KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ
ANABİLİM DALI

YÜKSEK LİSANS TEZİ

RISC-V TABANLI AES HIZLANDIRILMIŞ KIRMIK ÜSTÜ SİSTEMİN FİZİKSEL TASARIMI

EMRE GAYİR

KOCAELİ 2021

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

RISC-V TABANLI AES HIZLANDIRILMIŞ KIRMIK ÜSTÜ SİSTEMİN FİZİKSEL TASARIMI

EMRE GAYİR

Prof. Dr. Ali TANGEL		
Danışman,	Kocaeli Üniversitesi	.....
Dr. Öğr. Üyesi Anıl ÇELEBİ		
Jüri Üyesi,	Kocaeli Üniversitesi	.....
Dr. Öğr. Üyesi Oktay AYTAR		
Jüri Üyesi,	Bolu Abant İzzet Baysal Üniversitesi	.....

Tezin Savunulduğu Tarih: 24.06.2021

ÖNSÖZ VE TEŞEKKÜR

Akademik kariyerimde her zaman örnek aldığım, bilgi ve birikimini benden esirgemeyen değerli hocam, sayın Prof.Dr. Ali TANGEL'e,

Ülkemizde önemli çalışma alanlarına sahip FPGA ve ASIC konusunda bana çalışma fırsatı veren, her konuda her zaman desteklerini esirgemeyen değerli çalışma arkadaşlarım İnan Erdem ve Gökhan Işık'a,

Beni bugünlere getiren, desteklerini esirgemeyen ve her zaman arkamda duran canım annem Perihan Gayir ve canım babam Şaban Gayir'a çok teşekkür ederim.

Haziran - 2021 Emre GAYİR

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

GaAs : Galyum-Arsenik
MHz : Megahertz
nm : Nano metre
SiGe : Silisyum-Germanyum
µm : Mikro metre

Kısaltmalar

ASIC : Aplication Specific Integrated Circuit (Uygulama Özel Tümleşik Devre)
BJT : Bipolar Junction Transistor (Çift Kutuplu Eklemli Transistör)
CISC : Complex Instruction Set Computer (Genişletilmiş Komut Mimarisi)
CMOS : Complementary Metal Oxide Semiconductor (Bütünleyici Metal Yarı İletken)
FPGA : Field Programable Gate Array (Porgramlanır Kapı Dizisi)
I2C : Inter-Integrated Circuit (Entegre Devre)
I2S : Inter-IC Sound (Gömülü Çip Sesi)
LB : Load Byte (Bayt Yükle)
LH : Load Half (Yarım Kelime Yükleme)
LW : Load Word (Kelime Yükleme)
RISC : Reduced Instruction Set Computer (Kısıtlanmış Komut Mimarisi)
RTL : Register Transfer Level (Hafıza Seviyesi Kodlama)
RV32C : RISC-V 32 bit Compressed (Sıkıştırılmış 32 bit RISC-V Mimari)
RV32F : RISC-V 32 bit Floating (Sıkıştırılmış 32 bit RISC-V Mimari)
RV32I : RISC-V 32 bit Integer (Doğal Sayı 32 Bit RISC-V Mimari)
RV32M : RISC-V 32 bit Multipication (ÇArpma 32 Bit RISC-V Mimari)
SB : Store Byte (Bayt Depolama)
SPI : Serial Peripheral Interface (Seri Aygıt Arayüzü)
SW : Store Word (Kelime Depolama)
UART : Universal Asyncronous Receiver Transmitter (Genelgeçer Eşzamansız Alıcı-Verici)

RISC-V TABANLI AES HIZLANDIRILMIŞ KIRMIK ÜSTÜ SİSTEMİN FİZKSEL TASARIMI

ÖZET

Günümüzde havacılık, uzay, otomotiv, savunma gibi bir çok alanda mikroelektronik alanındaki gelişmeler çok büyük önem arz etmektedir. Bu alanlarda her daim daha hızlı, daha az alan kaplayan ve daha az güç tüketimi ile çalışan Kırmık Üstü Sistem ihtiyacı ortaya çıkmaktadır. Bunlara ek olarak tüm bu isteklerin yanında, lisans maliyeti gibi durumları minimize etme ihtiyacı rekabetçi küresel ortamda sıkça karşılaşılan bir durumdur. Bu doğrultuda küresel pazara Kırmık Üstü Sistemler için tüm yüksek performans isterlerinin yanında açık kaynak kodlu, hiçbir lisans maliyeti olmayan komut satır kümeleri ortaya çıkmıştır. Tüm bu gelişmeler sadece büyük firmalar için değil, birçok küçük firmaların maliyetlerini azalttığı için yeni çalışma alanları doğurmuştur.

Bu çalışmada, açık kaynak kodlu RISC-V mimarisine sahip olan bir Kırmık Üstü Sistemin fiziksel tasarımı aşamaları gösterilmiştir. Literatürde de yer alan bu alandaki bazı çalışmalar incelenmiştir.

Kırmık Üstü Sistemin fiziksel tasarımı fabrikadan gelen kapı seviyelerine sentezleyici program sayesinde dönüştürülmüştür. Bu sentezlenmiş devre fiziksel tasarım aracıyla yerleşim şeması haline getirilmiştir. Elde edilen sonuçlar ve çalışmalara ek olabilecek öneriler çalışmanın sonucunda verilmiştir.

Anahtar Kelimeler: Fiziksel Tasarım, Kırmık Üstü Sistem, RISC-V Komut Satırı Kümesi.

PHYSICAL IMPLEMETATION OF AN AES ACCELERATED RISC-V SYSTEM ON CHIP

ABSTRACT

Today's microelectronics research and development has very significant role at field of aviation, defence, space and automotive, Those fields always need more speed, less area and less power working system at their system on chip architecture. Moreover, beside of those requirements lower cost is so much important in phase of developing an microelectronic product. This kind of low cost requirement has brought new open source and no cost instruction set architecure to the microelectronics industrie. All those kind of news bring new application domain to the not only big companies even to small companies.

This thesis shows physical implementation steps of an RISC-V based System On Chip. Some related articles are analyzed during this thesis.

RISC-V based System On Chip architecture was synthesized to the foundary libraries cells and macros using synthesis tool after that, this synthesized netlist was converted to pyhsical representation of cells and macros. At final stage, some results and suggestions are given.

Keywords: Physical Implementaiton, System on Chip, RISC-V Instruction Set Architecture.

GİRİŞ

Entegre devre ilk olarak 1949 yılında Alman mühendis Werner Jacoibi tarafından entegre yapısına benzer bir yapı ile sinyal yükseltici devre şeklinde tasarlanmıştır. Entegre devreler için istek tüm sistemin bir yonga üzerinde oluşturulmasıydı. Bu doğrultuda Texas Instrument firmasından Kilby bir alet tasarladı ve bunu "tamamen bütünleşmiş elektronik devrelerin tüm bileşenlerini barındıran yarı iletken maddelerin cismi" olarak tanımladı ve ilk müşterisi Amerika Birleşik Devletleri Hava Kuvvetleri oldu. Kilby, entegre devreleri bulduğu için Nobel Ödülü almıştır. Daha sonra Fairchild Semiconductor firmasından Robert Noyce Kilby'ın fikrini geliştirmiştir ve Kilby'ın tasarımında germanyumdan yapılmış olan kısım, Noyce'un tasarımında silikondan yapılmıştır. Günümüzde tümleşik devre üretiminde düşük maliyette ve büyük tümdevrelerin üretimine olanak sağladığı için silikon temelli CMOS kullanılır. Uygulama hedefinden daha hızlı ve düşük gürültülü sistemlerde SiGe, BJT, GaAs temelli teknolojiler de kullanılabilmektedir [1].

Kırmık Üstü Sistemler daima bünyelerinde merkezi işlem birimi, hafıza elemanı, giriş ve çıkış birimlerini içermektedir. Bunlara ek olabilecek grafik işleme birimi, sinyal işleme birimleri gibi fonksiyonel birimlerini de içerebilmektedir. Kırmık Üstü Sistemler geleneksel anakart tabanlı bilgisayar mimarisinin aksine tüm bu sistemleri tek bir tümleşik devre üzerinde bulundururlar. Bu fonksiyonel birimlerden işlem birimi olan kısım bir ya da daha fazla çekirdekten oluşmakta olup RISC ve CISC tasarımlarında olabilmektedir. RISC sistemi daha az alan kapladığı için tercih sebebi olarak öne çıkmaktadır. Hafıza birimi sistemin tüm veri ve komut kümesinin içeriğini de tutmaktadır. Giriş ve çıkış birimleri dış dünya ile veri alıp vermede kullanılan blok yapılarıdır. Kırmık Üstü Sistemleri içeren tüm yapılarda yüksek performans, daha az güç tüketimi ve daha küçük yarı iletken alanı kapsamaktadır. Kırmık Üstü Sistemler uygulama alanı olarak akıllı telefon, tablet, bulut teknolojileri, wifi teknolojisi ve nesnelerin interneti gibi birçok alanda hesaplama, sinyal işleme gibi işlemler için kullanılmaktadır [2].

Kırmık Üstü Sistemlerin işlem birimlerinde lisans maliyetinden kurtulabilmek için RISC-V açık kaynak kodlu, hiçbir lisans gereksinimi olmayan, kısıtlanmış komut satırı mimarisini temel almış yerlerde kullanılabilir. RISC-V orjinal olarak Kaliforniya Üniversitesinde geliştirilmiştir. İlk dönemlerde üniversite ve bilgisayar mimarisi araştırmalarını desteklemek için ortaya çıkmış olsa da, endüstride birçok firmanın Google, Nvidia, Microsemi, SiFive, IBM, AMD, Qualcom ve Western Digital vb. desteğiyle açık kaynak mimari standartı haline gelmiştir. Bu doğrultuda birçok firma RISC-V temelli donanım ve açık kaynak kodlu işletim sistemlerini duyurmuşlardır. Hiçbir lisans maliyeti gerektirmediği için çekirdek tasarımı konusunda yeni bir çağ açmış olup bu alanda çalışmak isteyen herkese öncülük etmektedir [3].

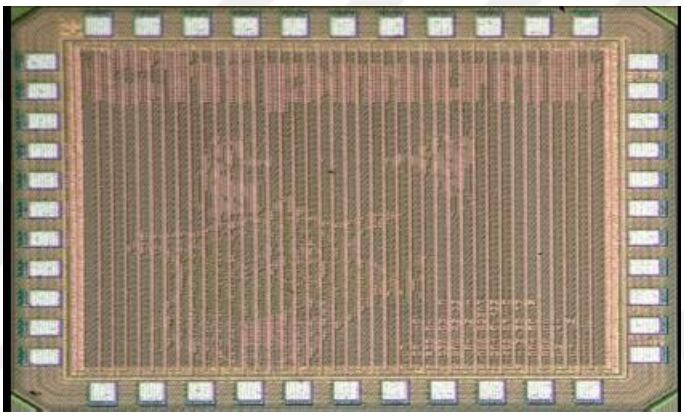
Kırmık Üstü Sistemin ve açık kaynak kodlu RISC-V mimarsinin birleştiği birçok sistem son dönemlerde ortaya çıkmıştır. Bunlardan biri Pulpino sistemidir. Bu sistem 32-bit RISC-V mimarisini hedef almış tek çekirdekli bir Kırmık Üstü Sistemdir. Pulpino Kırmık Üstü Sistemi, düşük güç tüketimli sinyal işleme uygulamalarını hedef alınarak geliştirilmiştir. Kırmık Üstü Sistemlerin olmassa olmazlarından çekirdek, hafıza elemanı ve dış dünya ile haberleşme imkanı sağlayan giriş çıkış birimleri bulunmaktadır. Bu sistem donanım kodları açık olduğu için uygulamamıza yönelik olarak yeni komut ekleme ya da yeni bir işlem bloğu sisteme eklenerek kullanılır [4].

Bu tezde Pulpino Kırmık Üstü Sistemin fiziksel tasarımı tasarımı yapılmıştır. Kırmık Üstü Sistem için öncelikli olarak ilgili veriler oluşturulmuştur. İkinci olarak sentezleme ve son olarak fiziksel tasarım işlemi yapılmıştır.

1. RISC-V MİMARİSİ

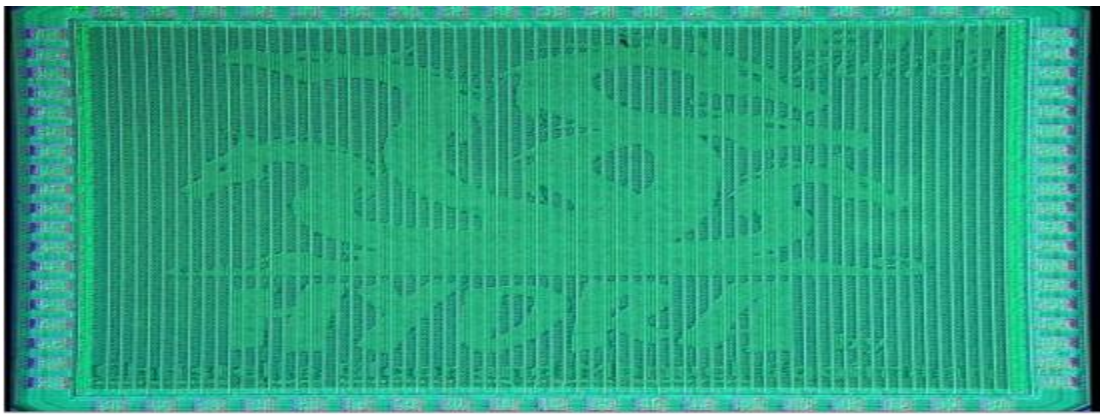
1.1. Giriş

RISC-V mimarisini temel alan birçok Kırmık Üstü Sistem, uygulamaya yönelik olarak özelliştirilmiş olup bu alandaki fiziksel gerçekleme çalışmaları farklı teknoloji frekans, farklı düğüm, farklı alan ve farklı voltaj seviyelerinde tasarımları yapılmıştır. Bu tasarımlardan bazıları aşağıda verilmiştir.



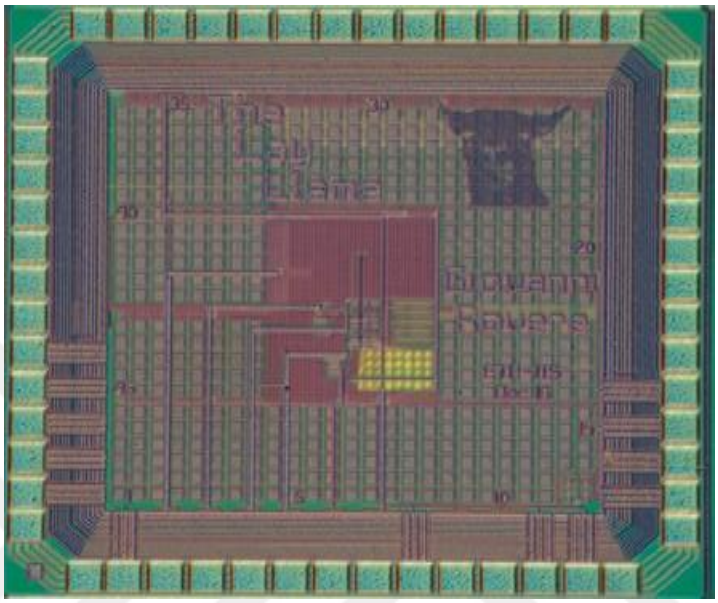
Şekil 1.1. Chipmunk Fiziksel Tasarım [5]

Şekil 1.1'de Chipmunk Kırmık Üstü sistemi verilmiştir. Chipmunk ses tanıma gibi akıllı saatlerde kullanılabilecek bir tasarıma sahiptir. UMC65 nm teknoloji kütüphaneleri kullanılmıştır. 100 MHz saat frekansında, 1,2 voltaj seviyesinde, yatay ve düşey eksende 1252 μm alan kaplamaktadır.



