T. C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

SHA256 KRİPTOGRAFİK ÖZETLEME(HASHING) ALGORİTMASI VE ANINDA(ON-THE-FLY) SHA256 GERÇEKLENMESİ

İnan ERDEM

YÜKSEK LİSANS TEZİ

ELEKTRONİK MÜHENDİSLİĞİ

GEBZE

2009

T. C.

GEBZE YÜKSEK TEKNOLOJİ ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

SHA256 KRİPTOGRAFİK ÖZETLEME(HASHING) ALGORİTMASI VE ANINDA(ON-THE-FLY) SHA256 GERÇEKLENMESİ

İnan ERDEM

Danışmanı

Yrd. Doç. Dr. Serdar S. ERDEM

YÜKSEK LİSANS TEZİ

ELEKTRONİK MÜHENDİSLİĞİ

GEBZE

2009

GEBZE YÜKSEK TEKNOLOJİ

ENSTİTÜSÜ

MÜHENDİSLİK VE FEN BİLİMLERİ ENSTİTÜSÜ

JÜRİ ONAY FORMU

JÜRİ

ÜYE (BAŞKAN) :

ÜYE :

ÜYE :

Gebze Yüksek Teknoloji Enstitüsü Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …/……/…… tarih ve ……/…… sayılı kararı ile yukarıdaki öğretim elemanlarından oluşmuş jüri tarafından düzenlenen ……/……/…… tarihli Tez Savunma Tutanağı neticesinde Yüksek Lisans öğrencisi …….'ın çalışması GYTE Mühendislik ve Fen Bilimleri Yönetim Kurulu ……/……/…… tarih ve …./…./…… sayılı kararıyla ……Anabilim Dalında Yüksek Lisans tezi olarak onaylanmıştır.

İMZA/MÜHÜR

ÖZET

TEZİN BAŞLIĞI: Sha256 kriptgrafik özetleme algoritması ve Verilog donanım tanımlama dili ile anında(on the fly) özetleme yapabilecek şekilde gerçeklenmesi.

YAZAR ADI : İnan ERDEM

Kriptografik özetleme algoritmaları temel olarak değişken uzunluktaki bir veriden sabit uzunluklu bir özet(hash) elde etmeyi sağlayan algoritma ailesidir. Bu tezde kriptografik özetleme algoritmalarından SHA(secure hashing algorithm) algoritma ailesinin 256 bitlik özet değeri üreten ve Sha256 olarak bilinen üyesi incelenmiştir.Bu algoritmanın anında(on the fly) bir donanım tasarımı Verilog HDL(Hardware Description Language) donanım tanımlama dili kullanılarak gerçeklenmiştir.

Özetleme algoritmaları çoğu zaman metinler üzerine uygulandığından bir çok özetleme algoritmasının gerçeklenmesi bir sekizlik(byte) temel alınarak yapılır yani en küçük girdi verisi bir sekizliktir; ancak daha genel uygulama alanlarını hedef alan gerçeklemelerde en küçük girdi verisi bir bitlik veridir. Bu çalışmada da bit tabanlı özetleme yapabilecek yetenekte bir tasarım gerçeklenmiştir.

SUMMARY

THESIS TITLE: Sha256 cryptographic hashing algorithm and an on the fly implementation with Verilog HDL language.

AUTHOR : Inan ERDEM

Cryptographic hashing algorithms are basically used to generate fixed length hash values out of a variable length data. In this thesis, Sha256 from SHA (secure hashing algorithm) family that is producing 256 bits of hash value is introduced. An on the fly hardware design of that algorithm is tried to be implemented using Verilog HDL (hardware description language).

Since most of the hashing algorithms are generally being applied upon texts, implementations of those algorithms are frequently accepts bytes as the minimum unit of its input data, but for those implementations targeting the general use the minimum unit is a single bit. In this work, a design having the ability to process its input data on a bit by bit bases is implemented.

TEŞEKKÜR

Bu tez çalışması sırasında beni yönlendiren ve yol gösteren sayın hocam Yrd. Doç. Dr. Serdar Süer ERDEM'e, sayısal tümdevre tasarımı bilgi ve deneyimime olan katkılarından dolayı STMicroelectronics firmasına ve eğitim hayatım boyunca beni her konuda destekleyen aileme teşekkür ederim.

İnan ERDEM

İÇİNDEKİLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

ASIC	: Application Specific Integrated Circuit
CPLD	: Complex Programmable Logic Device
CRC	: Cyclic Redundancy Check
DFF	: D-type flip flop
FIPS	: Federal Information Processing Standard
FPGA	: Field Programmable Gate Array
GCLK	: Gated-clock
HDL	: Hardware Description Language
ISO	: International Organization for Standardization
JTAG	: Joint Test Action Group
LCD	: Liquid Crystal Display
LED	: Light Emitting Diode
LUT	: Lookup table
MUX	: Multiplexer
NESSIE	: New European Schemes for Signatures, Integrity, and Encryption
NIST	: National Institute of Standards and Technology
RTL	: Register Transfer Level
SHA	: Secure Hash Algorithm
STA	: Static Timing Analysis

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Bu çalışmanın amacı kriptografik özetleme algoritmalarından olan Sha özetleme algoritma ailesinin bir üyesi, Sha256 kriptografik özetleme algoritmasının tanıtılması ve Verilog donanım tanımlama dili kullanılarak anında(on the fly) Sha256 özetleme yapabilecek bir donanımın tasarlanmasıdır.

Sha kriptografik özetleme algoritma ailesi günümüzde çok yaygın olarak kullanılan en popüler özetleme algoritmalarındandır.

2. Bölüm'de genel olarak kriptografik özetleme algoritmaları, özelde de Sha256 algoritması incelenmiştir.

3. Bölümde bu çalışmada izlenen yöntemler ele alınarak ayrıntılı olarak açıklanmıştır.

4. Bölümde ise tasarımı yapılan donanımın simülasyon sonuçları grafikler halinde verilerek tasarımın istenen fonksiyonu yerine getirdiği tüm uç durumlar(corner cases) için gösterilmiş ve bu simülasyonlarla ilgili açıklamalar verilmiştir.

5. Bölümde yazılan kodun sentezi yapılmış ve bir Xilinx Spartan 3E FPGA kiti üzerinde gerçeklenmiştir.

Sonuç bölümü olan 6. Bölüm'de uygulama ile elde edilen sonuçlar ve bu uygulamanın geliştirmeye açık yönleri anlatılmıştır. Ayrıca 64 adımda özetleme yapabilen bir referans tasarım ile performans, alan ve güç kriterlerini ele alan karşılaştırmalar verilmiştir.

Ek-1'de, genel olarak donanım tanımlama dilleri ve bu dillere olan gereksinim ve özelde de bu tezde gerçekleştirilen uygulamada kullanılan Verilog HDL donanım tanımlama dili incelenmiş ve kullanım gerekçeleri kısaca verilmiştir.

Ek-2’de, Sha256 algoritması için örnek bir kod taslağı verilmiştir

2. KRİPTOGRAFİ, ÖZETLEME ALGORİTMALARI VE SHA256

2.1 Giriş

Kriptografi bilgi güvenlik bilimidir. Bu terimin karşıtı olan kriptoanaliz ise bilgiyi açmanın, ifşa etmenin bilimidir.Çoğu kez bu iki terim birlikte tek olarak kriptografi olarak adlandırılırlar. Dolayısıyla kriptografiyi, gizli bilgiyi açıklamaya veya bilgiyi gizlemeye çalışan, bilgi güvenlik ilimi olarak adlandırlabiliriz.

Diğer taraftan özetleme algoritmaları ise rasgele uzunluğa sahip olabilecek bir veriden çoğu kez sabit uzunluktaki(sabit bit sayısına sahip) ikinci bir veriyi elde etme ilimidir.

Özet(hash) terimi ilk kez 1953 yılında IBM' de bilgisayar bilimci olarak çalışan Hans Peter Luhn tarafından kullanılmış, ancak teknik anlamda kullanılan bir kelimeden artık resmi bir terime dönüşmesi Robert Morris' in CACM(Communications of the ACM) dergisindeki makalesinden sonra olmuştur.[1]

Kriptografik özetleme algoritmaları ise yukarıda genel özetleme algoritmaları için verdiğimiz tanıma ek olarak şu beş temel özelliği de bulunduran algoritmalarıdır.[2]

1. Mesaj herhangi bir uzunluğa sahip olabilir.
2. Özet sabit uzunluğa sahip olmalıdır.
3. Özet hasabı herhangi bir mesaj için görece olarak kolay hesaplanabilir olmalı.
4. Verilen bir özet değerinden mesajın kendisi elde edilememeli, tek yön(one-way) prensibi
5. İki ayrı mesajın aynı özet değeri olmamalı, çarpışmama (collision-free) prensibi.

Bir özetleme fonksiyonu, verilen bir özet değeri için mesajın kendisinin hesapla bulunamaz olması durumunda tek yönlüdür denir.

Yine bir özetleme fonksiyonu için , aynı özet değerini elde edecek iki mesajın yine hesapla bulunamaz olması durumunda ise çarpışmama(collision-free) özetlliğine sahiptir denir.

2.2 Kriptografik Özetleme Algoritmaları ve Uygulama Alanları

2.2.1 Giriş

Bu bölümde yaygın olarak bilinen ve kullanılan, gerek kırılmış gerekse de henüz başarılı bir atak rapor edilememiş özetleme algoritmaları ele alınacak ve genel kullanım alanları bu bölümün 2. alt bölümünde kısaca verilecektir.

2.2.2 Kriptografik Özetleme Algoritmaları

Aşağıda Tablo-1’ de yaygın olarak bilinen kriptografik özetleme algoritmaları ve bunların özet değerlerinin bit uzunlukları verilmiştir.

Özetleme Algoritması	**Bit Uzunluğu**
HAVAL	128-256
MD4	128
MD5	
RIPEMD-X	64-320
SHA-0,SHA-1	160
SHA-2(SHA-224,SHA-256,SHA-384,SHA-512)	224-512
Whirlpool	512

Tablo-1 : Yaygın olarak bilinen kriptografik özetleme algoritmaları.

HAVAL algoritması 1992 yılında tasarlanmış ancak 1997 yılında tekrardan güncellenmiştir. Bu güncelleme, önceki gerçeklemede bulunan ve Paulo Barreto(pbarreto@uninet.com.br) tarafından ortaya çıkarılan bir hatayı düzeltmek için yapılmıştır. Emsalleri olan MD4 ve MD5 algoritmalarının kısmen veya tamamen kırlıdığı bilindiği halde, henüz HAVAL'ın tüm varyantları için böyle bir atak sözkonusu değildir. Bu algoritma 128, 160, 192, 224 ve 256 bit uzunluğunda özet değerleri üretebilmektedir. bunlara ek olarak aldığı bir parametre ile bir mesaj bloğunun(1024 bit) kaç adımda işleneceği de belirlenebilmektedir, böylelikle çeşitli ihtiyaçlara göre güvenlik seviyesi ayarlanabilmekte dolayısıyla da hızdan kazanılabilmektedir.[3]

MD4 ve MD5 algoritmaları yaygın olarak kullanıldıkları halde, bu algoritmalara ilişkin yayımlanmış başarılı ataklar sebebiyle otoriteler tarafınan daha güvenli özetleme algoritmalarının kullanımı önerilmiştir.MD5 algoritması Ron Rivest taraından MD4' ün geliştirilmiş bir versiyonu olarak sunulmuş ancak yukarıda da belirtildiği gibi bu algoritma da kırılmıştır.[4]

RIPEMD(RACE Integrity Primitives Evaluation Message Digest) algoritması temelde MD4 üzerine kurulu bir algoritmadır. Bu algoritmanın da başarılı ataklardan sonra değişik özet uzunluğuna sahip versiyonları geliştirilmiştir. Bunlardan en yaygın olanı RIPEMD-160'tır. RIPEMD-160 Hans Dobbertin, Antoon Bosselaers ve Bart Preneel tarafından geliştirilmiştir[5]. Henüz başarılı bir atak rapor edilmese de SHA ailesi özetleme algoritmaları kadar yaygın kullanıma sahip olmadıklarından güvenlirlikleri yeterince test edilmemiştir.

SHA-0 algoritması SHA algoritma ailesinin ilk üyesidir ve 1993 yılında NIST tarafından yayımlanmıştır. Ancak kısa süre sonra 1995'te şu anda SHA-1 olarak bilinen ve SHA-0'a eklenen fazladan bir kaydırma(shift) işlemi içeren yeni bir versiyon daha güvenli olma amacıyla yayımlanmıştır.SHA-1 geniş anlamda kullanım alanı bulmuş ancak 2005 yılında bir güvenlik kusurunun bulunması[6] daha güçlü versiyonlarının geliştirilmesine sebep olmuştur, böylece SHA-2 adıyla bilinen aile ortaya çıkmıştır.

SHA-2 dört değişik üyeden oluşan ve yine NIST tarafından yayımlanan kriptografik özetleme algoritma ailesidir. Bu aileden SHA-256 ve SHA-512 temelde birbirinden farklı diğer iki üye ise bu algoritmaların özet değerlerinin kırpılmasıyla elde edilir. Algoritmaların en güncel versiyonları NIST tarafından FIPS PUB 180-3 adıyla Ekim 2008'de standart olarak yayımlanmıştır.[7]

Gerek MD5 gerekse de SHA-1 algoritmalarına ilişkin başarılı atakların bildirilmesi en çok kullanım alanı bulan bu iki algoritmanın artık yerlerini SHA-2 algoritma ailesine bırakmalarına sebep olmuştur.

Whirlpool Paulo S. L. M. Barreto ve Vincent Rijmen tarafından tasarlanan ve 512 bitlik özet değeri üreten bir özetleme algoritmasıdır. Yapısal olarak AES şifreleme algoritmasına benzemektedir. Whirlpool-0 ve ardından da Whirpool-T olarak bilinen ve kırılmış iki versiyonundan sonra Whirpool olarak bilinen en son versiyonu ISO tarafından stardart olarak yayımlanmıştır.[8]

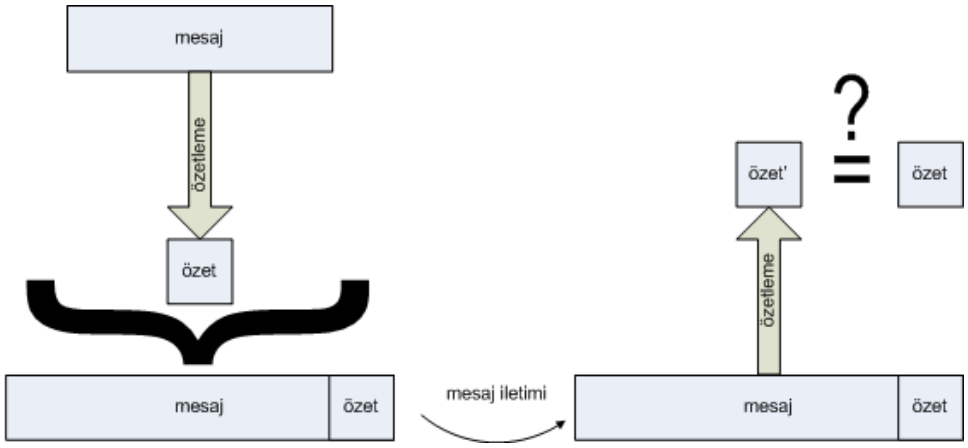
Whirpool, SHA-2 ailesi ile birlikte NESSIE tarafından kabul görmüş bir özetleme algoritmasıdır. Bu da Whirlpool' un gerek kullanımı ve gerekse de güvenirliği anlamında ona büyük avantaj sağlamaktadır.

