

Embedded Programmable Logic Core Enhancements for System Bus Interfaces

Bradley R. Quinton, Steven J.E. Wilton
Dept. of Electrical and Computer Engineering
University of British Columbia
{bradq,steve}@ece.ubc.ca

Abstract

Programmable logic cores (PLCs) offer a means of providing post-fabrication re-configurability to a SoC design. Circuits implemented in a PLC will inevitably have lower timing performance and logic density than fixed function circuits. This fundamental mismatch makes the design of the interface between the PLC and the rest of the SoC a challenging problem. In this paper we focus on interfaces between circuits implemented in PLCs and SoC system busses. We demonstrate problems with existing implementation options and then propose modifications to parts of the PLC architecture to enable more efficient system bus interfaces. Our results show that, on average, this modified architecture improves interface timing by 36.4%, reduces CLB usage by 7.9% and improves routability by 28.8% for circuits that require system bus interfaces. We show that the area overhead is less than 0.5% for circuits that do not require bus interfaces.

1. Introduction

Field-Programmable Gate Arrays (FPGAs) provide a low-cost low-risk implementation media for increasingly large and complex systems. However, for some applications, the area, speed, and power overhead of FPGAs makes them unsuitable. For these applications, a fixed-function chip, often designed as a System-on-Chip (SoC), may be required.

Configurability can be provided by embedding one or more programmable logic cores into the fixed-function chip. In such a chip, most of the design is implemented using fixed-function ASIC (Application Specific Integrated Circuit) gates, while programmable logic is used sparingly in those parts of the chip that are likely to change. These changes may be done to correct errors in the design or specification, to enable future upgrades, or to allow for the customization of an integrated circuit for multiple customers. Embedded Programmable Logic Cores (PLC's) can also provide a mechanism to add debug capability [1].

The construction of PLCs has been well studied, and commercial cores are now available [2-5]. However, the integration of these cores into a SoC remains a challenge.

Much of the communication within a large SoC occurs over one or more *system busses* that are used to transfer data between a master (often a processor) and slave IP blocks. Examples of industry-standard busses include AMBA, Wishbone and CoreConnect [6-9]. To seamlessly integrate an embedded PLC into such a SoC, the PLC must be able to participate on the system bus. This requires a flexible interface between the PLC and the system bus. In this paper we describe the implementation of such an interface. On the bus side, the interface is flexible enough to support many different system bus protocols; on the PLC side, the interface is flexible enough to provide a configurable set of interface registers so that the circuitry implemented in the PLC can perform an array of new functions.

The primary technical challenge in implementing such an interface is that it must be flexible enough to support a variety of bus protocols, yet be small and fast. Flexibility can be provided by implementing the interface with programmable logic, yet this will not provide sufficient speed. Speed is especially important, because, for many bus protocols, if one slave is slow, the speed of the entire bus is impacted. Implementing the interface as fixed ASIC circuitry (either outside the PLC or as an embedded block within the PLC) will not provide the required flexibility. Our approach is to provide an interface with *just enough* configurability to support the required flexibility. Most system bus protocols have many similarities, meaning most of our interface can be implemented using fixed logic. However, the bus protocols differ somewhat in their timing and data transfer policies. We provide configurable resources so that the implementation can be tailored for a specific bus protocol. Our architecture is configurable enough to support any of the commonly used standard system busses and should be useful for most proprietary system busses. In addition to protocol flexibility, different circuits implemented in the PLC will often require different interface registers. Our architecture allows the user to configure the number and behaviour of the interface registers.

A second challenge is the integration of the interface circuit into the PLC architecture. The PLC side of the interface circuit contains many signals and

we have found that without careful planning these signals overwhelm any reasonable routing architecture within the PLC. To address this, we break our interface into pieces, and distribute the pieces among certain configurable logic blocks (CLBs) within the PLC. This provides sufficient routability and, at the same time, provides a scalable implementation in that the number of interface registers that can be supported increases as the PLC size increases. We also use the concept of shadow clusters to reduce the area impact when these blocks are not being used [10].

Thus, the contribution of this paper is two-fold: we describe a flexible yet efficient system-bus interface for a PLC, and we describe the integration of this circuitry within an embedded PLC. We evaluate the timing, area overhead, routability, and logic density of a PLC containing our interface using a set of benchmarks that require system bus interfaces.