Şekil 1.2. Hydra Fiziksel Tasarım [6]

Şekil 1.2'de Hydra Kırmık Üstü sistemi verilmiştir. Hydra dönem projesi makine öğrenmesi alanını hedef alarak ortaya çıkmış bir tasarımdır. UMC65 nm teknoloji kütüphaneleri kullanılmıştır. 333 MHz saat frekansında, 1,2 voltaj seviyesinde, yatay ve düşey eksende 1875 µm alan kaplamaktadır.



Şekil 1.3. Lay_Llama Fiziksel Tasarım [7]

Şekil 1.3'de Lay_Llama Kırmık Üstü sistemi verilmiştir. Lay_Llama biyomedikal uygulamaları hedef alınmıştır. SMIC130 nm teknoloji kütüphaneleri kullanılmıştır. 1,2 voltaj seviyesinde, yatay ve düşey eksende 1240 µm alan kaplamaktadır.

Bunlara ek olarak Potato, Pulpino, Pulpisimo, CVA6 gibi farklı özelliklerdeki Kırmık Üstü Sistem özellikleri aşağıdaki tabloda verilmiştir.

Tablo 1.1. Farklı Kırmık Üstü Sistem Özellikleri

	Potato	Pulpino	Pulpisimo	CVA6
Çalışma Frekansı	50 MHz	40 MHz	10 MHz	50 MHz
İşetim Sistemi	Yok	Yok	Yok	Var
FreeRTOS	Yok	Var	Var	Yok
ISA Özelliği	RV32I	RV32IMAFC	RV32IMAFC	RV64IMAFDC
Arayüz	Wishbone	AXI4	AXI4	AXI4
Bellek Miktarı	128KB	64KB	512KB	DDR Desteği
ROM Miktarı	16KB	8KB	8KB	64KB

1.2. RISC-V Komut Satır Kümesi

RISC-V komut satırı kümesi endüstri, araştırma ekibi ve üniversitelerin iş birliğiyle ana katmanda temel komut kümesi olacak şekilde ve genişletilir modüler bir yapıdan oluşmaktadır. Bu temel yapı komutların kodlanması, kontol akışı, yazmaçların boyutları, hafıza ve adresleme, mantıksal işlemler ve yardımcı kümeden oluşmaktadır. Temel komut seti genel kullanım amaçlı bir bilgisayarı ifade etmek için kullanılmaktadır. Birçok bilgisayar temel komut kümesine ek olarak güç tüketimini, yazılım parçası boyutunu ve memory kullanımı iyileştirmek adına diğer eklentileri tasarım içinde tanımlamaktadır. Bunlara ek olarak işletim sistemi desteklemesi adına S, R, G, C komutlarını tasarımın içinde bulundurması gerekmektedir. RISC- V komut kümesi Tablo 1.1.' de verilmiştir.

Tablo 1.2. RISC-V Temel Komut ve Genişletilen Komut Satırı Kümeleri

Komut Satırı Kümesi'nin Temel ve Genişletilebilinen Komut Kümeleri	
RV32I	32 Bit Temel Komut Kümesi
RV64I	64 Bit Temel Komut Kümesi
RV128I	128 Bit Temel Komut Kümesi
M	Tamasayı Çarpma ve Bölme Kümesi
A	Atomik Komut Kümesi
F	32 Bit Kayan Noktalı Komut Kümesi
D	64 Bit Kayan Noktalı Komut Kümesi
Zicsr	Kontrol ve Durum Yazmaç Komut Kümesi
G	IMAFDzicsr Komutlarının Bütünün İfade Eder
Q	128 Bit Kayan Nokta Komut Kümesi
L	Ondalıklı Kayan Noktalı Komut Kümesi
C	Sıkıştırılmış Komut Satırı Komut Kümesi
B	Bit Operasyonu Komut Satırı Kümesi
V	Vektör Operasyonu Komut Satırı Kümesi
N	Kullanıcı Kesmeleri Komut Satırı Kümesi
P	Tek Komut Çoklu Veri Satırı Kümesi
H	Hipervizör Komut Satırı Kümesi
S	Süpervizör Kesmeleri Komut Satırı Kümesi

Bir tasarımın hangi komut kümesini içerdiğini anlamak için bir isimlendirme yöntemi kullanılmaktadır. Öncelikli olarak temel komut kümesinin belirtilmesi ardından genişletilen komut satırı kümeleri ifade edilmektedir. Temel komut kümesi ve IMAFD genişletilen komut kümelerinin bütünü genel kullanım amaçlı bir bilgisayarı ifade etmektedir. Bu komut kümelerinin bütününe G ifadesi kullanılmaktadır.

Tablo 1.3. RISC-V Yazmaç Kümesi Tablosu

Yazmaç İsmi	Sembolik İsmi	Tanım
x0	zero	Sıfır Yazmacı
x1	ra	Dönüş Adrws Yazmacı
x2	sp	Yığın yazmacı
x3	gp	Evrensel Göstergeci
x4	tp	Paralel İş Parçacığı Gösterğeci
x5	t0	Değişken ve Dönüş Adresi
x6-7	t1-t2	Değişkenler
x8	s0/fp	Saklama Yazmacı / Çerçeve Yazmacı
x9	s1	Saklama Yazmacı
x10-11	a0-1	Fonksiyon Argümanları
x12-17	a2-7	Fonksiyon Argümanları
x18-27	s2-11	Saklama Yazmacı
x28-31	t3-6	Değişken

RISC-V 32 tane tam sayı ve tasarıma eklenmiş ise 32 tane kayan noktalı yazmaç yapısına sahiptir. Bu yazmaç yapısından birinci yazmaç daima sıfıra bağlanmıştır. Bu sıfıra bağlı olan yazmaç yapısı komut satırı işlemlerini kolaylaştırmaktadır. Kontrol ve durum yazmaçları performans ve kayan noktalı yönetimi için kullanılmaktadır. Tablo 1.2.' de standart tam sayı yazmaç yapısı verilmiştir.

Format	Bit																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper immediate	imm[31:12]																				rd					opcode						
Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						
Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd					opcode						

Şekil 1.4. RISC-V 32 Bitlik Komut Formatı

Şekil 1.4'de görüldüğü gibi Risc-V komut satır kümesi temel olarak 6 farklı komut formatından oluşmaktadır. Bu komutlar temel olarak opcode, funct7, funct3, rs1, rs2, rd, imm alanlarından oluşmaktadır. Merkezi işlem birimlerinin içinde bulunan çözümleme birimi tarafından bu kısımlar ayrıştırılarak yapılacak işlem anlaşılmaktadır. Opcode instruction formatını belirlemek için, funct7 ve funct3 alanları opcode ile hangi işlemin yapılacağının çözümlenmesinde, rs1 ve rs2 sırasıyla hangi yazmaçların girdi olarak işlem birimlerine girdi olacağının, rd alanı işlem sonucunun hangi yazmaçta tutulacağının ve son olarak imm alanı ise komutlarda kullanılan sabit değerleri ifade etmek için kullanılmaktadır. Dikkat edilecek husus rs1, rs2, rd alanları 5 bitlik olarak yazılır, bu ifade 32 adet yazmaç adresleyebilmemize olanak sağlamaktadır.

1.2.1. Birtakım komut ifadeleri

Risc-V mimarisinde birçok komutlar sınıfı kullanılmaktadır. Bunlardan RV32I ile ifade edilen temel komut sınıfından bazıları hafızadan yazmaçlara veri kopyalamaya yarayan load&store komutları, yazmaçları değerlerini değiştirmek için kullanılan en değerlikli 20 biti için lui, en değerliksiz 12 bitini değiştirmek için addi, bir program içerisindeki alt fonksyonlara dallanmak için jal, aritmetik ve mantıksal ifadeleri gerçeklemek için add, subtract, or, and, xor logic gibi komutlar bulunmaktadır. Yukarıda belirtilen aritmetik ve mantıksal işlemlerden add, subtract, or, and, xor komutu register/register formatında bir komuttur. Şekil 1.4'de anlaşılacağı gibi bu komut iki tane kaynak yazmaç, bir tane hedef yazmaç ve operasyonun CPU içerisinde neye karşılık geldiğimi ifade etmeye yönelik func7 ,funct3 ve opcode kısımlarından oluşmaktadır. Register/register formatdaki bir komutun opcode' u ikilik sayı sisteminde "0110011" ifadesine karşılık gelirken add işleminin funct7 ifadesi ikilik sayı sisteminde "0000000" ve func3 ifadesi "000" sayı ifadelerine, subtract komutu için funct7 ifadesi ikilik sayı sisteminde "0100000" ve funct3 ifadesi "000" sayı ifadelerine, or mantıksal işleminin funct7 ifadesi ikilik sayı sisteminde "0000000" ve funct3 ifadesi "110" sayı ifadelerine, and mantıksal işleminin funct7 ifadesi ikilik sayı sistemin de "0000000" ve funct3 "111" sayı ifadesine son olarak xor funct7 ifadesi ikilik sayı sistemin de "0000000" ve funct3 ifadesi "100" sayı dizisine karşılık gelmektedir. Load & Store komutları yazmaçtan hafıza elemanına ya da tam tersi hafıza elemanından yazmaçlara veriyi kopyalamak için kullanılmaktadır. Load

komutlarının LB, LH, LW, LBU, LHU gibi farklı varyasyonları bulunduğu gibi Store komutlarının da SB, SH, SW gibi farklı varyasonları bulunmaktadır. Load komutları I tipi komut formatında olurken, Store komutları S tipi komut formatından oluşmaktadır. I tipi komutları temelde sabit sayıları tanımlamak için imm , kaynak yazmaç için rs, hedef yamaç kümesi için rd ve komutu tanımlamada kullanılan funct3 ve opcode ifadelerinden oluşmaktadır. S tipi komutları ise temelde sabit sayıları tanımlamak için imm, kaynak yazmaç için rs1 ve rs2 ve komut tanımlamada kullanılan funct3 ve opcode ifadelerinden oluşmaktadır. LW ve SW komutlarını ele alacak olursak LW için opcode ikilik sayı sisteminde “0000011” sayı dizisine funct3 “010” sayı dizisine karşılık gelirken SW komutları için bu sayı dizileri opcode için “0100011” ikilik sayı dizisine ve funct3 için “010” sayı dizisine karşılık gelmektedir. Temel olarak immediate data formatı yazmaçları belirli sayı değerleriyle işleme sokmayı sağlamaktadır. Bunlardan bazıları ADDI, ORI, ANDI komutlarıdır. Immediate data formatını ifade etmede I type format kullanılmaktadır. Bu I type sabit değerleri ifade etmek için imm, kaynak yazmaç birimini ifade etmek için rs, hedef yazmaç birimini ifade etmek için rd ve fonksiyonel olarak CPU’ da bu komutun anlaşılmasını sağlayan funct3 ve opcode yazmaç bit alanlarından oluşmaktadır.Tüm bu komutlar için opcode ifademiz ikilik sayı dizisi olarak “0010011” sayı dizilimine karşılık gelmektedir. ADDI için funct3 “000” sayı dizisine , ORI için “110” ikilik sayı dizisine, ANDI için “111” ikilik sayı dizisine karşılık gelmektedir.

Yazılım parçamızda alt programlara dallanmak için RISC-V komut setleri içerisinden JAL komutu ifadesi kullanılabilinmektedir. JAL komutu UJ format yapısına sahiptir. Bu format yapısında 20 bitlik imm, hedef yazmaç için rd ve 7 bitlik opcode kısmından oluşmaktadır. JAL komutu temelde program sayacı üstüne imm ile ifade edilen sabiti toplayıp oradaki adrese dallanma görevini yapmaktadır. JAL komutu için opcode ifadesi ikilik bit düzeninde “1101111” sayı dizisine karşılık gelmektedir.

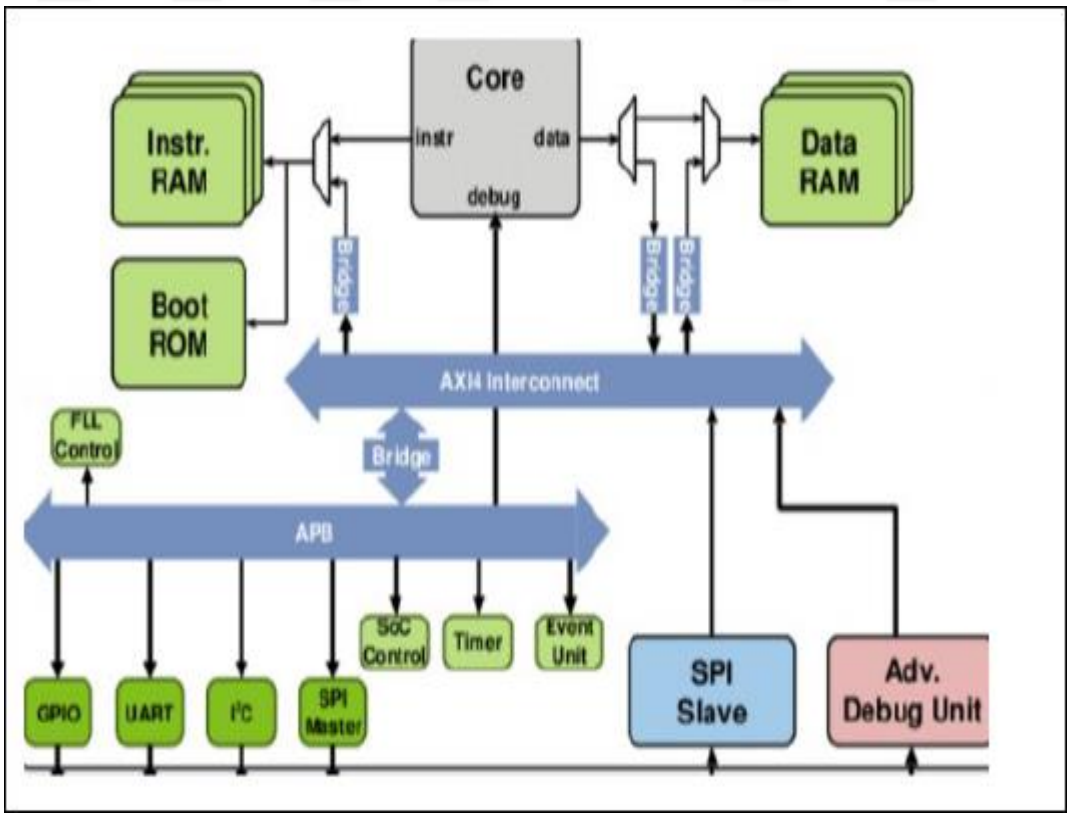
Bu kısım için bazı Risc-V komutlarının yapısı tez kapsamında incelenmiştir. Bu komutlar RV32I bölümünün bir parçasını oluşturmaktadır. Bu komutlar ve diğer komutlar işlemcimizin çözümleme birimlerin de en temelde belirli bit alanları incelenerek öncelikle hangi komutu ifade ettiği analiz edilmektedir. Ardından bu komutun yapacağı işlemler işlemcimizin aritmetik ve logic birimlerinde işlenmekte olup basit yapıda işlemcilerin çalışma mantığı incelenmiştir.

2. PULPINO KIRMIK ÜSTÜ SİSTEMİ

2.1. Pulpino Mikrokontrolcüsü

Pulpino açık kaynak kodlu 32-bit RiscV mimarisini hedef almış tek çekirdekli bir kırmık üstü sistemdir. Pulpino kırmık üstü sistemi, düşük güç tüketimli sinyal işleme uygulamalarını hedef alınarak ETH Zürih üniversitesinde geliştirilmiştir. Pulpino ayarlanılır RISCY ve ZERO-RISCY çekirdeğini kırmık üstü sisteminde kullanabilmektedir. Bu tezde Pulpino' nun RISCY çekirdeği kullanılmaktadır.

RISCY 4 aşamalı, temel tamsayı komut kümesi RV32I, sıkıştırılmış komut kümesi RV32C ve çarpma komut kümesi RV32M desteklemektedir. Ayrıca ayarlanılır tek hassasiyetli noktalı sayı komut kümesi RV32F desteği sunmaktadır. RISCY çekirdeği RISC-V mimarisinin çarpma&toplama birimi, tamsayı operasyonu, donanım çevrimi gibi özelliklerini gerçeklemektedir.