2.2.3 Kriptografik Özetleme Algoritmalarının Kullanım Alanları

Kriptografik özetleme algoritmalarının belki de en bilinen uygulama alanı mesaj bütünlüğünün sınanması ve dijital imza uygulamalarıdır. Böylelikle gönderenden alıcıya ulaşan mesajın iletim sırasında bozulmadığı ve kimliği bilinen bir göndericiden alındığı tesbit edilebilir.

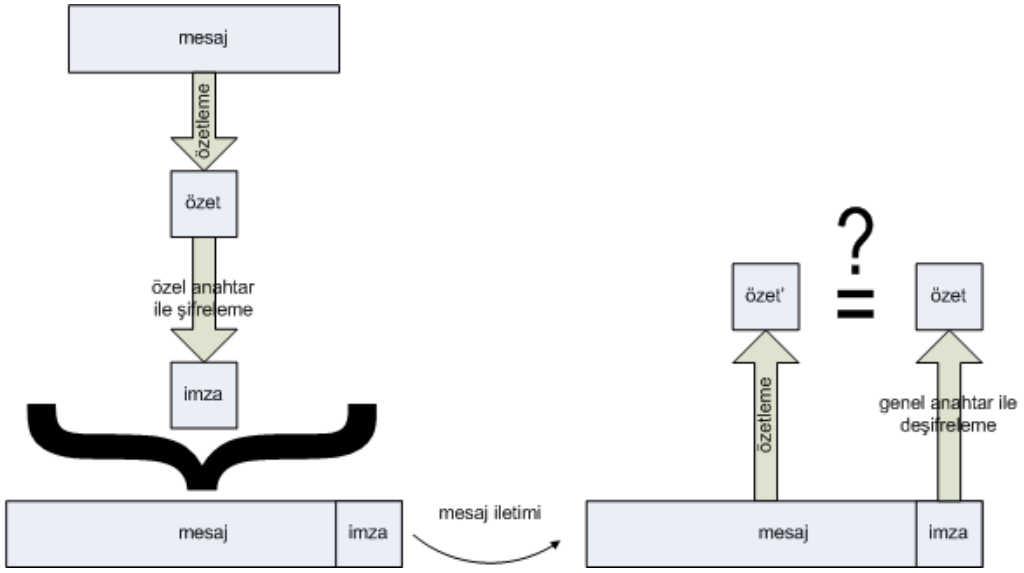
Mesaj bütünlüğü mesaja yukarıda açıklanmaya çalışılan özetleme algoritmalarından biri kullanılarak bir özet değerinin eklenmesi ile sağlanır; öyle ki, alıcı mesajın belirlenen özetleme algoritması ile elde edeceği özetin gönderen tarafından iletilen özet ile aynı olup olmadığını kontrol eder, böylece mesajın içeriğinin değişip değişmediği anlaşılabilir.

Mesaj bütünlüğü uygulaması basitçe Şekil-1’ deki gibi verilebilir, burada **özet’** değeri eğer **özet** değerine eşit çıkarsa bu durumda mesaj içeriğinin değişmediği söylenebilir.



Şekil-1 : Mesaj bütünlüğü uygulamasının basitçe gösterimi

Dijital imza uygulması ise, bu tezde ayrıntısı verilmeyen asitmetrik kripgorafi yöntemi kullanılarak gerçeklenir. Bu uygulamada imza değeri mesajın özet değerinin bir özel anahtar kullanılarak şifrelenmesi ile elde edilir. Mesaj iletimi yapıldıktan sonra, mesajın elde edilecek yeni özet değeri ile imzanın deşifre edilen değeri karşılaştırılır. Uygulama basitçe Şekil-2’deki gibi verilebilir, burada da **özet’** değeri **özet** değerine eşit ise alınan mesajın bilinen bir gönderen tarafından iletildiği anlaşılmış olur.



Şekil-2 : Dijital imza uygulamasının basitçe gösterimi

Yukarıda verilenler dışında özetleme algoritmalarının başka kullanım alanları da vardır, örneğin şifre doğrulama, sözde rasgele sayı üretimi, özet tabloları veri yapıları vb.

2.3 Sha256 Kriptografik Özetleme Algoritması

2.3.1 Giriş

Bu tezin asıl konusunu teşkil eden Sha256 özetleme algoritması, SHA-2 olarak yukarıda verilen ailenin bir üyesidir ve 256 bitlik özet değerine sahiptir. Bu bölümde bu algoritmanın açıklanması ve örnek bir kod taslağı verilecek ve tez konusu olan donanım tasarımının ayrıntıları 3. bölümde açıklanacaktır.

Sha256 kriptografik özetleme algoritmasının ayrıntılarına geçmeden önce, bu algoritmanın tüm Sha ailesi içerisindeki yeri aşağıdaki Tablo-2’ deki gibi basitçe verilebilir:

Algoritma	Mesaj Uzunluğu (bit)	Blok Uzunluğu (bit)	Kelime Uzunluğu (bit)	Özet uzunluğu (bit)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

Tablo-2 : Sha256 algoritmasının Sha ailesi içindeki yeri.

Tablodan da görüldüğü gibi Sha256 en çok 2^{64}-1 mesaj uzunluğuna kadar özetleme yapabilmektedir, ayrıca 512-bitlik bloklar üzerinde çalışmakta ve çalıştığı kelime uzunluğu 32 bittir. Son olarak adından da anlaşılabileceği gibi ürettiği özet değeri 256 bittir.

2.3.2 Örnek bir Kod Taslağı(Pseudo Code) İle Sha256 Algoritmasının Açıklanması

Algoritmanın açıklanması için aşağıdaki parametre, sembol ve terimler kullanılacaktır:

a, b, c, …, h : i. özet değerinin hesaplanması için kullanılan değişkenler

$H^{(i)}$: i. özet değeri, burada $H^{(0)}$ ilk özet değeri(initial hash value); $H^{(N)}$ ise sonuncu özet değeri, yani tüm mesajın özet değeridir.

$H_j^{(i)}$: i. özet değerinin j. kelimesi(32 bitlik değer)dir. Örneğin $H_0^{(i)}$ i. özet değerinin 0. kelimesidir, bu da en soldaki kelimedir.

K_t : t. özetleme iterasyonu için kullanılacak sabit değeridir.

k : mesaja eklenen toplam sıfır(0)' ların sayısıdır.

l : mesajın kaç bit uzunluğunda olduğunu gösteren değikendir.

m : bir mesaj bloğundaki toplam bit sayısıdır

M : özeti alınacak mesaj.

$M^{(i)}$: m bit sayısında sahip i. mesaj bloğudur.

$M_j^{(i)}$: i. mesaj bloğunun j. kelimesi(32 bitlik değer)dir. Örneğin $M_0^{(i)}$ i. mesaj bloğunun 0. kelimesidir, bu da en soldaki kelimedir.

n : üzerinde işlem yapılan kelimenin döndürülecek veya kaydırılacak bit sayısıdır.

N : dolgulanan(padded) mesajdaki toplam block sayısıdır

T : özet hesaplamada kullanılan geçici w-bit genişliğindeki kelimedir

w : bir kelimedeki toplam bit sayısıdır. Sha256 için bu değer 32'dir

W_t : t. mesaj çizelgeleme aşamasında kullanılan t. 32 bit uzunluğundaki kelimedir.

$\wedge$: bit tabanlı VE(AND) işlemi

$\vee$: bit tabanlı VEYA(iOR) işlemi

$\oplus$: bit tabanlı özel VEYA(XOR) işlemi

$\neg$: bit tabanlı tümleme(complement) işlemi

+ : mod 2^w ' e göre toplama işlemi

<< : sola kaydırma işlemi, burada solda bulunan ve kaydırma işlemiyle ezilen bitler atılır ve sağdan sıfır ile besleme yapılır.

>> : sağa kaydırma işlemi, burada sağda bulunan ve kaydırma işlemiyle ezilen bitler atılır ve soldan sıfır ile besleme yapılır.

Yukarıdaki parametre, sembol ve terim tanımlamalarının yanı sıra özetleme algoritmasının açıklanmasında aşağıdaki temel işlemler de tanımlanmıştır:

$ROTL^n(x)$: sola döndürme işlemi, burada x w-bit genişliğinde bir kelime, n toplam döndürme sayısı ve n bir tamsayı olmak üzere $0 \leq n \leq w$'dur.

$ROTR^n(x)$: sağa döndürme işlemi, burada x w-bit genişliğinde bir kelime, n toplam döndürme sayısı ve n bir tamsayı olmak üzere $0 \leq n \leq w$'dur.

$SHR^n(x)$: sağa kaydırma işlemi, burada x w-bit genişliğinde bir kelime, n toplam döndürme sayısı ve n bir tamsayı olmak üzere $0 \leq n \leq w$'dur.

Yukarıda verilenlere ek olarak, algoritmanın gerçeklenmesinde aşağıdaki fonksiyonlar kullanılacaktır:

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \quad ...1$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \quad ...2$$

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \quad ...3$$

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \quad ...4$$

$$\sigma_0^{\{256\}} = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x) \quad ...5$$

$$\sigma_1^{\{256\}} = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x) \quad ...6$$

Burada her bir x, y ve z değeri w-bit genişliğinde kelimeler ve sonuçlar da yine w-bit genişliğine sahip olan kellimelerdir. Üst indis olarak kullanılan 256, bu fonksiyonun Sha256 aloritmasını gerçeklemede kullanılan bir fonksiyon olduğunu göstermektedir. Ayrıca algoritmanın gerçeklenmesinde kullanılacak 32-bitlik 64 tane K_t sabiti de aşağıda verilmiştir:

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

Yukarıda verilenler ışığında algoritmanın nasıl çalıştığı şu şekilde verilebilir: Algoritma temelde iki adımdan oluşur, bunlar önişlem ve özet hesaplamadır, ayrıca önişlemi kendi içinde üç alt adıma ayırmak da mümkündür:

1. Önişlem
    a. dolgulama(padding)
    b. dolgulanmış mesajın m-bit'lik bloklara ayrılması
    c. özetlemede kullanılacak ilk değerlerin atanması
2. Özet hesaplama

Yukarıda verilen adımlardan önişlem bir örnek üzerinde şu şekilde gösterilebilir:

Dolgulama mesajın toplam uzunluğunun 512 bitin katları olmasını garantilemek için kullanılır, çünkü algoritma daha önce de belirtildiği gibi 512 bitlik bloklar üzerinde çalışmaktadır.

Dolgulama şu şekilde yapılır, l uzunluğuna sahip M mesajı için mesajın sonuna bir bitlik 1 değeri eklenir, ardından k tane 0 değeri eklenir, buarada k aşağıdaki eşitliği sağlayan en küçük pozitif değerdir,

$$l + 1 + k = 448 \bmod 512 ...7$$

Son olarak da mesajın son 64 bitlik kısmına l değeri ikilik sistemde yerleştirilir. Örneğin "abc" şeklindeki 24 bitlik M mesajı için 1 eklenme işlemi yapıldıktan sonra 448 – (24 + 1) = 423 tane de 0 ekleme işlemi yapılır, son olarak da 24 sayısı 64 bitlik ifade edilerek en sona eklenir:

$$\underbrace{01100001}_{"a"}\ \underbrace{01100010}_{"b"}\ \underbrace{01100011}_{"c"}\ 1\ \overbrace{00...00}^{43}\ \overbrace{00...\underbrace{011000}_{l=24}}^{64}$$

Sonuç olarak dolgulama işlemi sonrasında elde edilen yeni M' mesajı 512' nin katları şeklindeki bit uzunluğuna sahip olacaktır.

Dolgulama işlemi bittikten sonra M mesajı 512 bitlik N tane bloğa ayrılır ve yukarıda da verildiği gibi bunlar $M^{(1)}$, $M^{(2)}$... $M^{(N)}$ şeklinde ifade edilir. Ayrıca her block 16 tane 32-bitlik kelimeden oluştuğundan i. blok $M_0^{(i)}$, $M_1^{(i)}$...$M_{15}^{(i)}$ olarak da ifade edilebilir.

Mesaj bu şekilde bloklara ayrıştırıldıktan(parsing) sonra, sıra özette kullanılacak ilk değerlerin atanmasına gelir. Sha256 için bunlar aşağıdaki gibidir:

$$H_0^{(0)} = 6a09e667...8$$
$$H_1^{(0)} = bb67ae85...9$$
$$H_2^{(0)} = 3c6ef372...10$$
$$H_3^{(0)} = a54ff53a...11$$
$$H_4^{(0)} = 510e527f...12$$
$$H_5^{(0)} = 9b05688c...13$$
$$H_6^{(0)} = 1f83d9ab...14$$
$$H_7^{(0)} = 5be0cd19...15$$

Yukarıdaki ilk değerlerin atanmasından sonra önişlem aşaması tamamlanmış olur. Bir sonraki aşama özet hesaplama adımıdır. Özet işleme yukarıda verilen sabit ve fonksiyonları kullanarak şu şekilde gerçekleşir:

Öncelikle $M^{(i)}$ mesaj blokları aşağıda mesaj çizgeleme(scheduling) dediğimiz işlemden geçirlilerek 64 adet W_t değerleri elde edilir:

i 1'den N'ye kadar:
{

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15...16 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 15 \le t \le 63...17 \end{cases}$$

Ardından a, b, c, …, h özet değer hesaplama değişkenlerine, (i-1) özet değerleri ile ilk değer atama işlemi uygulanır:

$a = H_0^{(i-1)} ...18$

$b = H_1^{(i-1)} ...19$

$c = H_2^{(i-1)} ...20$

$d = H_3^{(i-1)} ...21$

$e = H_4^{(i-1)} ...22$

$f = H_5^{(i-1)} ...23$

$g = H_6^{(i-1)} ...24$

$h = H_7^{(i-1)} ...25$

Sonrasında özet hesaplamanın ana çevrim(main loop) işlemi denilen ve özet değer hesaplama değişkenlerinin güncellendiği 64'lük çevrim yapılır:

t 0'dan 63'e kadar

{

$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t ...26$

$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c) ...27$

$h = g ...28$

$g = f ...29$

$f = e ...30$

$e = d + T_1 ...31$

$d = c ...32$

$c = b ...33$

$b = a ...34$

$a = T_1 + T_2 ...35$

}

Son adım olarak da üzerinde işlem yapılan i. mesaj bloğu için ara özet değeri(intermediate hash value) hesaplanır.

$$H_0^{(i)} = a + H_0^{(i-1)} \quad ...36$$
$$H_1^{(i)} = b + H_1^{(i-1)} \quad ...37$$
$$H_2^{(i)} = c + H_2^{(i-1)} \quad ...38$$
$$H_3^{(i)} = d + H_3^{(i-1)} \quad ...39$$
$$H_4^{(i)} = e + H_4^{(i-1)} \quad ...40$$
$$H_5^{(i)} = f + H_5^{(i-1)} \quad ...41$$
$$H_6^{(i)} = g + H_6^{(i-1)} \quad ...42$$
$$H_7^{(i)} = h + H_7^{(i-1)} \quad ...43$$

}

Böylece yukarıdaki adımlar döngü sayısından da anlaşıldığı gibi N defa tekrar edilerek ara özet değerler hesaplanır ve sonuncu özet değer de algoritmanın ürettiği nihai özet değer olmuş olur. Nihai özet değer en sonuncu ara özet değerlerin birleştirilmesi(concatenation) işlemi ile şu şekilde gösterilebilir:

$$H_0^{(N)} \parallel H_1^{(N)} \parallel H_2^{(N)} \parallel H_3^{(N)} \parallel H_4^{(N)} \parallel H_5^{(N)} \parallel H_6^{(N)} \parallel H_7^{(N)} \quad ...44$$

Yukarıda açıklanmaya çalışılan algoritma için tam bir kod taslağı ise EK-2’ de verilmiştir.

