시간 상이점을 이용한 자체 검진 비교기의 설계에 관한 연구
A Study on The Design of The Self-Checking Comparator Using Time Diversity

신석균* , 양성현** , 이기서***

Abstract

This paper presents the design of self-checking comparator using the time diversity and the application to 8 bit CPU for the implementation of fault tolerant computer system. this self-checking comparator was designed with the different time points in which temporary faults were raised by electrical noise between duplicated functional blocks. also this self-checking comparator was simulated in the method of the fault injection using 4 bit shift register counter. we designed the duplicated functional block and the self-checking comparator in the single chip using the Altera EPLD and could verify the reliability and the fault detection coverage through the modeling of temporary faults ,especially intermittent faults. at the results of this research, the reliability and the fault detection coverage were implemented through the self-checking comparator using the time diversity.

1. 서론

현재 시스템의 신뢰성을 향상시키기 위해 사용되는 시스템 레벨의 접근은 결함 허용(Fault Tolerance)이라는 개념을 바탕으로 개발되어왔고, 이를 위해 최소 고장율을 가진 내구성이 좋은 소자를 사용한 접근에 추가되어 왔다. 시스템 고장의 90%를 차지하는 순시결함[8](Temporary Fault)의 검출[7](fault detection)은 결함의 잠재성을 최소화하고 고속의 결함검출과 복구는 시스템의 가용도를 향상시키기 위해 필요하다. 결함검출을 위해 사용되어지는 자체검진시스템[9](self-checking system)을 실현하기 위해 two-rail logic[5]과 M-out-of-N 코드[1]그리고 parity[5]코드와 같은 여분 코드(redundant code)를 이용한다. 그러나 여분코드의 단점은 모든 회로가 새로 설계되어야하고 엄격한 설계법에 의한 ad hoc[9]과정 속에서 구현된다. 게다가 설계된 여분회로사이에서의 지연시간의 최적화는 매우 어렵다.

본 논문에서 연구된 시간 상이점을 이용한 자체검진비교기는 이중화된 시스템간의 최적의 시간차이를 이용하여 결함의 발생시점을 비교함으로써 이미 설계되어진 시스템에 적용될 수 있고 하드웨어여분이나 정보여분에서 검출되기 힘든 간헐결함(Intermittent Fault)을 정확히 검출한다는 장점을 지닌다.

현대의 고도화된 VLSI화 기술로 자체검진비교기를 적용한 이중화된 시스템이 단일칩 내에 설계가 가능하고 그에 따른 비용 절감효과를 거둘 수가 있다.

* 광운대학교 제어계측공학과 석사과정

** 광운대학교 전자공학부 교수

*** 광운대학교 제어계측공학과 교수

2. 결함의 분석

2.1 결함의 특성분류

오랜 시간 결함의 원인과 결함, 오류, 고장의 차이점이 논의되어져 왔으나 결함의 개념을 정확히 분류하기 위해 결함의 원인보다는 특성분석이 필요하다. 결함의 원인 외에 추가적으로 뒷받침되는 결함의 특성은 크게 그림 1처럼 성질(nature), 지속성(duration), 범위(extent), 유용성(value)등 4가지로 나누어진다.

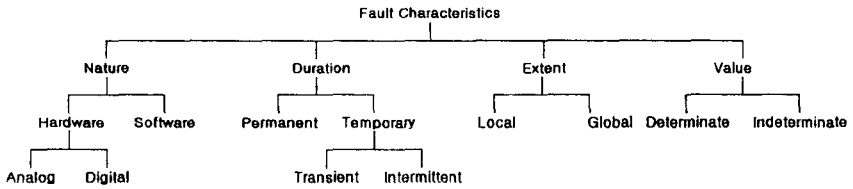

그림 1. 결함의 특성분류

1)결함 성질(fault nature)

결함 성질은 크게 소프트웨어 결함인지 하드웨어 결함인지를 구분하여 결함의 형태를 분류하는 것이다. 또한 하드웨어 중에서도 아날로그와 디지털을 구분하여 규정한다. 결함 성질은 엄격히 결함 형태를 유도 할 수 있게 하며 그 형태를 통해 고착 결함(stuck-at fault), 합선 결함(Bridging -fault), 절선 결함 (Stuck-open) 등의 모델링이 정의되어진다.

2)결함 지속성(fault duration)

결함 지속성은 결함의 활성시간에 따라 분류하며 영구 결함(Permanent Fault)과 순시 결함으로 크게 나눠진다. 영구 결함은 한번의 결함발생이 결국 오류로까지 이어지는 결함의 종류이다. 순시 결함은 매우 짧은 시간에 나타났다 사라지는 순간적인 결함이며 시스템에 영구적인 손실을 주지는 않는다.

