SMI: SLACK MEASUREMENT INSERTION FOR ONLINE TIMING MONITORING IN FPGAS

Joshua M. Levine, Edward Stott, George A. Constantinides and Peter Y.K. Cheung

Department of Electrical & Electronic Engineering
Imperial College London, SW7 2BT United Kingdom
email: {josh.levine, ed.stott, g.constantinides, p.cheung}@imperial.ac.uk

ABSTRACT

Shadow registers, driven by a variable-phase clock, can be used to extract useful timing information from a circuit during operation. This paper presents Slack Measurement Insertion (SMI), an automated tool flow for inserting shadow registers into an FPGA design to enable measurement of timing slack. The flow provides a parameterised level of circuit coverage and results in minimal timing and area overheads. We demonstrate the process through its application to three complex benchmark designs.

1. INTRODUCTION

The ability to measure timing slack using built-in self-test techniques opens many possibilities in modern digital circuits. Potential applications include on-silicon timing debug, degradation monitoring and dynamic voltage or frequency scaling. In [1] we reported an online timing measurement technique to precisely and continuously measure timing slack at selected registers while the system operates normally. In this paper, we show how this technique can be applied to circuits on mondern FPGAs, using a set of CAD tools, which automatically insert the necessary hardware into an application system. We investigate the cost of such online measurement circuitry in terms of extra hardware resources and timing overhead.

2. PRINCIPLE OF OPERATION

Figure 1 is a block diagram of the measurement technique. A register K, selected for slack monitoring, is a sink node for a block of combinatorial logic that is sourced from a set of registers C. The input D and output Q of register K are forked off to a dedicated measurement cell. D is sampled by a shadow register S on a shadow clock, which has the same frequency as the main clock but with a programmable phase offset ϕ. XOR gate X compares S to Q and produces a discrepancy signal that is latched in register Y.

When the shadow clock and main clock are in phase ($\phi = 0$), S and Q are identical and no discrepancies are generated. As ϕ is increased, S is triggered earlier than D and C and the slack available at S is reduced. Increasing ϕ further causes setup time violation at S and discrepancies are indicated at Y. Signal Q, used by the application circuit, is unaffected. Whilst a single discrepancy indicates that slack at S is exhausted, adding a counter can build up a discrepancy profile for the phase sweep and reveal more information about timing characteristics.

The previous description assumes the delays from source nodes C to registers K and S are identical – in practice, this cannot be achieved without compromising the application circuit. The difference in delays leads to a measurement error, which can be eliminated through a one-off calibration procedure. The calibration process is based on an offline measurement method [2] that uses a frequency sweep to measure the delay at K. Comparing the application circuit delay to that measured by the slack measurement cell gives an offset that is stored and used to correct future readings. No additional hardware is required in slack measurement cell. The resolution of online timing measurement is governed by the minimum possible phase step — this is dictated by the design and configuration of the PLL.

Monitoring all circuit paths is impractical and wasteful in real systems. Assuming that delays in a circuit generally track each other over temperature, voltage and other variations, it is only necessary to measure and monitor the slack at a small number of critical and near-critical registers in a system. We select registers based on the critical delay margin (CDM), where a register K is monitored if the slack S_K at its input satisfies $S_K < S_{crit} \times (100\% + CDM)$. Here, S_{crit} is the critical (minimum) slack for the whole circuit and CDM is the critical delay margin in percent.

Our method differs from others using shadow registers in its exploitation of the rich and flexible clock resources on an FPGA. Using different clock signals to drive the shadow and the main registers and then sweeping the phase of the shadow clock allows us to measure the slack directly, rather than just detecting timing errors [3] or warning of impending guardband violation [4].