2. Related work

Although we know of no previous work that directly addresses the problem of improving the efficiency of generic system bus interfaces implemented in programmable logic cores, there have been specific examples of such interfaces described. Winegarden described a proprietary bus to facilitate communications between a processor and user-configurable logic [11]. The bus protocol was specified, and the implementation was carefully pipelined, to ensure high performance and predictable latency. The improvements to area efficiency and timing achieved by this integration were not quantitatively described. While these results provide an indication of the benefits of designing the programmable logic fabric specifically for a given system bus, the results are not directly applicable to the problem addressed in this paper since, in general, the system bus will be defined without regard to the PLC.

Other work in this area has suggested the need for a system bus to PLC interface. Lertora and Borgatti described an SoC containing three PLCs and a microprocessor [12]. Inter-block communication was handled by a Network-On-Chip (NoC), however control and configuration of the PLCs and other peripherals was done using the AMBA AHB bus. The bus interface for the PLC was built using fixed logic. The costs and trade-offs of this decision were not detailed. Work by Madrenas described a Field Programmable System on Chip (FIPSOC) that made use of a PLC on a proprietary shared bus [13]. The details of the PLC/bus interface were not given. Other work has discussed integration without focusing on busses [14-16]. Finally, the current authors have described a post-silicon debug architecture that requires one or more SoC bus to PLC interfaces [1]. In this work it was assumed that the interface could be implemented using the logic provided by the PLC,

however the cost and performance of the interface were not described.

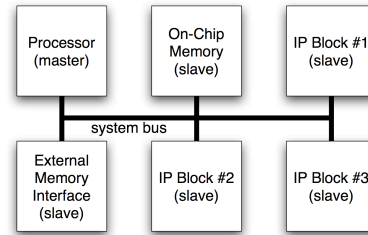


Figure 1: SoC System Bus

3. SoC System Architecture

In this section, we set the context of our work by describing a typical system architecture, and show how a programmable logic core fits into this architecture.

In general, SoC designs are built by integrating one or more embedded processors, along with a number of fixed-function IP blocks, on a single die, as shown in Figure 1. The communication between the software (or firmware) running on the embedded processor and the fixed-function IP is performed using the system bus. Each participant on the system bus is designated as a master or slave; masters, such as the embedded processor, initiate transactions on the bus, while slaves respond to requests initiated by the masters.

Slaves typically provide a memory-mapped interface through a set of control registers, data registers, and status registers. In this paper, we will refer to these registers as *interface registers*. These interface registers are mapped to a region of the software memory space, and masters can access the registers through 'load' and 'store' instructions.

In our architecture, each PLC participates on the system bus as a slave. Therefore the PLC should provide a slave interface to the bus (the PLC may also directly connect to other portions of the SoC, however in this paper we are focused on the system bus interface). A portion of the system bus address space is reserved for potential use by any circuit that is implemented in the PLC logic. This makes it possible to implement new functions in the PLC in a way that is consistent with other IP blocks, and consistent from a software perspective. Each new circuit implementation can provide a different set of interface registers to complement its specific functionality.

A specific example of an application that requires this type of connectivity is post-silicon debug using programmable logic cores [1]. In this application the programmable logic allows debug circuits to be implemented so they interact with the various portions of the fixed function IP blocks. At the same time, the interface to the system bus allows the software to communicate (in real time) with the debug circuit in

order to enhance the hardware debug, or even enable the debug of the system bus transactions.

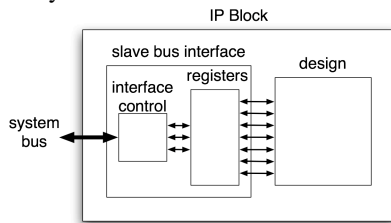


Figure 2: Slave Bus Interface

4. PLC to System Bus Interface

Figure 2 shows the interface circuitry between a system bus and the design implemented in a PLC. The circuitry contains interface registers as well as control logic to coordinate access to these registers. In this section, we present two implementation options, and show that neither of these options may be sufficient.

4.1. Programmable Logic (Soft Implementation)

One option is to use the programmable resources within the PLC to implement the interface. This would provide the flexibility to tailor the interface circuitry to the specific protocol used by the bus. It would also provide for easy and fast connections between the interface registers and the rest of the PLC.