3)결함 범위(fault extent)

결함 범위는 결함이 시스템에 미치는 영향의 범위에 대한 분류방법으로 부분적으로 하드웨어나 소프트웨어에 영향을 미치는지 혹은 전체시스템의 하드웨어나 소프트웨어에 영향을 미치는지를 구분하여 나타낸 것이다. 예를 들어 전원공급장치와 같은 경우에 발생하는 결함은 시스템 전체에 영향을 주며 메모리의 경우는 시스템의 부분적으로 오동작을 행하게 하는 결함이 되는 것이다.

4)결함 유용성(fault value)

결함 유용성은 결함의 결과를 구분 짓는 분류로써 확정(determinate)결함과 비확정(indetermina te)결함으로 구분된다. 확정 결함은 외부적으로 동작하는 전체시간동안 변화되지 않는 상태로 남아있는 결함이다. 고착 결함(Stuck at fault)이 확정 결함에 해당되며 비확정 결함은 t 시간의 상태가 t 시간을 기준으로 임의로 다른 값을 가지고 하드웨어 결함의 값이 1과 0을 반복하는 경우이다.

2.2 순시 결함의 개념과 모델링

결함의 지속성에 관련된 순시 결함은 디지털 시스템 오동작의 대부분을 차지한다. 순시 결함은 순간적인 성분을 가지므로 결함이 검출되기 힘들고 또한 고립적인 성격을 가지고 있기 때문에 전

체 시스템 고장의 90% 이상으로 추정한다. 이런 순시 결함은 일반적인 의미상으로는 "간헐" 혹은 "과도 " 결함으로 동일한 뜻으로 언급되나 엄밀히 분류를 한다면 그 차이점이 명확히 구분지을 수 있다.

2.2.1 과도 결함과 간헐 결함의 분류

순시 결함의 분류를 위해 결함의 활성여부에 따라 "결함 활성 상태"와 "결함 비활성 상태"로 나누게 되는데 "결함 활성 상태"의 경우는 결함이 회로의 안전상태에서 활성화된 것을 의미하며 고장의 원인이 된다. "결함 비활성 상태"는 회로 내 결함이 존재하나 활성이 되지 않은 상태를 의미한다. 이 두 가지 상태를 통해서 과도 결함과 간헐 결함의 정확한 구분이 가능하다.

1) 과도 결함

과도 결함은 비반복적으로 발생하는 순시 결함으로 유한 시간 내에 결함이 존재하고 결함이 활성화된 후 양성상태로 천이하여 영구적으로 유지되는 경우이다. 주로 α 방사선 혹은 전원공급 불안 등에 의해 발생하고 하드웨어 상에 물리적인 피해가 없기 때문에 복구 또한 쉽지 않다. 그 예로 시스템내의 반도체 메모리 칩 상에 발생하는 결함이 이것에 해당하며 고장의 원인이 되기도 한다.

2)간헐 결함

간헐 결함은 과도 결함과는 달리 결함의 활성상태와 양성상태를 일정한 규칙 하에 재현하여 반복하는 결함이다. 간헐 결함은 주로 회로 내에 연결부위의 느슨함 그리고 부분적으로 불완전한 소자의 사용이나 불완전한 설계 등으로 인해 발생한다. 또한 일부 간헐 결함은 환경요인 즉, 온도, 습도, 진동 등에 의해 영향을 받는데 그런 간헐 결함과 같은 발생 가능성은 시스템의 접지, 필터링, 냉각 등등 환경으로부터 얼마나 보호가 되어 있는가에 의존하게 된다. 간헐 결함은 기본적인 규칙을 가지고 발생하지만 비교적 임의적이므로 확률적인 방법으로만 모델링이 가능하다.

2.2.2 결함의 모델링

본 연구에서 논의될 결함에 대한 결함 검출 능력과 최종 시스템의 신뢰도평가를 위해 간헐 결함의 수학적 모델링을 해보면 그림2와 같이 마코브(Markov) 모델로 나타낼 수 있다.

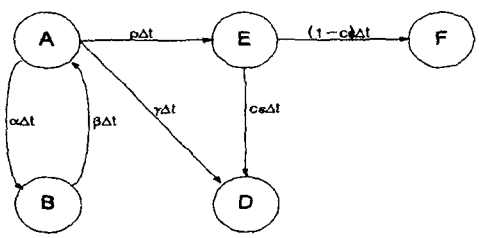

그림 2. 결함의 모델링

A 상태 : 결함이 활성된 상태.
B 상태 : 결함이 양성인 상태.
E 상태 : 결함이 발생한 오류상태.
D 상태 : 결함을 검출한 상태.
F 상태 : 결함이 검출되지 않은 상태에서 발생한 오류의 확장으로부터 생긴 고장상태.
α : 결함 활성로부터 양성상태의 천이확률.
β : 양성상태로부터 결함 활성 상태로의 천이가 일어나는 비율.
ρ : 오류 generation의 발생비.
γ : 결함 detection의 발생비.
ε : 오류 propagation의 발생비.
C : 확률 결함이 어떤 damage로 이어지기 전에 오류를 detecting할 확률

그림 2의 상태도를 기본으로 하여 앞 절에서 설명한 영구 결함과 과도 결함 그리고 이번 절에서 논의될 간헐 결함을 각각 마코브 모델링을 하면 아래와 같다.

