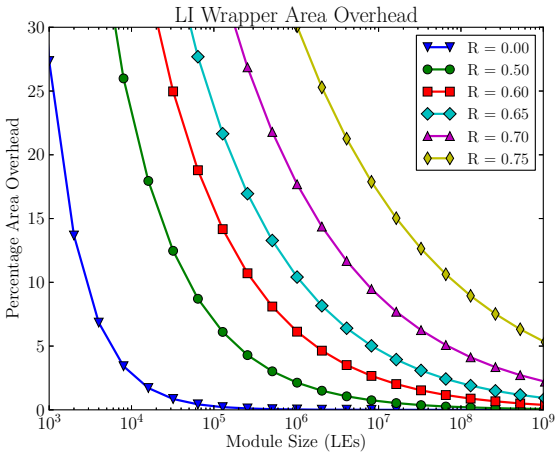


Figure 10: Estimated latency insensitive module area overhead for various rent parameters, assuming equal numbers of input/output pins, 64-bit wide ports and FIFO depths of 4 words.

tinue to improve. This mismatch, along with increasing design sizes, make LI techniques attractive to simplify timing closure, since they allow pipelining decisions to be made late in the design cycle; possibly even by new physical CAD tools. An improved LI wrapper that addresses some of the frequency limitations of conventional LI wrappers was presented, and was used to evaluate the area and frequency overheads of LID. On an example system the area and frequency overheads were found to be only 9% and 8% respectively, with the frequency overhead reducible with further pipelining. The pipelining efficiency of LID was also compared to conventional non-LI pipelining and found to have an overhead of 14-17%. Finally, a more general exploration of the scalability of LI wrappers was conducted, and used to provide guidelines to designers regarding the level of granularity at which latency insensitive communication should be implemented to maintain reasonable area overheads.

While this work shows that the frequency and area overhead of LI systems can be manageable, it remains untenable for some classes of designs, such as those with poorly localized communication ($R > 0.7$) and those unwilling to accept a 14-17% reduction in pipelining efficiency. Previous work on statically scheduled LI systems [5] helps address this, but does so by removing much of the flexibility at late stages of the CAD flow that LID promises. Future work to develop higher performing and lower area overhead LI systems would be beneficial. One potential method to do so would be to improve support for low cost FIFOs requiring 'new data' behaviour in future FPGA architectures.

It would also be useful to extend the overhead quantification to include a power analysis of LID, particularly since unlike ASICs, stalled modules on FPGAs do not have their clocks gated. Similarly further work on evaluating the holistic costs and benefits of LID on real world systems, and with larger more complex benchmarks, would be of value.

While LID will have a cost in area and frequency compared to a perfectly pipelined non-LI system, as design sizes continue to grow and CAD run time for each design iteration increases, the design costs of such systems become large enough to make LID attractive at the system level to interconnect large modules. However, to fully exploit the promise and benefits of LID it must be integrated into CAD flows and exploited by CAD tools to improve designer productivity and design quality.

6. ACKNOWLEDGEMENTS

This work was supported by Altera, Texas Instruments, the Semiconductor Research Corporation, and a QEII-GSST scholarship. We would also like to thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] M. S. Abdelfattah and V. Betz. Design Tradeoffs for Hard and Soft FPGA-based Networks-on-Chip. In *FPT*, pages 95–103, Dec. 2012.

[2] Altera Corporation. *Stratix IV Device Handbook*, September 2012.

[3] D. Capalija and T. Abdelrahman. A High-Performance Overlay Architecture for Pipelined Execution of Data Flow Graphs. In *FPL*, 2013.

[4] L. P. Carloni, K. L. McMillan, and A. Sangiovanni-Vincentelli. Theory of Latency-Insensitive Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[5] M. R. Casu and L. Macchiarulo. A New Approach to Latency Insensitive Design. In *DAC*, pages 576–581, June 2004.

[6] E. S. Chung, J. C. Hoe, and K. Mai. CoRAM: An In-Fabric Memory Architecture for FPGA-based Computing. In *FPGA*, pages 97–106, Feb. 2011.

[7] K. Eguro and S. Hauck. Armada: Timing-Driven Pipeline-Aware Routing for FPGAs. In *FPGA*, pages 169–178, Feb. 2006.

[8] K. E. Fleming, M. Adler, M. Pellauer, A. Parashar, A. Mithal, and J. Emer. Leveraging Latency-Insensitivity to Ease Multiple FPGA Design. In *FPGA*, pages 175–184, Feb. 2012.

[9] S. Hauck. Asynchronous Design Methodologies: An Overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

[10] R. Ho, K. W. Mai, and M. A. Horowitz. The Future of Wires. *Proceedings of the IEEE*, 89(4):490–504, 2001.

[11] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu. Elastic CGRAs. In *FPGA*, pages 171–180, Feb. 2013.

[12] M. Krstic, E. Grass, F. K. Gürkaynak, and P. Vivet. Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook. *IEEE Design & Test of Computers*, 24(5):430–441, Sept. 2007.

[13] B. Landman and R. Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479, Dec. 1971.

[14] C.-H. Li, R. Collins, S. Sonalkar, and L. P. Carloni. Design, Implementation, and Validation of a New Class of Interface Circuits for Latency-Insensitive Design. In *International Conference on Formal Methods and Models for Codesign*, pages 13–22, 2007.

[15] R. Lu and C. Koh. Performance Optimization of Latency Insensitive Systems Through Buffer Queue Sizing of Communication Channels. In *International*

Conference on Computer Aided Design, pages 227–231, 2003.

[16] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. Titan: Enabling Large and Complex Benchmarks in Academic CAD. In *FPL*, 2013.

[17] A. Royal and P. Y. K. Cheung. Globally asynchronous locally synchronous FPGA architectures. In *FPL*, pages 355–364, 2003.

[18] D. P. Singh and S. D. Brown. The Case for Registered Routing Switches in Field Programmable Gate Arrays. In *FPGA*, pages 161–169, Feb. 2001.

[19] P. Teehan, M. Greenstreet, and G. Lemieux. A Survey and Taxonomy of GALS Design Styles. *IEEE Design & Test of Computers*, 24(5):418–428, Sept. 2007.

[20] P. Teehan, G. G. Lemieux, and M. R. Greenstreet. Towards reliable 5Gbps wave-pipelined and 3Gbps surfing interconnect in 65nm FPGAs. In *FPGA*, pages 43–52, Feb. 2009.

[21] H. Wong, V. Betz, and J. Rose. Comparing FPGA vs. Custom CMOS and the Impact on Processor Microarchitecture. In *FPGA*, pages 5–14, 2011.

[22] Xilinx Inc. *7 Series FPGAs Clocking Resources*, March 2011.

[23] A. Yakovlev, P. Vivet, and M. Renaudin. Advances in Asynchronous logic: from Principles to GALS & NoC, Recent Industry Applications, and Commercial CAD tools. In *Design, Automation and Test in Europe*, pages 1715–1724, Mar. 2013.