This approach may not provide sufficient speed. Most modern industry-standard SoC system busses are synchronous [6-9]. The slowest path on the bus dictates the maximum operating frequency, and in most cases this determines the maximum throughput of the bus, as shown in Figure 3. Since programmable logic is slower than fixed logic, it is likely that the paths through the PLC will become the critical path in this architecture. For example, in our experiments using 0.18 μ m technology, 64 bytes of registers with an AMBA APB slave interface implemented in generic programmable logic had a critical path of approximately 11ns whereas the same interface implemented with standard cells had a critical path of approximately 1ns. Based on these numbers, it is reasonable to assume that a SoC bus with only fixed-function blocks could operated at 200 MHz, while on a bus with a PLC, the performance would decrease to 62 MHz. This can be a major limitation to the usefulness of the PLC, since the performance of the overall system may deteriorate when the PLC is used.

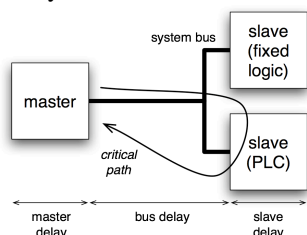


Figure 3: System Bus Critical Path

4.2 Fixed Logic (Hard Implementation)

Another potential strategy is to implement the bus interface using fixed logic (standard cells). The interface could then be placed either outside the PLC or as an embedded block within the PLC. While this addresses the problem of timing performance, it has significant disadvantages. First, since the functionality to be implemented in the PLC is not known when the chip is built, the number and nature of the interface registers needs to be flexible. Second, in some cases, there are several different system busses in a chip, and it may be desirable to provide the flexibility to interface to any of them. An example is the debug application in [1]; in this case, it is impossible to predict (when the chip is designed) which bus will require monitoring during debugging. Third, even if the system bus and number of interface registers can be determined when the chip is designed, using a hard block to implement the interface would complicate the integration, since a separate hard block would be required for each potential system bus. The final difficulty is that the use of a hard block will stress the PLC routing architecture. The PLC side of the interface is a set of interface registers; each bit of each interface register would require a connection to the programmable logic. We have conducted routability experiments and have found that the high concentration of such connections would lead to routing difficulty in the PLC.

5. Proposed PLC Enhancements

As described in the previous section, the soft implementation of the bus interface circuitry may not provide sufficient speed, while the hard implementation may not be flexible enough and may result in an unroutable programmable logic core. In this section, we describe an alternative approach; we create generic bus interface circuitry, divide this circuitry into small subcircuits, and embed each subcircuit into one of several selected CLBs. To minimize area overhead, the shadow cluster approach from [10] is employed. We describe the interface circuitry in Section 5.1, and the integration of this circuitry in our PLC in Section 5.2.

5.1 Functional Requirements

One of the key requirements of our architecture is the flexibility to support a variety of system bus protocols. As described in Section 3.1, there are many common system bus protocols [6-9]; our interface circuitry should support any of these protocols. Note that this is different than if the interface circuitry is implemented strictly in programmable logic; in that case, the interface circuitry can be designed specifically for the protocol being considered.

All of the system bus protocols we have considered are similar, but differ in terms of timing and data transfer policies. We take advantage of these

similarities by creating an architecture that is mostly fixed, but with enough programmability that the differences can be implemented. In this section, we describe how these protocols are similar and how they differ, and discuss how this impacts the architecture of our flexible interface.

5.1.1 Basic Operation

In all protocols, the slave bus interface works by translating master requests, which appear on the bus, into interface register accesses, as shown in Figure 2. The design implemented in the PLC is directly connected to these interface registers.

5.1.2 Protocol Timing

Different bus interfaces have different protocol timing requirements. For instance, AMBA APB requires control information and data to be valid on the same clock cycle, whereas AMBA AHB requires the control information to be valid one clock cycle before the data. In addition, some protocols provide means to allow for variable read and write delays using an acknowledgement signal. To support this, our circuit provides for configurable clock cycle delays for all control, data and acknowledgement signals.

