파워 로직 디자인

고집적도 FPGA에서 최적의 결과를 얻기 위한 설계지침

오늘날의 FPGA 애플리케이션은 ASIC에 버금가는 복잡성과 성능 요건에 도달하고 있다. 경우에 따라서 FPGA는 대형 ASIC 설계의 시제품으로써 사용된다. 이러한 복잡한 설계에서는 좋은 설계습관이 FPGA의 타이밍 성능과 로직 활용, 시스템 신뢰성에 커다란 영향을 미치게 된다. 이 글에서는 최적의 동기화된 FPGA 설계습관의 이점과 다른 기술에 포함되어 있는 위험요소에 관하여 논의할 것이다.

자료제공: Altera Corporation/www.altera.com

복잡한 설계에서는 좋은 설계습관이 FPGA의 타이밍 성능과 로직 활용, 그리고 시스템 신뢰성에 큰 영향을 미치게 된다. 최적으로 코딩된 설계는 다른 디바이스 제품군이나 속도 등급의 디바이스로 다시 타깃 되더라도 예측이 가능하고 신뢰할 수 있게 된다. 또한 좋은 설계습관은 시제품 및 양산을 위한 FPGA와 ASIC 구현간의 성공적인 설계 마이그레이션을 도와준다. FPGA를 설계할 경우에는 최적의 성능을 위하여 동기화된 설계습관이 주는 영향력을 이해하고, 권장하고 있는 설계 기술을 따르며 FPGA의 향상된 아키텍처 특성을 이해하고 이를 이용해야 한다.

동기화된 FPGA 설계습관

좋은 설계 방법론을 위한 첫 단계는 설계습관 및 기술의 관계를 이해하는 것이다. 지금부터 최적의 동기화된 FPGA 설계습관의 이점과 다른 기술에 포함되어 있는 위험요소들에 대하여 논의할 것이다. 동기화된 좋은 설계습관은 설계 목표를 일관되게 충족시켜 준다. 이와 다른 설계 기술의 근본적인 문제는 디바이스 내부에서의 전파지연(Propagation Delay)에 관한 의존성과 불완전한 타이밍 분석, 그리고 발생 가능한 글리치(glitch) 등을 들 수 있다.

동기화된 설계

동기화된 설계의 기본원칙은 클록 시그널이 모든 이벤트를 트리거(trigger)한다는 것이다. 모든 레지스터의 타이밍 요건이 충족되는 한, 동기화된 설계는 모든 프로세스와 온도(PVT) 조건에서 예측가능하고 신뢰할 수 있는 방식으로 동작한다. 원래 설계자들은 동기화된 설계를 서로 다른 디바이스 제품이나 속도 등급으로 손쉽게 변경할 수 있다. 더욱이 여러분의 설계를 알테라의 HardCopy™ 디바이스와 같은 대량생산형 솔루션으로 마이그레이션 할 계획이라면, 혹은 ASIC의 시제품으로써 사용할 경우 동기화된 설계습관은 마이그레이션이 성공할 수 있게 도와준다.

동기화된 설계의 기본

동기화된 설계에 있어 모든 것은 클록 시그널과 관련 있다. 클록의 모든 액티브 에지(edge)에서(대개의 경우 rising edge에서) 레지스터의 설계 입력은 샘플되고 출력으로 이동된다. 액티브 클록 에지를 따라, combinational logic(레지스터의 데이터 입력을 피딩하는)은 값을 변경한다. 시그널이 여러 번 전환을 하는 동안 로직간의 전파지연 때문에, 이 변경은 instability

의 주기를 트리거하고 결국 새로운 값을 보유하게 된다. 레지스터의 데이터 입력에서 발생하는 변화는 다음 액티브 클록 에지 때까지 그 출력의 값에 영향을 주지 않는다.

레지스터의 내부 회로가 출력으로부터 데이터 입력을 분리하기 때문에 다음의 타이밍 요건이 만족되는 한 combinational logic의 instability는 설계 동작에 영향을 주지 않는다:

- 액티브 클록 에지 전에 적어도 레지스터의 셋업 타임에 대하여 데이터 입력이 정해진다.
- 액티브 클록 에지 후에 적어도 레지스터의 홀드 타임(hold time)에 대하여 데이터 입력이 안정되게 유지된다.

레지스터의 셋업이나 홀드가 위배될 때, 출력은 metastable state라고 불리는 High와 Low 레벨 사이의 중간 전압 레벨로 정해질 수 있다. 그와 같은 상태는 완전히 안정된 것은 아니며 파워 레일의 잡음과 같이 적은 perturbation이 레지스터를 유효상태로 복구시킬 수 있다. 전파지연의 증가나 출력 스테이트의 오류 등 바람직하지 않은 여러 효과들이 발생할 수 있다. 어떤 경우에 출력은 상대적으로 오랜 시간 동안 두 유효상태 사이를 공진할 수도 있다.

비동기화된 설계의 위험 요소

과거에, 설계자들은 종종 FPGA 설계에 리플 카운터(ripple counter)나 펄스 제너레이터(pulse generator)와 같은 비동기식 기술을 사용하였으며, 설계자들은 "short cut"을 택하여 디바이스 자원을 절약할 수 있었다. 비동기화된 설계 기술에는 디바이스 내의 전파지연 의존과 같은 고질적인 문제가 있으므로 불완전한 타이밍 constraints를 발생, glitch나 스파이크 문제를 발생시켰다. FPGA는 고성능 로직 게이트와 레지스터 그리고 메모리를 많이 보유하고 있기 때문에 자원과 성능간의 트레이드-오프가 변화하고 있다. 따라서 자신의 설계 목표를 일관되게 만족시킬 수 있는 설계습관에 중점을 두어야 한다.