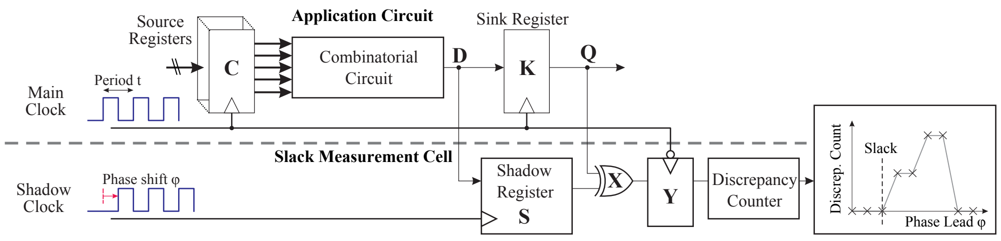

Figure 1: Block diagram of the online timing slack measurement method

3. THE SMI FLOW

The slack measurement insertion (SMI) flow is an end-to-end process that compliments the existing FPGA CAD flow. Similar to scan-path register insertion, slack measurement cells are automatically added after the design has been compiled. However, unlike the ASIC process, SMI must accommodate the architectural restrictions of an FPGA fabric — particularly that certain signals are not visible to the general routing network. Additionally, the shadow register circuit should have minimal impact on the performance of the application.

In this section, the steps in the flow are explained in the context of Altera's Cyclone IV FPGA. Cyclone IV logic elements (LEs) contain features that are found in all FPGAs, although the technique is demonstrated for this architecture, the general principles are transferable.

3.1. Mapping to Cyclone logic elements

Figure 2a is a simplified block diagram of the architecture of a Cyclone LE. Each LE consists of a 4-input lookup table (LUT) to produce a combinatorial signal COMBOUT. This can be registered by the D flipflop to produce the synchronous output Q. The multiplexer circuit MUX, with a select signal SLOAD can be used to bypass the LUT circuit if desired, and the gate G provides synchronous clear capability for the register. In this diagram, all the signals in bold are accessible outside the LUT. LEs are grouped in clusters of 16 to form a Logic Array Block (LAB).

In order to add a slack measurement cell, both signals D and Q need to be monitored by the shadow circuit. The signal D is not available outside the LE and, unless SLOAD and SCLEAR are unused, a solution is needed. We have developed four possible approaches:

Mirroring The shadow register duplicates the configuration of the sink (Figure 2b). The timing behaviours of the two registers will be well matched and the measurement cell can be accurately calibrated. However, the synchronous load and clear signals are LAB-wide, potentially making it more difficult to find a practical placement.

MUX emulation The MUX of the main LE is emulated by the LUT adjacent to the shadow register (Figure 2c). The LUT output is routed directly via COMBOUT to the shadow register. Since the delays in the MUX of the sink LE are different to those in the LUT-emulated MUX, this configuration can introduce measurement error.

Measurement cell enable If the near-critical delay paths arrive exclusively at the COMBOUT then the measurement cell needs only to monitor this signal. The SLOAD signal is used to disable the discrepancy checker so that it operates only when the sink register selects COMBOUT (Figure 2d). The converse can be applied if the critical timing is at SDATA (Figure 2e). This method ensures that there is only one timing-critical path to the shadow register.

Dummy register Another approach for sink registers that are timing critical at only one input is to implement a dummy register. An example is given for SLOAD in Figure 2f. The dummy register is clocked by the main clock and is used for comparison to the shadow instead of the application sink register. Just one signal is forked off to the measurement cell, however, an additional register is required and an alternative calibration method must be used to establish the timing statistics of the sink register.

3.2. Mapping to embedded logic blocks

Embedded blocks also present problems with signal observability. On the Cyclone IV, embedded circuitry comes in the form of 18×18 multipliers and 9kb memories. The SMI strategy for these as are as follows:

Multipliers These have optional registers at their inputs and output. Since one side of these registers is not visible to soft logic, they cannot be monitored by slack measurement cells. Instead, the registers must be pulled out of the multiplier and implemented in soft logic. This could adversely affect application timing circuit performance.