Table 1: Register Bit Types

Type	Description
RW	Read/Write by Master
RO	Read Only by Master
WIC	Write 1 to clear
RWS	Read/Write Sticky
IND	Indirect

5.1.3 Data Granularity

Many system busses support transfers at a data granularity that is smaller than the maximum single-transfer data size. For instance, a system bus may support dword (32-bit) transfers, with byte (8-bit) granularity. There are a number of ways to indicate which of the data bytes should be involved in a data transfer. For instance, CoreConnect OPB provides dword-aligned addresses with a corresponding set of byte enables, whereas AMBA AHB provides byte aligned addressing with byte, halfword, and dword size indications. To support this, our circuit provides both byte enables and size indications, and the address interpretation are also configurable.

5.1.4 Interface Register Bit Types

The behaviour of individual register bits is usually not defined in the system bus specification, instead they are usually design-specific. However, a survey of design standards and IP blocks demonstrates that there is a common set of behaviour that will allow for the implementation of most required designs. These basic behaviours are summarized in Table 1. The RW bit

type is a register bit that can be written and read by a bus master. The RO bit type is a register that only supports read operations by the bus master. The WIC bit type is a register that is used to implement interrupt status registers; it supports normal master read operations, however it only supports master writes of value ‘1’. These master writes clear the register to ‘0’. The register value is set to ‘1’ based on an event internal to the slave. The RWS bit type supports a handshaking operation between the master and slave, allowing the slave to control the timing of events. Finally, the IND bit type supports the triggering of immediate actions in the slave. Our design supports all of these bit types.

5.1.5 Data Bursting

Some system busses, such as AMBA AHB, and CoreConnect OPB support data bursting. Data bursting is used to increase the bus throughput for data transfers involving contiguous address locations in a single slave. Our interface supports these types of transactions.

5.2 Implementation

Based on the requirements outlined in Section 5.1, we designed a flexible circuit that can implement an interface for generic system busses. We expect that our circuits will be useful for any system bus, but we have targeted the following common, industry-standard busses: AMBA APB, AMBA AHB, CoreConnect OPB, CoreConnect DRC, Wishbone Classic, and Wishbone Registered Feedback [6-9]. We have successfully implemented each of the above bus interfaces using our circuitry. In this section we describe our architecture and how it was embedded into the PLC.

5.2.1 Baseline Programmable Logic Architecture

Our technique can be applied to any island-style LUT-based PLC architecture. To make our results concrete, we assume an architecture from [17]. Each clustered logic block (CLB) contains four 4-input LUTs and has 10 inputs and 4 outputs. The configurable logic blocks are arranged in a grid and surrounded by routing channels. The routing channels are composed of length 1 segments. The CLBs inputs are connected to the routing channel with a connection factor, F_c , of 0.5 (meaning each CLB input pin can be selected from half the tracks in the neighbouring channel), and the outputs are connected to the routing channel with a connection factor, F_o , of 0.25 (meaning each CLB output pin can drive up to 25% of the tracks in the neighbouring channel). The switch blocks use the ‘subset’ pattern described in [17]. All switches are buffered.

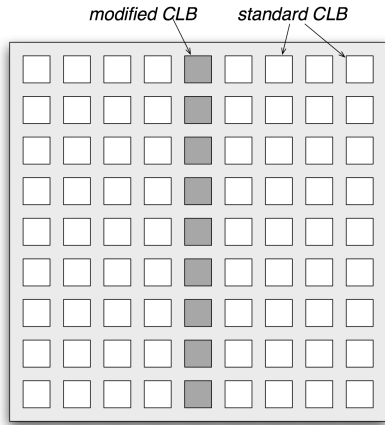


Figure 4: PLC Architecture

5.2.2 PLC Integration

As described earlier, we divided our flexible interface into subcircuits, and embedded each subcircuit in a CLB. As shown in Figure 4, the modified CLBs were placed in a single column in the centre of the chip. In addition to the added circuitry, each modified CLB also contained the normal set of four 4-LUTs. Using the shadow cluster approach from [10], when the interface is required, these LUTs can be disabled, and the dedicated interface circuitry within each LUT used. When the interface is not required, the dedicated circuitry is disabled, and the LUTs can be used as normal.