1)영구 결함의 모델링

영구 결함은 결함의 활성상태에서 양성상태로의 천이가 없으므로 결함 활성상태 후 바로 오류상태로 이어지며 오류상태에서 일반적인 오류검출기를 통한 검출이 가능하다.

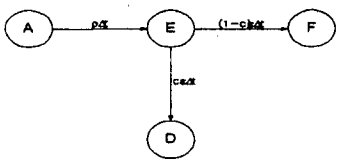

그림 3. 영구결함의 모델링

그림3의 마코브 모델을 시간에 대한 함수로 표현을 하면

$$\begin{bmatrix} P_A(t+\Delta t) \\ P_E(t+\Delta t) \\ P_D(t+\Delta t) \\ P_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1-\rho\Delta t & 0 & 0 & 0 \\ \rho\Delta t & 1-\varepsilon\Delta t & 0 & 0 \\ 0 & C\varepsilon\Delta t & 1 & 0 \\ 0 & (1-C)\varepsilon\Delta t & 0 & 1 \end{bmatrix} \begin{bmatrix} P_A(t) \\ P_E(t) \\ P_D(t) \\ P_F(t) \end{bmatrix}$$

이 된다. 따라서 영구결함의 발생확률은

$P(t) = P_A(t) + P_E(t) + P_D(t) + P_F(t)$가 된다.

2)과도 결함의 모델링

과도 결함은 결함의 활성상태에서 양성상태로 가는 경우 영구적으로 양성상태를 유지하며 만일 결함 활성상태가 유지되면 영구 결함경우처럼 오류와 고장으로 이어진다. 따라서 결함 활성상태로 천이 될 비율은 없다. 과도 결함은 결함 활성상태에서 양성상태로 가는 시간에 따라서 의존하는 데 긴 경우는 짧은 경우의 평균 10배정도의 차이를 나타낸다.

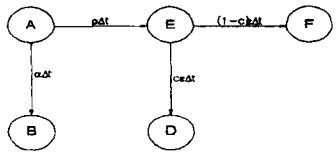

그림 4. 과도결함의 모델링

그림 4의 마코브 모델을 시간에 대한 함수로 표현을 하면

$$\begin{bmatrix} P_B(t+\Delta t) \\ P_A(t+\Delta t) \\ P_E(t+\Delta t) \\ P_D(t+\Delta t) \\ P_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1 & \alpha\Delta t & 0 & 0 & 0 \\ 0 & 1-(\alpha+\rho)\Delta t & 0 & 0 & 0 \\ \rho\Delta t & 0 & 1-\varepsilon\Delta t & 0 & 0 \\ 0 & 0 & C\varepsilon\Delta t & 1 & 0 \\ 0 & 0 & (1-C)\varepsilon\Delta t & 0 & 1 \end{bmatrix} \begin{bmatrix} P_B(t) \\ P_A(t) \\ P_E(t) \\ P_D(t) \\ P_F(t) \end{bmatrix}$$

이 된다. 따라서 과도 결함의 발생 확률은

$P(t) = P_B(t) + P_A(t) + P_E(t) + P_D(t) + P_{(F)}(t)$이 된다.

3)간헐 결함의 모델링

간헐 결함은 결함의 활성상태와 양성상태를 반복적으로 천이하는 경우이므로 그림 2의 상태천이도와 동일하다. 이때 결함이 활성상태에서 보내는 시간에 따라 주기적일 수도 있고 즉 α와 β의 값이 동일 할 수도 있고 비주기성을 지닌 경우는 평균 1000배 정도의 차이를 나타낸다.

간헐 결함의 마코브 모델을 수식으로 전개를 해보면

$$\begin{bmatrix} P_B(t+\Delta t) \\ P_A(t+\Delta t) \\ P_E(t+\Delta t) \\ P_D(t+\Delta t) \\ P_F(t+\Delta t) \end{bmatrix} = \begin{bmatrix} 1-\beta\Delta t & \alpha\Delta t & 0 & 0 & 0 \\ \beta\Delta t & 1-(\alpha+\rho+\gamma)\Delta t & 0 & 0 & 0 \\ 0 & \rho\Delta t & 1-\varepsilon\Delta t & 0 & 0 \\ 0 & \gamma\Delta t & C\varepsilon\Delta t & 1 & 0 \\ 0 & 0 & (1-C)\varepsilon\Delta t & 0 & 1 \end{bmatrix} \begin{bmatrix} P_B(t) \\ P_A(t) \\ P_E(t) \\ P_D(t) \\ P_F(t) \end{bmatrix}$$ 이 된다.