3. ANINDA(ON THE FLY) SHA256 GERÇEKLENMESİ

3.1 Giriş

Bölüm 2’ de örnek bir kod taslağıyla açıklanmaya çalışılan Sha256 algoritmasının tez konusu olan anında özetleme yapabilecek şekilde nasıl tasarlandığı ve bunun ayrıntıları bu bölümde verilecektir.

Bölüm 2’ de verilen kod yalnızca özetlemenin nasıl yapılacağına yönelik olmakla birlikte gerçek bir sistemde merkezi işlem birimi veya host ile olan haberleşme de önemli bir tasarım problemi olmaktadır. Dolayısıyla host ile olan haberleşme ve verinin Sha256 çekirdeği(core) içerisine alınması ve sonucun host’a aktarılması da burada anlatılmıştır.

Bu bölümde tez konusu olan donanım için , verinin okunması(message reading), önişlem(padding), özet hesaplama(hash calculation) ve özet değerinin dışarıya aktarılması (hash reading) adımları ayrı ayrı ele alınarak anlatılmıştır; ancak öncesinde sistemin genel olarak nasıl çalıştığı anlatılmaya çalışılmıştır.

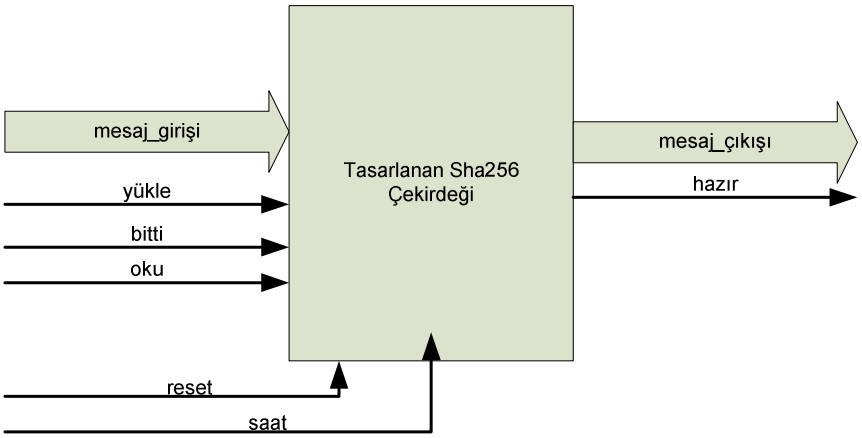
3.2 Anında Sha256’ya Sistem Seviyesi Bakış

Günümüz sayısal sistemlerinin en yaygın kullandığı veri yolu (data bus) genişliklerinden biri de 32 bitlik veri yolu genişliğidir, bu tezde de veri yolu genişliği 32 bit olarak seçilmiştir. Dolayısıyla verinin Sha256 içerisine alınması paralel bir arayüz(interface) üzerinden gerçekleştirilmiştir. Veri message_inp adında 32 bitlik bir kapı(port)’ dan alınmaktadır.

Sistem tek bir saat(clock) işareti ve bu saat işaretinin yükselen kenarına senkron olarak çalışmaktadır. Dolayısıyla gerek verinin alınması gerekse de özet değerinin hesaplanması saat işaretinin yükselen kenarına senkrondur.

Sistem düşük seviye(low level) asenkron bir reset ile resetlenmekte ve reset durumuna(reset state) geçmektedir.

Şekil-3' te sistemin genel bir şeması yer almaktadır.



Şekil-3 : Tasarlanan anında Sha256 donanımının sistem seviyesi şeması

Ayrıca bunların dışında mesajın yüklenmesini kontrol eden iki adet giriş kapısı(input port) ve hesaplanmış özet değerini okumayı başlatan bir adet giriş kapısı daha bulunmaktadır, bunlar load(yükle), finish(bitir) ve read(oku) kapılarıdır.

Sistemin ayrıca iki adet çıkış kapısı bulunmaktadır, bunlardan birincisi 32 bitlik özet değerini dışarıya veren message_out, diğeri ise tek bitlik ready(hazır) kapısıdır. Ready işareti Sha256 çekirdeğinin özet hesaplama işlemini bitirdiğini bildiren ve host'a giden bir kontrol işarettir.

Bu basit arayüz sayesinde tasarlanan donanımın herhangi bir sisteme gömülmesi ve kullanılması oldukça kolay olacak ve aşağıdaki gibi basit bir akış söz konusu olacaktır:

→başla
→masajı oku
→ mesajı oku + özet hesapla

→ mesajı oku + özet hesapla

.

.

.

→ mesajı oku + özet hesapla

→ özet hesapla

→özeti dışarıya aktar.

Anında(on the fly) bir yaklaşım olmadığını düşündüğümüzde ise akış şu sekilde olacaktır:

→başla

→masajı oku

→ özet hesapla

→ mesajı oku .

.

.

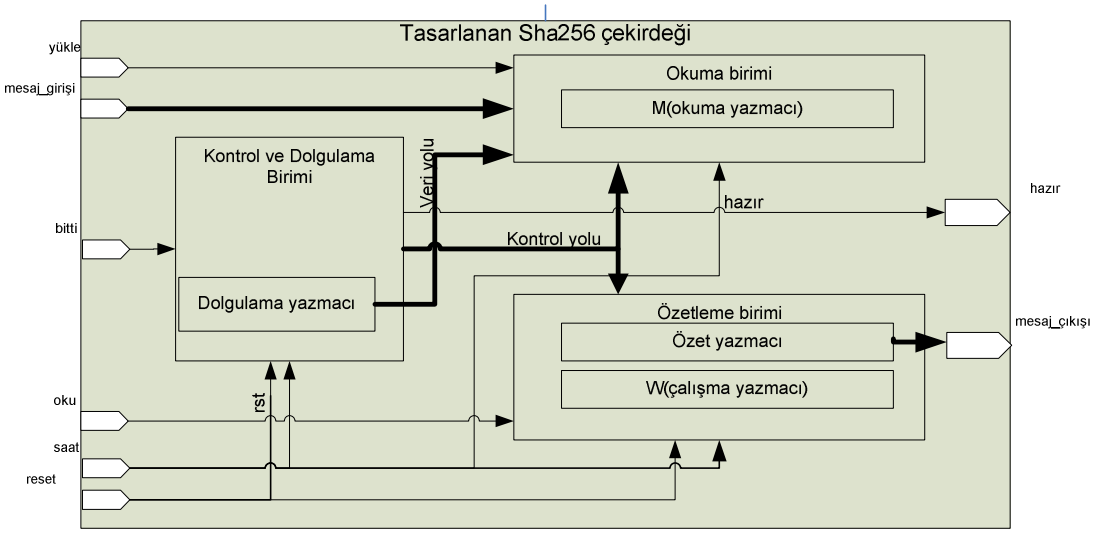
.

→ özet hesapla

→özeti dışarıya aktar.

Yukarıdan da görüldüğü gibi ilk durumdaki iş çıkarma yeteneği(throughput) ikinci duruma göre daha yüksek olacaktır.

Tezde tasarlanan Sha256 çekirdeğini yine sistem seviyesinde ayrıntılı olarak Şekil-4’ teki gibi gösterebiliriz, burada sistem içerisindeki tüm kontrol ve veri yolları çizilmemiş, daha çok kavramsal olarak ve basit bir gösterim yapılmıştır



Şekil-4 : Tasarlanan anında Sha256 donanımının sistem seviyesi ayrıntılı şeması

Yukarıdaki ayrıntılı şemadan kısaca şu sonuçları çıkarabiliriz:

1. Sistem kendi aralarında kontrol işaretleri ile haberleşen üç bağımsız ve paralel çalışabilen alt birimlerden oluşmaktadır.

2. Sistemin tamamını kontrol eden birim(Kontrol ve Dolgulama Birimi/Control and Padding Unit) aynı zamanda dolgulama işlemini de yapmaktadır.

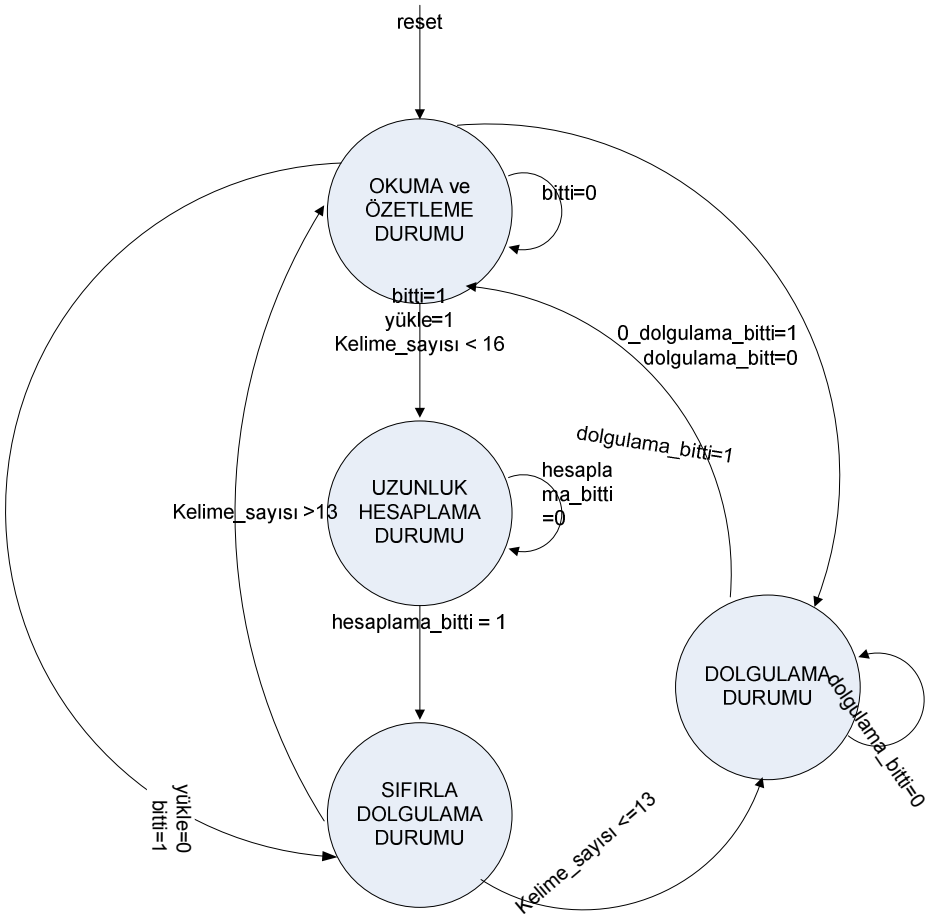
3. Mesaj 32 bitlik message_inp kapısından M adlı bir yazmaca(register) okunmaktadır.

4. Özetleme işlemini yapan birim ise okunan mesajı W yazamacında işlemektedir.

5. Hazır(ready) işareti yine kontrol birimi tarafından üretilmektedir.

6. Son olarak okunacak özet değeri özet yazmacı(hash register)' ndan message_out kapısı yardımıyla dışarıya verilmektedir.

Sistem bir sonlu durum makinesi(finite state machine, FSM) olarak gerçeklenmiş ve dört temel durumdan(state) oluşmuştur. Bu durumlar ve durumlararası geçiş yine kavramsal olarak ve tüm durum geçiş işaretleri çizilmeden basitçe Şekil-5' te verilmiştir.



Şekil-5 : Tasarlanan anında Sha256 donanımının durum geçiş diyagramı

Yukarıdaki temel durumlara ek olarak, aynı durum içerisinde birden çok saat işaretinde gerçeklenecek işlemler için ise bir sayaç tutulmuş ve bu sayacın değeri kontrol edilerek o anki durumda kalınmasına veya sonraki duruma geçilmesine karar verilmiştir.

Sistem seviyesi böylece açıklandıktan sonra her bir alt birimin nasıl çalıştığına bakılabilir. Bu bölümün geriye kalan kısımlarında bu birimler ele alınmaktadır.

3.3 Mesajın Okunması

Mesajın okunması işlemi M yazmacı bir kaydıran yazmaç şeklinde kullanılarak gerçekleştirilmiştir, böylece bir çoğullayıcı yapısı yerine daha yüksek performansta çalışabilen bir yapı tercih edilmiştir.

Verilog donanım tanımlama dili bağlamında, bu işlem ayrı bir always bloğu olarak tasarlanmıştır ve basitçe aşağıdaki gibi verilebilir:

```
//Okuma işlemi
always @(posedge clk)
begin
case (state)
  STATE_READ_AND_HASH:
  begin
    if(load)
    begin
        M <= M >> 32;
        M[511:480]  <= message_inp;
...
```

Görüldüğü gibi okuma işlemi saat işareti(clk)' nin yükselen kenarı(posedge) ile senkrol olarak çalışmakta ve eğer aktif durum STATE_READ_AND_HASH ise ve load işareti de aktif ise okuma gerçeklenmektedir.

M yazmacının en yüksek anlamlı 32 biti üzerine message_inp kapısındaki değer yazılmakta ve yazmaç 32 bit sağa kaydırılarak bir sonraki okunacak 32 bitlik değerin halihazırda okunan değeri ezmesi engellenmektedir.

Yukarıda verilen okuma akışı, olası çok sayıdaki diğer akışlardan sadece biridir. Örneğin veri yolunun seri bir veri yolu olması durumunda yapı tamamen değişecek ve okuma bit bit yapılacaktır. Aynı şekilde okuma kapısının genişliğinin 32 bitten farklı olması da yine okuma sayısının değişmesine ve yazmacını da yeniden tasarlamaya sebep olacaktır.

2. bölümde ayrıntısı verilen algoritmadan da hatırlanacağı gibi, dolgulama işlemi içerisinde 512 bitlik bloğun son 64 bitlik kısmı mesajın uzunluğu ile doldurulmaktadır. Bu uzunluğun nasıl hesaplanacağı ise donanım tasarlanırken karar verilmesi gereken en önemli problemlerde biri olmuştur.

Mesaj uzunluğunun belirlenmesinde iki temel yol izlenebilir; bunlar uzunluğun doğrudan host tarafından 64 bit olarak verilmesi, diğer ise bu uzunluğun Sha256 çekirdeği tarafından içeride hesaplanmasıdır. Birinci yöntem host' un yükünü artıracağından ve fazladan yazılım eforu gerektirdiğinden, bu işlemi donanım içerisinde gerçekleme yolu seçilmiştir. Ayrıca bu yöntem basitleştirilmiş arayüze de daha uygun olacaktır, çünkü böylece sistem başka sitemlere daha kolay şekilde adapte edilebilecektir.

Sonuç olarak okunan son 32 bitlik kelime kendinden önce yüklenen 32 bitlik kelimenin kaç bitinin anlamlı olduğu bilgisini içerecek şekilde bir tasarım yapılmış ve bu bilgi kullanılarak mesajın toplam uzunluğu Sha256 çekirdeği tarafından hesap edilmiştir.