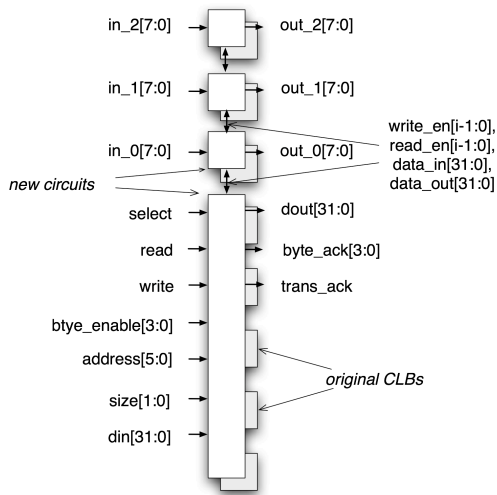


Figure 5: Modified CLBs

The modified CLBs can be classified into two types. The first implements the interface control portion of the bus interface. These CLBs contain the control and data interfaces as well as the address decode and control logic. The generic bus connections of these five CLBs are shown in Figure 5; the lower five CLBs represent the control CLBs. These I/O connections are connected directly to the regular programmable routing fabric of

the PLC. Therefore the system bus can be connected to the regular I/O of the PLC and then routed to the inputs and outputs of the new modified CLBs as needed. The connection to the regular routing fabric adds important flexibility to our architecture since the bus interface signals could potentially be manipulated with programmable logic before connecting to our circuit.

The remaining CLB locations in the column of modified CLBs implement the interface register portion of the bus interface. Each modified CLBs contain eight configurable register bits that represent a single byte in the address map of the bus interface. The system bus access to these bits is controlled by signals generated in the interface control circuits. These control signals (*write_en*, *read_en*, *data_in*, and *data_out*) are hardwired from the interface control circuitry. Each interface register bit has an input and output that connects to the programmable routing fabric. These connections allow the registers to connect the rest of the design implemented in the PLC.

Note that the maximum number of register bits provided by the bus interface depends on the number of rows in the PLC. However, if a single column does not provide sufficient register bits another column of modified CLBs can be added.

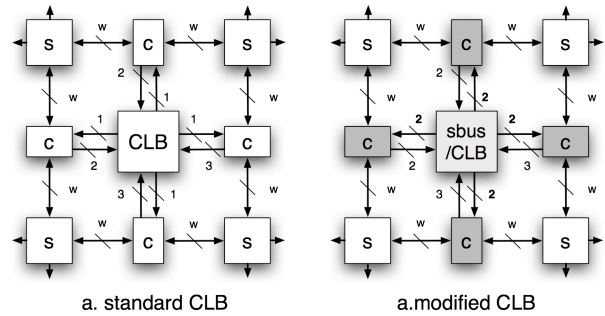


Figure 6: Modified Connection Blocks

5.2.3 CLB to Routing Track Connections

As described earlier, the inputs and outputs of the circuitry embedded into each CLB is multiplexed with the inputs and outputs of the 4x4 LUT structure, and then connected to the regular routing resources within the PLC. For both types of modified CLBs, the number of inputs to the embedded circuitry is the same as the number of inputs to the original 4x4-LUT structure. However, the number of outputs from the modified CLBs is eight, whereas a regular CLB has only four outputs. To accommodate this, additional switches have been added to the connection blocks in the channels on all four sides of the modified CLBs, as shown in Figure 6. As will be shown in Section 8, this has a small impact on the overall area of the PLC.

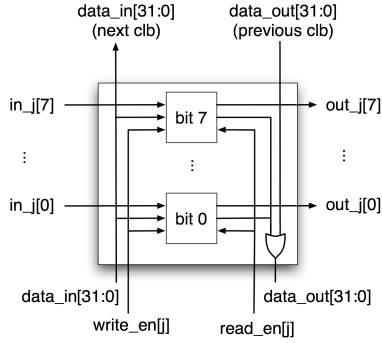


Figure 7: Register Type CLB

5.2.4 New Circuit Details

Each of the CLBs that contain interface registers has 8 configurable bits. These 8 register bits are controlled by a common read and write enable since the minimum address granularity is a byte. Independent read and write enables are generated by the interface control circuits for each set of 8 registers. These are routed directly to the appropriate registers. Each of the bits corresponds to one of 32 bit positions on the data bus, depending on the address of the register bit, as shown in Figure 7.