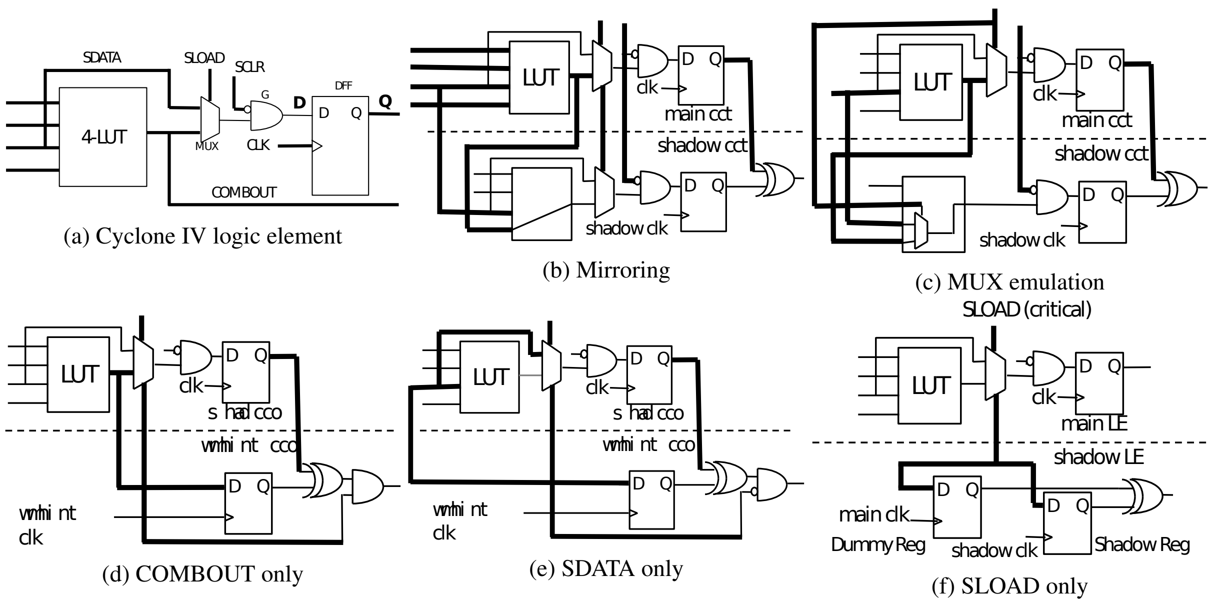

Figure 2: The Cyclone IV logic element and shadow register configurations

Block RAM This used in several ways by the design compiler. It is employed not only as RAM, but as ROM, shift registers, FIFOs and barrel shifters. Registers in block RAM cannot be bypassed, so it is not possible for them to be monitored. In some cases the functionality can be implemented in soft logic instead, for example a barrel shifter, which often performs faster in soft logic than memory.

3.3. The Compilation Flow

Figure 3 depicts compilation using our automatic slack measurement insertion (SMI) flow with Altera Quartus II. Our SMI flow plugs into to the vendor's tools, and can readily be used by a designer without knowing the details of our technique. The input of the flow is a conventional HDL design.

First, a design is synthesised in Quartus and a technology-mapped netlist is extracted. Then, SMI is invoked to add a dummy wrapper to the netlist, isolating the design from the I/O pins. Quartus II is called again for its place-and-route and timing analysis, producing the placement and routing constraints that fix the design while shadow registers are added. The timing report will be used to select which registers to monitored.

Once target registers are identified, SMI checks to make sure that the necessary signals are accessible. If they are not, changes are made to the original design, such as pulling registers out of a hard block, and the design is recompiled. Once the netlist is suitable, slack measurement cells are added by tapping into the relevant signals. Also added is a controller, which is responsible for cycling through a phase sweep and recording the amount of timing slack. This is again sent to Quartus to implement the final design including slack measurement circuitry. Thanks to the constraints extracted earlier, the application performance is largely unaffected by the additional logic.

Table 1: Resource utilisation and timing estimation for the benchmark designs

Benchmark	f_{max} (MHz)	LUT	Reg	Mult	Mem
Adder	204.96	578	481	0	109
Multiplier	126.50	210	320	8	0
Divider	142.90	1514	1412	0	204

4. RESULTS

We applied the SMI tool flow to three benchmark circuits: a single precision floating point adder, multiplier and divider, generated by FloPoCo [5]. The resource usage of the benchmarks and the model-predicted maximum operating frequency (F_{max}) are given in Table 1. The compiled benchmark circuits utilised embedded multipliers and block memory, making them realistic tests for the technique and for our SMI flow. For these benchmark circuits, the mirroring method was used as this is the most generally applicable.