따라서 간헐 결함의 발생확률은 $P(t) = P_B(t) + P_A(t) + P_E(t) + P_D(t) + P_{(F)}(t)$ 가 된다.

2.3 순시 결함 검출

순시 결함을 검출하기 위해 이용된 기존의 방법중의 하나가 시간 여분(Time Redundancy)을 이용한 방법이다. Time 여분은 시스템 레벨에서 하드웨어 여분(hardware redundancy)이나 정보여분

(information redundancy)이 지닌 하드웨어의 추가문제를 해결한 방법이다. 최근 결함 검출이나 결함허용을 실현하기 위해 개발되는 시스템은 하드웨어여분을 줄이는 목적에 주안점을 두며 따라서 time 여분을 이용한 방법은 각광을 받고 있다.
시간 여분의 주요한 개념은 순시 결함을 검출할 수 있도록 반복적으로 계산하는 것이며 그 기본 구조는 그림 5와 같다. 그 원리는 두 번 혹은 여러 번을 일정한 시간 차이를 두어 반복계산하고 그 결과값의 불일치 유무를 비교하는 것이다. 만일 오류가 검출되면 먼저 발견된 불일치가 남아 있는지 아니면 사라졌는지를 보기 위해 다시 계산한다. 이런 접근법은 순시 결함으로 초래되는 오류를 검출 하는데 유용하지만 그림 3과 같은 일반적인 방법은 결과의 저장상태를 비교하여 오류의 검출을 하기 때문에 Off-line으로 동작하며 그 결과를 저장하는 레지스터의 신뢰성에 의존도가 높다.

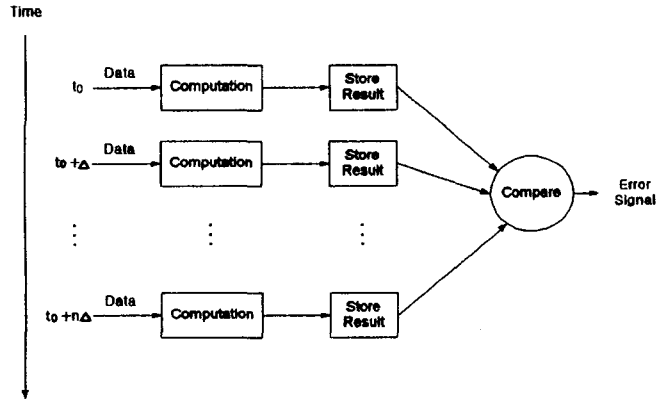

그림 5. 시간여분을 이용한 시스템

본 논문에서 연구한 자체 검사 비교기는 저장된 출력 결과에 따르는 기존의 방법을 개선하고 시간 차이를 이용할 경우 기본원리상으로는 시간여분과 유사하지만 데이터의 연속적인 비교를 통해 on-line 의 방법을 사용한다는 점에서 차이점이 나타난다. 또한 고속의 결함검출이 가능하므로 전체신뢰도와 결함 검출능력이 높은 장점이라 할 수 있다.

3. 시간 상이점을 이용한 자체 검사 비교기의 설계

시간 상이점을 이용한 자체 검사 비교기의 설계를 위해 본 연구에 필요한 3가지 개념 즉, 자체검사, 여분코드, 시간 상이점의 개념을 소개하고자 한다.

3.1 자체 검사의 개념

자체 검사는 외부 테스트 입력이 없이 회로 내의 결함 유무를 자동으로 검증 할 수 있는 능력으로 정의된다. 자체 검사가 만족되기 위해서는 결함 보안(fault-secure) 와 자체 시험(fault-testing) 등 두 가지의 정의를 만족해야한다.

F: 결함 집합 , I : 입력 , Y1: 출력의 code word space , Y2: 출력의 non-code word space

Z: 출력 space

$Z=Y1 \cup Y2$, $i \in I$, $y1 \in Y1$ (correct code) , $y1' \in Y1$ ($f \in F$)

비코드워드(non-code word) : 출력 상에 비코드(non-code)로 불리는 code space를 Y로 정의

비정확성 코드(incorrect code) : 출력 상에 결함이 존재하나 검출되지 않는 경우를 나타낸다.

1) 결함보안 : 주어진 결함 집합내에서 비정확성 코드가 발생되지 않게 하는 회로이다.
$y1 \in Y1$이면 $y1' \in Y1$, 출력이 코드 워드이면 그 값은 정확하다.

2) 자체시험 : 주어진 결함 집합 내의 모든 결함에 대해 적어도 하나의 입력 코드워드에 대한 비코드워드를 발생하는 회로이다. 각 $f \in F$에 대해 결과 출력 값이 비코드워드

y2′∈Y2 인 적어도 하나의 i2 ∈I 가 존재한다. 위의 결함 보안과 자체시험이 모두 충족되면 전체 자체 검사(Totally self checking)가 된다.