어떤 비동기화된 설계구조는 올바른 동작에 대해 시그널의 전파지연에 상대적으로 의존한다. 디바이스에서 설계가 각각의 컴파일 동안 place-and-route 되는 방식에 따라 FPGA의 설계구조는 다양한 타이밍 딜레이를 갖는다. 그러므로 사전에 로직의 특정 블록과 관련된 타이밍 딜레이를 결정하는 것은 거의 불가능하다. 프로세서의 발전으로 디바이스가 보다 빨라짐에 따라 비동기화된 설계의 딜레이는 감소할 수도 있으며, 기대한 바 대로 설계에서 그 기능을 발휘하지 않을 수도 있는 것이다. 특정 딜레이에 의존하게 되면 비동기화된 설계를 HardCopy 디바이스나 ASIC과 같은 서로 다른 디바이스 및 속도 등급의 디바이스로 마이그레이션하는 것이 매우 어렵다.

때때로 비동기화된 설계구조의 타이밍은 타이밍 assignment 및 constraint을 가지고 설정하기에 어렵거나 불가능하다. 만약 완전한 혹은 정확한 타이밍 constraint를 갖고 있지 않다면 자신의 합성툴, place-and-route 툴의 timing-driven 알고리즘을 최적화할 수 없을 수도 있으며, 그 결과는 완벽하지 않을 것이다.

어떤 비동기식 설계구조는 클록 주기에 비하여 매우 짧은 펄스와 같은 glitch를 만들 수 있다. 대부분의 glitch는 combinational logic에 의하여 만들어진다. combinational logic의 입력이 변경되는 경우 출력은 새로운 값을 갖기 전에 많은 glitch를 나타낸다. 그러므로 비동기식 설계에서 적절하지 않은 값이 combinational logic을 통하여 전파될 수 있으며 출력에 부적절한 값을 불러오게 된다. 동기화된 설계에서는 데이터가 클록 에지 때까지 처리되지 않기 때문에 레지스터 데이터 입력의 glitch는 부정적인 결과를 만들지 않는 평범한 사건일 뿐이다.

권장 설계 기술

HDL 코드로 설계할 경우, 합성 툴이 서로 다른 HDL 코딩 스타일을 어떻게 번역하는 지와 그 결과를 이해하는 것이 중요하다. 여러분의 코딩 스타일이 로직 활용과 타이밍 성능에 영향을 줄 수 있다. 이 섹션에서는 신뢰성 저하와 불안정성에 대한 원인들을 제거하면서 FGPA 설계에 있어 최적의 합성 결과를 가져 올 몇 가지 기본적인 설계 기술을 살펴볼 예정이다. 잠재적인 문제의 소지를 피하면서 여러분의 combinational logic을 세심하게 설계하고 동기화된 기능을 유지할 수 있는 클록킹 구조에 만전을 기하자.

Combinational Logic 구조

Combinational loop

디지털 설계에 있어 불안정성과 신뢰성 저하의 원인은 대부분 Combinational loop에 있다. 동기화된 설계에서 모든 피드

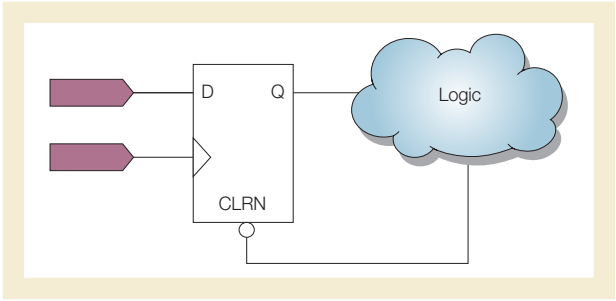

그림 1. 비동기화된 핀을 통한 Combinational loop

백 루프는 레지스터를 포함하여야 한다. Combinational loop는 레지스터 없이 직접적인 피드백을 구축함으로써 동기화된 설계원칙에 위배된다. 예를 들어, 연산의 좌변이 우변에 나타날 때 combinational loop가 발생한다. 또한 Combinational loop는 **그림 1**과 같이 combinational logic을 통하여 레지스터의 출력을 동일한 레지스터의 비동기화된 핀으로 피드백 할 때 발생한다.

다음과 같은 이유로, Combinational loop는 본질적으로 높은 리스크를 수반하는 설계구조이다.

- Combinational loop의 동작은 일반적으로 루프 내의 로직을 통한 relative propagation delay에 달려있다. 논의된 대로 전파지연은 변경될 수 있으며 루프의 동작도 변경되기도 한다.
- Combinational loop는 여러 종류의 설계 툴에서 계산 루프를 끝없이 발생시킬 수도 있다. 대부분의 설계 툴은 진행을 지속하기 위하여 오픈 combinational loop를 단절한다. 설계 플로에서 사용된 다양한 툴은 서로 다른 방식으로 주어진 루프를 오픈 할 수도 있으며 기존의 설계 의도와 다른 방식으로 처리할 수도 있다.

Delay Chain

싱글 fan-in 및 싱글 fan-out을 가진 2개 이상의 연속적인 노드가 딜레이를 유발하도록 사용되는 경우 Delay Chain이 발생한다. 일반적으로 delay chain은 비동기식 설계습관의 결과이지만, 때때로 다른 combinational logic에 의하여 생긴 race condition을 해결하는데 사용되기도 한다. 위에서 논의된 바와 같이, FPGA 딜레이는 각각의 place-and-route와 함께 변경될 수 있다. Delay chain은 설계를 동작조건에 대하여 너무 민감하게 만들거나 설계의 신뢰성을 저하시키고 혹은 다른 디바이스 아키텍처로의 마이그레이션을 어렵게 하는 등 다양한 설계문제를 발생시킬 수 있다.