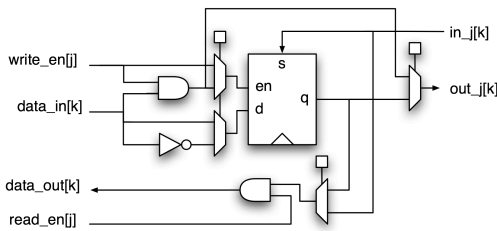


Figure 8: Configurable Register Bit

In order to ensure that the circuit can implement any required slave interface, each register bit can be configured to any of the types in Table 1. This ability is implemented using small amount of combinational logic and a flip-flop, as shown in Figure 8.

6. Results

In this section, we estimate the area and delay overhead of our proposal, and determine how efficiently our new architecture can implement circuits containing bus interfaces. We first describe the set of benchmark circuit used to evaluate our architecture.

6.1 Benchmark Circuits

Because there was no large set of existing benchmarks that use system bus interfaces, we created a new set of benchmarks by combining 20 MCNC benchmarks with bus interface logic. This was done for each of the 20 MCNC benchmarks and for each of the six bus interface standards described earlier. In the new benchmarks a register bit in the bus interface drives each of the inputs of the MCNC designs, and the value

of each output would be determined by a read operation on the bus interface. Some bus interface register bits were configured to be interrupt style bits. Each new benchmark circuit was named to reflect its base MCNC circuit and its bus interface standard, for example 'alu4_apb' for the 'alu4' circuit with an AMBA APB bus interface.

Because of space constraints, we only present the results for the 20 benchmarks that use the AMBA APB bus interface. We believe that this is sufficient since, as indicated in Section 5, the proposed architecture will function correctly for the other interfaces, and we expect the place and route results to be similar for other bus interface standards.

6.2 Area Overhead

The proposed architecture changes increase the number of transistors required to implement the PLC, and therefore the physical area. Tiles containing the modified CLBs required more area than a regular tile because of both the extra transistors required to implement the additional interface circuitry, and the extra switches required in the adjacent connection blocks to connect the extra CLB outputs. The following two subsections consider each of these components.

6.2.1 CLB Overhead

To estimate the area required by the additional circuitry in each CLB, we used the technique of transistor enumeration described in [17]. In [17], it was determined that 1678 minimum-width transistor equivalents were required for the baseline 4x4-LUT CLB architecture. Using the same technique, we found that the modified CLB will require approximately 33% more area than a basic CLB. This overhead is quite modest since the CLB area is a minor portion of the overall tile area.

6.2.2 Overall Overhead

The transistor increase for the routing resources is dependent on the number of normal and modified CLBs in the PLC, and the channel width, W . Since each benchmark requires a different number of CLBs and a different channel width, we extracted the size and channel width required for each benchmark using HHVPR, a modified version of VPR that handles heterogeneous architectures. We then determined the number of transistors required for the routing resources for each PLC configuration using the count provided by HHVPR. We combined these results with the calculations from the previous section to produce total transistor requirements as shown in Table 2. As shown, area overhead for the proposed architecture is quite small, with an average transistor count increase of less than 0.5%. Since each modified CLB can also implement a standard 4x4-LUT, this overhead number represents the area overhead for designs that do not

Table 2 Area Overhead

Bench-mark	Size	W	Base Arch. (10 ⁶ trans)	New Arch. (10 ⁶ trans)	Incr. (%)
alu4_apb	21x21	28	4.34	4.36	0.43
apex2_apb	23x23	34	6.14	6.16	0.36
apex4_apb	19x19	39	4.72	4.74	0.42
bigkey_apb	25x25	26	5.81	5.85	0.78
clma_apb	48x48	43	32.2	32.4	0.33
des_apb	25x25	30	6.51	6.55	0.73
diffeq_apb	20x20	29	4.09	4.10	0.24
dsip_apb	23x23	28	5.20	5.25	0.79
elliptic_apb	31x31	42	13.3	13.3	0.25
ex1010_apb	36x36	35	15.3	15.3	0.24
ex5p_apb	18x18	34	3.78	3.80	0.47
frisc_apb	31x31	41	13.0	13.0	0.26
misex3_apb	20x20	31	4.29	4.31	0.44
pdc_apb	35x35	53	20.6	2.07	0.21
s298_apb	23x23	24	4.61	4.63	0.43
s38417_apb	41x41	29	17.0	17.0	0.23
s38584.1_apb	42x42	34	20.3	20.4	0.41
seq_apb	22x22	37	6.04	6.06	0.38
spla_apb	32x32	47	15.5	15.6	0.23
tseng_apb	18x18	25	2.94	2.97	1.09
				avg:	0.44

make use of the new circuits. For designs that make use of the new circuits the number of required CLBs will be reduced (see Section 6.2) and there will be an overall area savings.