3.2 여분 코드의 개념

여분 코드는 정보여분에 해당되며 결함 검출과 결함 마스킹(masking), 그리고 결함허용을 위해 원래의 데이터에 여분의 정보를 첨가하는 방법이다. 대표적인 여분 코드로는 패리티(Parity) 코드, m-of- n 코드, 이중(Duplication) 코드, 주기(Cyclic) 코드 등이 있다.
일반적으로 여분 코드를 이용한 자체 검사 비교기는 비교기 자체내의 결함도 검출할 수 있게 설계가 되지만 만일 단순한 입력 데이터의 불일치를 알아내고자 하는 단일 경로(single path)를 사용하는 경우 값이 비교기에 입력되어도 그 입력된 데이터의 불일치를 일으킬 수 있는 functional block 내부의 결함 발생이 매우 드물기 때문에 좀처럼 결함이 활성화되지 않는다. 또한 이런 여분 코드는 모든 회로가 새로 설계되어야 하고 엄격한 설계법에 의한 ad hoc 설계 과정 속에서 구현되어야 하기 때문에 일단 설계된 여분 회로에서의 지연(delay) 시간의 최적화는 매우 어렵다.

3.3 시간 상이점의 개념

시간 상이점을 이용한 자체 검사 비교기의 장점은 크게 4가지로
1)이중화된 functional block을 사용함으로써 단일 경로가 가지는 고착결함이 검출가능하다.
2)기존의 설계된 시스템의 core를 이용할 수 있기 때문에 non ad hoc 설계가 가능하다.
3)동기방식으로 이중화된 functional block에서 존재하는 간헐 결함의 검출이 완벽하다.
4)현대의 발전된 VLSI화를 이용해 시스템의 이중화뿐만이 아니라 다중화 설계시에도 설계공간상의 제약을 줄일 수가 있고 비용상의문제를 줄일 수 있다.

시간 상이점의 개념은 이중화된 시스템사이에서 만일 왜란이 발생할 경우 상관된 결함으로 인한 고장을 방지하기 위하여 결함 고립(fault-isolation)을 시키기 위한 기술로 알려져있다. 전기회로에 대한 왜란은 비록 시스템이 아무리 이상적일지라도 왜란의 순간적인 발생시점이 다르면 이중화된 시스템사이에서 발생하는 비교시점에 차이가 존재한다는데 근본 원리를 가진다. 시간 상이점을 이용하여 결함 검출 능력을 향상시키거나 검출되지 않은 오류의 수를 줄이는 효과는 크게 macro측면과 micro측면으로 나눌 수 있다.

3.3.1 시간 상이점의 macro 측면의 효과

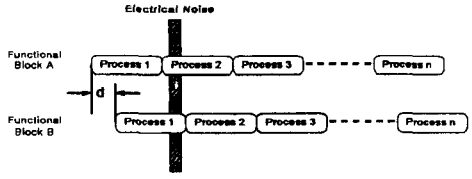

그림 6. macro 측면의 효과

그림 6에서처럼 전기적으로 왜란이 발생시 functional block A는 process2를 수행중이고 functional block B는 process 1을 수행한다. 그 후 functional block A에서의 process 2와 functional block B의 process 1은 전기적인 왜란으로부터 동시에 영향을 받으나 processing 1과 2가 동일한 Process에 영향을 받지 않는다. 상관된 오류의 확률은 이런 시간 차이로써 감소되고 macro 측면에서는 명령어 레벨에서의 시간 상이점의 효과가 발생한다.

3.3.2 시간 상이점의 micro 측면의 효과

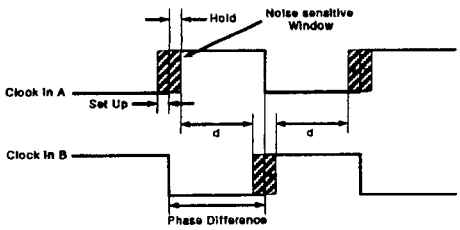

그림 7. micro 측면의 효과

시간 상이점은 macro 측면에서처럼 micro 측면에서도 역시 상관된 오류 감소 효과를 가진다.
일반 논리회로는 그림 7처럼 클럭 edge앞의 setup시간으로부터 클럭 edge 뒤의 hold 시간까지의 시간구간에서 매우 민감하다. 그러므로 이중화된 functional block사이에 일정한 클럭 시간 차이로 동작을 하면 상관된 오류의 발생을 막을 수 있다. 상관된 오류의 확률은 시점차이가 그림 7처럼 n+1/2 클럭 cycle 일 때 즉, 두 클럭의 위상차이가 180도 일때 왜란에 대해 민감한 구간 사이의 거리가 최고로 크기 때문에 상관된 오류의 확률은 최소가 된다.

3.4 시간 상이점을 이용한 자체 검사 비교기의 설계

3.4.1. 자체 검사 비교기의 설계

논리회로에 적용되는 비교기는 EX-OR를 이용한 설계며 그림 8은 비교기의 기본 구성도를 나타낸다.