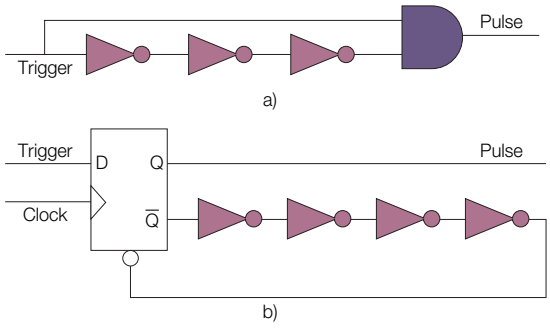

그림 2. 비동기 펄스 제너레이터

딜레이가 칩 주변에서 라우팅되기 때문에 ASIC 설계에서 딜레이는 시그널을 버퍼링하는데 사용될 수 있다. FPGA에서는 로컬 라우팅이 디바이스 전체에 버퍼를 제공하기 때문에 이 기능이 필요하지 않다. 따라서 설계에 delay chain을 사용하지 않고 대신 동기화 설계습관을 가져야 한다.

Pulse Generator 및 Multi-Vibrator

때때로 설계자는 하나의 펄스(pulse generator)나 여러 개의 펄스(multi-vibrator)들을 생성하는데 delay chain을 사용한다. **그림 2** a) 및 b)와 같이 pulse generation에는 2가지 일반적인 방법이 있다; 이 기술들은 순수 비동기식이며 반드시 피해야 할 방법들이다:

a. 트리거 시그널이 2-입력 AND 혹은 OR 게이트의 두 입력을 모두 피딩하지만, 설계는 하나의 delay chain을 그 중 하나의 입력에 invert하거나 add한다. 펄스의 폭은 게이트를 직접적으로 피딩하는 경로와 딜레이를 통과하는 경로의 상대적인 딜레이에 따른다. 이것은 입력의 변화를 따르는 combinational logic의 glitch 발생과 동일한 메커니즘이다. 이러한 기술은 delay chain을 사용함으로써 glitch의 폭을 인위적으로 증가시킨다.

b. 레지스터의 출력은 하나의 delay chain을 통하여 동일한 레지스터의 비동기화된 리셋 시그널을 드라이브한다. 어떤 딜레이 후에는 레지스터는 반드시 그 자신을 비동기식으로 리셋한다.

이러한 설계의 주요 문제는 여러분의 합성 및 place-and-route 소프트웨어가 펄스 폭을 결정하거나, 설정 혹은 검증하는 것이 어렵다는 점이다. 배치 및 배선(placement & routing)을 하고 나서 라우팅 및 전파지연을 알게 되었을 때에만 실제적인 펄스 폭이 결정될 수 있다. 그래서 HDL 코드를 생성할 때 펄스 폭을 신뢰성 있게 결정하는 것은 어려운 일이며, 여러분의 EDA 툴에서 할 수 있는 일이 아니다. 펄스가 모든 PVT 조건에 있는 애플리케이션에 적합하도록 충분히 크지 않을 수도 있으며, 여러분이 서로 다른 디바이스로 마이그레이션 할 경우 펄스의 폭이 바뀌게 될 것이다. 더욱이 static timing analysis가 펄스 폭을 검증하는 데에 사용될 수 없기 때문에 검증이 매우 어렵다.

Multi-vibrator는 회로를 오실레이터로 바뀌게 하는 combinational loop 뿐만 아니라 펄스를 생성하는 데에 "glitch generator"의 원칙을 이용하고 있다. 다수의 펄스를 만드는 구조는 내포된 많은 펄스 때문에 pulse generator보다도 더 많은 문제를 발생시킨다. 게다가 구조가 다수의 펄스를 만드는 경우 또한 펄스들이 설계 내에서 새로운 인위적인 클록을 만든다.

pulse generator를 사용할 필요가 있다면, 그것은 그림 3과 같이 완전 동기화된 기술에 기반하여 구현되어야 한다.

이 설계에서, 항상 펄스 폭은 클록 주기와 동일하다. 이 펄스 제너레이터는 예측 가능하며 타이밍 분석으로 검증될 수 있고, 서로 다른 아키텍처나 디바이스, 혹은 속도 등급으로 손쉽게 마이그레이션될 수 있다.

Latch

디지털 로직에서 Latch는 새로운 값을 갖게 될 때까지 시그널 값을 홀드하고 있다. 또한 설계자가 latch의 사용을 원하지 않을 때에는 값을 홀드하기 위하여 Latch는 HDL 코드로부터 infer 될 수 있다. FPGA는 레지스터-집중적이다: 그러므로 latch를 가지고 설계하는 것은 레지스터를 가지고 설계하는 것보다 더 많은 로직을 사용하게 되며 낮은 성능을 갖게 된다.

Latch는 설계상의 여러 가지 어려움을 유발할 수 있다. 비록 latch가 레지스터와 같은 메모리 요소이기는 하지만 그들은 기본적으로 다르다. Latch가 feed-through나 transparent 모드에 있는 경우, 데이터 입력과 출력간의 직접적인 경로가 있다. 데이터 입력에 있는 glitch은 출력으로 전달될 수 있다. 또한 Latch에 대한 타이밍은 본래 모호한 편이다. D latch를 가