6.2 CLB Usage

To determine the effect on logic density of the proposed changes, we measured the number of CLBs (regular and modified) used to implement the benchmarks for the base architecture and our new architecture. The results are shown in columns 3, 4 and 5 of Table 3. The results show an average CLB decrease of 7.9%.

6.3 Congestion

To determine the effect on congestion of the proposed changes, we measured the minimum required channel width to route each of the benchmark circuits. The results demonstrate significant improvements in congestion as shown in columns 6, 7 and 8 of Table 3. The average reduction in the channel width is 28.8%. We believe this decrease in routing width is primarily due to the fact the new architecture ‘hides’ much of the data bus routing and multiplexing. Note that this decrease in channel width occurs even though the CLBs and CLB pins of the new interface circuits are considered unique and fixed by the placer and router.

Table 3 Place and Route Results

Benchmark	Reg Bits	Base Arch (CLBs)	New Arch (CLBs)	Decr. (%)	Base Arch. (W)	New Arch. (W)	Decr. (%)	Base Arch. (ns)	New Arch. (ns)	Decr. (%)
alu4_apb	20	427	398	6.8	43	28	34.9	5.30	3.65	31.1
apex2_apb	41	536	497	7.3	55	34	38.2	3.98	3.87	2.8
apex4_apb	26	336	337	-0.3	33	39	-18.2	3.98	4.05	-1.8
bigkey_apb	398	648	598	7.7	51	26	49.0	9.75	4.18	57.1
clma_apb	130	2445	2131	12.8	94	43	54.3	9.82	3.83	61.0
des_apb	459	627	573	8.6	50	30	40.0	9.43	4.14	56.1
diffeq_apb	96	452	393	13.1	49	29	40.8	6.47	4.31	33.4
dsip_apb	398	507	388	23.5	45	28	37.8	9.45	4.19	55.7
elliptic_apb	227	992	933	5.9	40	42	-5.0	8.29	4.17	49.7
ex1010_apb	18	1210	1210	0.0	35	35	0.0	3.98	3.66	8.0
ex5p_apb	69	316	295	6.6	44	34	22.7	6.68	4.09	38.8
frisc_apb	144	1011	914	9.6	73	41	43.8	7.18	4.05	43.6
misex3_apb	26	389	369	5.1	46	31	32.6	4.09	3.85	5.9
pdc_apb	54	1307	1198	8.3	80	53	33.8	6.70	3.85	42.5
s298_apb	10	508	498	2.0	39	24	38.5	4.09	3.87	5.4
s38417_apb	131	1781	1627	8.6	88	29	67.0	9.14	4.16	54.5
s38584.1_apb	334	1765	1659	6.0	33	34	-3.0	12.47	4.17	66.6
seq_apb	71	473	458	3.2	36	37	-2.8	6.65	3.86	42.0
spla_apb	58	1049	967	7.8	69	47	31.9	4.08	2.98	27.0
tseng_apb	165	340	289	15.0	41	25	39.0	7.22	3.67	49.2
avg:	144	856	787	7.9	52	34	28.8	6.94	3.93	36.4

6.4 Timing

To determine the effect on timing of the proposed PLC changes we measured the system bus critical path for each of the benchmarks. To ensure that we were focused on the system bus portion of the benchmark, we reported the critical path of the clock domain containing only the system bus interface. The results show a very significant improvement for the benchmarks with a large number of register bits, and an average critical path improvement of 36.4% as shown in columns 9, 10 and 11 of Table 3. Another important aspect of these results is that the standard deviation of the critical paths is much smaller in the proposed architecture (0.30) than in the base architecture (2.51). This is important in practical terms because it means that regardless of the circuit implemented in the PLC the system bus timing will be predictable.