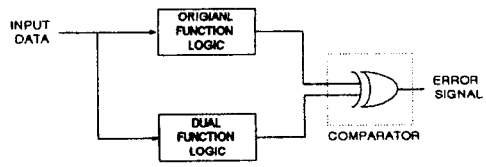

그림 8. 비교기의 기본 구조

그림 8처럼 이중화된 두 개의 값이 각각 비교되어 값이 다른 경우를 검출하는 논리로 구성된다.
만일 동일한 값이 비교기에 입력되지 않으면 오류로 처리를 한다.

3.4.2 결함 주입(Fault injection)

본 논문에서 설계한 시간 상이점을 이용한 자체 검사 비교기를 시험하기 위해서 여분 코드와 같은 기존의 static 코드 회로에 의존한 자체 검사가 아닌 결함의 잠재성 문제를 해결하고자 on-line 결함 주입 방법을 제시했다.

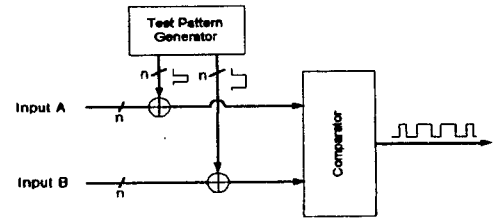

그림 9. 결함 주입

결함 주입은 시험대상의 회로에 그림 9에서 나타난 것처럼 이중화된 functional block의 각각에 Test Pattern Generator에 의한 강제 결함을 비교기의 입력데이터에 주입한다. 특정한 패턴을 가진 비교기의 출력신호는 입력 데이터의 입력 값이 일치하고 Test Pattern Generator와 비교기가 이상이 없다는 것을 나타낸다. 그러므로 입력 데이터가 불일치한다면 혹은 Test Pattern Generator와 비교기가 고장난다면 불규칙한 패턴을 가진 신호가 비교기로부터 나온다.

3.4.3 TPG의 이중화 functional block에의 적용

그림 9는 TPG를 이용해 결함 주입을 함으로써 이중화된 시스템 동작 검증 방법을 블록다이어그램으로 나타내었다.

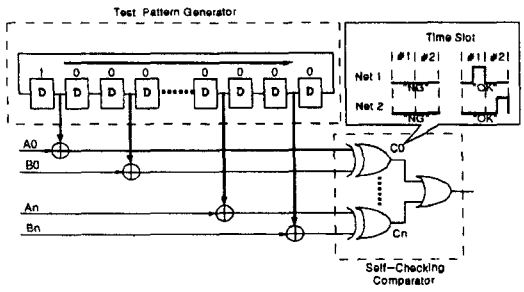

그림 10. 결함주입을 통한 시스템의 검증방법

그림 10에서 An과 Bn은 동일한 값이 입력되는 이중화된 Bus로서 두 Bus가 동기화되어 동작한다. 두 입력이 비동기로 동작하는 것처럼 가상적으로 구현하기 위하여 TPG에 의해 An과 Bn의 입력이 1 클럭 차이로 강제적인 결함이 주입됨으로써 실제 설계될 이중화된 CPU가 일정한 클럭 차로 비동기 동작한다는 가정을 뒷받침한다. 결함 주입을 통해 결과로 출력되는 C0-Cn은 왜란이 검출되는 과정을 나타내고 최종출력은 결함주입을 통해서 검출되는 오류 신호들의 급수형태로 나타나게 된다. 그러나 실제 적용될 CPU설계시에는 TPG부분은 제거되며 최종 자체 검사 비교기로 들어가는 데이터 값은 동기화되어 비교된다.

3.5 8 Bit CISC CPU에 적용한 시간 상이점을 이용한 자체 검사 비교기의 설계

그림 11은 본 연구에서 설계될 최종적인 시스템의 구성도를 나타내었다.