Şekil 2.1. Pulpino Kırmık Üstü Sistemi Mimarisi [8]

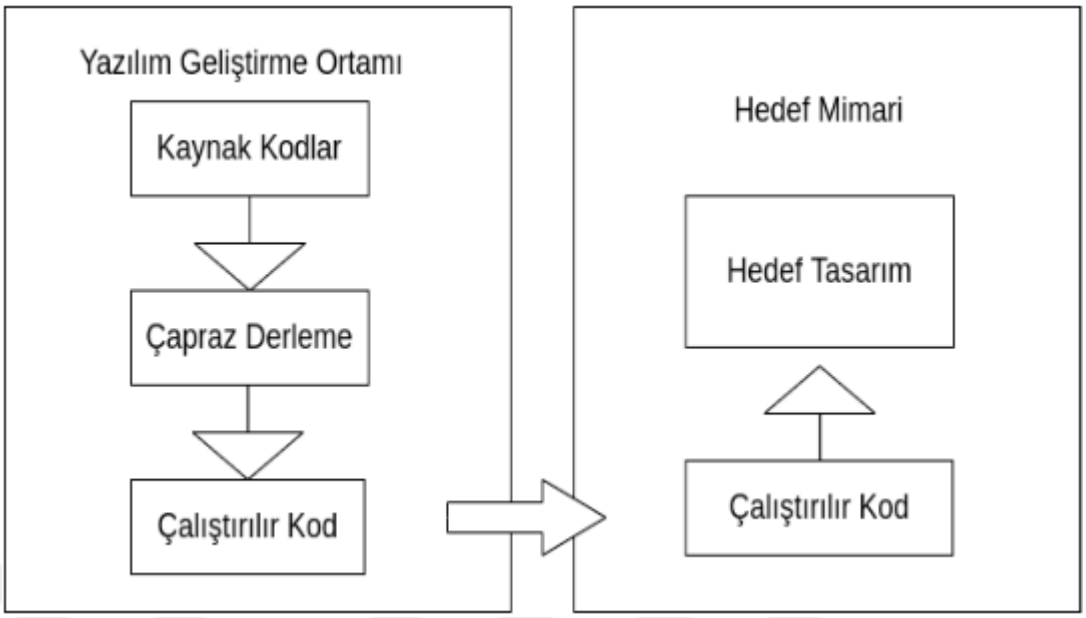
Şekil 2.1'de görüldüğü gibi Kırmık Üstü Sistem dış dünya ile haberleşilebilmesi için I2S, I2C, SPI, UART, GPIO, Zamanlayıcı birimi, Olay birimi gibi birçok donanım birimine sahiptir.

Çekirdek herhangi bir işlem yapmıyorken platform düşük güç tükemi özelliği active edilebilir olay biriminden gelebilecek olan kesmeler ile normal çalışma moduna alınabilmektedir. Bu durumda diğer birimler saat kapılama yöntemiyle kapatılıp ihtiyaç duyulduğu zaman aktive edilir. Bu özellik ile düşük güç tüketimi gerektiren uygulamalarda Kırmık Üstü Sistem ihtiyacını kolaylıkla karşılayabilmektedir. Pulpino platformu RTL simülasyon ve FPGA üzerinde hızlı prototipleme için hazır bir haldedir. Ayrıca bu sistem 2016 yılında fabrikada üretilmiştir.

Pulpino sistemi Şekil 2.1'de görüldüğü gibi Boot-ROM üzerinden çalışmaya başlamaktadır. Boot-ROM temel seviyede sistemde bulunan herhangi bir SPI, I2C, UART gibi standart arayüzden yine Şekil 2.1'de görülen Inst. Ram ve Data Ram hafıza birimlerine çalıştırılması planlanan ana kodumuzun komutlarını Instr. Ram birimine ve yine kodumuzun veri birimlerini Data Ram hafıza birimine yazmaktadır. Bu birimlerin Boot-Rom tarafından doldurulmasının ardından Inst. Ram hafıza biriminin start adresine atlandıktan sonra sistem ana kodumuzu çalıştırmaya başlar.

2.2. Yazılım Geliştirme Ortamı

Kırmık Üstü Sistemlerin hedef uygulamaya yönelik çalışabilmesi için öncelikli olarak yapmak istediğimiz sistemin özellikleri belirlenmeli daha sonrasında bu özellikler doğrultusunda sistem için yazılım parçalarının oluşturulması gerekmektedir. Basit bir sıcaklık göstergesi düşünülür ise, mikrokontrolcü burada sıcaklık bilgisini bir sensörden alırken daha sonrasında bu sıcaklık bilgisini bir ekranda son kullanıcıya gösterebilmektedir. Tüm bu senaryoyu düşünecek olursak en tepeden bir sistem belirlendikten sonra alt seviyede sıcaklık sensörünün hangi arayüz ile çalışacağı, sistemin hangi güç tasarruf modunda çalışacağı ve son kullanıcıya bu bilginin ekranda nasıl gösterileceği, herhangi bir gerçek zamanlı işletim sistemi ya da geleneksel bir işletim sistemine ihtiyacı gibi alt gereksinimler ve bunların da alt detay gereksinimleri oluşturulmaktadır. Tüm bu yazılımsal tasarım isterlerinden sonra artık nihayi yazılım kısmına geçilerek sistemimiz kodlanmaya başlanır.



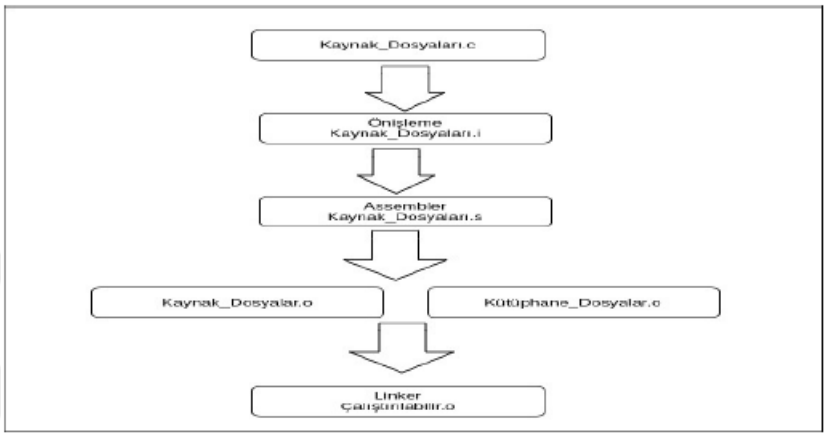
Şekil 2.2. Çapraz Derleme

Kırmık Üstü Sistemlerin yazılımı geliştirmesi denildiğinde çapraz derleyici gibi kavramlar ortaya çıkmaktadır. Kırmık Üstü Sistemlerimiz kısıtlı çevresel birimlere sahip sistemler olduğundan bu sistemler için yazılım geliştirmeleri daha güçlü bir platformda oluşturulduktan sonra en nihayi oluşturulan çalıştırılır kod Kırmık Üstü Sistemimize aktarılmaktadır. İşte farklı bir tasarım mimarisine sahip bir geliştirme ortamında hedef farklı bir tasarım mimarisi platformuna çalıştırılır kod derleme işine çapraz derleme işlemi denilmektedir. Örneğin X86 platformumuzda ARM platformu için gerekli çapraz derleyicilerimizi kurduktan sonra artık hedef ARM mimarisi için çalıştırılır kodlar üretebilmekteyiz. Tüm bu çarpraz derleme işlemi Şekil 2.2'de verilmiş olup bu tez kapsamında Pulpino Kırmık Üstü Sistemimiz Risc-V mimarisine sahip olduğundan çapraz derleyici olarak ETH Zürih üniversitesinin geliştirmiş olduğu "riscv32-unknown-elf" ön uzantılı çapraz derleyicisi kullanılmaktadır.

2.3. Yazılım Oluşturma Aşamaları

Pulpino Kırmık Üstü Sistemimiz için gerekli olan çapraz derleme aracımız ETH Zürih Üniversitesi'nden temin edilebilmektedir. Genel anlamda tüm derleyicilerde olduğu gibi çapraz derleyicimiz kod üretimi için gerekli olan tüm alt araçlara sahiptir. Genel aşamada derleme işlemi önişleme, derleme, assembler ve linklemeaşamalarından oluşmaktadır. Önişleme birimine giren dosya uzantımız .c uzantılı yani bizim yazdığımız yazılım parçasını temsil etmektedir. Önişlemci çıktısı olan kodumuz .i

uzantılıdır. Bu dosya derleyiciye girdi olarak kullanılır ve .s uzantılı hedef mimariyi içeren anahtar kelimelerinden oluşmaktadır. Bu .s uzantılı dosyamız daha sonradan assembler tarafından object dosyasına çevirilir ve son aşamada object dosyamız bir linkleme aracılığıyla çalıştırılır bir koda dönüştürülür. Şekil 2.3'de temel çalışabilecek kod üretim aşamaları verilmiştir.



Şekil 2.3. Temel Çalıştırılır Kod Üretim Aşamaları

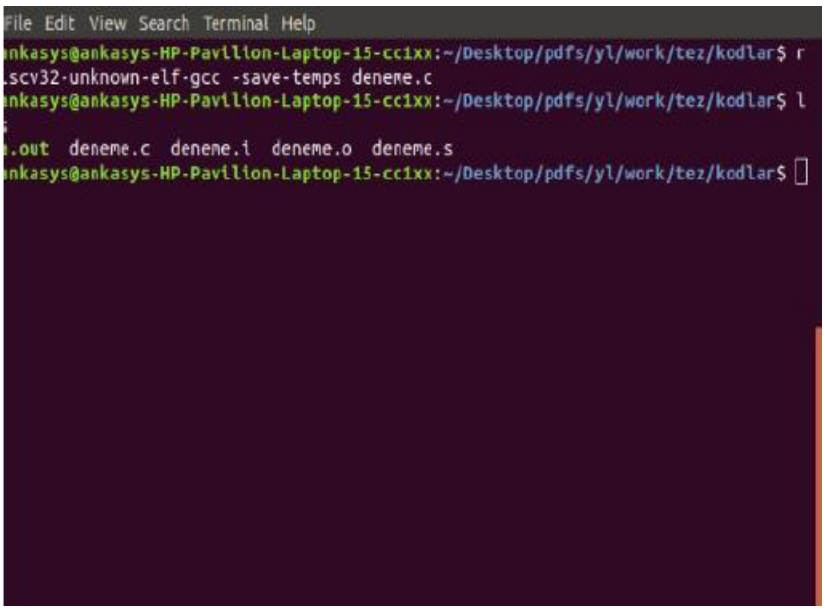
2.4. Önişleme-Derleme-Assembler-Linkleme

Derleme işleminden önce yazılımımıza önişleme işlemi yapılmaktadır. "#" ifadesiyle başlayan tüm satırlar önişlemeden geçmektedir. Önişleme aşamasında yazılım parçamızda yorum satırları kaldırılır, gerekli dosyaların içeriği alınır, koşullu kodlar ayrıştırılır ve makro değişimleri kod içerisinde yapılmaktadır. Şekil 2.4'de örnek bir C kodu verilmiştir ve aynı kodun önişlemciden geçtikten sonraki çıktısı Ek-A' da verilmiştir. Ek-A' da önişlemcinin bütün adımları gerçekleşmiş olup yazılım parçasının izlenebilirliği oldukça azalmıştır.

```
/*
Basit bir C kodu
*/

#include <stdio.h>
int main(){

printf("Merhaba Risc-V\n");

return 0;
}
```

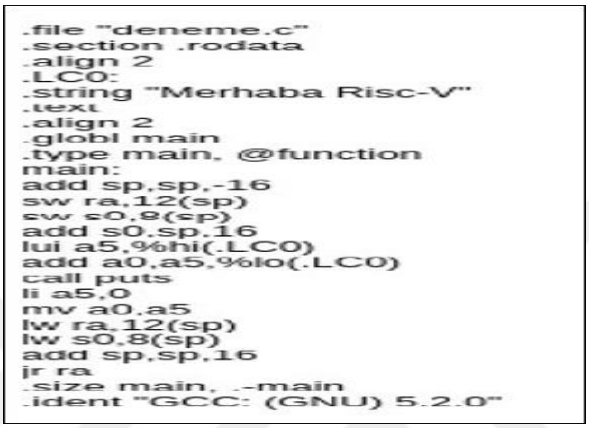
Şekil 2.4. Örnek Bir C Yazılımı

Derleme işlemi yapılırken bir çok opsiyon parametreleri derleme sırasında derleyiciye verilerek aslında oluşturulacak olan kodumuzun hangi formda olmasını istediğimizi belirtmiş oluyoruz. Bunlardan bazıları "-o" bu parametre en nihayi çalışabilir dosyamıza isim vermeye yaramaktadır, "-O0, -O1, -O2, -O3" parametresi yazılım parçamızın optimizasyonu için kullanılır, örneğin gereksiz kod tekrarları gibi durumlar söz konusuyla derleyicimiz vereceğimiz parametreye göre kod optimizasyonu yapmaktadır. "-save-temps" opsiyonu tüm derleme aşamalarındaki dosyalarımızı üretmemizi sağlamaktadır. "-mfpdouble=float -fsingle-precisionconstant" parametresi özellikle kod performansı için çok önemlidir, kodumuzun içinde kayan noktalı işlemler yapılıyorsa bunu donanım üzerinde yapacak komutları oluşturmamıza imkan sağlamaktadır. Eğer bu noktada derleyicimize kayan noktalı işlemleri yaptırsaydık çok kötü performans sonuçlarıyla karşı karşıya kalırdık. "-g" parametresi kodumuzu debug etmemiz için gerekli olan parametreleri kodumuza eklemektedir. "-lm" parametresi kodumuz içerisinde cos, sin gibi matematiksel işlemleri kullanmamıza olanak sağlamaktadır. "–Wall" parametresi kodumuzun derleme işleminde karşılaşılan tüm uyarıları görmemizi sağlar ki bu durum ileride yazılım ile ilgili karşılaşılan hataların çözümünde önem arz etmektedir. "-T" ile kendi linkleme dosyamızı derleyicinin kullanmasını sağlayabiliriz. "-m32" parametresi ile mimarimizin 32 bitlik bir mimari olduğu bilgisini verebilmekteyiz. Şekil 2.5'de derlenmiş kod üretimi için gerekli komut verilmiştir.