Table 2 gives the number of registers that are selected for measurement in the benchmarks for three CDM settings. The resource overhead after the insertion of the online measurement circuitry to the benchmarks is shown in Table 3.

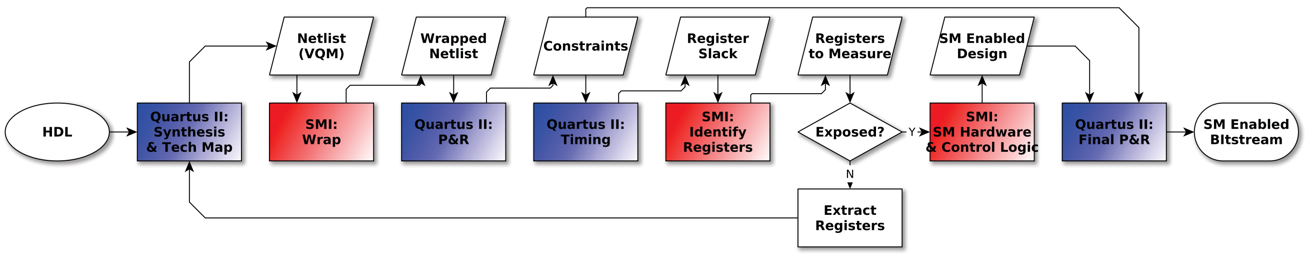

Figure 3: Slack Measurement Insertion Design Flow

Table 2: Measurement cell count for the benchmark designs

Benchmark	5% CDM	7.5% CDM	10% CDM
Adder	9	34	63
Multipler	11	20	25
Divider	17	38	54

Table 3: Resource overhead for measurement cells

	5% CDM		7.5% CDM		10% CDM	
Benchmark	LUT	Reg	LUT	Reg	LUT	Reg
Adder	-31	156	-6	206	23	264
Multipler	11	22	20	40	25	50
Divider	17	34	38	76	54	108

For the multiplier and divider the increase in hardware is minimal and corresponds to two LEs per monitored node, one for the shadow register and the second for the comparison and discrepancy register. In the adder the figures are dominated by the remapping of the barrel shifter from block RAM to soft logic. This has saved a number of LUTs (plus 109 block memory bits) while increasing register usage.

The measurement controller is excluded from these counts, the design and size of this depends on the host system. A basic implementation would require ~50 LEs and be amortised in large applications.

Table 4 shows the timing model performance overhead of SMI for 5% CDM. Since the flow preserves placement and routing of the application circuit, the reported critical path delays are virtually identical. The biggest change is seen in the adder circuit, which has become faster. This is also due to remapping — implementing the shifter in soft logic and not RAM removes the long routes needed to reach the nearest memory block.

Table 4: Online slack measurement performance overhead

	Before	After	
Benchmark	t_{crit} (ns)	t_{crit} (ns)	Increase
Adder	4.88	4.79	-1.8%
Multipler	7.91	7.94	0.4%
Divider	7.00	7.00	0.1%

5. CONCLUSION

SMI is a useful tool flow for automatic addition of timing measurement circuitry to FPGA designs. Applying SMI to benchmark circuits shows that the timing and resource overheads of the technique are low. This work opens opportunities for degradation monitoring and power/performance optimisation in FPGA applications.

The authors acknowledge the support of EPSRC (grants EP/C549481 and EP/H013784). They are also grateful for support from Altera and Terasic.

6. REFERENCES

[1] J.M. Levine, E. Stott, G.A. Constantinides, and P.Y.K. Cheung, “Online measurement of timing in circuits: For health monitoring, dynamic voltage frequency scaling,” in *Proc. IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, 2012.

[2] J.S.J. Wong, P. Sedcole, and P.Y.K. Cheung, “Self-measurement of combinatorial circuit delays in FPGAs,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 2, 2009.

[3] D. Ernst, et al., “Razor: a low-power pipeline based on circuit-level timing speculation,” in *Proc. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2003.

[4] A. Amouri and M. Tahoori, “A low-cost sensor for aging and late transitions detection in modern fpgas,” in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2011.

[5] F. de Dinechin, C. Klein, and B. Pasca, “Generating high-performance custom floating-point pipelines,” in *Proc. Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2009.