3.4 Dolgulama İşleminin Yapılması

Dolgulama işlemi sonlu durum makinesinin yukarıda da gösterilen STATE_PAD0 ve STATE_PAD durumlarında gerçekleştirilmektedir.

Görüldüğü gibi özetleme durumuna hem STATE_PAD0' dan hem de STATE_PAD durumundan geçilebilmektedir. STATE_PAD0'dan doğrudan özetleme durumuna geçilmesi şu koşulda gerçekleşir: dolgulanan 512 bitlik bloğun mesajın son bloğu olmaması (yani uzunluk yerleştirecek 64 bitlik bir kısım kalmaması) 0' la dolgulama işleminden sonra önce bir özet hesaplamayı gerektirecektir. Bu özet hesaplandıktan sonra Şekil-5' ten de görüleceği gibi

pad0_done işareti 1 olacağından, özet durumundan STATE_PAD0’ a gidilmeden doğrudan STATE_PAD durumuna geçilecek ve buradaki işlem de tamamlandıktan sonra en son özet işlemi için STATE_READ_AND_HASH durumuna geçilerek özet alma(hashing) işlemi tamamlanmış olacaktır.

3.5 Özet Hesaplama

Özet hesaplama daha önce de belirtildiği gibi mesaj okuma ile aynı zamanda paralel olarak yapılan(concurrent) bir işlem olarak tasarlanmıştır. Bunun gerçeklenmesi için 64 döngülük Sha256 algoritmasının 16 döngüye indirgenmesi yoluna gidilmiştir.

Bu indirgeme bir anında(on the fly) gerçekleme için şarttır, çünkü 512 bitlik mesaj blokları 32 bitlik bir veri yolundan toplamda 16 saat işareti(clock cycle) içerisinde okunup tamamlanmaktadır. Dolayısıyla bir sonraki okumanın bekletilmemesi özetlemenin de toplamda 16 saat işareti içerisinde yapılmasını zorunlu kılmaktadır.

Aslında daha yakından bakıldığında her ne kadar ana döngü(main loop) toplamda 64 ise de, W yazmacının M yazmacından ilk değerini alması ve ara özet değerlerinin güncellenmesi işlemleri ile birlikte bu sayının etkin olarak 66 olduğunu söyleyebiliriz.O yüzden tez çalışmasında öncelikli olarak bu 66’ lık döngü önce 64’ lük bir döngüye indirilmeye çalışılmış ve ardından da 64’ lük döngü de 16’ lık bir döngüye indirilmiştir.

İlk indirgemeler için kullanılan yolu gösteren kısmi kod aşağıdadır:

```
...
if(counter == 6'd15)
 begin
            hash[255 : 224] <= hash[255 : 224] + a4;
             hash[223 : 192] <= hash[223 : 192] + b4;
             hash[191 : 160] <= hash[191 : 160] + c4;
```

```
    hash[159 : 128] <= hash[159 : 128] + d4;
    hash[127 :  96] <= hash[127 :  96] + e4;
    hash[ 95 :  64] <= hash[ 95 :  64] + f4;
    hash[ 63 :  32] <= hash[ 63 :  32] + g4;
    hash[ 31 :   0] <= hash[ 31 :   0] + h4;

    a <=hash[255 : 224] + a4;
    b <=hash[223 : 192] + b4;
    c <=hash[191 : 160] + c4;
    d <=hash[159 : 128] + d4;
    e <=hash[127 :  96] + e4;
    f <=hash[ 95 :  64] + f4;
    g <=hash[ 63 :  32] + g4;
    h <=hash[ 31 :   0] + h4;
    W <=  W >> 128;
    //Yeni W değerlerinin güncellenmesi...
  end
  else if(counter == 6'd0)
  begin
    a <= a4;
    b <= b4;
    c <= c4;
    d <= d4;
    e <= e4;
    f <= f4;
    g <= g4;
    h <= h4;
    W <= M >> 128;
    //Yeni W değerlerinin güncellenmesi
  end
  else
  begin
    a <= a4;
    b <= b4;
```

```
            c <= c4;
            d <= d4;
            e <= e4;
            f <= f4;
            g <= g4;
            h <= h4;
            W <=  W >> 128;
            //Yeni W değerlerinin güncellenmesi
          end
...
```

Yukarıdaki kod parçacığında counter sayacı 0’ dan 15’ e kadar değerler almakta ve Sha256 çekirdeğinin ana döngülerinden herbirini kontrol etmede kullanılmaktadır.

Kod parçacığından da analşılabileceği gibi counter sayacının 0(sıfır) değerinde olması durumu için W yazmacı doğrudan M yazmacından yüklenmektedir, bu da 512 bitlik bloğun son okuması gerçekleştirildikten sonra M yazmacının W yazmacına yüklenmesi için bir saat işareti daha kaybedilmesini engellemiştir böylece okuma işlemi biter bitmez yeni mesaj bloğu öncekini ezmeden(overwirte) okunan block W yazmacına çekilmiş olmaktadır.

Diğer taraftan counter sayacının 15 değerini alması durumunda da ara özet değerlerinin güncelleme işlemi hemen yapılmaktadır, dolayısıyla bu güncellemenin bir durum daha işgal etmesinin önüne geçilerek yine bir saat işaretinden daha kazanılmış olmaktadır.

Diğer durumlarda yapılanlar ise(counter sayacının 0 ve 15’ ten farklı olması durumları) bölüm 2’ de açıklandığı gibi a, b, c, ... h değişkenlerinin güncellenmesinden ibarettir, ancak buna ek olarak W yazmacı da sağa ötelenerek bir sonraki döngü için hazır hale getirilmektedir.

Sonuç olarak 66’ lık döngü böylelikle 64’ lük bir döngü haline getirilmiştir. Bir sonraki adım ise 64’ lük ana döngünün 16’lık bir döngü haline indirgenmesidir. Bunun için ise şu yol izlenmiştir:

Nasıl ki a, b, c, ...h değişkenleri 64’ lük Sha256 döngüsündeki her bir adımda güncellenerek toplamda 64 değer alıyor ise; bu mantıktan yola çıkılarak her dört adım sonrasındaki a, b, c, ... h değerlerinin lojik ifadeleri çıkarılarak bunlar hesap edilmiş böylece toplamda 16 adımda hesap edilebilecek a4, b4, c4 ... h4 değerleri donanım içerisine konulmuştur. Buna ilişkin kod parçacığı ise aşağıdadır:

```
assign h1 = g;
assign g1 = f;
assign f1 = e;
assign e1 = d + h + S1MainLoop(e) + Ch(e, f, g) + current_K[31:0] + currentM[31:0]
;
assign d1 = c;
assign c1 = b;
assign b1 = a;
assign a1 = (h+S1MainLoop(e)+Ch(e, f, g) + current_K[31:0] + currentM[31:0]) +
(S0MainLoop(a) + Maj(a,b,c)) ;

assign h2 = g1;
assign g2 = f1;
assign f2 = e1;
assign e2 = d1 + h1 + S1MainLoop(e1) + Ch(e1, f1, g1) + current_K[63:32] +
currentM[63:32] ;
assign d2 = c1;
assign c2 = b1;
assign b2 = a1;
assign a2 = (h1+S1MainLoop(e1)+Ch(e1, f1, g1) + current_K[63:32] +
currentM[63:32]) + (S0MainLoop(a1) + Maj(a1,b1,c1));

assign h3 = g2;
assign g3 = f2;
```

```
assign f3 = e2;
assign e3 = d2 + h2 + S1MainLoop(e2) + Ch(e2, f2, g2) + current_K[95:64] +
currentM[95:64];
assign d3 = c2;
assign c3 = b2;
assign b3 = a2;
assign a3 = (h2+S1MainLoop(e2)+Ch(e2, f2, g2) + current_K[95:64] +
currentM[95:64]) + (S0MainLoop(a2)+Maj(a2,b2,c2)) ;

assign h4 = g3;
assign g4 = f3;
assign f4 = e3;
assign e4 = d3 + h3 + S1MainLoop(e3) + Ch(e3, f3, g3) + current_K[127:96] +
currentM[127:96];
assign d4 = c3;
assign c4 = b3;
assign b4 = a3;
assign a4 = (h3+S1MainLoop(e3)+Ch(e3, f3, g3) + current_K[127:96] +
currentM[127:96]) + (S0MainLoop(a3)+Maj(a3,b3,c3));
```

Yukarıdaki kod parçacığından da anlaşılacağı gibi herbir a4, b4, c4 ... h4 değişkenleri a, b, c ... h değikenlerinden, ve bunlara karşılık gelen K sabitleri ve M yazmaç değerlerinden elde edilmiştir.

3.6 Özet Değerinin Dışarıya Aktarılaması

Özet değeri daha önceki kısımlarda da belirtildiği gibi read(oku) kontrol işareti ile dışarıya aktarılmaya başlanır, buna ilişkin Verilog kodu basitçe aşağıda verilmiştir:

```
...
assign message_out =  hash[255 : 224];
...
if(read)
    begin
      hash <= (hash << 32) | (hash >> 224);
      //Aşağıdaki kısım fpga_cpld gerçeklenmesi içindir.
      //hash <= (hash << 4) | (hash >> 252);
end//if (read) begin
```

Kod parçacığından da anlaşılabileceği gibi, eğer read kontrol işareti aktif ise hash özet yazmacı döndürme işlemine tabi tutulmakta böylece yüksek anlamlı 32 bitlik kısmı message_out çıkış kapısından dış dünyaya verilmektedir.

4. SİMÜLASYONLAR

4.1 Giriş

Bu bölümde tasarımı gerçekleştirilen Sha256 ile ilgili tüm uç durumlar(corner case) için simulasyon sonuçları verilmiştir. Böylece tasarımın istenen işlevi doğru şekilde yerinde getirdiği gösterilmeye çalışılmıştır. Bu uç durumlar sırasıyla:

1. Sıfır uzunluklu mesajın özet değerinin hesaplanması

2. 0’dan büyük 448’den küçük uzunluğa sahip mesajların özet değerlerinin hesaplanması

3. 448 bit’lik uzunluğa sahip mesajların özet değerlerinin hesaplanması

4. 448’den büyük 512’den küçük uzunluğa sahip mesajların özet değerlerinin hesaplanması

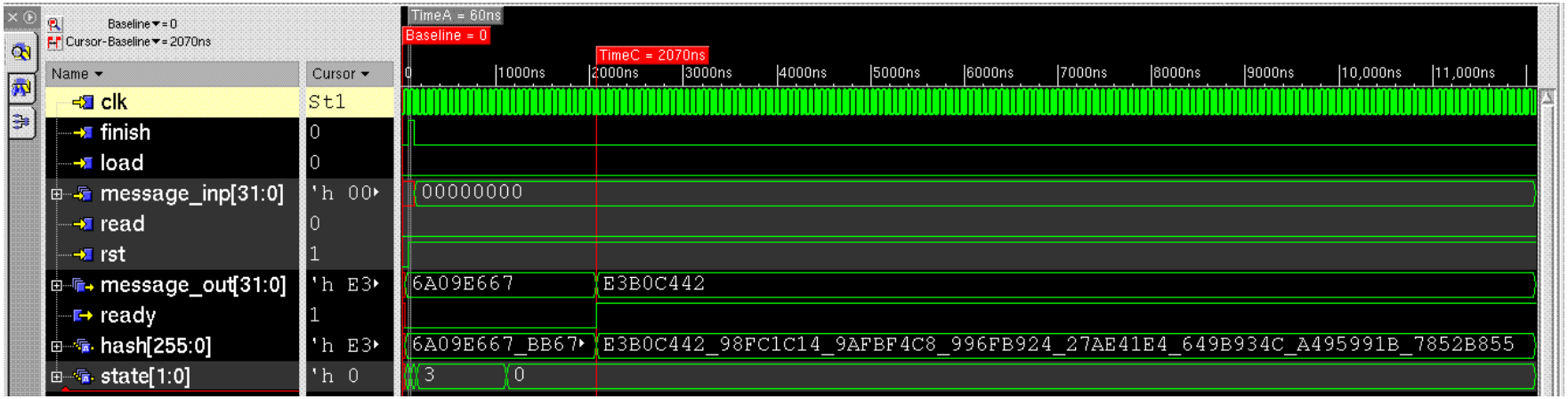
5. 512 bit’lik uzunluğa sahip mesajların özet değerlerinin hesaplanması

6. 512’den büyük uzunluğa sahip mesajların özet değerlerinin hesaplanması

4.2 Uç Durumlar İçin Simulasyon Sonuçları

Bu bölümde tasarımı yapılan Sha256 donanımının simulasyon sonuçları tüm uç durumlar için verilecektir. Böylece tasarımın tüm bu durumlar için doğru sonuçlar verdiği gösterilmeye çalışılacaktır. Referans alınacak özet değerleri için [10] numaralı kaynaktan yararlanılmıştır.

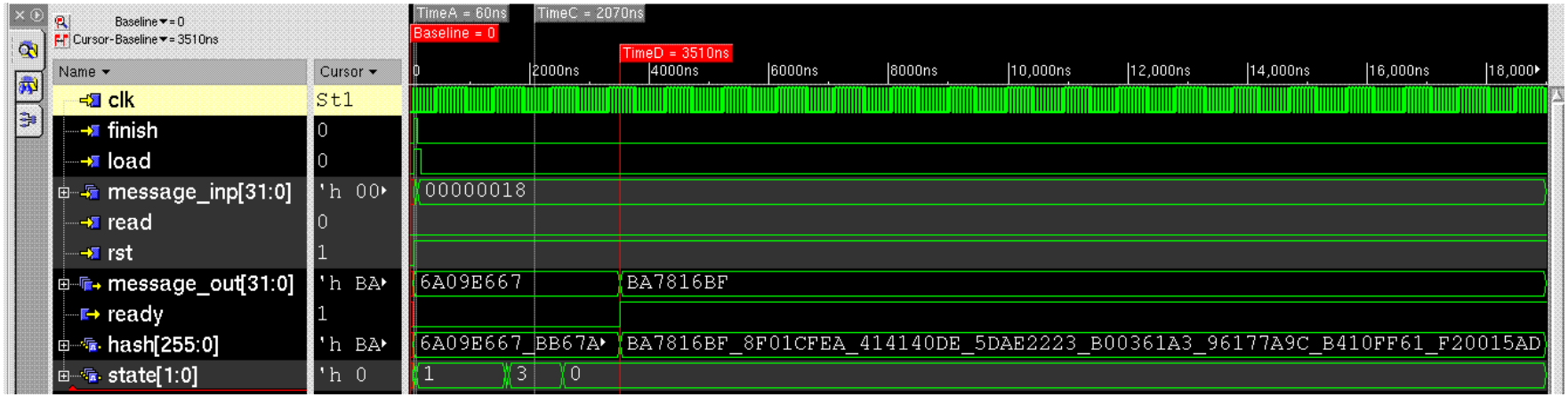
2.3.1 0 Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-6 : 0 Uzunluklu Mesaj için Simulasyon Sonucu

Simulasyon sonucu tasarımın 0 uzunluklu bir mesaj için doğru olan özet değerini(e3b0c44298fc1c149afbf4c8996fb924 27ae41e4649b934ca495991b7852b855) ürettiğini göstermektedir. Ayrıca sistemin reset durumuna girmesi, reset durumundan çıkması, 0 uzunluklu bir mesajın okunması için gerekli olan load işaretinin oluşturulması, dolgulama ve özet hesaplama adımlarının gerçeklenmesi ve son olarak da ready(işaretinin) tüm işlemler sonucu 1' e çekilmesi adımları da Şekil-6' dan görülebilmektedir. Son olarak bu işlemler sırasında durum(state) geçişlerinin de olduğu yine state değişkeninin değer değiştirmesinden anlaşılabilir.