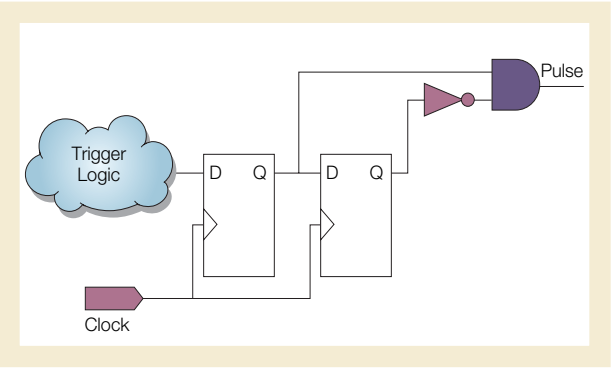

그림 3. 권장 Pulse-Generation 기술

진 설계를 분석해 보면, 예를 들어 소프트웨어는 여러분이 클록의 leading edge에서 데이터를 출력으로 이동하려고 하는지 혹은 trailing edge에서 그런지를 결정할 수 없다. 많은 경우에 오직 초기의 설계자만이 설계의 전체 내용을 알고 있으며, 이것은 다른 설계자가 설계를 마이그레이션하거나 혹은 코드를 재사용하기가 쉽지 않다는 것을 의미한다.

Combinational Logic을 설계하는 경우, 어떤 코딩 스타일은 전혀 의도하지 않았던 latch를 만들 수도 있다. 예를 들어, CASE 혹은 IF 구문이 있을 수 있는 모든 입력 조건을 커버하지 않는 경우, 새로운 출력 값이 배정되지 않는다면 latch는 출력을 홀드해야 할 수도 있다. IF 혹은 CASE 구문에서 최종 ELSE 조항이나 WHEN OTHERS 조항을 생략하는 것 또한 latch를 발생시킬 수도 있다. 의도하지 않았던 latch 발생을 막기 위하여 default CASE 혹은 final ELSE 구문을 "don't care" 값에 배정하여야 한다.

클록킹 Scheme

내부적으로 생긴 클록

내부적으로 생긴 클록은 설계상에서 기능상 혹은 타이밍상의 문제를 야기할 수 있기 때문에 가능하다면 그들을 사용하지 말아야 한다. Combinational logic으로 만들어진 클록은 기능상의 문제를 일으키는 glitch을 발생시킬 수도 있으며 combinational logic 때문에 생기는 딜레이는 타이밍 문제를 초래한다.

만약 여러분이 combinational logic의 출력을 클록 시그널이나 비동기식 리셋 시그널로써 사용한다면, 여러분의 설계에

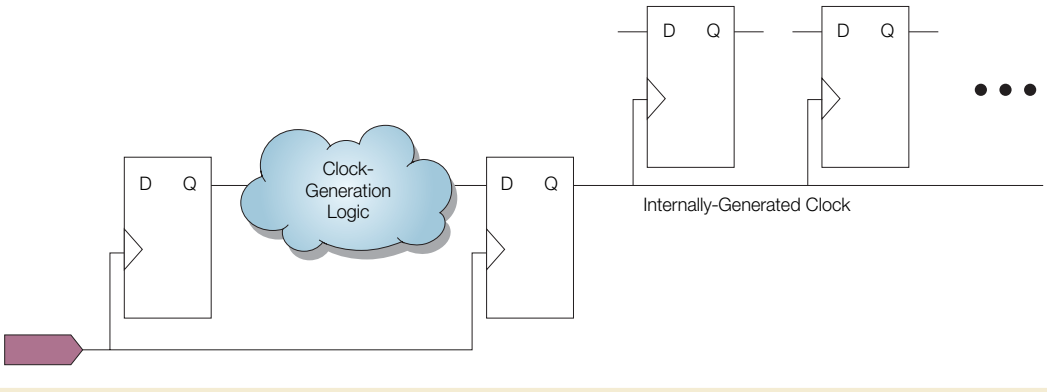

그림 4. 권장 Clock-Generation 기술

glitch가 생기는 것을 발견하게 될 것이다. 동기화된 설계에서 레지스터의 데이터 입력상의 glitch는 일상적인 일로서 문제될 것이 없다. 그러나 클록 입력(혹은 레지스터의 비동기식 입력)에서의 glitch나 스파이크는 중대한 문제를 일으킨다. Narrow glitch은 레지스터의 최소 펄스 폭 요건을 위반한다. Glitch가 클록 입력에 도달하는 경우, 레지스터의 데이터 입력이 바뀌게 된다면 셋업 및 홀드 타임 또한 방해된다. 비록 설계가 타이밍 요건을 위반하지 않더라도 갑자기 레지스터 출력 값이 변경되어 설계의 다른 곳에 기능상의 위험요소를 유발할 수 있다.

이러한 문제때문에 클록 시그널로서 combinational logic을 사용하기 전에 항상 그 출력을 레지스터링 하는 것이 좋다(그림 4 참조).

이렇게 레지스터링을 하면 combinational logic에 의하여 생긴 glitch가 레지스터의 데이터 입력에 블록되게 해준다. 내부 클록을 발생시키는데 사용된 Combinational logic은 또한 클록 라인에 딜레이를 증가시켜 준다. 어떤 경우에 클록 라인 위의 로직 딜레이는 clock skew가 두 레지스터 사이의 데이터 경로 길이보다 더 크게 만들기도 한다. 만약, Clock skew가 데이터 딜레이보다 크다면 레지스터의 타이밍 매개변수는 손상될 것이며, 설계는 제대로 동작하지 않을 것이다. 클록 도메인 내의 clock skew를 감소시키려면, 생성된 클록 시그널을 FPGA 디바이스 내의 high-fan-out 및 low-skew 클록 트리의 하나(존재한다면)로 배정하자. 알테라의 글로벌 시그널과 같은 low-skew 클록 트리를 사용하면 시그널에 대한 전체 clock skew를 줄일 수 있을 것이다. 알테라의 Quartus® II 소프트웨어에서, 여러분은 글로벌 시그널로서 Assignment Editor를 사용하여 노드를 지정할 수 있다.