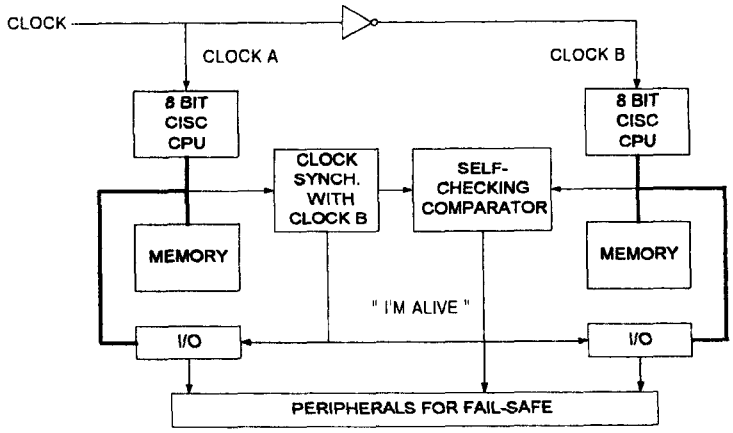

그림 11. 최종시스템의 구성도

전체시스템의 클럭은 공통으로 사용되나 CPU B로 들어가는 클럭 B 는 클럭 A를 위상을 180 ° 반전시킨 것으로 클럭 A 보다 1/2 클럭 cycle 만큼 강제 지연되어 시작된다. 클럭 A와 클럭 B가 1/2 클럭 cycle 만큼 차이를 둔 이유는 두 cycle의 위상 차가 180 °일 때 왜란에 민감한 구간사이의 거리가 가장 크므로 상관된 오류의 발생 확률이 최소가 되기 때문이다. 각 CPU 와 memory, 그리고 I/O Device에 사용되는 데이터 bus는 클럭 B에 동기화되어 자체-검사 비교기에 입력된다. 1/2 클럭 만큼 선행한 CPU A 가 다시 1/2 클럭 만큼 지연되어 동작함으로써 최종적으로 CPU B와 동기화되어 최종적으로 비교됨을 나타낸다. 만일 데이터 bus상에 noise 가 입력되어 CPU A, B 에 모두 영향을 주더라도 시간 상이점의 효과로 각각 다른 위치에서 영향을 받기 때문에 최종적인 동기화 시에는 자체 검사 비교기에서 간헐 결함이 검출될 수 있는 것이다. 그림 11의 설계는 현대의 고도화된 VLSI 화를 이용하여 이중화된 CPU를 하나의 칩속에 설계가 가능하다. 이를 위해 8bit cisc cpu는 각각 AHDL(Altera Hardware Description Language)로 설계되고 클럭 A와 클럭 B를 동기화하는 부분과 자체-검사 비교기 역시 AHDL 로 설계를 하였다.

4.실험 및 결과

본 연구는 Altera 사의 MaxplusII 8.0 이라는 설계툴을 사용하여 AHDL로 설계 및 시뮬레이션 하였다. AHDL은 ALtera사의 HDL로서 Schematic이 아닌 순수 Language로 설계가 가능하며 본 연구를 위해 Schematic으로 구현시 발생하는 1/2 클럭 지연과 최종 두 클럭의 동기화 문제를 해결하는 방법으로 상태를 language를 통해 하드웨어로 구현 가능한 장점을 갖고 있다. 또한 전체의 시스템을 하나의 FPGA 형태로 설계됨으로써 하나의 모듈로 구성 가능함에 따라 또 다른 하나의 CPU 설계가 가능하였다.

4.1 TPG를 통한 결함 주입 결과

시간 상이점을 이용한 자체 검사 비교기를 test 하기 위해서 그림 10을 기본으로 하여 TPG 를 통한 시뮬레이션을 하였다.

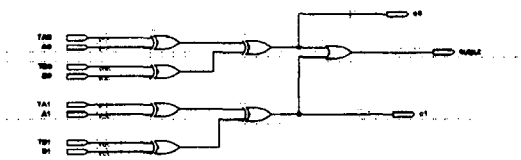

그림 12. 자체 검진 비교기의 가상 설계

그림 13처럼 A0와 A1은 CPU A, B0와 B1는 CPU B의 bus line 중 두 line 이고 두 CPU A,B는 동기화 되어 동작을 한다.
최종 설계될 CPU A와 B는 1/2 클럭 cycle 만큼 차이를 나타내며 동작을 하지만 가상적으로 시간 상이점을 시험하기 위해서 CPU A와 B는 동기화되어 동작을 한다.

결함주입을 위해 TPG는 그림 14처럼 shift register counter로 설계하였고 TPG의 출력은 Table 1과 같다.

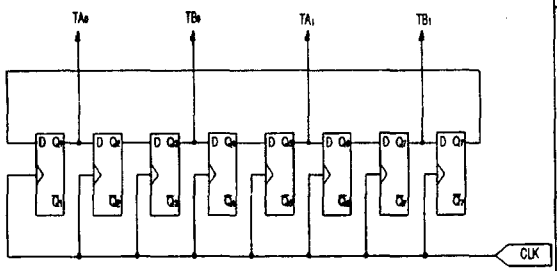

그림 13. 시험 패턴 생성기의 설계

표 1. 시험 패턴 생성기의 출력

TPG입력	TA0		TB0		TA1		TB1	
출력 / 시간	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
t0	1	0	0	0	0	0	0	0
t0+Δt	0	1	0	0	0	0	0	0
t0+2Δt	0	0	1	0	0	0	0	0
t0+3Δt	0	0	0	1	0	0	0	0
t0+4Δt	0	0	0	0	1	0	0	0
t0+5Δt	0	0	0	0	0	1	0	0
t0+6Δt	0	0	0	0	0	0	1	0
t0+7Δt	0	0	0	0	0	0	0	1
t0+8Δt	1	0	0	0	0	0	0	0
t0+9Δt	0	1	0	0	0	0	0	0
t0+10Δt	0	0	1	0	0	0	0	0
·	·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·	·
t0+nΔt	·	·	·	·	·	·	·	·

첫 번째 D Flip-Flop 의 초기값은 1이고 나머지는 0으로 정해진다.