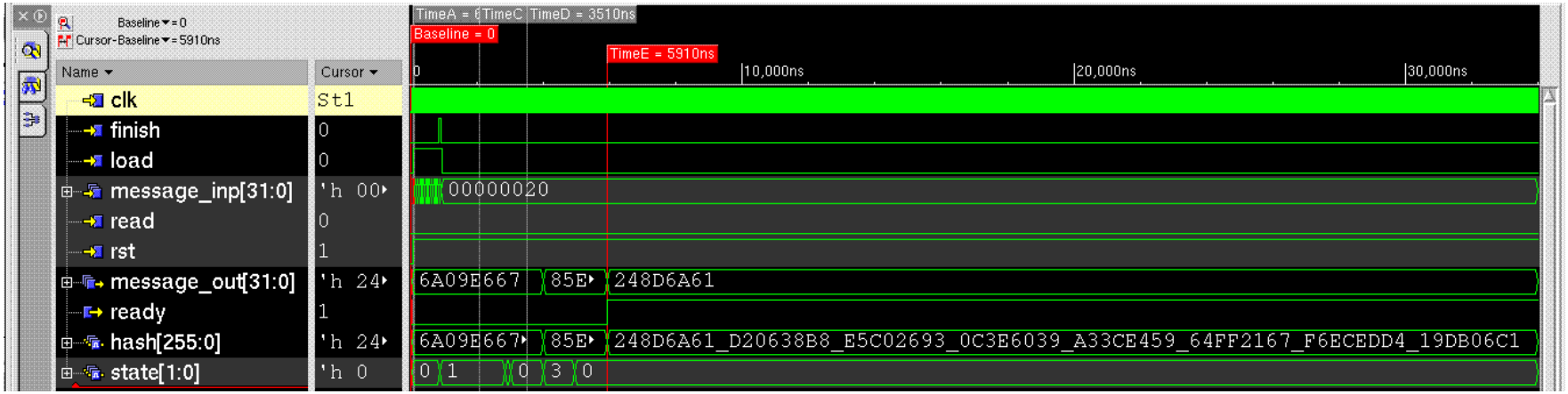
2.3.2 0’dan Büyük 448’den Küçük Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-7 : “abc” Mesajı İçin Simulasyon Sonucu

Simulasyon sonucu tasarımın “abc” mesajı için doğru olan özet değerini(ba7816bf8f01cfea414140de5dae2223 b00361a396177a9cb410ff61f20015ad) ürettiğini göstermektedir. Ayrıca 0 uzunluklu mesajdan farklı olarak bu mesajın yüklenmesi için load işaretinin de 1’e çekilip tekrardan sıfırlandığı görülmektedir.

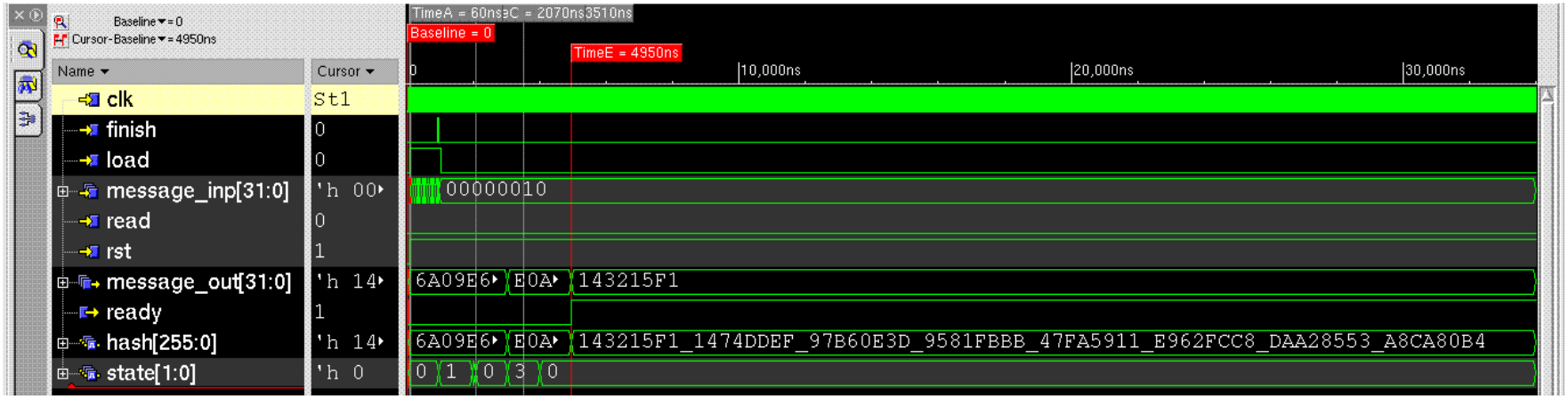
2.3.3 448’e Eşit Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-8 : “abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq” Mesajı İçin Simulasyon Sonucu

Simulasyon sonucu tasarımın “abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq” mesajı için doğru olan özet değerini(248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1) ürettiğini göstermektedir. Ayrıca 448 uzunluklu bir mesajın sonuna 1 eklendiğinde geriye uzunluk için 64 bitlik alan kalmayacağından iki kere özetleme yapılacağı da simulasyondan görülmektedir, gerçekten de hash yazmacı önceki iki simulasyonda bir kere değiştiği halde burada iki kere değişmiştir.

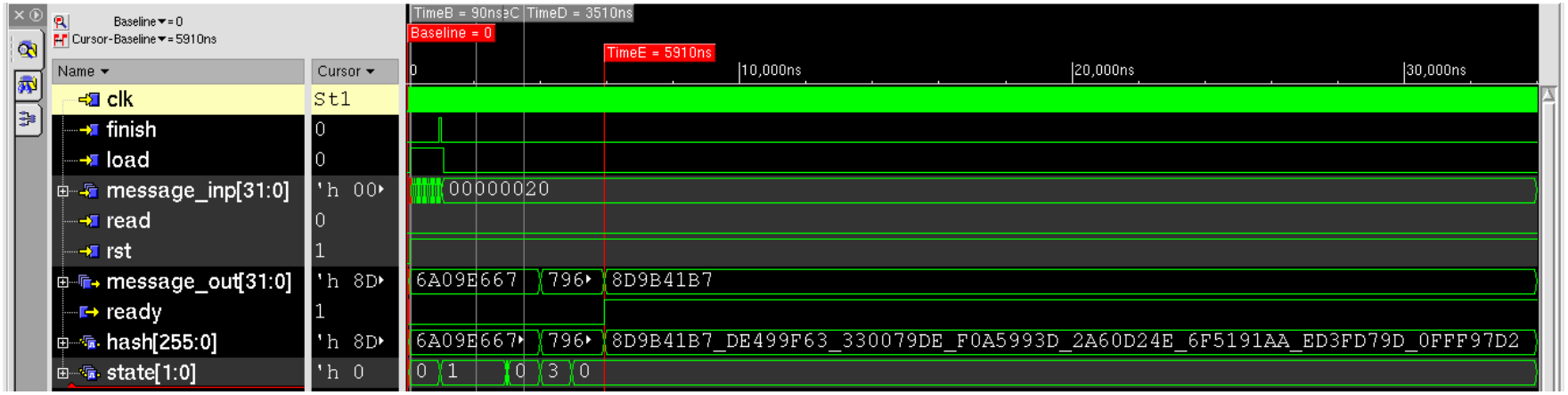
2.3.4 448’den Büyük 512’den Küçük Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-9 : “GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusu” Mesajı İçin Simulasyon Sonucu

Simulasyon sonucu tasarımın “GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusu” mesajı için doğru olan özet değerini(143215f11474ddef97b60e3d9581fbbb 47fa5911e962fcc8daa28553a8ca80b4) ürettiğini göstermektedir. Burada da 448 uzunluklu bir mesajda olduğu gibi mesajın sonuna 1 eklendiğinde geriye uzunluk için 64 bitlik alan kalmayacağından iki kere özetleme yapılmıştır.

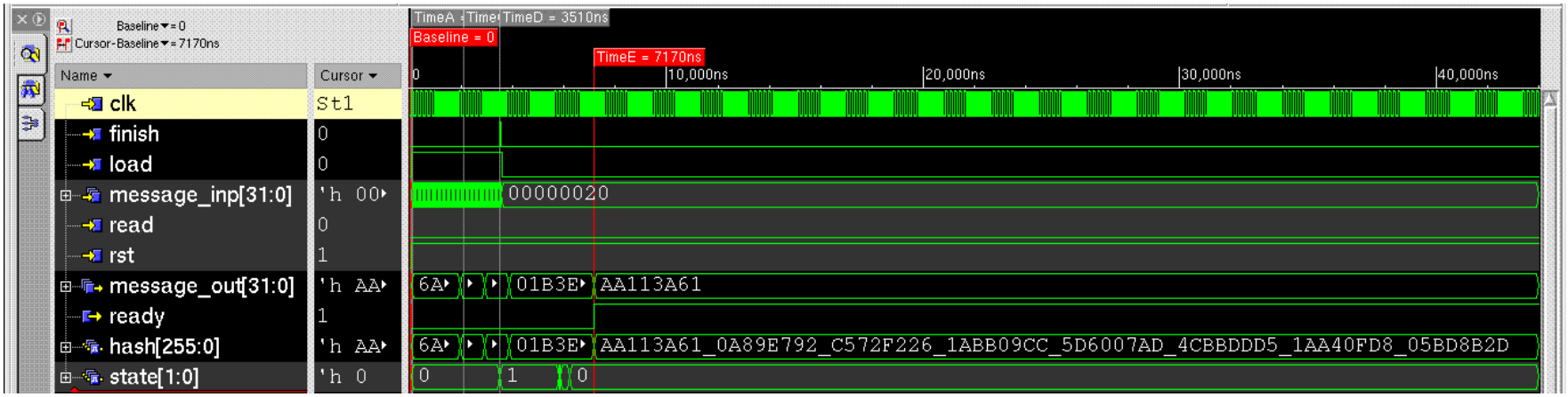
2.3.5 512’ye Eşit Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-10 : “GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGYTE09” Mesajı İçin Simulasyon Sonucu

Simulasyon sonucu tasarımın “GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGYTE09” mesajı için doğru olan özet değerini(8d9b41b7de499f63330079def0a5993d2a60d24e6f5191aaed3fd79d0fff97d2) ürettiğini göstermektedir. Burada da yine mesajın sonuna 1 eklendiğinde geriye uzunluk için 64 bitlik alan kalmayacağından iki kere özetleme yapılmıştır.

2.3.6 512’den Büyük Uzunluklu Mesaj İçin Simulasyon Sonucu



Şekil-11 :
“GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusu” Mesajı İçin Simulasyon Sonucu

Simulasyon sonucu tasarımın “GebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusuGebzeYuksekTeknolojiEnstitusu” mesajı için doğru olan özet değerini(aa113a610a89e792c572f2261abb09cc5d6007ad4cbbddd51aa40fd805bd8b2d) ürettiğini göstermektedir. Mesaj uzunluğu toplamda 232 bayt(byte) olduğundan özet değeri toplamda dört kere güncellenmelidir, bu da yukarıdaki grafikten görülebilir.

5. TASARIMIN SENTEZİ VE XILINX SPARTAN3E STARTER KIT UZERINDE GERCEKLENMESI

5.1 Giriş

Bu bölümde tasarımı gerçekleştirilen Sha256'nın Xilinx Spartan 3E Starter Kit üzerindeki gerçeklenmesi açıklanacaktır. Öncelikle örnek kit ile ilgili bilgi verilecek ve ardından da Sha çekirdeğinin, onu test edecek ve sonuçları gösterecek donanımın nasıl tasarlandığı anlatılacaktır.

5.2 Xilinx Spartan 3E Starter Kit

Xilinx Spartan 3E Starter Kit yaklaşık olarak 150 dolar civarında yurdışında satışı yapılan ve en yaygın olarak kullanılan FPGA geliştirme kartlarından biridir. Bu geliştirme kiti aşağıdaki bileşenlere sahip olarak verilmektedir:

Xilinx XC3S500E Spartan-3E FPGA
232 kullanıcı-I/O pinleri
320-pin FBGA paket
10,000' in üzerinde lojik hücre(logic cells)
Xilinx 4 Mbit Platform Flash konfigürasyon PROM
Xilinx 64-macrocell XC2C64A CoolRunner CPLD
64 MByte (512 Mbit) DDR SDRAM, x16 veri arayüzü, 100+ MHz
16 MByte (128 Mbit) paralel NOR Flash (Intel StrataFlash)
FPGA konfigürasyon depolama birimi (configuration storage)
16 Mbits SPI seri Flash (STMicro)
2 satırlı, 16 karakterli LCD ekran
PS/2 fare ve klavye portu
VGA görüntü(display) portu
10/100 Ethernet
İki tane 9-pin RS-232 portu (DTE- ve DCE)
Bord üzerinde USB-tabanlı FPGA/CPLD download/debug arayüzü
50 MHz saat osilatörü

Değişik özelliklere sahip ADC ve DAC bileşenleri
8 tane ayrık LED
Değişik özellikte çeşitli butonlar ve diğer özellikler.

Görüldüğü gibi bu bord genel anlamda prototip tasarımlar için oldukça çok sayıda özellik sunabilmektedir; zaten pazarlama stratejisi de daha çok prototip üretimler içindir.[9]

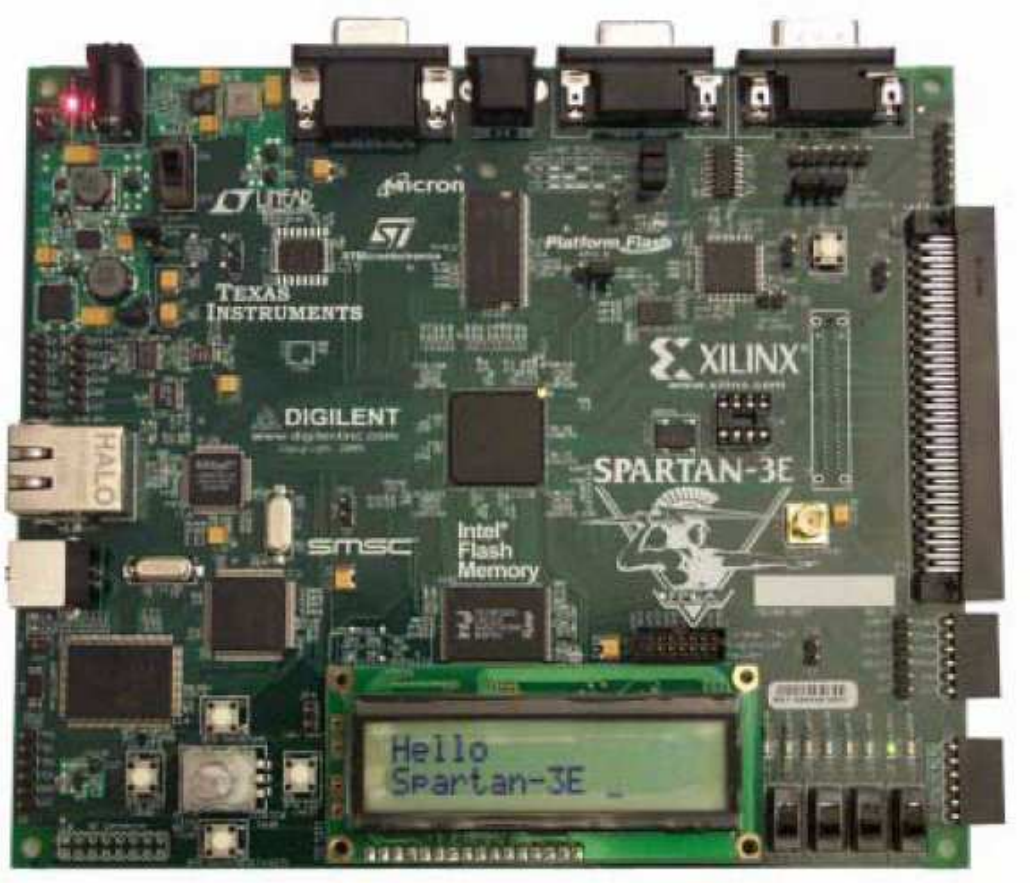
Digilent firması tarafından geliştirilen bu bordun resmi Şekil-12'de verilmiştir, merkezde bulunan çip Xilinx Spartan 3E FPGA ve XILINX ambleminin üzerinde bulunan çip ise Xilinx CoolRunner CPLD çipidir. Daha sonraki bölümde açıklanacağı gibi tasarımın testi için bu iki çipin de kullanılması gerekmektedir.