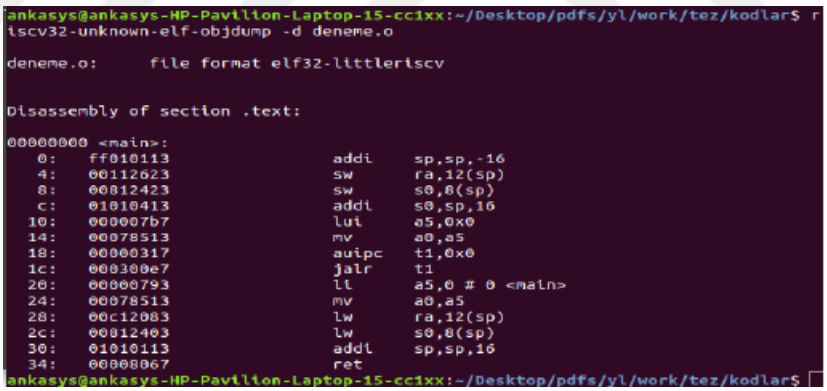
Şekil 2.5. C Kodu Derlenme İşlemi

Derleme işleminden sonrasındaki adım assembler adımıdır. Bu aşamada yazılım parçamız makina anahtar kelimelerine dönüşmüş durumda olup dosya uzantımız .s şeklindedir. Derleyicimizin assembler programı ile 1 ve 0' lardan oluşan makine kodumuza çevrim yapabilmektedir. Şekil 2.6'da assembler programına girdimizi ve ayrıca assembler sonucunda oluşan object dosyalarımızın içeriğini yine derleyicimizin alt bir aracı olan objdump' ı kullanabiliriz.



```
.file "deneme.c"
.section .rodata
.align 2
.LC0:
.string "Merhaba Risc-V"
.text
.align 2
.globl main
.type main, @function
main:
add sp,sp,-16
sw ra,12(sp)
sw s0,8(sp)
add s0,sp,16
lui a5,%hi(.LC0)
add a0,a5,%lo(.LC0)
call puts
li a5,0
mv a0,a5
lw ra,12(sp)
lw s0,8(sp)
add sp,sp,16
jr ra
.size main, .-main
.ident "GCC: (GNU) 5.2.0"
```

Şekil 2.6. Assembler Program Girdisi

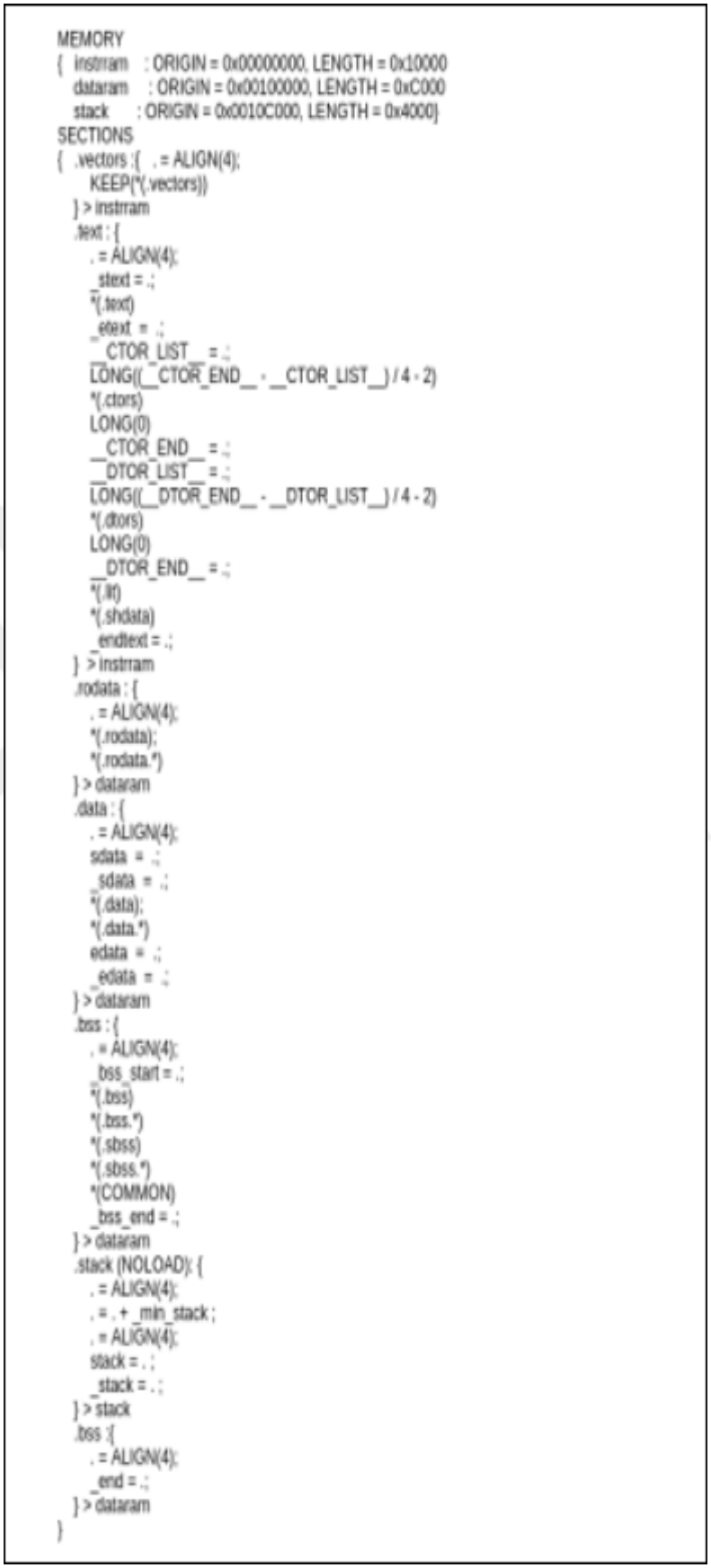


```
ankasys@ankasys-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/pdfs/yl/work/tez/kodlar$ r
iscv32-unknown-elf-objdump -d deneme.o

deneme.o:     file format elf32-littleriscv


Disassembly of section .text:

00000000 <main>:
   0:   ff010113                addi    sp,sp,-16
   4:   00112623                sw      ra,12(sp)
   8:   00812423                sw      s0,8(sp)
   c:   01010413                addi    s0,sp,16
  10:   000007b7                lui     a5,0x0
  14:   00078513                mv      a0,a5
  18:   00000317                auipc   t1,0x0
  1c:   000300e7                jalr    t1
  20:   00000793                li      a5,0 # 0 <main>
  24:   00078513                mv      a0,a5
  28:   00c12083                lw      ra,12(sp)
  2c:   00812403                lw      s0,8(sp)
  30:   01010113                addi    sp,sp,16
  34:   00008067                ret
ankasys@ankasys-HP-Pavilion-Laptop-15-cc1xx:~/Desktop/pdfs/yl/work/tez/kodlar$
```

Şekil 2.7. Assembler Çıktısı Object İçeriği

Object dosyalarımızın içeriğini görmek için çalıştırılması gereken komut ve obje dosyamızın içeriği Şekil 2.7'de verilmiştir. Çalışabilir kod üretimindeki son aşama olan Linkleme aşaması temelde projemizde var olan tüm object dosyalarını alıp son object dosyasını oluşturma işlemidir. Linkleme işlemi belirli bir kurallar dizinine göre yapılmaktadır. Aslında bu kurallar Linker dosyası dediğimiz bir dosya ile derleyiciye -T parametresi ile verilmektedir. Şekil 2.8'de linkleme dosyası verilmiştir. Linkleme dosyası temel olarak üretilen komut ve verilerin Kırmık Üstü Sisteminin hafıza alanında nasıl konumlandırılacağını ifade etmektedir.

Şekil 2.8'de bakacak olursak linkleme dosyamız MEMORY ve SECTION diye adlandırılan iki farklı alandan oluşmaktadır. MEMORY olarak adlandırılan kısım Kırmık Üstü Sistemimizin hafıza alanlarını temsil etmektedir. "instram" olarak adlandırılan hafıza kısmı, komutlarımızın hafıza adresleme sisteminde hangi başlangıç adresine ve hangi boyutta olduğunu, "dataram" olarak adlandırılan hafıza kısmı, verilerimizin hafıza adresleme sistemibde hangi başlanğıç adresine ve hangi boyutta olduğunu belirtir ve son olarak "stack" olarak ifade edilen hafıza kısmı, yazılım parçasında kullanılan stack hafıza bölgesi için ne kadarlık hafıza alanı ayrıldığını göstermektedir.Kullandığımız derleyici obje dosyalarını oluştururken yazılımımızı belirli alanlara ayrımaktadır. Bu alanlar Linkleme dosyamızda da görülen .vector, .text, .rodata, .data, .bss, .stack alanlarıdır. .vector ile ifade edilen alan yazılım parçamızda kullanılan vektör tablosunu, .text ile ifade edilen alan komutlarımızın içinde bulunduğu alanı, .rodata ile ifade edilen alan yazılımında sabit olarak adlandırdığımız ifadeleri, .data ile ifade edilen alan ilk değerleri verilmiş global alandaki değişkenlerimizi, .bss ilk değerleri verilmemiş global bölgedeki değişkenleri ve son olarak .stack olarak verilmiş alan fonksyon değişkenlerinin tutulduğu bölgeler şeklinde ifade edilebilir. Tüm bu alanlar obje dosyalarından en nihayi kullanılır obje dosyasını üretirken linkleme dosyamız yardımıyla birleştirilmektedir. Bunlara ek olarak bazı değişkenlere "." ifadesiyle atamalar yapılmıştır. Bu değişkenler esas olarak o anki adres bilğisini tutmaktadır ki bu değişkenler bazı sistem fonksyonlarında kullanılmaktadır. ">" ile ifade edilen işlem ise bu bölgelerin hangi hafıza bölgesine yazılacağı konusunda bilgi vermektedir. ALIGN(4) ile ifade edilen kısım adresleme bilgilerinin 4 bayt olacak şekilde düzenlememize olanak sağlamaktadır. Çünkü Kırmık Üstü Sistemimiz 32 bitlik adresleme ve okuyabilme olanağına sahiptir. Biraz önce değindiğimiz vektör tablomuzda temel olarak stak yazının ilk değeri, Kırmık Üstü Sistemi reset verildikten sonraki çalışacak ilk fonksyonun adresi ve kesmelere karşılık gelen fonksyonların adres değerlerinin tutulduğu bir diziye karşılık gelmektedir. Bu dizi çalıştırılır kodumuzun en tepesinde son çalıştırılır kodumuzun içine eklenir. Klasik mikrokontrollerde reset verildikten sonra çalışmaya başlayan fonksiyon temel olarak stak yazmanun ilklendirme, global alanda ilklendirilmemiş değişkenlere değer verme, gerekiyorsa bazı sistemdeki modülleri ilklendirme ve yazılımımız için de bulunan main kodumuzu çağırma işlemini ve main kodumuzu sonuna kadar çalışma işlemini yerine getirmektedir.



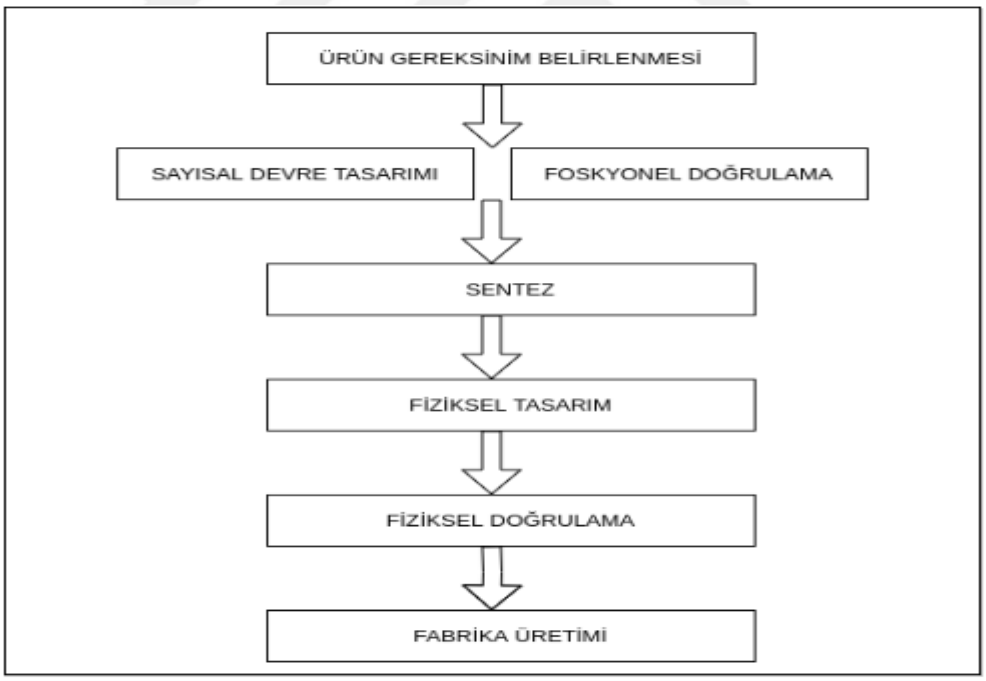
```
MEMORY
{  instrram   : ORIGIN = 0x00000000, LENGTH = 0x10000
   dataram    : ORIGIN = 0x00100000, LENGTH = 0xC000
   stack      : ORIGIN = 0x0010C000, LENGTH = 0x4000}
SECTIONS
{  .vectors :{   . = ALIGN(4);
      KEEP(*(.vectors))
   } > instrram
   .text : {
      . = ALIGN(4);
      _stext = .;
      *(.text)
      _etext  =  .;
      __CTOR_LIST__ = .;
      LONG((__CTOR_END__ - __CTOR_LIST__) / 4 - 2)
      *(.ctors)
      LONG(0)
      __CTOR_END__ = .;
      __DTOR_LIST__ = .;
      LONG((__DTOR_END__ - __DTOR_LIST__) / 4 - 2)
      *(.dtors)
      LONG(0)
      __DTOR_END__ = .;
      *(.lit)
      *(.shdata)
      _endtext = .;
   } > instrram
   .rodata : {
      . = ALIGN(4);
      *(.rodata);
      *(.rodata.*)
   } > dataram
   .data : {
      . = ALIGN(4);
      sdata  =  .;
      _sdata  =  .;
      *(.data);
      *(.data.*)
      edata  =  .;
      _edata  =  .;
   } > dataram
   .bss : {
      . = ALIGN(4);
      _bss_start = .;
      *(.bss)
      *(.bss.*)
      *(.sbss)
      *(.sbss.*)
      *(COMMON)
      _bss_end = .;
   } > dataram
   .stack (NOLOAD): {
      . = ALIGN(4);
      . = . + _min_stack ;
      . = ALIGN(4);
      stack = . ;
      _stack = . ;
   } > stack
   .bss :{
      . = ALIGN(4);
      _end = .;
   } > dataram
}
```

Şekil 2.8. Linkleme Dosyası

3. SENTEZ VE FİZİKSEL TASARIM

3.1. Giriş

Günümüz elektronik cihazlarının hepsinde o cihazın işini yapmasını kontrol eden bir kontol birimi vardır. Bu kontrol birimleri medikal, otomotiv, askeri, uzay ve haberleşme gibi birçok alanda karşımıza çıkmaktadır. Bu devreleri tasarım açısından düşündüğümüzde hız, alan ve güç tüketimi gibi kavramlar oldukça önem arz etmektedir. Bu noktada Uygulama Yönelik Devre Tasarımı konusu ortaya çıkmaktadır. Uygulama Yönelik Devre Tasarımının en büyük avantajı kontrolcü biriminin, hafıza elemanları ve dış dünyadan bilgi almak için gerekli olan donanımların hepsi bir paket içinde olmasıdır.



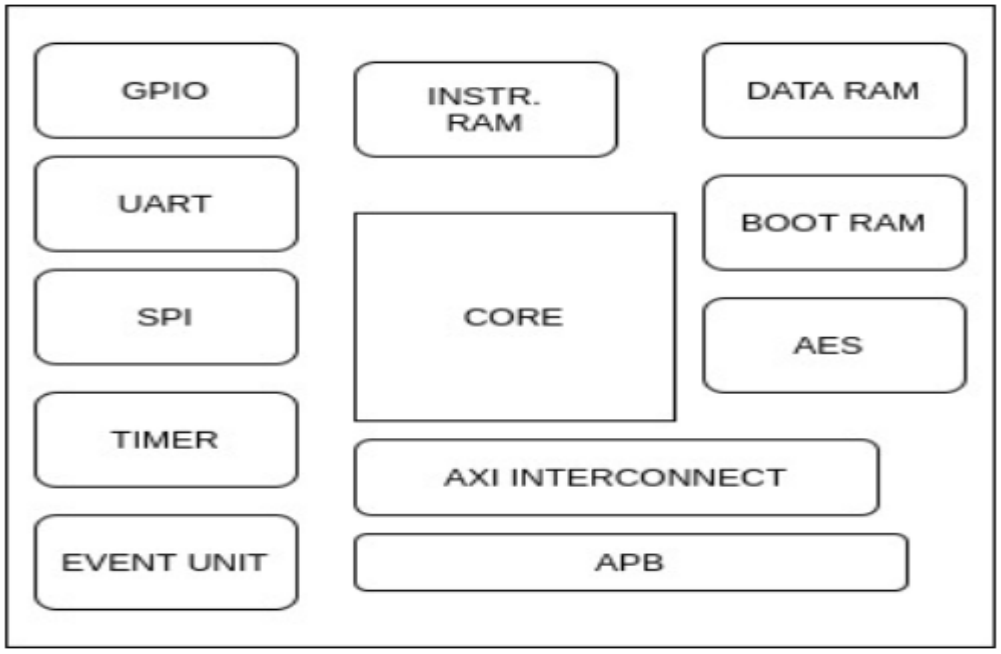
Şekil 3.1. Uygulamaya Yönelik Devre Tasarım

Uygulamaya Yönelik Devre Tasarım süreçleri Şekil 3.1'de verilmiştir. Bu süreç ilk olarak Ürün Gereksinimlerinin Belirlenmesi, Sayısal Devre Tasarımı ve bu devrenin hedef fonksiyonellikte çalıştığının testinin yapıldığı Fonksiyonel Doğrulama, sayısal devrenin hedef teknoloji kapılarına dönüştürüldüğü Sentez işlemi, sentezlenen sayısal

devrenin fiziksel olarak gerçeklendiği Fiziksel Tasarım, fiziksel tasarımın doğruluğunun hedeflendiği Fiziksel Doğrulama ve son olarak nihayi ürünün elde edileceği Fabrika Üretimi aşamalarından oluşmaktadır. Bu bölümde Uygulamaya Yönelik Devre Tasarım içerisinde bulunan ve tez için kullanılan bu yöntemler ve çıktıları verilmiştir.