분주된 클록

많은 설계가 마스터 클록을 분주함으로써 생기는 클록들을 필요로 한다.

많은 FPGA는 알테라의 PLL(phase-locked loop)과 같은 클록 분주을 위한 전용 회로를 제공하고 있다. PLL 회로를 사용하여, 비동기식 클록 분주 로직에 의하여 발생할 수도 있는 많은 문제를 방지할 수 있다.

로직을 사용하여 하나의 마스터 클록을 분주하는 경우, 항상 동기식 카운터나 스테이트 머신을 사용하기 바란다. 또한 위에서 상세히 설명한 바와 같이 레지스터가 항상 분주된 클록 시그널을 직접 발생하도록 여러분의 설계를 구축하라. 여러분의 설계는 절대로 클록 시그널을 만들기 위하여 카운터나 스테이트 머신의 출력을 디코딩해서는 안된다; 이러한 구현 유형은 가끔 glitch를 생기게 하기 때문이다.

Ripple Counter

FPGA 설계자는 설계의 용이성과 동기화된 카운터에 비하여 보다 적은 게이트를 사용하는 2의 승수로 클록을 분주하는 ripple counter를 구현하기도 한다. Ripple Counter는 cascade된 레지스터를 사용하며, 이 경우 각각의 레지스터의 출력 핀이 다음 단계의 레지스터 클록 핀을 피딩한다. 카운터가 각각의 단계에서 리플(ripple) 클록을 발생하기 때문에

cascade하는 것이 문제를 일으킬 수도 있다. 이 리플 클록은 타이밍 분석에서 그런 것처럼 취급되어야만 하며, 따라서 매우 어려운 일이며 적어도 여러분은 합성 및 place-and-route 툴에 적절한 타이밍 assignment를 입력해야 할지도 모른다. 여러분은 검증작업을 간편하게 하려면 되도록 이러한 유형의 구조를 피하고자 노력하여야 한다.

Multiplexed Clock

클록 다중화는 서로 다른 클록 소스로 동일한 로직 기능을 동작시키는데 사용될 수 있다. 어떤 multiplexing logic 종류는 클록 소스를 선택한다. 예를 들어, 다수 주파수 표준을 취급하는 통신 애플리케이션은 때때로 다중화된 클록을 사용하기도 한다.

클록 시그널에 multiplexing logic을 추가하면은 앞의 섹션에서 언급한 문제를 야기할 수 있지만, multiplexed clock에 대한 요건은 애플리케이션에 따라 매우 다양하다. 클록 multiplexing은 다음의 경우에 허용할 수 있다.

- 클록 multiplexing 로직이 초기 컨피규레이션 후에 변하지 않는다.
- 설계에서 테스트를 목적으로 클록을 선택하는 데에 multiplexing logic을 사용한다.
- 여러분은 클록을 스위칭할 때 항상 리셋을 적용한다.
- 클록 스위칭 이후에 칩이 잠깐 나타내는 부적절한 반응에 별 의미가 없다.

만약, 설계가 리셋 없이 실시간 연속적으로 클록을 스위칭 하거나 칩의 부적절한 반응을 임시적으로 toleration 할 수 없다면 여러분은 그러한 경우에 레지스터의 타이밍이 위반되지 않고 클록 시그널에 glitch가 생기지 않도록 그리고 race condition이나 다른 로직상의 문제가 없도록 동기화된 설계를 사용하여야 한다.

Gated Clock

Gated Clock은 gating circuitry 종류를 컨트롤하는 enable signal을 사용하여 클록 시그널을 켜고 끈다. 하나의 클록이 꺼지면, 관련 클록 도메인이 폐쇄되고 기능상 휴지상태가 된다. Gated clock은 전력사용을 절감하는데 좋은 기술이 될 수 있다. 클록을 Gating 할 때, 클록 트리와 레지스터 모두 더 이상 토글링하지 않기 때문에 스위칭에 드는 전력소모를 없애준다.

Gated Clock은 동기화된 scheme의 일부가 아니므로 어떤 경우에는, 문제점을 발생시켜 설계의 구현이나 검증을 복잡하게 할 수 있다. Gated Clock은 클록 스큐의 원인이 되기도 하고 디바이스 마이그레이션을 힘들게 만든다. 이러한 클록들은 또한 glitch에 민감하며 설계가 실패하게 되는 원인이 되기도 한다.

기능적 관점에서 볼 때, 여러분은 동기화된 클록 enable을 사용하여 순수 동기화된 방식으로 클록 도메인을 폐쇄할 수 있다. 그러나 동기화된 클록 enable 구조를 사용할 때 클록 트리는 토글링을 계속하게 되며, 각각의 레지스터의 내부 회로는 활성상태를 유지하여(비록 출력 값이 변하지 않더라도), 전력소모를 줄이지 않게 된다. 그러므로 여러분은 대부분의 경우 다음 장에 기술되어 있는 동기화된 scheme을 사용하여야 하며, 다만 전력소모를 크게 절감하려면 아래의 clock-gating 권장사항을 참조하자.

동기화된 클록 enable

동기화된 방식으로 클록 도메인을 닫으려면 동기화된 clock enable을 사용하라. 디바이스 레지스터 위에 클록 enable 시그널이 있는 대부분의 FPGA에서 클록 enable 시그널은 효율적으로 지원된다. 이 구조는 클록 트리와 레지스터 내부 회로가 계속하여 토글링하기 때문에 전력소모를 절감하지는 않지만, 레지스터 세트를 disable 시킴으로써 Gated Clock으로서 동일한 기능을 실행한다. 새로운 데이터를 로딩하거나 레지스터 출력을 카피하도록 각각의 레지스터의 데이터 입력 앞에 multiplexer를 집어넣는다(그림 5참조).