5.3 Sha256 Çekirdeğinin Spartan 3E FPGA ve CoolRunnerII CPLD Kullanılarak Kit Üzerinde Gerçeklenmesi ve Test Edilmesi

Bu bölümde 5.2' de verilen kit kullanılarak tasarımın nasıl teset edildiği gösterilecektir.

Sayısal tümdevre tasarım akışı içerisinde istenen fonksiyonu yerine getiren bir donanımın RTL kodu yazıldıktan ve testleri de yapıldıktan sonra bir ASIC haline dönüşmeden önce bir FPGA üzerinde prototip denemeler yapılarak, görece olarak çok çok pahalı olan ASIC üretimi öncesi düşük maliyette donanım testleri yapılmış olur. Prototip aşamasında çıkan sorunların bulunması ve çözümü hem zaman hem de maliyet açısından ASIC üretimi aşamasına kadar gelen bir tasarıma göre karşılaştırılamayacak derecede avantajlıdır. Tasarımın geliştirme, simulasyon ve sentez aşamalarında aşağıdaki araçlar kullanılmıştır:

RTL kod yazma ortamı→ Xilinx ISE 8.2i
Simulasyonlar → Cadence NCSIM
Sentezleyici → XST ve Synplify
Geliştirme Ortamı → ISE



Şekil-12 : Xilinx Sparten 3E Starter Kit

Geliştirilen Sha256 çekirdeği tek başına Spartan 3E FPGA'e sığdığı halde, bu tasarımın çalıştığını göstermek için gerekli olan diğer birimler de sentezlenip FPGA' ya gömülmek istendiğinde Spartan 3E' nin kapasitesini aşmaktadır. Kit üzerindeki gerçekleme için Sha256 dışında şu bileşenlere de ihtiyaç duyulmuştur:

1. Stim Generator (Test üreteci) : test için kullanılacak test vektörlerini üreten birim
2. Clock Divider (Saat bölücü) : Çalışmanın demo halinde sunulabilmesi ve sonuçların LCD ekrana yazılabilmesi için gerekli olan saat frekansını üreten birim.

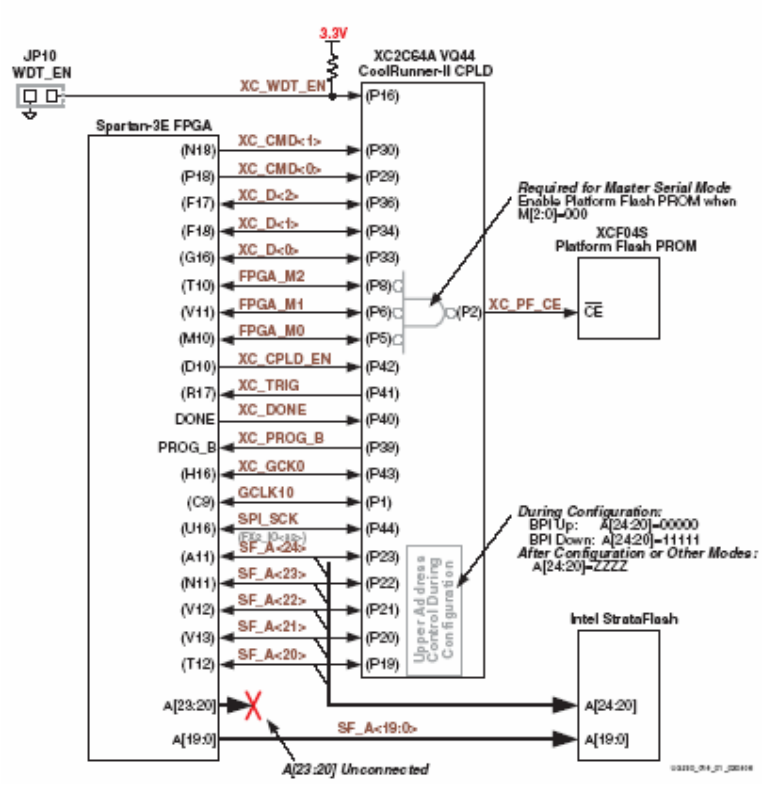
3. LCD Driver (LCD Sürücü) : Bord üzerinde bulunan LCD ekranı kontrol etmek ve bu ekrana yazmayı sağlamak için kullanılan birim.

4.Stim_Sha_Wrapper: Stim Generator, Sha256 birimlerini sarmalayan birim (Test ve Sha için ana block)

5. And Gate (Ve kapısı) : LCD için gerekli olan lcd_enable işaretini üreten basit kombinezonsal lojik kapı.

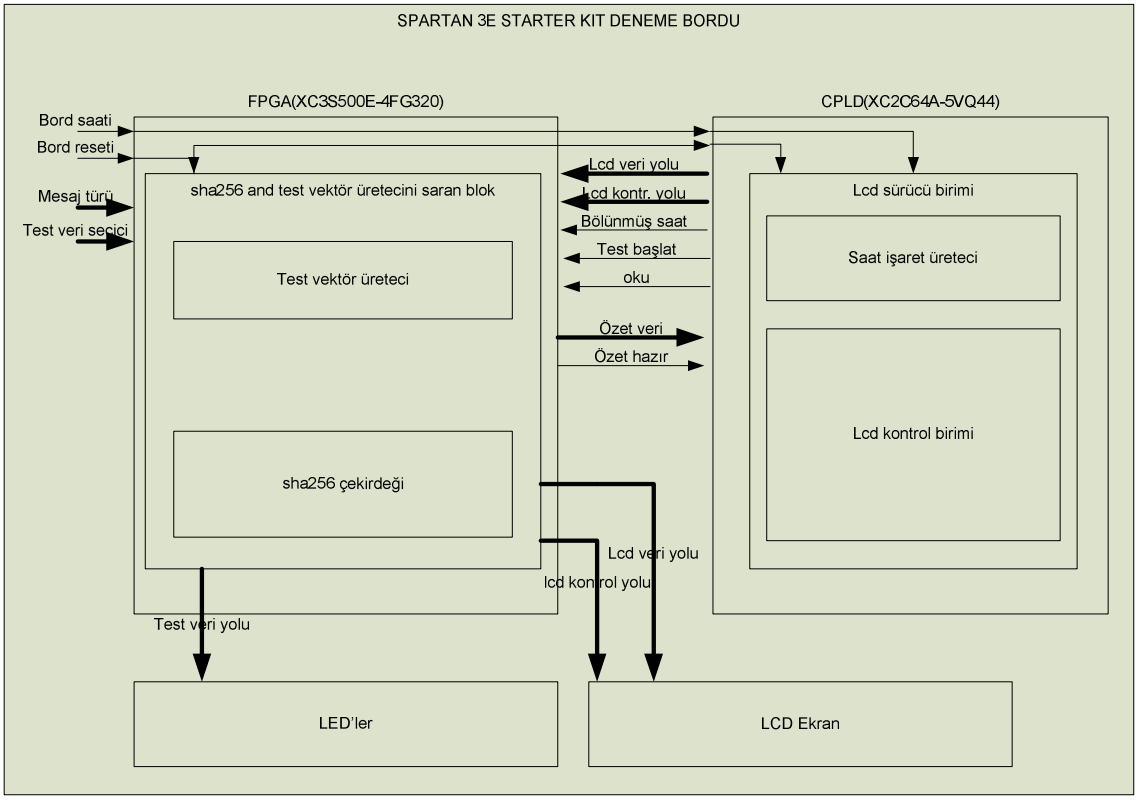
6. LCD Control Unit (LCD Kontrol Birimi) : LCD Driver ve Clock Divider birimlerinin kontrolünü sağlayan kontrol birimi.

Yukarıda sıralanan tüm birimler yalnız başına Spartan 3E FPGA çipine sığmadığından, bord üzerinde buluna ve FPGA'ya göre görece olarak daha düşük kapı kapasitesine sahip olan CoolRunnerII CPLD çipi de kullanılmıştır. Bu iki çipin bord üzerindeki bağlantısı Şekil-13' te verilmiştir.



Şekil-13 : Xilinx Sparten 3E Starter Kit Üzerindeki FPGA ve CPLD'nin bağlantısı

Görüldüğü gibi CPLD' ye giden işaretler öncelikle FPGA üzerinden geçmektedir. Bu yüzden örneğin CPLD üzerinde üretilecek saat işreti için gerekli olan referens saat işareti bord üzerinden doğrudan CPLD' ye bağlanamamış öncelikle FPGA' ya oradan da CPLD' ye bağlanmıştır. Böylece 50 MHZ' lik bord üzerindeki(on-the-board) saat önce FPGA' ya oradan doğrudan CPLD' ye gitmiş, CPLD üzerinde Clock Generator tarafından bölünerek gerek FPGA gerekse de CPLD' nin ihtiyaç duyduğu saat işareti üretilmiştir. FPGA' nın bu saat işareti CPLD' den tekrardan FPGA' ya bağlanmıştır. Diğer bazı işaretler için de benzer durumlar sözkonusu olmuştur. Şekil-13 FPGA ve CPLD'den oluşan donanımın sistem seviyesi gerçeklenmesini göstermektedir.



Şekil-14 : FPGA ve CPLD' den Oluşan Donanımın Sistem Seviyesi Gerçeklenmesi

Yukarıdan da görüldüğü gibi CPLD temelde , saat üretimi ve LCD' ye yazma işlemlerini gerçekleştirirken; FPGA de test vektör üretimi ve Sha256 hesaplamasını

gerçeklemektedir. Ayrıca çeşitli işaretlerin gözlenebilmesi için bunlar bord üzerindeki LED' lere de bağlanmıştır.

Tablo-3 CPLD için kullanılan kaynakları, Tablo-4 ise FPGA için kullanılan kaynakları göstermektedir, bu rapor doğrudan ISE geliştirme ortamından alınmıştır. Görüldüğü gibi Sha256 çekirdeği ile Stim Generator FPGA çipinin kaynaklarını büyük çoğunluğunu kullanmışlardır

Macrocell Kullanımı	Pterms Kullanımı	Yazmaç Kullanımı	Pin Kullanımı	Fonksiyonel Blok Giriş Kapı Kullanımı
51/64 (80%)	109/224 (49%)	47/64 (74%)	17/33 (52%)	122/160 (77%)

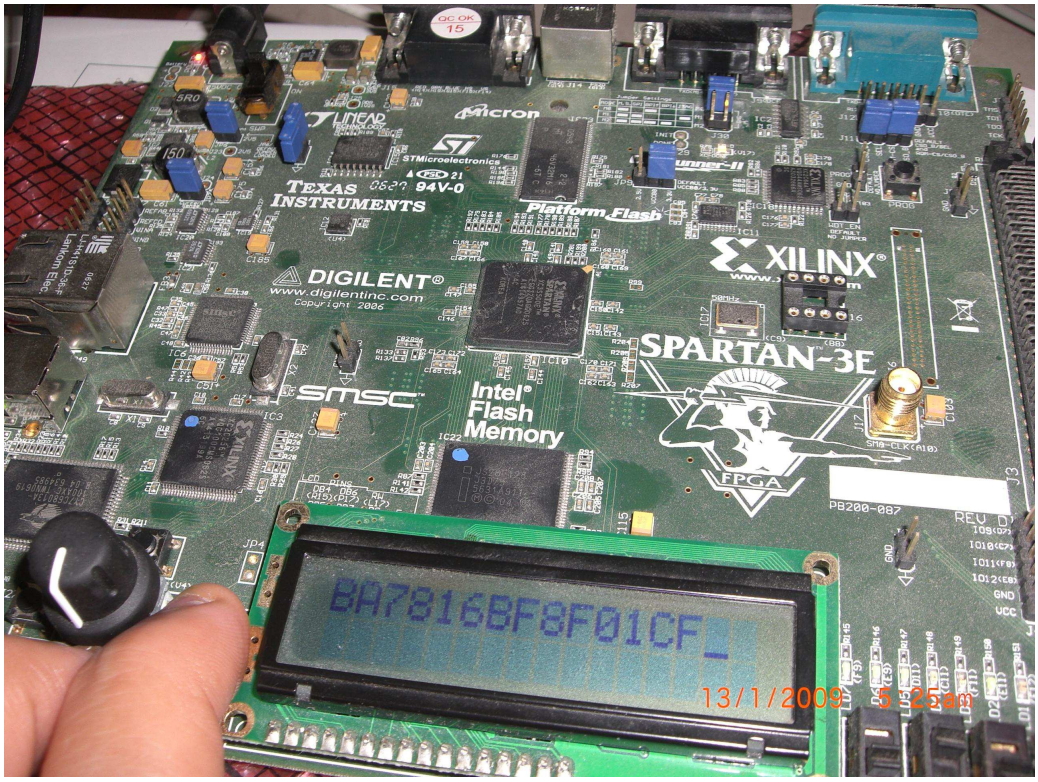
Tablo-3 : CPLD Kaynak Kullanım Raporu.

Araç Kullanım Özeti			
Lojik Kullanım	**Kullanılan**	**Kullanılabilir**	**Kullanım**
Dilim flip flop sayısı	1,716	9,312	18%
4 girişli LUT sayısı	8,757	9,312	94%
Lojik dağılım			
Lojik yerleştirilmiş dilim sayısı	4,598	4,656	98%
Doğrudan ilişkili lojik içeren dilim sayısı	4,598	4,598	100%
İlişkili olmayan lojik içeren dilim sayısı	0	4,598	0%
4 girişli toplam LUT sayısı	8,829	9,312	94%
Lojik olarak kullanım sayısı	8,757		
Geçiş olarak kullanım sayısı	72		
Bağlanmış giriş/çıkış block olarak kullanım sayısı	40	232	17%
GCLK sayısı	1	24	4%
Tasarım için toplam eşdeğer kapı sayısı	85,731		
Giriş/çıkış blokları için kullanılan ek JTAG kapı sayısı	1,920		

Tablo-4 : FPGA Kaynak Kullanım Raporu.