3.2. Veri Hazırlama

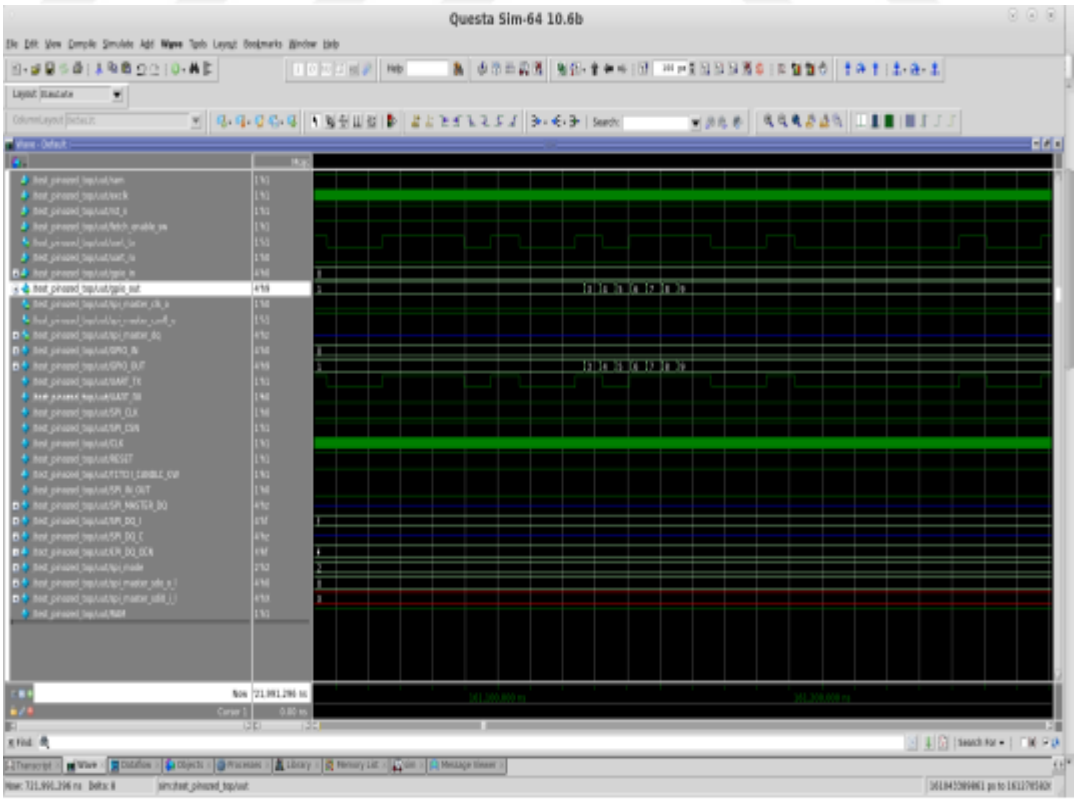
Şekil 3.2'de fiziksel tasarımı gerçeklenmiş Risc-V tasarım mimarisine sahip bir Kırmık Üstü Sistem verilmiştir. Bu Kırmık Üstü Sistem tek çekirdekli Risc-V mimarili bir yapıdadır. Çevresel birimler olarak UART, SPI, GPIO ve TIMER gibi birimlere sahiptir. Bu sistem için öncelikli olarak verilerin hazırlanıp efektif bir çalışma ortamı hazırlanmıştır. Bu ortam hazırlama aşamaları kullanılan .lef, .lib gibi teknoloji dosyaların elde edilmesi, tasarım kodu içinde yine teknoloji hafıza dosyalarının tasarıma eklenmesi, tasarıma giriş ve çıkış noktalarının eklenmesi ve bu eklentilerin doğru çalışıp çalışmadığını gösterecek bir simülasyon yapılmıştır. Yapılan bu simülasyonda kullanılan yazılım parçası Şekil 3.3'de verilmiştir. Aynı simülasyonun program çıktısı Şekil 3.4'de verilmiştir. Yazılım parçasına bakacak olursak öncelikli olarak UART birimi üzerinden dış dünyaya bir ifade gönderilmiş ve GPIO' lara bazı değerler basılmıştır.



Şekil 3.2. Fiziksel Gerçekleşmesi Yapılan Tasarım

```
#include  "math.h"
#include <string.h>
#include "utils.h"
#include "uart.h"
#include "event.h"
#include "timer.h"
#include "int.h"
#include "bench.h"
#include <gpio.h>
#include <spi.h>
#include <stdlib.h>
#define FIXED_POINT


int *data_address;

int main() {

printf("Enter elements: ");
writeGpio(3);
writeGpio(4);
writeGpio(5);
writeGpio(6);
writeGpio(7);
writeGpio(8);
writeGpio(9);


return 0;
```

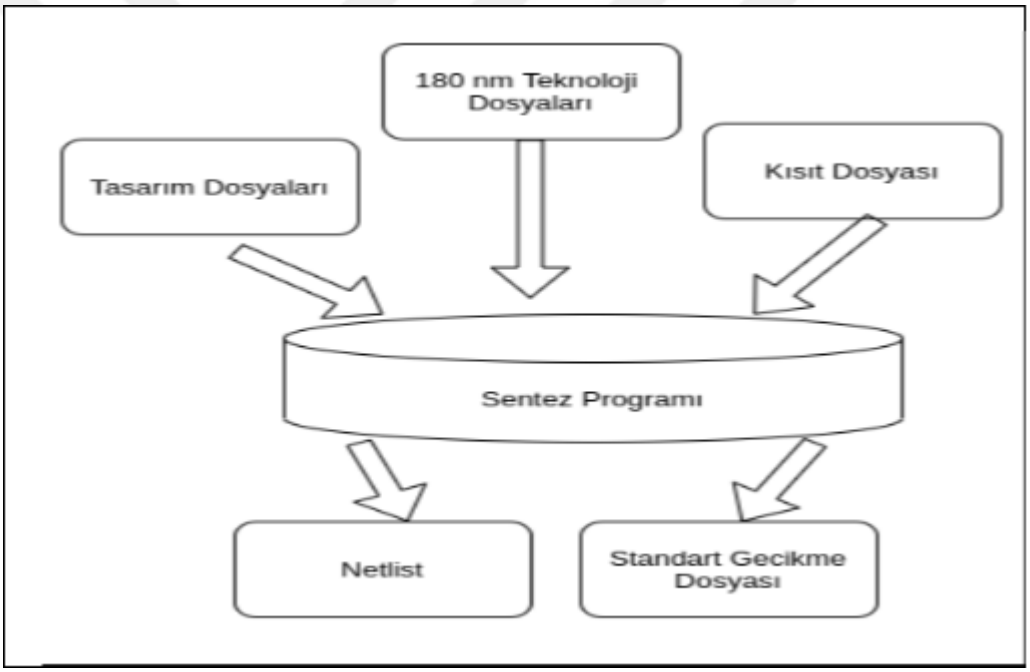
Şekil 3.3. Simülasyon Yazılım Parçası



Şekil 3.4. Simülasyon Çıktısı

3.3. Sentez İşlemi

Sentez işleminin genel amacı güç, zamanlama ve alan hedefi doğrultusunda donanım tanımlama dilleriyle yazılmış olan tasarımı, hedeflenen teknoloji düğümündeki kapı seviyelerine döndürme işlemidir. Şekil 3.5'de görüldüğü gibi Sentez işleminde sentez programcısına hedefteki tasarım kodlarımızı, seçilen teknolojideki ilgili teknoloji dosyası ve zamanlama, güç, alan, fiziksel kısıtlar ve gecikmelerin içinde bulunduğu tasarım kısıt dosyamızı girdi olarak veririz. Sentez sonrasında çıktı olarak ilgili teknoloji düğümünden oluşan tasarım dosyamız, oluşan bu tasarımın içindeki net ve kapıların gecikmelerinden oluşan standart gecikme dosyası da ayrıca çıktı olarak üretilmektedir.



Şekil 3.5. Sentez İşlemi

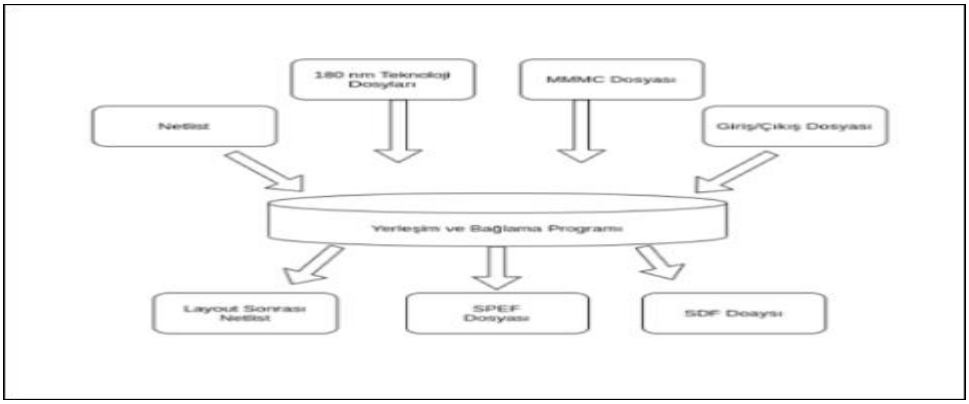
Sentez sonrası oluşan netlist tasarımımızın fonksiyonel karşılığı olan kapılar, giriş çıkış noktaları ve hafıza elemanlarını gösterecek makrolardan oluşmaktadır. Sentez işlemi bu makro, giriş çıkış noktaları ve kapıların olabilecek en kötü zamanlama bilgileriyle yapılmaktadır. Bu aşamada 40 MHz tasarım saat frekansı 0,2 ns belirsizlik, 0,2 ns seviye değişim gecikmesi ve 0,15 kaynak gecikmesi şeklinde modellenerek tasarlanmıştır. Bunlara ek olarak giriş ve çıkış noktalarındaki gecikmeler 1 ns olarak belirlenmiştir. Tablo 3.1.' de bu belirtilen koşullar altında oluşan netlist dosyasındaki birimlerin sayısını ve toplam alan bilgisini vermektedir.

Tablo 3.1. Sentez Makrosu, Standart Kapıları ve Giriş/ Çıkış Birimleri

	Birim Adedi	Toplam Alan
Standart Kapılar	38573	830376.104
Giriş Çıkış Birimleri	19	131100
Makro	2	12773641

3.4. Fiziksel Yerleştirme ve Bağlama İşlemi

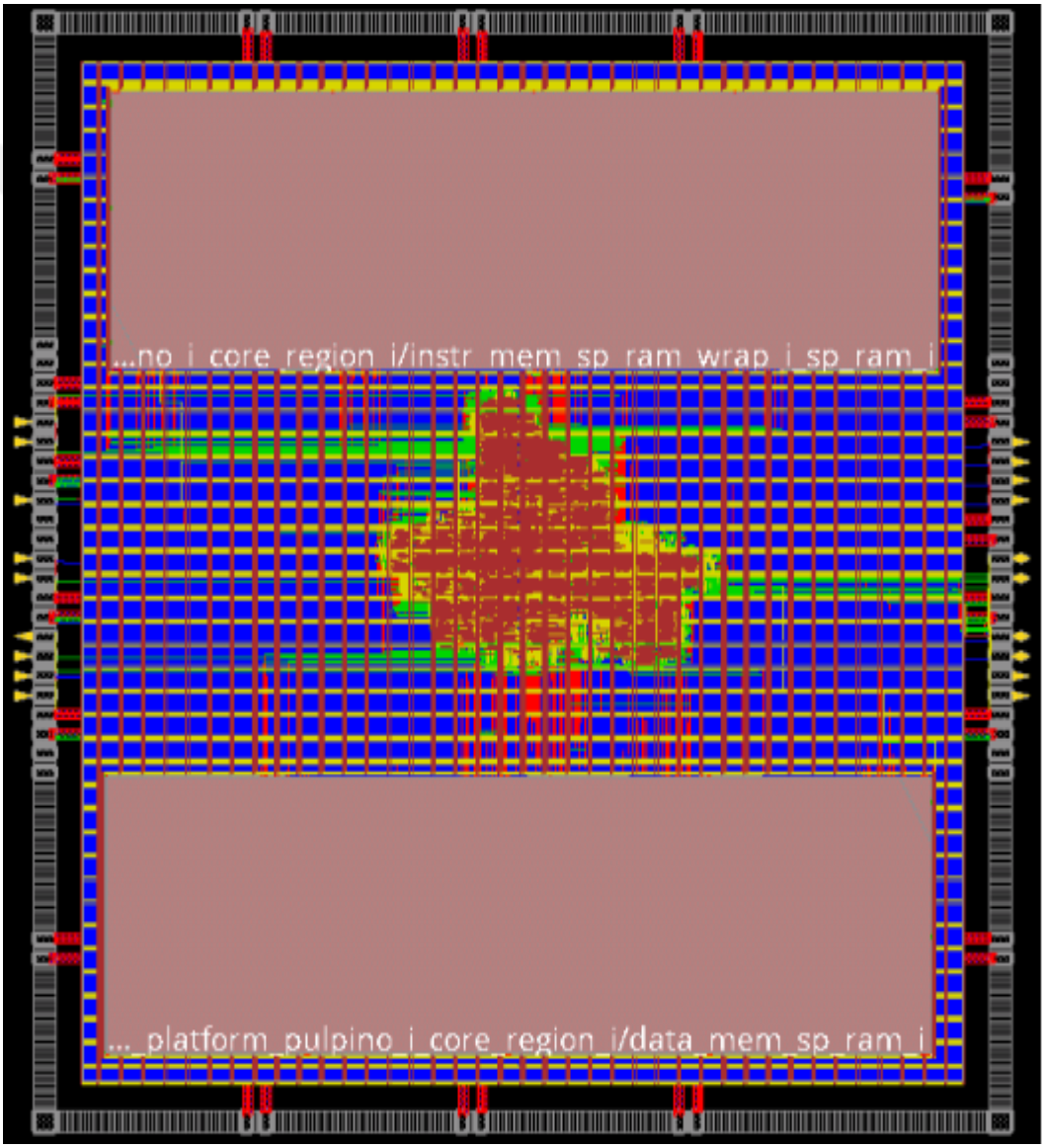
Fiziksel yerleşim ve bağlama işlemi temel olarak mantıksal kapıların fiziksel gösterimlerine dönüşmesi işlemine denilmektedir. Bu yerleşim ve bağlama işlemi konum belirleme, yerleştirme, saat ağacı oluşturma işlemi ve bağlama gibi birkaç adımdan oluşmaktadır. Şekil 3.6'da görüldüğü gibi yerleşim ve bağlama program sentez işleminden gelen neltist, fabrikadan gelen 180nm teknoloji dosyalarını, Farklı mod ve sınır değerlerinin belirtildiği MMMC dosyasını ve çip üzerinde giriş çıkış pinlerinin nasıl konumlandırıldığını gösteren girdileri alıp yerleşim ve baglama işlemini gerçeklemektedir. Bu işlemlerden sonra en nihayetinde tüm kapı, makro ve giriş çıkış pinlerinin yerleşiminin olduğu bir netlist, tüm devre elemanlarının ve bağlantılarının ifade edildiği SDF dosyası ve devredeki bağlantıların direnç, kapasite ve indüktans gibi değerlerin gösterildiği parazitiklerin ifade edildiği SPEF dosyasından oluşmaktadır. Oluşturulan Netlist' in en üst modül tanımı ve kullanılmış olan kapılar Ek-B' de verilmiştir.