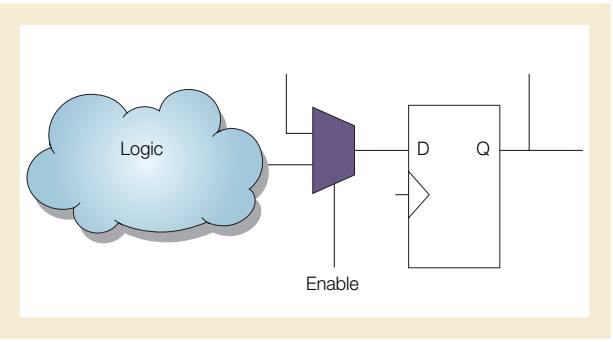

그림 5. 동기화된 Clock Enable

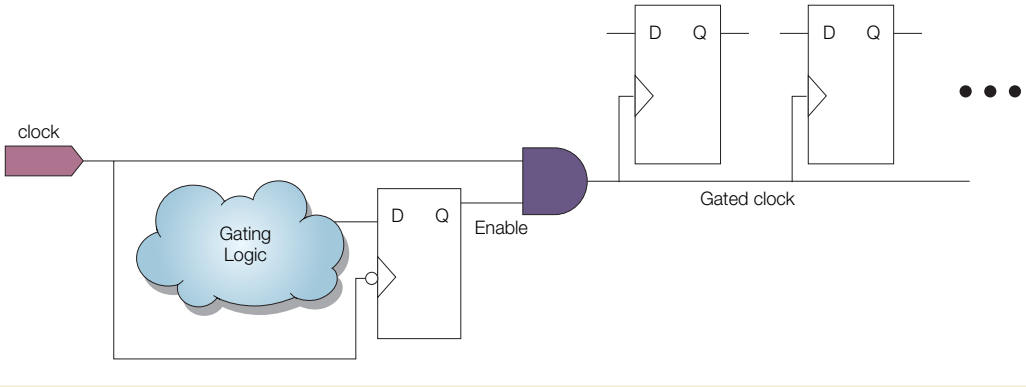

그림 6. 권장 Clock Gating 기술

권장 Clock-Gating 방식

여러분의 타깃 애플리케이션이 전력절감을 크게 필요로 하는 경우에만 Gated Clock을 사용하라. 만약 여러분이 Gated Clock을 사용해야 한다면 그림 6의 Clock-Gating 기술을 사용하여 구현하는 것이 좋다.

여러분은 클록 트리의 root와 leaves 그리고 그 사이 다른 곳에서 클록 시그널을 Gating 할 수 있다. 클록 트리가 스위칭 파워를 제공하기 때문에 여러분이 leaves에서 클록 트리를 따라 gating 하는 대신에 전체 클록 트리를 폐쇄할 수 있도록 root 부분에서 항상 클록을 만든다.

그림 6에서 보는 바와 같이 레지스터는 glitch와 스파이크가 없도록 확실히 하기 위하여 enable 명령어를 만든다. 설계는 gate되는 클록의 비활성 edge에서 enable 명령어를 만들어 주는 레지스터를 클록한다(그림 6에서와 같이 rising edge에서 활성인 클록을 gating 할 때 falling edge를 사용). 이러한 구현으로 한번에 클록을 켜고 끄는 게이트 입력 한 개만이 변하게 되며 출력에 glitch나 스파이크를 발생시키지 않는다. AND 게이트를 사용하여 rising edge에 활성인 클록을 게이트한다. Falling edge에서 활성인 클록에 대하여 OR 게이트를 사용하여 클록을 gate하고 positive-edged 레지스터와 함께 enable 명령어를 레지스터한다.

이러한 구조를 사용할 때, 여러분은 오직 온-타임만이 enable 명령어를 생성할 수 있기 때문에 클록의 duty cycle에 주의를 기울여야 한다. 이러한 상황은 만약 enable 명령어를 생성시키는 로직이 특별히 복잡할 경우, 혹은 클록의 duty cycle이 몹시 불균형을 이루고 있다면 문제를 일으킬 수도 있다. 그러나 duty cycle은 다른 gating clock 방식의 문제점에 비한다면 사소한 이슈 일 뿐이다.

FPGA 아키텍처 특성을 대상으로 하기

다음의 일반적인 FPGA 설계지침뿐만 아니라 여러분의 설계를 원하는 목표 기술로 코딩하는 것이 중요하다. FPGA는 디바이스 전체에 대한 컨트롤 시그널을 제공, 성능을 향상시킬 수 있도록 해준다. FPGA 디바이스는 고성능 로직 게이트, 레지스터, 메모리 및 전용 DSP 회로와 같은 첨단 특성을 포함하고 있다. 다음의 권장 가이드라인을 따라 여러분의 FPGA 아키텍처를 활용하라. 여러분은 여러분의 합성 툴이 서로 다른 HDL 설계기술을 어떻게 전환할 지, 그리고 여러분의 설계가 여러분의 대상 디바이스에 어떻게 매핑될 지를 이해하고 있어야 한다. 특정 FPGA 아키텍처 특성을 활용하면 성능을 높이고 주어진 설계에서 필요로 하는 로직의 양을 줄일 수 있다. ASIC을 프로토타이핑할 때, 설계에 메모리와 다른 특성에 대한 컨피규레이션을 제한할수록 대부분의 FPGA에 여러분의 설계를 더 쉽게 프로토타이핑할 수 있다.