Şekil-15 gerçeklenen FPGA, CPLD sisteminin "abc" mesajınının özetini almasını ve LCD' ye yazma işlemi başladıktan sonraki durumunu göstermektedir. Şekilden de görüldüğü gibi LCD' ye "abc"' nin özet değeri olan

ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad değeri LCD Driver birimi tarafından yazılmaya başlanmıştır.



Şekil-15 : FPGA ve CPLD' den Oluşan Donanımn "abc" Mesajının Özetini Doğru Şekilde Aldığını Gösteren Resim

6. SONUÇLAR

6.1 Giriş

Bu bölümde ilk olarak tasarımı gerçekleştirilen ve anında(on-the-fly) özetleme işlemi yapabilen Sha256 ile, bu işlemi 64 adımda gerçekleştirebilen ve bir el sıkışma protokolüne ihtiyaç duyan referens bir tasarım karşılaştırılacak, ardından da kısaca tez özetlenecektir.

6.2 Özetleme İşlemini 64 Adımda Gerçekleştirebilen Bir Referans Tasarım İle Karşılaştırma Sonuçları

Karşılaştırma için Synopsys firmasının Design Compiler adlı sentezleyici aracı(tool) ve STMicroelectronics firmasının 65 nm CMOS teknolojisi kullanılmıştır. Sentez yapılırken her iki tasarım için de Design Compiler aracının öntanımlı(default) optimizasyon parametreleri kullanılmış, dolayısıyla her iki tasarım için de sentez aynı koşullar altında yapılmaya çalışılmıştır.

Aşağıdaki Tablo-5'te sentez öncesi verilen saat frekansları ve sentez sonrası elde edilen alan ve güç değerleri verilmiştir.

	Saat frekansı	Toplam Alan	Güç Tüketimi	
			Dinamik	Statik
Tezde gerçeklenen tasarım	41,6MHz	135,505um^2	1,07445mW	7,3902uW
Referans Tasarım	111MHz	67,145um^2	1,0022mW	3,6939uW
Oran	2,66	2,01	1,07	2

Tablo-5 : Tezde Gerçeklenen Tasarım ile Referans Tasarımın Karşılaştırılması

Tablo-5'tan görüldüğü gibi iki tasarım arasındaki performans(saat frekansı) oranı 2.67, toplam alan oranı 2,01 ve güç oranları sırasıyla 1.07 ve 2 olmuştur.

Tasarımlar arası karşılaştırma için toplam alan ve güç oranları tablodaki değerler kullanılarak verilebilir ancak performans için doğrudan saat frekansı yerine iş çıkarma yeteneğine(throughput) bakmak daha gerçekçi bir yaklaşım olacaktır.

Bölüm 3.2’ de her iki tasarım için de özetleme akışının aşağıdaki gibi olacağı verilmişti, görüldüğü gibi birinci yaklaşım iş çıkarma yeteneği bakımından ikiciye göre daha yüksek olacaktır. Aşağıda her bir adım için toplam kaç saat işareti gerektiği de yazılmıştır. Bu bilgi iş çıkarma yeteneğinin hesaplanmasında kullanılacaktır.

Anında(on the fly) bir yaklaşım olması durumunda akış şu sekilde olacaktır:

→başla
→masajı oku.....16 saat işareti
→ mesajı oku + özet hesapla .....16 saat işareti
→ mesajı oku + özet hesapla.....16 saat işareti
.
.
.
→ mesajı oku + özet hesapla.....16 saat işareti
→ özet hesapla.....16 saat işareti
→özeti dışarıya aktar.....8 saat işareti

Anında(on the fly) bir yaklaşım olmadığını düşündüğümüzde ise akış şu sekilde olacaktır:

→başla
→masajı oku.....16 saat işareti
→ özet hesapla .....64 saat işareti
→ mesajı oku.....16 saat işareti
.
.
.
→ özet hesapla.....64 saat işareti

→özeti dışarıya aktar.....8 saat işareti

Tablo-5' te verilen saat frekans oranı ve yukarıda verilen akış işleminde kullanılan saat işaretleri sayıları kullanılarak N tane 512-bitlik bir mesaj için her iki durumda da geçmesi gereken zaman miktarları ve oranları şu şekilde hesaplanabilir.

Tezde gerçeklenen tasarım için : (16 + (N x 16) + 16 + 8) x (1/41,6 us)...45

Referans tasarım için : (16 + (N x 80) + 64 + 8) x (1/111 us)...46

$$Oran = \frac{(16 + (\mathrm{N\,x\,16}) + 16 + 8)\ \mathrm{x}\ (1/41{,}6\ \mathrm{us})}{(16 + (\mathrm{N\,x\,80}) + 64 + 8)\ \mathrm{x}\ (1/111\ \mathrm{us})} = \frac{(40 + (\mathrm{N\,x\,16}))\ \mathrm{x}\ (24\ \mathrm{ns})}{(88 + (\mathrm{N\,x\,80}))\ \mathrm{x}\ (9\ \mathrm{ns})} \quad ...47$$

Aşağıda Tablo-6' da N' nin çeşitli değerleri için her iki durumda geçmesi gereken zamanlar ve bunlar arasındaki oranlar verilmiştir.

N	Tez Taarım(ns)	Referans(ns)	Oran
1	1344	1512	0,888889
2	1728	2232	0,774194
3	2112	2952	0,715447
4	2496	3672	0,679739
5	2880	4392	0,655738
6	3264	5112	0,638498
7	3648	5832	0,625514
8	4032	6552	0,615385
9	4416	7272	0,607261
10	4800	7992	0,600601
11	5184	8712	0,595041
12	5568	9432	0,590331
13	5952	10152	0,586288
14	6336	10872	0,582781
15	6720	11592	0,57971
16	7104	12312	0,576998
17	7488	13032	0,574586
18	7872	13752	0,572426
19	8256	14472	0,570481
20	8640	15192	0,56872
21	9024	15912	0,567119
22	9408	16632	0,565657
23	9792	17352	0,564315
24	10176	18072	0,563081
25	10560	18792	0,561941

Tablo-6 : 512-bitlik blok sayısı olan N' nin değerlerine göre her iki tasarımın özet alması için geçecek toplam zaman ve bunların oranları

Tablo-6'dan görüldüğü gibi N değeri arttıkça oran küçülmektedir.Çok büyük N değerleri için eşitlik 47' nin 0,53 değerine yakınsadığı görülebilir. O halde tezde gerçeklenen tasarımın iş çıkarabilme yeteneğinin referans tasarımın iş çıkarabilme yeteneğine oranı 1/0.53 = 1,88'dir...48

Bu da anında özetleme yapabilen tasarımın sonuç olarak daha çok iş çıkarabileceği anlamına gelir, Tablo-7'deki veriler iş çıkarabilme yeteneği oranını ve toplam özetleme zamanlarını vermektedir.

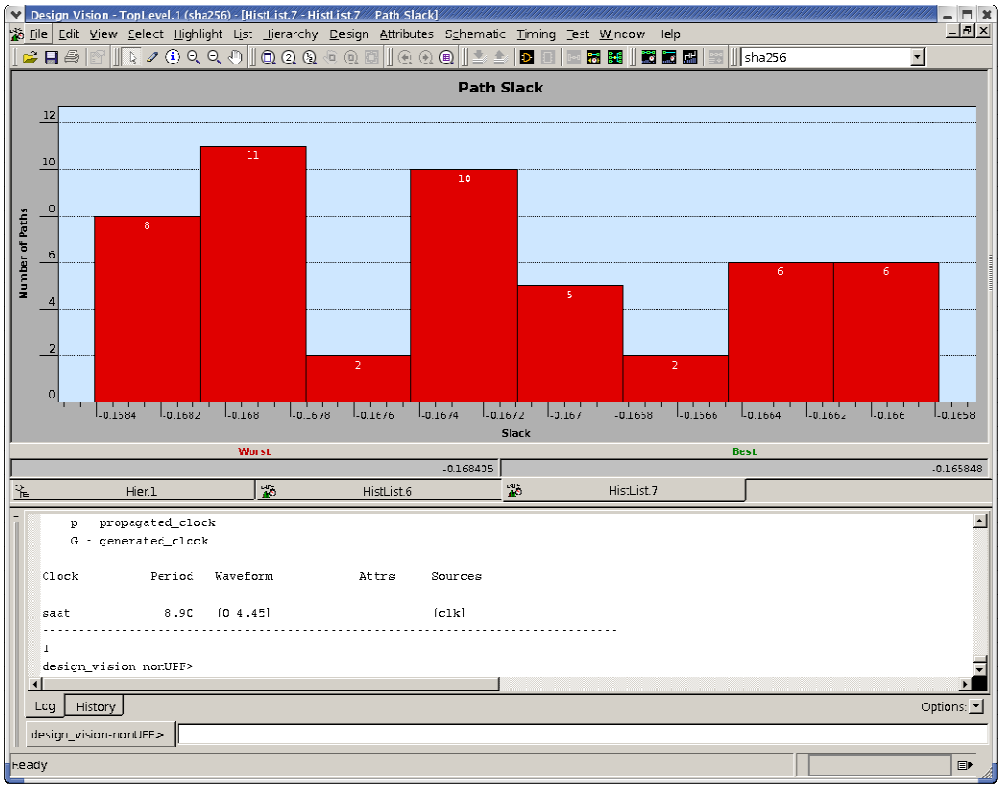
	Büyük N değerleri için toplam özetleme zamanı	İş Çıkarabilme Yeteneği Oranı
Tezde gerçeklenen tasarım	N x 16 x 24 ns	1.88
Referans Tasarım	N x 80 x 9 ns	

Tablo-7 : Tezde Gerçeklenen Tasarım ile Referans Tasarımın İş Çıkarabilme Yetenekleri Açısından Karşılaştırılması

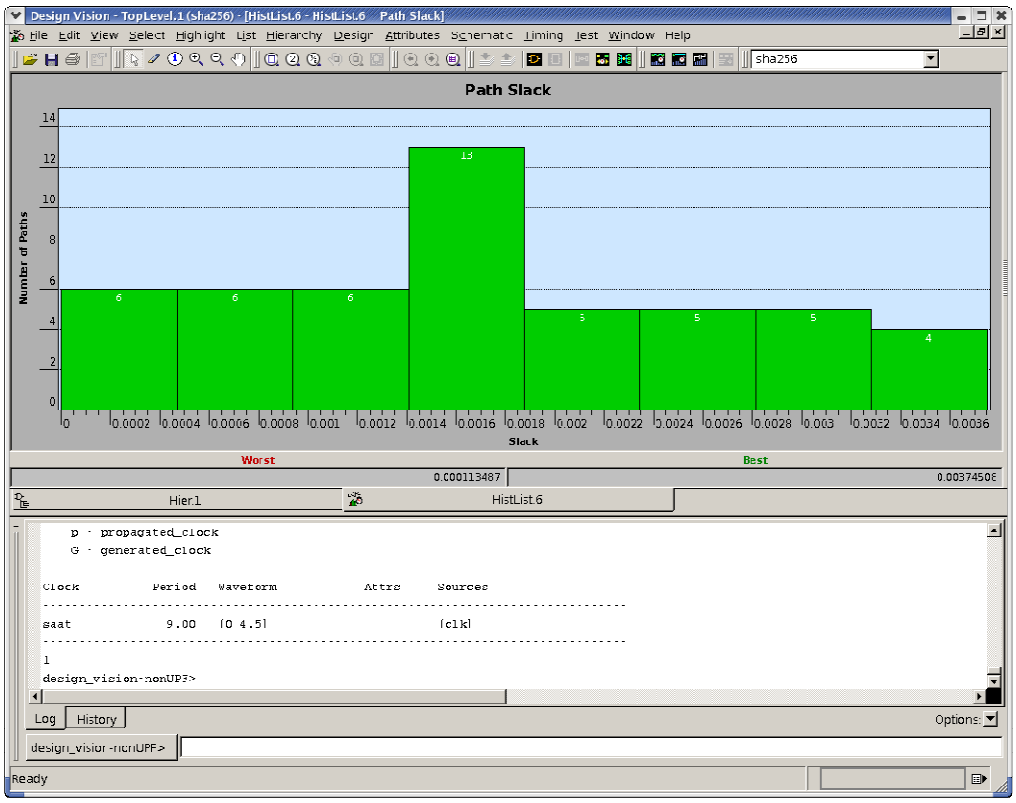
Şekil-16, referans tasarımın 8.9ns' lik saat işareti uygulandığında sentezinin başarılı olarak yapılamadığnı göstermektedir. Şekil-17 ise referans tasarımın 9 ns' lik saat işareti için sentezinin başarılı olarak yapılabildiğini göstermektedir.

Şekil-18, tezde tasarımı yapılan Sha256' nın 23ns'lik saat işareti için sentezinin yapılamadığını, Şekil19 ise 24ns' lik saat işaresi için ise sentezinin başarılı bir şekilde yapılabildiğini göstermektedir.

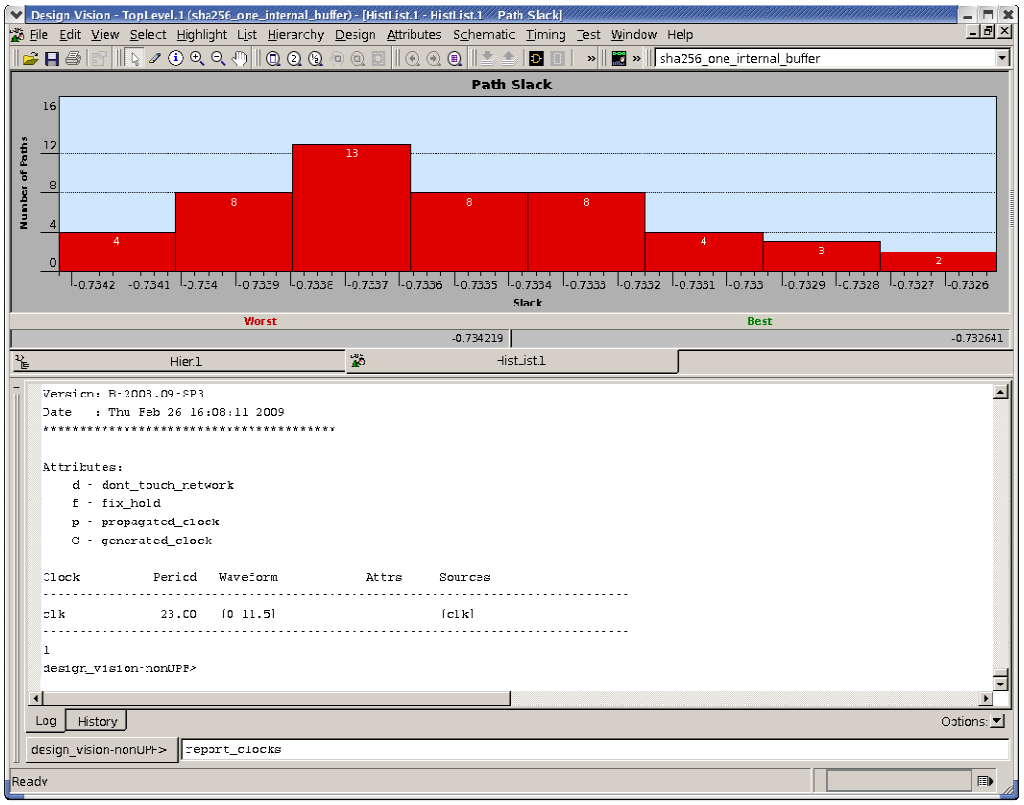
Bu sonuçlar Design Compiler aracı kullanılarak alınmış STA sonuçlarını göstermektedir. Benzer şekilde alan ve güç tüketimi raporları da yine aynı araç kullanılarak verilmiştir.