Şekil 3.6. Fiziksel Yerleşim ve Bağlama İşlemi

Yerleşim aşamasında çekirdek büyüklüğü, giriş çıkış birimlerinin yerleştirilmesi, devredeki büyük makroları yerleşimi ve genel güç ve toprak netlerinin yerleşimi yapılmaktadır. Yerleştirme işleminde hedeflenen zamanlama çerçevesi içinde kalarak

devremiz içindeki kapıların yerleşimi yapılmaktadır. Üçüncü aşama saat işaretinin devremizin tüm gerekli elemanlarına dağılma işlemidir. Devremizin fiziksel ortamda çalışması için kullanılan saat zamanı içerisine sığması gerekmektedir. Bu doğrultuda tüm elemanların saat işaretini aynı anda alması hedeflenir. Saat işareti oluşturulmasında dolayı oluşabilecek negative etkilerin görülmesinin önüne geçinilebilir. Son aşama olan bağlama aşaması kapıların ve giriş çıkış noktaların birbirine bağlanmasından oluşmaktadır.



Şekil 3.7. Fiziksel Yerleşim ve Bağlama İşlemi Gerçekleşmiş Devre

MMMC dosyamız içerisinde üç temel gecikme sınır ifadeleri içermektedir. Bu gecikme ifadeleri teknoloji dosyalarından gelen kapıların, giriş çıkış noktalarının ve

hafıza birimlerinin zamanlama açısından normal, yavaş ve hızlı kombinasyonlarından oluşmaktadır. Giriş çıkış dosyamızda devremizin dış dünya pinlerinin yerleşimi köşe noktasından 1000 um uzaklık ve 40 um aralılarla yerleştirilmiştir. Çipimiz x düzleminde 5250 um ve y düzleminde 5750 um genişliğe sahiptir. Şekil 3.7'de sayısal devremizin fiziksel olarak yerleştirilmiş ve bağlanmış hali verilmiştir.

3.4.1. Saat ağacı sentezleme işlemi

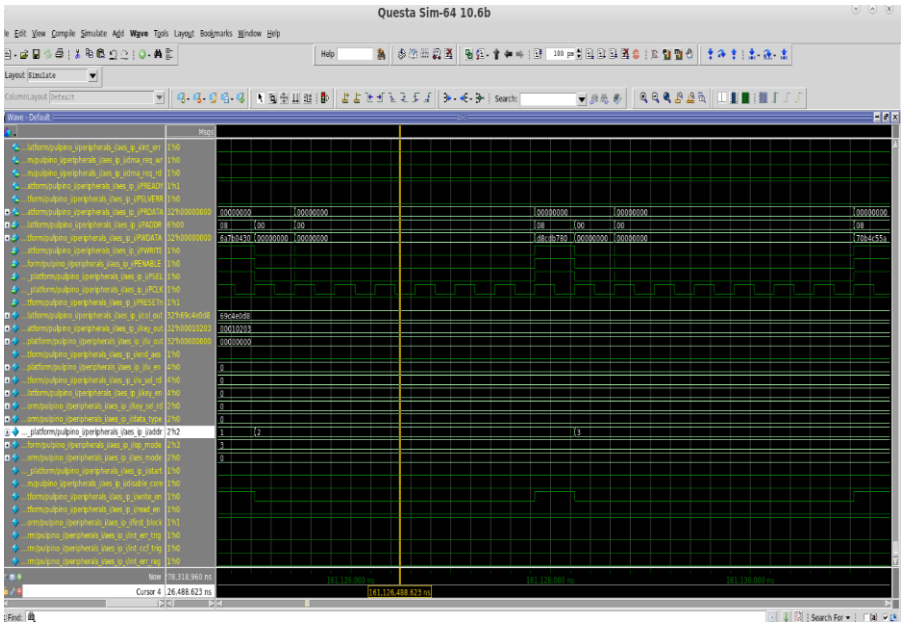
Fiziksel tasarım adımlarından olan saat üretim aşaması, ASIC devrelerini FPGA' lerden ayıran en büyük kısımdır. Bir FPGA şemasında, saat ağacı devrede hazır biçimde bulunmaktadır. ASIC sürecinde ise saat ağacı fiziksel tasarımın bir adımı oalrak tasarıma eklenir. Saat ağacı, en temelde saate ihtiyaç duyan birimlerin aynı anda saati almasından ibarettir. Bu doğrultuda saat ağacına buffer ve inverterlar eklenir. Devremizde kullanılan buffer elemaları CKBD0BWP7T, CKBD10BWP7T, CKBD12BWP7T, CKBD1BWP7T, CKBD2BWP7T, CKBD3BWP7T, CKBD4BWP7T, CKBD6BWP7T ve CKBD8BWP7T bunlara ek olarak inverter elemaları ise CKND0BWP7T, CKND10BWP7T, CKND12BWP7T CKND1BWP7T, CKND2BWP7T, CKND3BWP7T, CKND4BWP7T, CKND6BWP7T ve CKND8BWP7T' dir. Bu elemanlar özel elemanlardır ve düzgün bir saat işareti üretimi için özel tasarlanmıştır. Bu elemaların özelliği voltaj geçişlerinin zamanlama açısından eşit olmasıdır. Saat ağacı tüm devremizin %30 - %40 oranında güç tüketen kısmıdır, dolayısıyla saat ağacını kapılamak devremizin güç tüketimini azaltmaktadır. Devremizde kullanılan saat ağacı kapılayıcıları CKLNQD1, CKLNQD12, CKLNQD2, CKLNQD3, CKLNQD4, CKLNQD6 ve CKLNQD8' dir. Saat ağacını devremizde oluştururken öncelikle bir bağlama kuralı oluşturulur. Ardından kullanılacak buffer, inverter ve saat kapılama elemanları belirlendikten sonra saat ağacı özellikleri oluşturulur. Bu özellikler kaynak gecikmesi, saat frekansı, ve gecikmelerin tanımlandığı bir kümedir. Son olarak fiziksel tasarım aracının saat ağacı sentezleme birimi çalıştırılarak hedeflenen doğrultuda saat birimi devremiz için oluşturulur.

4. SİMÜLASYON VE ÇIKTILAR

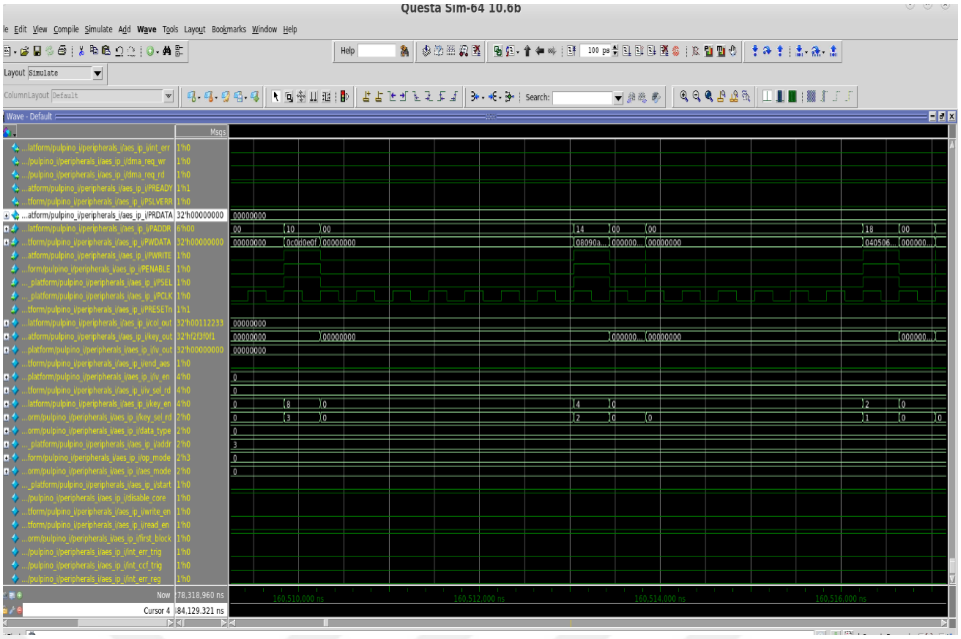
Bu bölümde Şekil 4.1'de AES Yazılım Parçası, Şekil 4.2'de AES İlk Veri Girişi, Şekil 4.3'de AES İkinci Veri Girişi, Şekil 4.4'de Kapı Seviyesi Simülasyon C Kodu, Şekil 4.5'de Tasarımın Simülasyon Görünümü, Şekil 4.6'da Kapı Seviyesi Simülasyon GPIO Çıktıları, Şekil 4.7'de Kapı Seviyesi Simülasyon Uart Çıktısı, Şekil 4.8'de Annotation Çıktısı, Şekil 4.9'da Setup Çıktısı ve Şekil 4.10'da Hold Çıktısı verilmiştir.

```
33 {
34     uart_set_cfg(0, 10);
35     int key[4]  = { 0x0c0d0e0f, 0x08090a0b, 0x04050607, 0x00010203 }; // enc
36     int data[4] = { 0x70b4c55a, 0xd8cdb780, 0x6a7b0430, 0x69c4e0d8 }; //dec
37     int_enable();
38     EER = 0x3 << 21;
39     IER = 0x3 << 21;
40     aes_set_key(key);
41     aes_set_cr(AES_ENABLE | AES_CCFIE | AES_KEY_DECRYP); // dec
42     aes_send_data(data);
43     while(1);
44     sleep_busy(10000000);
45     aes_read_data(ctext);   // read data to default location.
46     printf("ctext:%x=%x=%x=%x",ctext[3],ctext[2],ctext[1],ctext[0]);
47     return 0;
48 }
49
50 void ISR_AES_DONE(void)
51 {
52         ICP = (1 << 22);                // clear interrupt pending register.
53         aes_read_data(ctext);   // read data to default location.
54         printf("ctext:%x-%x-%x-%x",ctext[3],ctext[2],ctext[1],ctext[0]);
55         ctext += 16;                    // increase the ctext location start 4 integers.
56 }
57 void ISR_AES_ERR(void)
58 {
59     ICP = (1 << 21);        // clear interrupt pending register.
60     aes_read_data(ctext);   // read data to default location.
61     printf("eerrrrctext:%d%d%d%d",ctext[3],ctext[2],ctext[1],ctext[0]);
62     ctext += 16;            // increase the ctext location start 4 integers.
63 }
```

Şekil 4.1. AES Yazılım Parçası



Şekil 4.2. AES İlk Veri Girişi

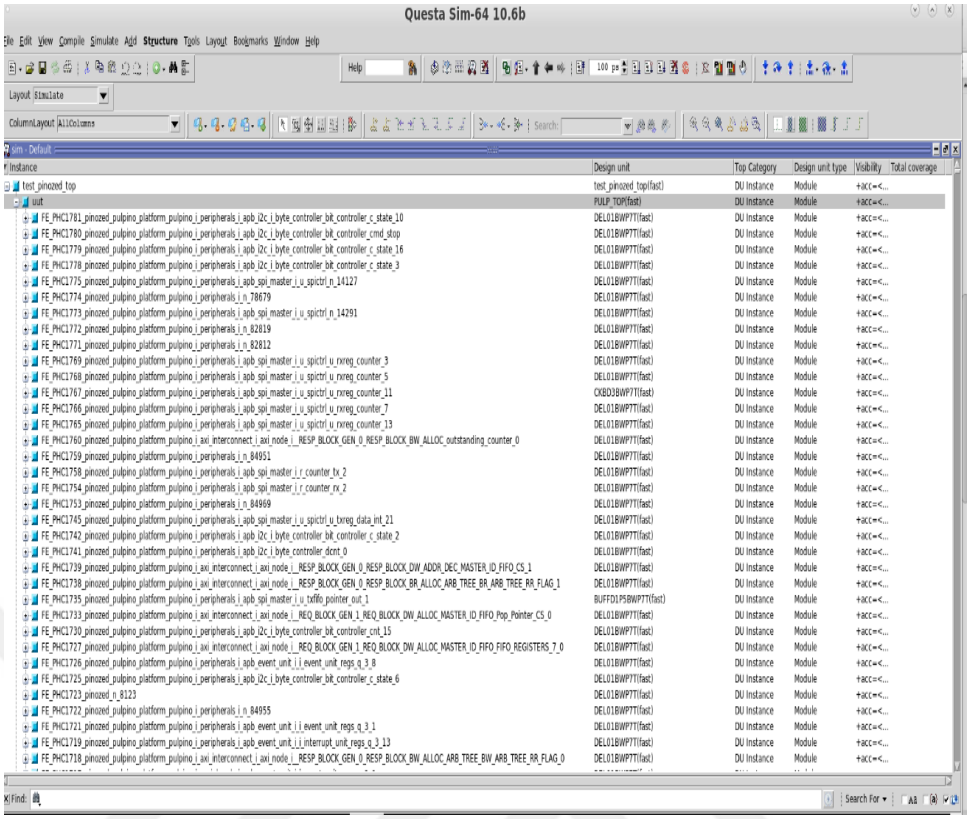


Şekil 4.3. AES İkinci Veri Girişi

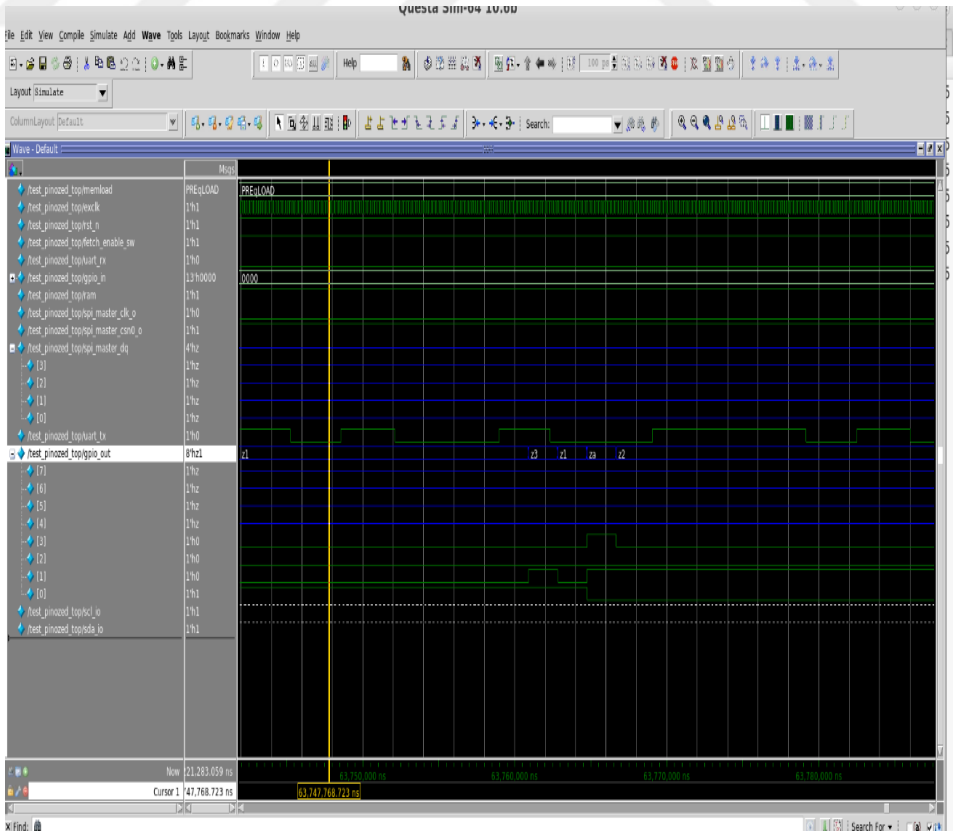
helloworld.c

```
10
17 #include <string.h>
18 #include "utils.h"
19 #include "uart.h"
20 #include "event.h"
21 #include "timer.h"
22 #include "int.h"
23 #include "gpio.h"
24 #include <gpio.h>
25 #include <aes.h>
26 #include <int.h>
27 #include <event.h>
28 #include <pulpino.h>
29
30 int main()
31 {
32
33     writeGpio(3);
34     writeGpio(1);
35     writeGpio(10);
36     writeGpio(2);
37 while(1);
38     return 0;
39 }
40
```