클록 트리와 디바이스 컨트롤 시그널 자원

ASIC 설계에서, 설계 과정상 가장 중요한 부분은 클록 딜레이가 칩 전반에 분배되기 때문에 그들끼리의 균형을 맞추는 것

이다. 대부분의 FPGA 기술은 클록에 대한 디바이스의 전체 라우팅을 제공하므로 수동으로 클록 트리의 딜레이 균형을 맞출 필요가 없다. FPGA의 low-skew와 high-fan-out, 전용 라우팅을 활용하자.

예를 들어 알테라의 FPGA는 글로벌 클록 라우팅 자원과 전용 입력을 제공하고 있다. 클록 입력을 이러한 전용 클록 핀의 하나에 배정함으로써, 혹은 로직을 글로벌 시그널에 배정함으로써 여러분은 클록 시그널용으로 사용가능한 전용 라우팅을 활용할 수 있다.

최상의 성능을 위하여 설계상의 글로벌 클록 수를 FPGA에 있는 전용 글로벌 클록 자원의 개수로 제한하자.

오늘날의 FPGA에서는 많은 클록 도메인을 가진 대형 설계에 부응하도록 글로벌 클록의 수가 증가되었다. 예를 들어, 알테라의 Stratix 디바이스는 16개의 전용 글로벌 클록 네트워크와 16개의 지역 클록 네트워크(디바이스 4분의 1 당 4개씩) 그리고 5개의 전용 fast regional 클록 네트워크를 제공한다. 글로벌 혹은 지역 클록 네트워크 중의 하나를 드라이브하는 16개의 전용 클록 핀이 있다. Stratix의 고속 PLL 출력은 또한 글로벌 및 지역 클록 네트워크를 드라이브할 수 있으며, 설계 내의 내부 시그널은 Quartus II 소프트웨어 내의 글로벌 로직 배정(assignment)을 사용하여 클록 네트워크로 라우팅될 수 있다. 이 클록들은 계층적 클록 구조로 구성되어 있으며, 낮은 skew와 딜레이와 함께 디바이스 지역 당 최대 22개의 클록을 허용한다. 따라서 Stratix 디바이스내에서 최대 48개의 unique 클록 도메인을 제공한다.

이러한 라우팅 자원을 최대한 활용하도록 설계의 클록 시그널 자원(입력 클록 핀이나 혹은 내부적으로 만들어진 클록)은 레지스터의 입력 클록 포트만을 드라이브해야 한다. 만약 클록 시그널이 레지스터의 데이터 포트를 피딩한다면 시그널은 전용 라우팅을 사용할 수 없을 수도 있으며 성능이 저하될 수도 있다.

리셋 자원

대부분의 FPGA에서 지원되는 디바이스 전체의 비동기식 리셋 핀을 활용하자. 이 시그널은 디바이스의 낮은 skew 라우팅을 제공한다. ASIC 설계는 시그널에서 딜레이가 오래 발생하는 것을 피하기 위하여 로컬 리셋을 사용할 수도 있다; 그러나 글로벌 리셋 시그널은 이러한 문제를 해결해 준다.

레지스터 컨트롤 시그널

설계의 대상 디바이스 아키텍처가 비동기식 로딩을 위한 전용 회로를 가진 레지스터를 포함하고 있지 않다면 비동기식 로드 시그널을 사용하지 말자. 또한 아키텍처가 그러한 컨트롤 시그널 하나만을 지원한다면 비동기식 clear 및 preset 모두 사용하는 것을 피하자. 대상 디바이스가 시그널을 직접 지원하지 않을 경우, place-and-route 소프트웨어는 같은 기능을 구현하도록 combinational logic을 사용해야 한다. 그러한 구현은 Combinational logic의 효율성을 떨어뜨리며 glitch와 같은 다른 문제 등을 유발할 수 있으므로 가능하다면 피하는 것이 좋다.

벤더-특화된 IP 기능

FPGA 벤더는 일반적으로 여러분의 설계과정을 보다 손쉽게 하기 위하여, 많은 기능에 대하여 IP(Intellectual Property) 블록을 제공하고 있다. 예를 들어, 알테라는 알테라 디바이스 아키텍처에 최적화되어 있고 매개변수화된 메가펑션을 공급한다. 메가펑션에는 매개변수화된 모듈의 라이브러리(LPM)와 PLL, DSP 블록, LVDS 드라이버, 알테라 자체의 MegaCore® IP, 그리고 AMPP(Altera Megafunction Partners Program) 회원사들의 IP와 같은 디바이스 특화된 임베디드 메가펑션들이 있다.

여러분의 로직을 코딩하는 대신, IP 기능을 사용하면 설계기간을 절약할 수 있다. 더욱이 이러한 기능은 가장 효율적인 로직 합성과 디바이스 구현을 제공한다. 또한 간단히 매개변수를 설정함으로써 손쉽게 메가펑션을 서로 다른 크기로 스케일할 수도 있다. 여러분은 또한 메가펑션을 사용하여 메모리, DSP 블록, LVDS 드라이버, PLL 그리고 DDR I/O 회로와 같은 디바이스-특화된 특성을 액세스할 수 있다. 그것들을 여러분의 HDL 코드에 instantiation하거나 어떤 경우에는 일반 HDL 코드로 infer 함으로써 벤더-특화된 기능을 사용할 수 있다.

벤더-특화된 IP를 instantiation하기

여러분은 포트와 매개변수 정의에 따라 다른 모듈이나 요소처럼 IP 기능을 여러분의 HDL 설계에 instantiation 할 수 있다. 또한 알테라의 MegaWizard® 플러그-인 관리자와 같은 기능을 매개변수화하고 wrapper 파일을 만들도록 도와주는 소프트웨어 마법사도 있다. 이 마법사는 메가펑션을 커스터마이징하고 매개변수화하는데 있어 그래픽 인터페이스를 제공하며,

여러분이 모든 기능 매개변수를 올바로 설정할 수 있도록 해준다. 여러분이 설정 매개변수를 끝마쳤을 때, 대개 마법사는 VHDL이나 Verilog HDL wrapper 파일을 만들어 올바른 매개변수를 가진 메가펑션을 instantiation 해준다.