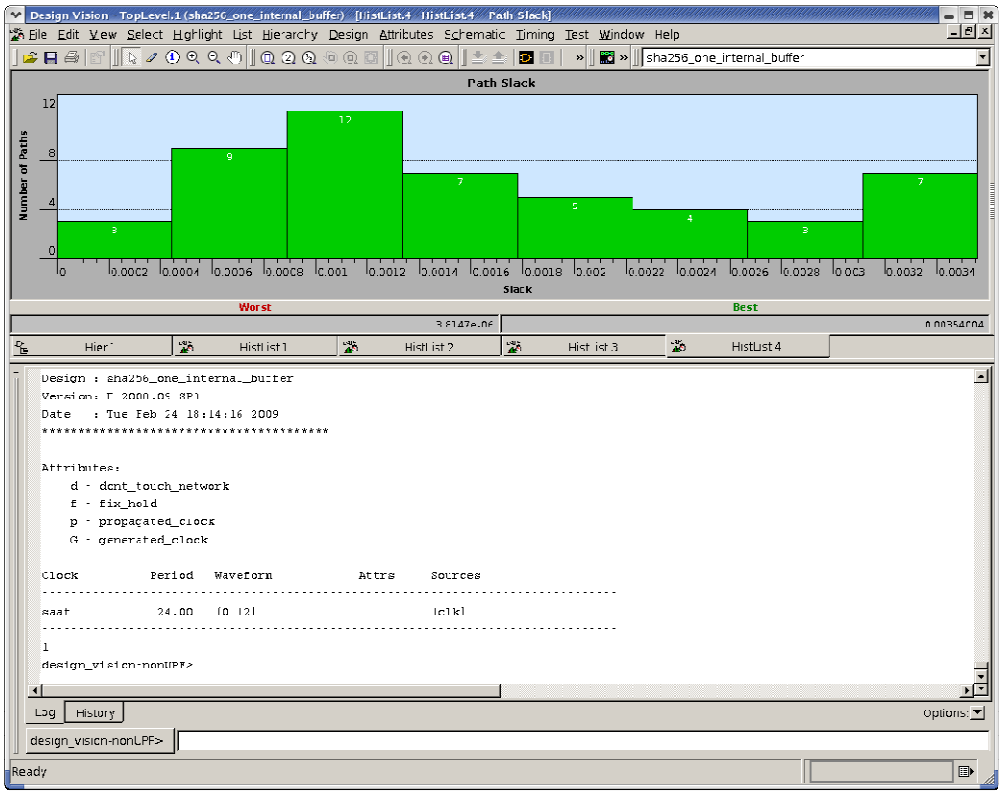
Şekil-16 : Referans Tasarım 8.9 ns'lik Saat Periyodu İçin Sentezlenememiştir.



Şekil-17 : Referans Tasarım 9 ns'lik Saat Periyodu İçin Sentezlenebilmiştir.



Şekil-18 : Tezdeki Tasarım 23 ns'lik Saat Periyodu İçin Sentezlenememiştir.



Şekil-19 : Tezdeki Tasarım 24 ns'lik Saat Periyodu İçin Sentezlenebilmiştir.

6.3 Tezin Özeti ve Geliştirmeye Açık Yönleri

Bu çalışmada Sha256 kriptografik şifreleme algoritmasının Verilog HDL dili kullanılarak anında(on the fly) özetleme yapabilecek bir donanım tasarımı gerçekleştirilmiştir. Böylece merkezi işlem birimi veya host ile özetleme yapacak yardımcı işlem birimi(co-processor) arasında bir el sıkışma(hand shaking) protokolüne olan gereksinim ortadan kaldırılmıştır. Dolayısıyla aynı anda hem merkezi işlem biriminden okuma yapılmakta ve daha önceden okunarak depolanmış(buffered) veri de özetlenebilmektedir. Böylece okuma için kullanılan saat işaretlerinde özetleme de yapılabilmekte ve özetleme birimi boş(idle) bekletilmemektedir .

Tasarım genelinde kaydırma(shifting) tabanlı bir yaklaşım kullanılmıştır. Bu sayede donanım içerisinde çok büyük gecikmelere neden olabilecek çoğullayıcı kapıları(Multiplexer)' ndan kaçınılmış, böylece donanım içerisindeki kombinezonsal lojik seviye en aşağı düzeyde tutulmaya çalışılmıştır.

Bu tasarımın geliştirilmeye ihtiyaç duyabilecek en önemli noktası döngü sayısı 16' ya indirildiği için, ardarda seri olarak bağlanmış olan toplayıcılardır. Eğer bu toplayıcıların yüksek performansta çalışmasına yönelik bir en iyileştirme(optimizasyon) yapılırsa, tasarımın toplam performansı da artacaktır.

Sonuç olarak toplam alanda 2.01, toplam dinamik güç tüketiminde 1.07, toplam statik güç tüketiminde 2 kat kötüleşme yaşanmakla birlikte, toplam iş çıkarabilme yeteneğinde 1.88 kata kadar iyileştirme sağlanmış ve en önemlisi anında(on-the-fly) özetleme yapabilecek bir tasarım gerçeklenmiştir.

KAYNAKLAR DİZİNİ

[1] Knuth, Donald, ‘The Art of Computer Programming’, Volume 3, Sorting and Searching, pp. 506–542, 1973

[2] RSA Laboratories Inc., ‘RSA Laboratories' Frequently Asked Questions About Today's Cryptography’,Chapter 2 Cryptography, Version 4.1, 2000

[3] Yuliang Zheng, Josef Pieprzyk, Jennifer Seberry ‘HAVAL,a one-way hashing algorithm with variable length of output’, Advances in Cryptology AUSCRYPT'92, Lecture Notes in Computer Science, Vol.718, pp.83-104, Springer-Verlag, 1993

[4] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu, ‘Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD’, Cryptology ePrint Archive, Report 2004/199, 2004

[5] Hans Dobbertin, Antoon Bosselaers, Bart Preneel, ’RIPEMD-160: A Strengthened Version of RIPEMD’, http://homes.esat.kuleuven.be/~bosselae/ripemd160.html, Erişim Tarihi Ocak 2009

[6] Xiaoyun Wang, Andrew Yao and Frances Yao, ’ Finding Collisions in the FullSHA-1’, Crypto 2005 The 25th Annual International Cryptology Conference August 14-18, 2005 Santa Barbara, California, USA, 2005

[7] National Institute of Standards and Technology Information Technology Laboratory, ’ Secure Hash Standard (SHS)’, FIPS PUB 180-3, 2008

[8] ISO, ’ Information technology -- Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions’, ISO/IEC 10118-3, 2004

[9] http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm, Erişim Tarihi Ocak 2009

[10] http://www.paulschou.com/tools/xlate/, Erişim Tarihi Ocak 2009

ÖZ GEÇMİŞ

Adı-Soyadı **:** İnan Erdem

Doğum Tarihi/Yeri **:** 20.10.1979 / Iğdır

Eğitim

İlköğretim **:** Yukarı Aratan Köyü İlköğretim Okulu, Iğdır, 1990

Ortaokul **:** Aralık Lisesi, Iğdır, 1993

Lise **:** Bağcılar Lisesi, İstanbul, 1996

Lisans **:** İstanbul Teknik Üniversitesi, Elektronik ve Haberleşme Mühendisliği Bölümü, İstanbul, 2005

Yüksek Lisans **:** Gebze Yüksek Teknoloji Enstitüsü, Elektronik Mühendisliği Bölümü, Gebze, 2009

Sürekli Adres **:** Kemalpaşa Mah. Namıkkemal Cad. 14/2 Sok No:5/6 Bağcılar / İstanbul

Telefon **:** 0 505 4019177

E-Posta **:** inanerdem@gmail.com

EK-1. DONANIM TANIMLAMA DİLLERİ VE VERILOG HDL DİLİ

Donanım tanımlama dili(Hardware Description Language) genel olarak bir elektronik devrenin davranışını bir yazı formatında modelleyen ve çoğu zaman bir standart tarafından ortaya konmuş yapı taşlarından(türler, sabitler, ifadeler vs.) oluşan kurallar bütünüdür. Donanım tanımlama dili kullanılarak hem amaçlanan donanım tasarlanabilir hem de onun testleri yapılabilir.

Bir donanım tanımlama dili ile yazılım dilini birbirinden ayırt edebilmek için şu örnek verilebilir:

Çok düşük seviyeli makina dilinden yüksek seviyeli bir uygulama geliştirme diline kadar tüm yazılım dilleri ile yazılmış olan programlar üzerlerinde çalışacakları bir donanıma muhtaç durumdadırlar. Bu donanımın sunduğu komut setinin dışına çıkmaları mümkün değildir ve her bir komut donanımın belirlediği saat işareti kadar sürede çalışmak zorundadır.

Donanım tanımlama dilini kullanarak geliştirdiğimiz programlar ise doğrudan donanıma dönüşecek nihai program olduğundan, çalışmak istediğimiz donanımı kendimizin tasarlaması mümkün olacaktır. Bu durumda örneğin bir işlemciye yapması gerekenleri söylemek yerine; doğrudan işlemciyi tasarlayan konumunda olmuş olacağızdır.

Yukarıda verilen açıklamalardan sonra bir donanım tanımlama dilinin hedef olarak donanımı modellemesi sebebiyle yazılım diline göre şu iki temel farklılığa sahip olabileceğini söyleyebiliriz.

1. İçerisinde zaman bilgisinin bulunması gerekir böylece tasarlayacağımız donanımın örneğin maksimum çalışma frekansını modelleyebilir ve tasarımımızı buna göre yapabiliriz.

2. Eş zamanlı(concurrent) çalışabilecek kod yazmayı sağlaması gerekir. Her ne kadar bir yazılım programı yukarıdan aşağıya doğru bir akışa sahip ve satır satır ele alınıyorsa da aynı şeyi donanımdan bekleyemeyiz çünkü bir donanım içerisinde farklı görevlere ship kısımlar aynı anda paralel olarak çalışmak zorunda kalabilirler. Örneğin bir haberleşme donanımının gerçek zamanlı olarak hem veri alması hem de veri göndermesi ancak böyle bir eş zamanlı çalışabilmeyle mümkün olacaktır.

Verilog dili de bu donanım tanımlama dillerinden en yayın olarak kullanılanlardan biridir ve VHDL dili ile beraber en çok kullanılan 1. veya 2. dildir denebilir. Bu kısımda Verilog dilini anlatmak yerine bir DFF(D-tipi flip flop)' in Verilog ile nasıl yazılacağı gösterilip önemli kısımları kısaca verilecektır.

Aşağıdaki kod, asenkron resetlenebilir ve saatin yükselen kenarında tetiklenen basit bir DFF' in Verilog kodudur:

```
module DFF(clk, rst, D, Q);
input clk, rst, D;
output Q;
reg Q;
        always @ (posedge clk or negedge rst)
        begin
                if(!rst)
                begin
                        Q <= 1'b0;
                end
                else
                begin
                        Q <= D,
                end
        end
endmodule
```

Yukarıdaki kod kısaca şöyle açıklanabilir,

Verilog ile yapılan tasarımlar sırasıyla module ve endmodule anahtar kelimeleri ile başlar ve biter

Ardından donanımın giriş ve çıkış kapıları input ve output anahtar kelimeleri ile belirlenir. Eğer çıkış kapıları yazmaçlı ise(registered) bunlar reg anahtar kelimesi ile belirtilirler

Saatli bir prosedür, yani ardışıl bir devreye karşılık gelecek kod genellikle always @ ifadesi ile başlar ve ardışıl devrenin saat ve reseti (posedge clk or negedge rst) gibi bir ifade ile verilir. Yukarıdaki örnekte, flip flop' un saatin yükselen kenarında tetikleneceği belirtilmiş ve always bloğunun gövdesi içerisinde resetlemenin de saatten bağımsız olarak asenkron olacağı ifade edilmişti.r

Aşağıda ise aynı D flip flop' un senkron resetlenmesi durumunda always bloğunun nasıl olacağı verilmiştir, görüldüğü gibi tek fark reset işaretinin düşen kenarının saatin yükselen kenarında kontrol edilmesidir:

```
...
always @ (posedge clk or negedge rst)
        begin
                if(!rst)
                begin
                        Q <= 1'b0;
                end
                else
                begin
                        Q <= D,
                end
        end
...
```

Son olarak Verilog dilinin görece olarak daha küçük tasarımlar için kullanılmasının hızlı simulasyon imkanı veridiği için uygun olduğu ve söz dizim

kurallarının(syntax) diğer dillere göre C programlama diline daha çok benzediği söylenebilir. Böylece yazılım deneyimine sahip insanların donanım tasarım öğrenme aşamasında Verilog dilini rahatlıkla kullanabileceğini söyleyebiliriz.

EK-2. SHA-256 İÇİN KOD TASLAĞI

1. Önişlemenin yapılması

```
h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19
```

2. Sabitlerin atanması:

```
k[0..63] :=
   0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b,
0x59f111f1, 0x923f82a4, 0xab1c5ed5,
   0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74,
0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
   0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f,
0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
   0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3,
0xd5a79147, 0x06ca6351, 0x14292967,
   0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354,
0x766a0abb, 0x81c2c92e, 0x92722c85,
   0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819,
0xd6990624, 0xf40e3585, 0x106aa070,
   0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3,
0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
   0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa,
0xa4506ceb, 0xbef9a3f7, 0xc67178f2
```

2. Özet değerinin hesaplanması:

Mesajı 512-bitlik parçalara(chunk) böl ve herbir parça için aşağıdaki işlemleri yap:

Üzerinde işlem yapılan parçayı 32-bitlik 16 adet kelimeye(w) böl ve diğer 48 kelimeyi hesapla:

```
    for i from 16 to 63
        s0=(w[i-15] rotr 7) xor (w[i-15] rotr 18) xor (w[i-15] >> 3)
```

```
        s1=(w[i-2] rotr 17) xor (w[i-2] rotr 19) xor (w[i-2] >> 10)

        w[i] := w[i-16] + s0 + w[i-7] + s1
```

Ara özet değerlerinin atamasını yap:

```
    a = h0
    b = h1
    c = h2
    d = h3
    e = h4
    f = h5
    g = h6
    h = h7
```

Ana çevirimi(main loop) yap:

```
    for i from 0 to 63
        s0 = (a rotr 2) xor (a rotr 13) xor (a rotr 22)

        maj = (a and b) xor (a and c) xor (b and c)

        t2 = s0 + maj

        s1 = (e rotr 6) xor (e rotr 11) xor (e rotr 25)

        ch = (e and f) xor ((not e) and g)

        t1 = h + s1 + ch + k[i] + w[i]

        h = g
        g = f
        f = e
        e = d + t1
        d = c
        c = b
        b = a
        a = t1 + t2
```

ara özet değerini ata

```
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
    h4 := h4 + e
    h5 := h5 + f
    h6 := h6 + g
    h7 := h7 + h
```

Yukarıdaki adımlar tüm bloklar için tekrarlandıktan sonra son olarak nihai özet değerini(hash value) son ara özet değerlerini birleştirerek(appending) bul:

```
özet = h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7
```