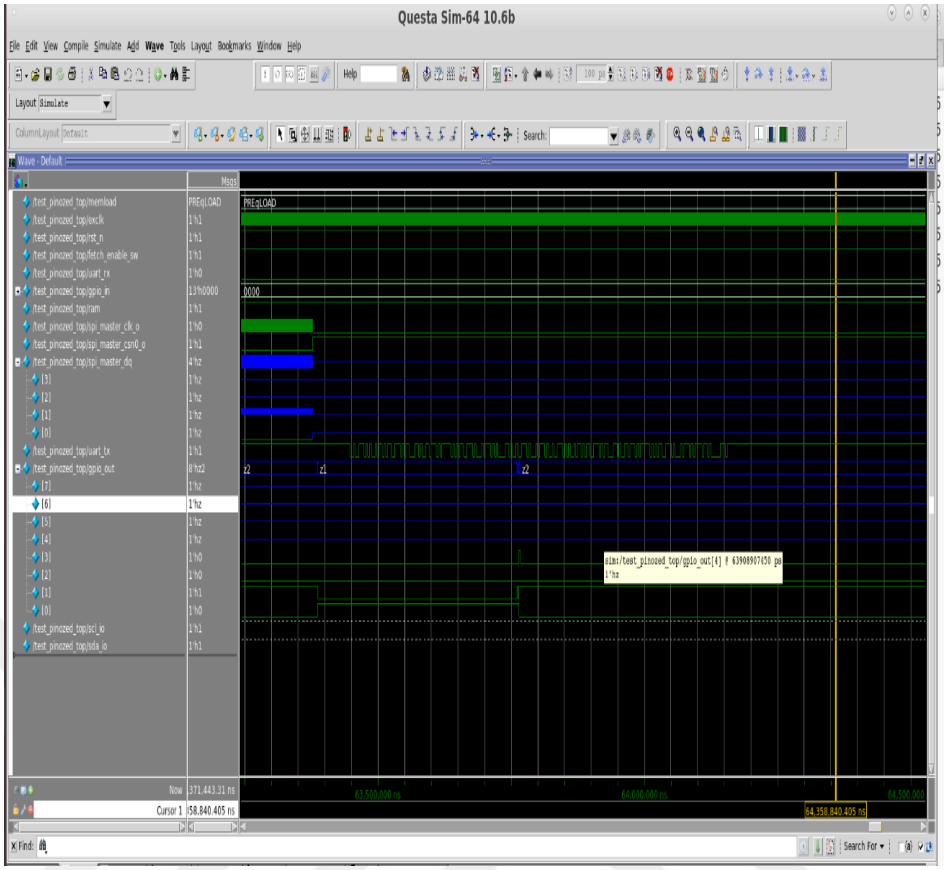
Şekil 4.4. Kapı Seviyesi Simülasyon C Kodu



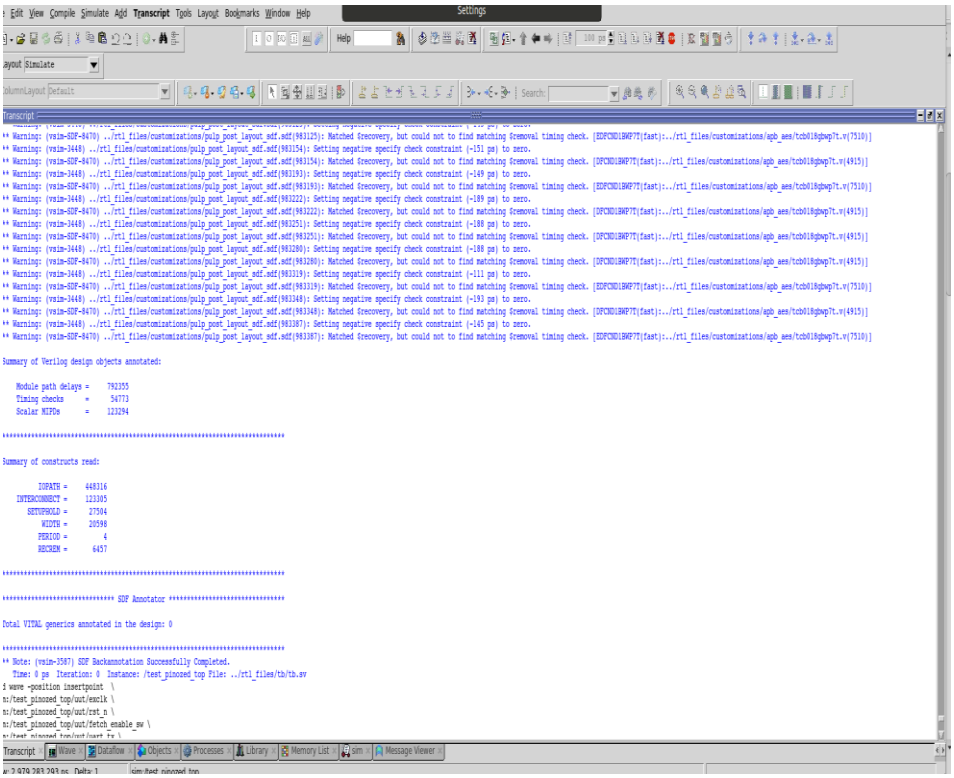
Şekil 4.5. Tasarımın Simülasyon Görünümü



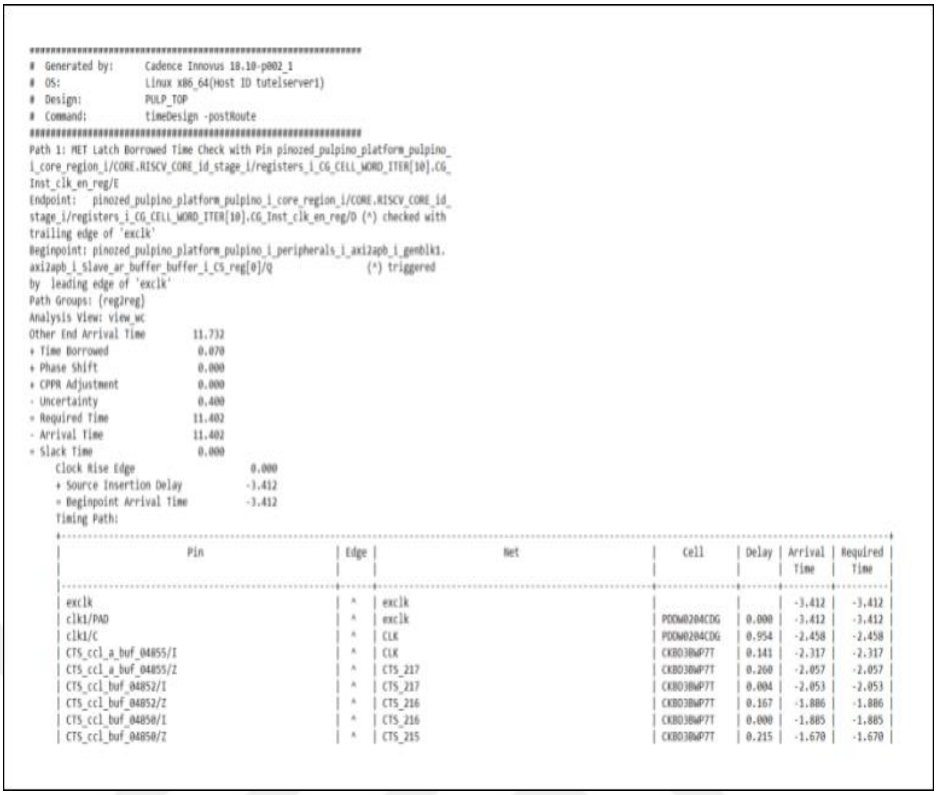
Şekil 4.6. Kapı Seviyesi Simülasyon GPIO Çıktıları



Şekil 4.7. Kapı Seviyesi Simülasyon Uart Çıktısı



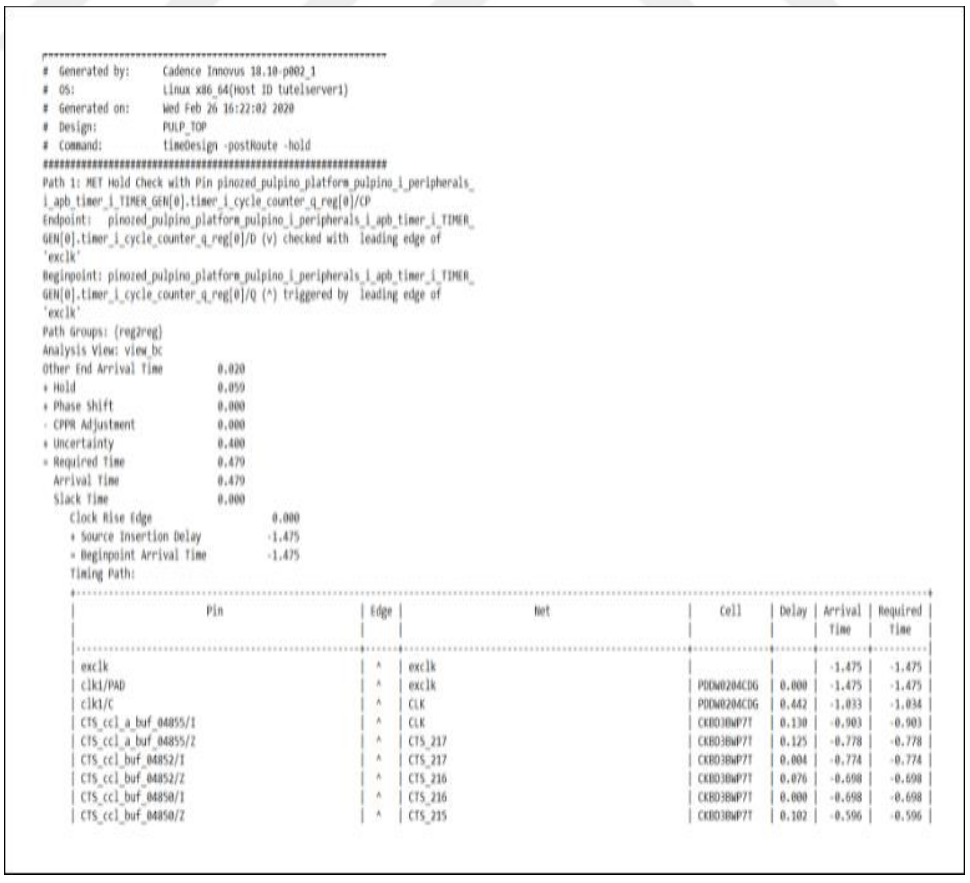
Şekil 4.8. Annotation Çıktısı



```
#  Generated by:      Cadence Innovus 18.10-p002_1
#  OS:                Linux x86_64(Host ID tutelserver1)
#  Design:            PULP_TOP
#  Command:           timeDesign -postRoute
Path 1: MET Latch Borrowed Time Check with Pin pinozed_pulpino_platform_pulpino_
i_core_region_i/CORE.RISCV_CORE_id_stage_i/registers_i_CG_CELL_WORD_ITER[10].CG_
Inst_clk_en_reg/E
Endpoint:   pinozed_pulpino_platform_pulpino_i_core_region_i/CORE.RISCV_CORE_id_
stage_i/registers_i_CG_CELL_WORD_ITER[10].CG_Inst_clk_en_reg/D (^) checked with
trailing edge of 'exclk'
Beginpoint: pinozed_pulpino_platform_pulpino_i_peripherals_i_axi2apb_i_genblk1.
axi2apb_i_Slave_ar_buffer_buffer_i_CS_reg[0]/Q                (^) triggered
by  leading edge of 'exclk'
Path Groups: {reg2reg}
Analysis View: view_wc
Other End Arrival Time          11.732
+ Time Borrowed                  0.070
+ Phase Shift                    0.000
+ CPPR Adjustment                0.000
- Uncertainty                    0.400
= Required Time                 11.402
- Arrival Time                  11.402
= Slack Time                     0.000
     Clock Rise Edge                      0.000
     + Source Insertion Delay            -3.412
     = Beginpoint Arrival Time           -3.412
     Timing Path:
```

Pin	Edge	Net	Cell	Delay	Arrival Time	Required Time
exclk	^	exclk			-3.412	-3.412
clk1/PAD	^	exclk	PDDW0204CDG	0.000	-3.412	-3.412
clk1/C	^	CLK	PDDW0204CDG	0.954	-2.458	-2.458
CTS_ccl_a_buf_04855/I	^	CLK	CKBD3BWP7T	0.141	-2.317	-2.317
CTS_ccl_a_buf_04855/Z	^	CTS_217	CKBD3BWP7T	0.260	-2.057	-2.057
CTS_ccl_buf_04852/I	^	CTS_217	CKBD3BWP7T	0.004	-2.053	-2.053
CTS_ccl_buf_04852/Z	^	CTS_216	CKBD3BWP7T	0.167	-1.886	-1.886
CTS_ccl_buf_04850/I	^	CTS_216	CKBD3BWP7T	0.000	-1.885	-1.885
CTS_ccl_buf_04850/Z	^	CTS_215	CKBD3BWP7T	0.215	-1.670	-1.670

Şekil 4.9. Setup Çıktısı



```
#  Generated by:      Cadence Innovus 18.10-p002_1
#  OS:                Linux x86_64(Host ID tutelserver1)
#  Generated on:      Wed Feb 26 16:22:02 2020
#  Design:            PULP_TOP
#  Command:           timeDesign -postRoute -hold
Path 1: MET Hold Check with Pin pinozed_pulpino_platform_pulpino_i_peripherals_
i_apb_timer_i_TIMER_GEN[0].timer_i_cycle_counter_q_reg[0]/CP
Endpoint:   pinozed_pulpino_platform_pulpino_i_peripherals_i_apb_timer_i_TIMER_
GEN[0].timer_i_cycle_counter_q_reg[0]/D (v) checked with  leading edge of
'exclk'
Beginpoint: pinozed_pulpino_platform_pulpino_i_peripherals_i_apb_timer_i_TIMER_
GEN[0].timer_i_cycle_counter_q_reg[0]/Q (^) triggered by  leading edge of
'exclk'
Path Groups: {reg2reg}
Analysis View: view_bc
Other End Arrival Time          0.020
+ Hold                          0.059
+ Phase Shift                   0.000
- CPPR Adjustment               0.000
+ Uncertainty                   0.400
= Required Time                 0.479
  Arrival Time                  0.479
  Slack Time                    0.000
     Clock Rise Edge                      0.000
     + Source Insertion Delay            -1.475
     = Beginpoint Arrival Time           -1.475
     Timing Path:
```

Pin	Edge	Net	Cell	Delay	Arrival Time	Required Time
exclk	^	exclk			-1.475	-1.475
clk1/PAD	^	exclk	PDDW0204CDG	0.000	-1.475	-1.475
clk1/C	^	CLK	PDDW0204CDG	0.442	-1.033	-1.034
CTS_ccl_a_buf_04855/I	^	CLK	CKBD3BWP7T	0.130	-0.903	-0.903
CTS_ccl_a_buf_04855/Z	^	CTS_217	CKBD3BWP7T	0.125	-0.778	-0.778
CTS_ccl_buf_04852/I	^	CTS_217	CKBD3BWP7T	0.004	-0.774	-0.774
CTS_ccl_buf_04852/Z	^	CTS_216	CKBD3BWP7T	0.076	-0.698	-0.698
CTS_ccl_buf_04850/I	^	CTS_216	CKBD3BWP7T	0.000	-0.698	-0.698
CTS_ccl_buf_04850/Z	^	CTS_215	CKBD3BWP7T	0.102	-0.596	-0.596

Şekil 4.10. Hold Çıktısı

5. SONUÇ VE ÖNERİLER

Bu tez çalışmasında; açık kaynak kodlu, hiçbir lisans maliyetine ihtiyaç duymayan RISC-V mimarisini hedef alarak oluşturulmuş Kırmık Üstü Sistemin fiziksel tasarımı yapılmıştır. Sentezleme işlemi Kırmık Üstü Sistemin tasarım dosyaları, 180 nm teknoloji kütüphaneleri ve tasarım kısıt dosyaları ile 40 MHz saat frekansı ile oluşturulmuştur. Bu aşamadan sonra fiziksel tasarım kısmında oluşturulan netlist, ilgili teknoloji düğümündeki kütüphane dosyaları, tasarımımız üzerindeki fiziksel dış dünya pinlerinin çip üzerindeki konumunu gösteren giriş ve çıkış dosyamız, farklı varyasyonlarda çipimizi analiz etmemize yarayan MMMC dosyamız kullanılarakçipimizin fiziksel görünüm devresi elde edilmiştir. Fiziksel tasarım sonrasında devremiz üzerinde bulunan zamanlama ihlalleri ECO aşamasında çözülmüş olup zamanlama açısından en nihayi fiziksel gerçeklenmiş devremiz ortaya konulmuştur.