벤더-특화된 IP 기능의 instantiation에 대한 단점으로는 기능이 쉽게 다른 벤더의 기능으로 대체될 수 없을 때, 그것이 여러분의 코드를 벤더-특화된 상태로 만든다는 점이다. 벤더 IP 기능을 평가할 때 이러한 요인을 따져 볼 필요가 있다. 그들이 일반적으로 같은 벤더 내의 다른 디바이스 제품군으로 마이그레이션될 수 있기 때문에 이러한 기능을 사용하는 경우 대체적으로 같은 벤더의 디바이스 마이그레이션과 설계 재사용이 지원되고 있다.

HDL로부터 벤더-특화된 IP를 infer하기

여러 합성 툴이 자동으로 HDL 코드 유형을 인식하고 적합한 벤더-특화된 기능을 infer 할 수 있다. 즉, 비록 여러분이 기능을 특별히 instantiation 하지는 않았지만, 여러분의 place-and-route 소프트웨어가 설계를 컴파일 할 때 벤더의 IP 코드를 사용할 것이다. 일반 HDL로부터 벤더-특화된 기능을 infer하는 것은 설계를 더욱 벤더-독립적으로 만들어 준다. 위에서 언급한대로 이러한 기능(카운터나 multiplier와 같은)들이 FPGA 아키텍처에 대하여 최적화되어 있기 때문에 소프트웨어는 inference를 사용하고, 그러므로 차지하는 density나 혹은 성능이 일반 HDL 코드에 비하여 더 나을 수도 있다. 더욱이 여러분은 특정 기능을 사용하여 일반적으로 기본 로직이나 레지스터에 비하여 성능을 높여주는 아키텍처-특화된 특성(RAM, 시프트 레지스터 및 알테라의 DSP 블록의 multiply-accumulators 혹은 multiply-adder)을 액세스할 수 있다. Inference는 올바르고 효율적으로 표현된 HDL이 사용되고, 그리고 이것과 부합하는 특정 기능에 대하여서만 지원된다.

여러분의 합성 툴이나 혹은 FPGA 벤더가 권장하는 대로 벤더-특화된 기능을 infer하는 것은 특정 HDL 코딩 스타일을 필요로 할 수도 있으며 디바이스 아키텍처로 인한 어떤 제한이 있을 수도 있다. 어떤 경우에는 특성의 기능이 여러분의 본래 HDL 소스 코드와 다소 달라질 수도 있다. 여러분이 벤더가 제공하는 특성 대신에 일반적인 로직이나 레지스터를 사용하고자 한다면 합성 및 Place-and-route 툴은 여러분이 inference를 disable하도록 할 수 있는 옵션을 대부분 제공하고 있다.

한 예로써, 메모리 블록은 종종 특정 FPGA 벤더에 대하여 특화되어 있으며 ASIC 메모리 컴파일러는 일반적으로 FPGA를 지원하지 않는다. EDA 합성 툴은 일반 HDL 코드로부터 알테라의 메모리 블록을 infer 할 수 있으며, 블록을 벤더-독립적으로 만들어준다. 알테라 및 다른 합성 툴에서는 서로 다른 디바이스 아키텍처에서 싱글-포트 및 심플 듀얼-포트 메모리를 infer하는 데 있어서 설계지침을 제공하고 있다. read와 write에 관련된 언어 제약 때문에 Verilog HDL이나 VHDL로 동시에 FPGA의 트루 듀얼-포트 RAM을 기술할 수 없다. 더욱이 RAM이 같은 위치에서 read하고 write 할 경우, 분리된 read 및 write 클록을 가진 심플 듀얼-포트 RAM에 대한 알테라의 RAM 메가펑션을 사용하는 것은 설계 특성을 다소 변경시키게 될 수도 있다. 또한 이러한 차이는 HDL 언어에 대한 정의 때문이다. 이러한 경우, 소프트웨어는 경고를 보내고 특성이 바뀐 조건에 대하여 설명한다. 여러분의 대상 아키텍처 역시 특정 요건을 가지고 있을 수도 있다. 예를 들어, 알테라의 Stratix™ 디바이스는 동기화된 메모리 블록을 포함하고 있으며, 그러므로 어떤 시그널은 디바이스 메모리로 이동할 때 레지스터링 되어야만 한다.

FPGA의 첨단 아키텍처의 특성을 활용하고 로직 블록에 대한 최적화된 코드를 사용함으로써 벤더-특화된 기능을 infer하면 여러분의 FPGA 설계 성능을 높여줄 수 있다. 기능을 직접적으로 instantiation하는 것보다 일반 코드를 writing하는 것이 여러분의 설계를 보다 벤더-독립적이 되게 해준다. 특화된 코딩 유형 지침에 대한 합성 툴 자료를 참조하고 HDL 제약이나 FPGA 아키텍처로 인한 모든 코딩 요건을 인지하고 있어야 한다.

결론

이 글에서는 설계결과에 따라 서로 다른 설계방식의 유형을 논의하였다. 기본적인 FPGA 설계지침과 아키텍처-특화된 권장사항이 설명되었다. 여러분의 FPGA 설계 혹은 최적의 ASIC 프로토타이핑 결과를 얻으려면 각자의 설계 유형의 영향을 이해하고 권장된 설계기술을 이용, FPGA의 첨단 아키텍처 특성을 활용하도록 하자. RTime