6.5 Overall PLC Area and Timing

To further understand these results, it is informative to consider the impact of the proposed PLC architecture changes on a specific SoC/PLC implementation. To do this we created two representative PLC implementations targeting an SoC. Each implementation was sized to have enough CLBs and routing resources to implement all the new benchmark circuits with the exception of clma (clma was excluded because it is significantly larger than the other benchmarks). Architecture 1 is based on the baseline programmable logic architecture and Architecture 2 used our proposed system bus enhancements. The results showed that Architecture 1 contains 44x44 tiles with a channel width of 88 and required a total of 51.3 million equivalent min-width transistors. Architecture 2 contains 42x42 tiles with a channel width of 53 and required a total of 29.6 million equivalent min-width transistors. The worst-case critical path of the bus interface in Architecture 1 was 12.47 ns, while in Architecture 2 it was 4.31 ns. Therefore, in the case of a PLC targeted at SoCs with the types of designs in the new benchmark set, the area of the PLC would be reduced by 42.3% and the worst-case critical path would be reduced by 65.4% by using the proposed architecture.

7. Conclusions

In this paper, we have outlined the importance of ensuring a circuit implemented in a PLC can interact with SoC software by participating as a slave on the system bus. We have shown the problems with current PLC to system bus interface strategies. To address these problems, we have implemented a configurable bus interface circuit and demonstrated an effective integration of this circuit into the programmable logic fabric. We have shown that the addition of this circuit to the fabric will improve the timing, logic density and routability of designs with system bus interfaces. We

have also shown that this change has minimal area overhead for designs that do not require a system bus interface.

References

- [1] B.R. Quinton and S.J.E. Wilton, "Post-Silicon Debug Using Programmable Logic Cores", *Proc. IEEE Int. Conf. on Field-Programmable Technology*, 2005.
- [2] S.J.E. Wilton, *et al.*, "Design Considerations for Soft Embedded Programmable Logic Cores", *IEEE Journal of Solid-State Circuits*, Vol. 40, No. 2, 2005.
- [3] S. Phillips and S. Hauck, "Automatic layout of domain-specific reconfigurable subsystems for systems-on-a-chip," in *Proc. ACM Int. Symp. Field-Programmable Gate Arrays*, Feb. 2002.
- [4] M2000 FLEXEOS Configurable IP Core [Online]. Available: <http://www.m2000.fr>
- [5] eASIC 0.13um Core [Online]. Available: <http://www.easic.com/products/easicore013.html>
- [6] Opencores.org, *Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*, Revision B.3, Sept. 2002.
- [7] IBM Corporation, *On-Chip Peripheral Bus Architecture Specification*, Version 2.1, April 2001.
- [8] IBM Corporation, *Device Control Register Bus Architecture Specification*, Version 3.5, Jan. 2006.
- [9] ARM Ltd., *AMBA Specification*, Revision 2.0, May 1999.
- [10] P. Jamieson and J. Rose, "Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters", *IEEE Int. Conf. on Field-Programmable Technology*, 2006.
- [11] S. Winegarden, "Bus Architecture of a System on a Chip with User-Configurable System Logic", *IEEE Journal of Solid-State Circuits*, Vol. 35, No. 3, March 2000.
- [12] F. Lertora and M. Borgatti, "Handling Different Computational Granularity by Reconfigurable IC featuring Embedded FPGAs and Network-On-Chip", *Proc. of the IEEE Symp. on Field-Programmable Custom Computing Machines*, April 2005.
- [13] J. Madrenas, "Rapid Prototyping of Electronic Systems Using FIPSOC", *Proc. of the IEEE Intr. Conf. on Emerging Technologies and Factory Automation*, 1999.
- [14] S.J.E. Wilton and R. Saleh, "Programmable Logic IP Cores in SoC Design: Opportunities and Challenges", *Proc. IEEE Custom IC Conf.*, San Diego, CA, 2001.
- [15] P. Zuchowski, *et al.*, "A Hybrid ASIC and FPGA Architecture", *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, 2002.
- [16] P. Magarshack and P. Paulin, "System-on-Chip Beyond the Nanometer Wall", *Proc. Design Automation Conference*, 2003.
- [17] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, 1999.
- [18] J. Lamoureux and S.J.E. Wilton, "On the Interaction between Power-Aware Computer-Aided Design Algorithms for Field-Programmable Gate Arrays", *Journal of Low Power Electronics*, Vol.1, No. 2, 2005.