그림 14는 동기화된 두 CPU A, B 에 결함 주입을 함으로써 나타난 시뮬레이션의 결과이다. TA0, TA1, TB0, TB1 은 각각 TPG에서 생성되는 신호이며 TA0와 TB0의 시간차이는 1 클럭 cycle 이다. C0와 C1은 각각 TPG의 결과이며 output은 오류의 최종출력이며 결함 주입을 통한 오류가 완벽히 검출된다.

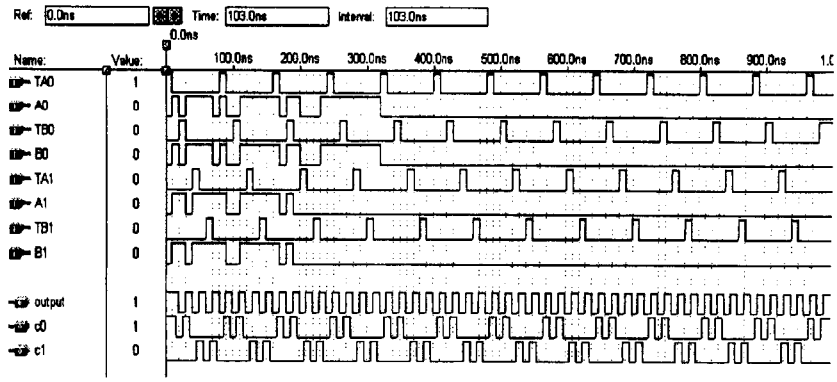

그림 14. TPG를 통한 자체 검진 비교기의 시뮬레이션 결과

5. 결론

본 논문에서는 시스템 고장의 90%를 유발하는 순시결함들 중에서 간헐결함을 모델링하고 시간 상이점을 이용한 자체 검진 비교기를 설계하였으며 on-line 결함 주입방법을 통해 간헐 결함과 과도 결함에 대한 검출을 실험하였다. 그 결과 간헐결함에 대한 결함검출능력이 향상되었고 시간 상이점을 이용한 자체 검진 비교기를 시스템에 적용시킴으로써 시스템의 신뢰도를 향상시켰다. 결함허용시스템에서 흔히 고려되는 하드웨어 여분상의 문제점을 줄이기 위해 ALTERA 사의 EPLD를 이용하여 단일 칩에 동일한 두 개의 CPU를 독립적으로 탑재시켰다. 이것은 현대 추세인 비용상의 절감 효과를 고려한 것이다. 앞으로 시간 상이점을 이용한 자체검진 비교기를 이용해 결함이 발생한 시점에서 다시 복구가 되어 연차적으로 시행될 수 있는 시스템의 개발과 여러개의 FUNCTIONAL BLOCK을 설계하여 시스템의 복구율을 높이는 문제가 수행되어야 할 것이며 단일 칩내에 두 CPU가 설계됨으로써 발생한 상관된 결함을 줄이기 위해 제조 공정상에서 철저한 공간상의 분리가 향후에 수행되어야할 연구이다.

참고문헌

[1]"Design of Totally Self-Checking Circuits for m-out-of-n Codes" IEEE Trans. on Computers Vol.c-22 No.3 pp.263-269: March 1973. D.A Anderson & G. Metze

[2]"A Totally Self-Checking Checker for Borden's Code" IEEE Trans. on Computer-Aided Deisgn Vol.8 No.7 pp 731-736 July 1989 N.K. Jha

[3]"efficient Design of Self-Checking checkers for Any M-out-of-n Code" IEEE Trans. on Computers Vol.c-27 No.6 pp482-490 June 1978 M. Marouf and A.D Friedman

[4]"A Totally Self-Checking Checker for a Parallel Unordered Coding Scheme" IEEE Trans on Computer Vol.43 pp490-495 1994

[5]"Fault Detction in CVS Parity Trees with Application to Strongly Self-Checking Parity and Two-Rail Checkers" IEEE Trans. on Computers Vol.42 ppl79-189 1993

[6]"Optional Design of Checks for Error Detection and Location in Fault-Tolerant Multiprocessor Systems" IEEE Trans. on Computers Vol.42 pp 780-793 1993

[7]Fault Tolerant AND Fault Testable Hardware Design - Parag K. Lala Ch. 5

[8]Design and Analysis of Fault-Tolerant Digital System - Barry W. Johnson Ch. 3-5

[9]Digital Circuit Testing and Testability - Parag K. Lala Ch. 5

[10]Practical Digital Logic Design and Testing - Parag K. Lala ch. 2