İlerideki akademik çalışmalara öneri olarak fiziksel yapısı oluşturulan devremizin fabrikaya gönderilmeden önceki fiziksel doğrulama testleri yapılabilir ve çipin üretiminden sonraki aşaması olan tasarım testleri için gerekli olan test kapıları,tasarım içine tüm bu işlemlerden önce ya da sentez aşamasında eklenebilir. Bu doğrultuda hem çipimizin fonsiyonel olarak çalışması sağlanır ve tüm bu süreç hem insan hem de üretim maliyeti açısıdan maliyetlerin boşa harcanmasının önüne geçmiş olur. Bunlara ek olarak farklı teknoloji düğümlerinde ve daha hızlı saat frekanslarında devremiz tekrardan oluşturularak istenilen sistem gereksinimleri doğrultusunda zaman, alan ve güç bakımından daha verimli bir devre elde edilebilir.

KAYNAKLAR

[1] Reid T. R., *The Chip: How Two Americans Invented the Microchip and Launched a Revolution*, 1st ed., Random House, New York, 2001.

[2] Qurat-ul-Ain M., Aqeel I., Next Generation High Speed Computing Using System-on-Chip (SoC) Technology, *International Journal of Advancements in Technology*, 2011, **2**(2), 243-256.

[3] Andrew W., Krste A., *The RISC-V Instruction Set Manual*, 2nd., Univesity of California, USA, 2019.

[4] Abdelfattah M., Mina M., Samer E., Sherouk N., Sameh E., Ahmed S., Fast Reliable Verification Methodology for RISC-V Without a Reference Model, *International Workshop on Microprocessor and SOC Test and Verification*, DOI: 10.1109/MTV.2018.00012.

[5] Gianna P., Igor S., Francesco C., Lukas C., Chipmunk:Asystolically Scalable 0.9 mm2, 3.08 Gop/s/mW @1.2 mW Accelerator for Near-Sensor Recurrent Neural Network Inference, *IEEE Custom Integrated Circuit*, DOI: 10.1109/CICC.2018.8357068.

[6] Christelle G., Manuel E., Michael S., Lukas C., Hydra, ETH Zurih, https://asic.ethz.ch/2016/Hydra.html (Ziyaret tarihi: 21 Aralık 2019).

[7] Giovani R., Lay_Llama, ETH Zurih, https://asic.ethz.ch/2016/Lay_Llama.html (Ziyaret tarihi: 22 Aralık 2019).

[8] Cadence Design Sys., *Genus User Guide*, 1st ed., Cadence Design System Inc., San Jose, 2018.

[9] Cadence Design Sys., *Innovus User Guide*, 1st ed., Cadence Design System Inc., San Jose, 2018.

[10] Mentor Graphics Corp., *Questa SIM User's Manuel*, 1st ed., Mentor Graphics Corporation, Oregon, 2017.

[11] Himanshu B., *Advanced Asic Chip Synthesis*, 2nd ed., Kluwer Academic Publishers, USA, 2001.

EKLER

Ek-A

```
# 1 "deneme.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "deneme.c"
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 1 3
# 29 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/_ansi.h" 1 3
# 15 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/_ansi.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/newlib.h" 1 3
# 16 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/_ansi.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/config.h" 1 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/ieeefp.h"
1 3
# 5 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/config.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/features.h" 1
3
# 6 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/config.h" 2 3
# 17 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/_ansi.h" 2 3
# 30 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/cdefs.h" 1 3
# 43 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/cdefs.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 1 3
# 27 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
# 27 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef signed char __int8_t;
typedef unsigned char __uint8_t;
# 41 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef short int __int16_t;
typedef short unsigned int __uint16_t;
# 63 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef long int __int32_t;
typedef long unsigned int __uint32_t;
# 89 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef long long int __int64_t;
typedef long long unsigned int __uint64_t;
# 120 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef signed char __int_least8_t;
typedef unsigned char __uint_least8_t;
# 146 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
```

```
_default_types.h" 3
typedef short int __int_least16_t;
typedef short unsigned int __uint_least16_t;
# 168 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef long int __int_least32_t;
typedef long unsigned int __uint_least32_t;
# 186 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef long long int __int_least64_t;
typedef long long unsigned int __uint_least64_t;
# 200 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_default_types.h" 3
typedef int __intptr_t;
typedef unsigned int __uintptr_t;
# 44 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/cdefs.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 1 3 4
# 216 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 3 4
typedef unsigned int size_t;
# 46 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/cdefs.h" 2 3
# 36 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 1 3 4
# 149 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 3 4
typedef int ptrdiff_t;
# 328 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/

stddef.h" 3 4
typedef int wchar_t;
# 426 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 3 4
typedef struct {
long long __max_align_ll __attribute__((__aligned__(__alignof__(long long))));
long double __max_align_ld __attribute__((__aligned__(__alignof__(long
double))));
} max_align_t;
# 37 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stdarg.h" 1 3 4
# 40 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stdarg.h" 3 4
typedef __builtin_va_list __gnuc_va_list;
# 40 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 1 3
# 13 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/_ansi.h" 1 3
```

```
# 14 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 1 3 4
# 15 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 1 3
# 12 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
_types.h" 1 3
typedef unsigned long long __dev_t;
typedef unsigned int __uid_t;
typedef unsigned int __gid_t;
typedef long long _off_t;
typedef unsigned long long _fpos_t;
typedef unsigned long long time_t;
typedef unsigned long long __ino_t;
typedef unsigned int __nlink_t;
typedef long signed int _ssize_t;
# 13 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 2
3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/lock.h" 1 3

typedef int _LOCK_T;
typedef int _LOCK_RECURSIVE_T;
# 14 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 2
3
# 31 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 3
__extension__ typedef long long _off64_t;
# 67 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 1 3 4
# 357 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 3 4
typedef unsigned int wint_t;
# 68 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/_types.h" 2
3
typedef struct
{
int __count;
union
{
wint_t __wch;
unsigned char __wchb[4];
} __value;
} _mbstate_t;
typedef _LOCK_RECURSIVE_T _flock_t;
typedef void *_iconv_t;
# 16 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 2 3
typedef unsigned long __ULong;
# 38 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
```

```
struct _reent;
struct _Bigint
{
struct _Bigint *_next;
int _k, _maxwds, _sign, _wds;
__ULong _x[1];
};
struct __tm
{
int __tm_sec;
int __tm_min;
int __tm_hour;
int __tm_mday;
int __tm_mon;
int __tm_year;
int __tm_wday;
int __tm_yday;
int __tm_isdst;
};
struct _on_exit_args {
void * _fnargs[32];
void * _dso_handle[32];
__ULong _fntypes;
__ULong _is_cxa;
};
# 91 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
struct _atexit {
struct _atexit *_next;
int _ind;
void (*_fns[32])(void);
struct _on_exit_args _on_exit_args;
};
# 115 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
struct __sbuf {
unsigned char *_base;
int _size;
};
# 179 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
struct __sFILE {
unsigned char *_p;
int _r;
int _w;
short _flags;
short _file;
struct __sbuf _bf;
int _lbfsize;
void * _cookie;
_ssize_t (* _read) (struct _reent *, void *, char *, int);
_ssize_t (* _write) (struct _reent *, void *, const char *, int) ;
```

```
_fpos_t (* _seek) (struct _reent *, void *, _fpos_t, int);
int (* _close) (struct _reent *, void *);
struct __sbuf _ub;
unsigned char *_up;
int _ur;
unsigned char _ubuf[3];
unsigned char _nbuf[1];
struct __sbuf _lb;
int _blksize;
_off_t _offset;
struct _reent *_data;
_flock_t _lock;
_mbstate_t _mbstate;
int _flags2;
};
# 285 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
typedef struct __sFILE __FILE;
struct _glue
{
struct _glue *_next;
int _niobs;
__FILE *_iobs;
};
# 317 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
struct _rand48 {
unsigned short _seed[3];
unsigned short _mult[3];
unsigned short _add;
};
# 569 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
struct _reent
{
int _errno;
__FILE *_stdin, *_stdout, *_stderr;
int _inc;
char _emergency[25];
int _current_category;
const char *_current_locale;
int __sdidinit;
void (* __cleanup) (struct _reent *);
struct _Bigint *_result;
int _result_k;
struct _Bigint *_p5s;
struct _Bigint **_freelist;
int _cvtlen;
char *_cvtbuf;
union
{
struct
```

```
{
unsigned int _unused_rand;
char * _strtok_last;
char _asctime_buf[26];
struct __tm _localtime_buf;
int _gamma_signgam;
__extension__ unsigned long long _rand_next;
struct _rand48 _r48;
_mbstate_t _mblen_state;
_mbstate_t _mbtowc_state;
_mbstate_t _wctomb_state;
char _l64a_buf[8];
char _signal_buf[24];
int _getdate_err;
_mbstate_t _mbrlen_state;
_mbstate_t _mbrtowc_state;
_mbstate_t _mbsrtowcs_state;
_mbstate_t _wcrtomb_state;
_mbstate_t _wcsrtombs_state;
int _h_errno;
} _reent;
struct
{
unsigned char * _nextf[30];
unsigned int _nmalloc[30];
} _unused;
} _new;
struct _atexit *_atexit;
struct _atexit _atexit0;
void (**(_sig_func))(int);
struct _glue __sglue;
__FILE __sf[3];
};
# 762 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/reent.h" 3
extern struct _reent *_impure_ptr ;
extern struct _reent *const _global_impure_ptr ;
void _reclaim_reent (struct _reent *);
# 48 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 1 3
# 69 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
# 1 "/opt/ri5cy_gnu_toolchain/install/lib/gcc/riscv32-unknown-elf/5.2.0/include/
stddef.h" 1 3 4
# 70 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 2 3
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/types.h"
# 19 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/machine/
types.h" 3
typedef long int __off_t;
typedef int __pid_t;
__extension__ typedef long long int __loff_t;
```

```
# 71 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 2 3
# 93 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef unsigned char u_char;
typedef unsigned short u_short;
typedef unsigned int u_int;
typedef unsigned long u_long;
typedef unsigned short ushort;
typedef unsigned int uint;
typedef unsigned long ulong;
typedef unsigned long clock_t;
# 130 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
struct timespec {
time_t tv_sec;
long tv_nsec;
};
struct itimerspec {
struct timespec it_interval;
struct timespec it_value;
};
typedef long daddr_t;
typedef char * caddr_t;
typedef __ino_t ino_t;
# 188 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef _off_t off_t;
typedef __dev_t dev_t;
typedef __uid_t uid_t;
typedef __gid_t gid_t;
typedef int pid_t;
# 208 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef long key_t;
typedef _ssize_t ssize_t;
# 224 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef unsigned int mode_t __attribute__ ((__mode__ (__SI__)));
typedef __nlink_t nlink_t;
# 256 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef long fd_mask;
typedef struct _types_fd_set {
fd_mask fds_bits[((((64)+(((sizeof (fd_mask) * 8))-1))/((sizeof (fd_mask) * 8)))];
} _types_fd_set;
# 287 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/types.h" 3
typedef unsigned long clockid_t;
typedef unsigned long timer_t;
typedef unsigned long useconds_t;
typedef long suseconds_t;
# 49 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
typedef __FILE FILE;
typedef _fpos_t fpos_t;
# 1 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/sys/stdio.h" 1 3
# 64 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 2 3
```

```
# 164 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 3
FILE * tmpfile (void);
char * tmpnam (char *);
char * tempnam (const char *, const char *);
int fclose (FILE *);
int fflush (FILE *);
FILE * freopen (const char *restrict, const char *restrict, FILE *restrict);
void setbuf (FILE *restrict, char *restrict);
int setvbuf (FILE *restrict, char *restrict, int, size_t);
int fprintf (FILE *restrict, const char *restrict, ...) __attribute__ ((__format__
(__printf__, 2, 3))) ;
int fscanf (FILE *restrict, const char *restrict, ...) __attribute__ ((__format__
(__scanf__, 2, 3)));
int printf (const char *restrict, ...) __attribute__ ((__format__ (__printf__, 1, 2)));
int scanf (const char *restrict, ...) __attribute__ ((__format__ (__scanf__, 1, 2)));
int sscanf (const char *restrict, const char *restrict, ...) __attribute__ ((__format__
(__scanf__, 2, 3)));
int vfprintf (FILE *restrict, const char *restrict, __gnuc_va_list) __attribute__
((__format__ (__printf__, 2, 0)));
int vprintf (const char *, __gnuc_va_list) __attribute__ ((__format__ (__printf__, 1,
0))) ;
int vsprintf (char *restrict, const char *restrict, __gnuc_va_list) __attribute__
((__format__ (__printf__, 2, 0))) ;
int fgetc (FILE *);
char * fgets (char *restrict, int, FILE *restrict);
int fputc (int, FILE *);
int fputs (const char *restrict, FILE *restrict);
int getc (FILE *);
int getchar (void);
char * gets (char *);
int putc (int, FILE *);
int putchar (int);
int puts (const char *);
int ungetc (int, FILE *);
size_t fread (void * restrict, size_t _size, size_t _n, FILE *restrict);
size_t fwrite (const void * restrict , size_t _size, size_t _n, FILE *);
int fgetpos (FILE *restrict, fpos_t *restrict);
int fseek (FILE *, long, int);
int fsetpos (FILE *, const fpos_t *);
long ftell ( FILE *);
void rewind (FILE *);
void clearerr (FILE *);
int feof (FILE *);
int ferror (FILE *);
void perror (const char *);
FILE * fopen (const char *restrict _name, const char *restrict _type);
int sprintf (char *restrict, const char *restrict, ...) __attribute__ ((__format__
(__printf__, 2, 3))) ;
int remove (const char *);
```

```
int rename (const char *, const char *);
# 235 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 3
int fseeko (FILE *, off_t, int);
off_t ftello ( FILE *);
int asiprintf (char **, const char *, ...) __attribute__ ((__format__ (__printf__, 2, 3)))
;
char * asniprintf (char *, size_t *, const char *, ...) __attribute__ ((__format__
(__printf__, 3, 4))) ;
char * asnprintf (char *restrict, size_t *restrict, const char *restrict, ...) __attribute__
((__format__ (__printf__, 3, 4)));
int asprintf (char **restrict, const char *restrict, ...) __attribute__ ((__format__
(__printf__, 2, 3))) ;
int diprintf (int, const char *, ...) __attribute__ ((__format__ (__printf__, 2, 3)));
int fiprintf (FILE *, const char *, ...) __attribute__ ((__format__ (__printf__, 2, 3))) ;
int fiscanf (FILE *, const char *, ...) __attribute__ ((__format__ (__scanf__, 2, 3)));
int iprintf (const char *, ...) __attribute__ ((__format__ (__printf__, 1, 2)));
int iscanf (const char *, ...) __attribute__ ((__format__ (__scanf__, 1, 2)));
int siprintf (char *, const char *, ...) __attribute__ ((__format__ (__printf__, 2, 3))) ;
int siscanf (const char *, const char *, ...) __attribute__ ((__format__ (__scanf__, 2,
3))) ;
int snprintf (char *restrict, size_t, const char *restrict, ...) __attribute__ ((__format__
(__printf__, 3, 4))) ;
int sniprintf (char *, size_t, const char *, ...) __attribute__ ((__format__ (__printf__,
3, 4))) ;
int vasiprintf (char **, const char *, __gnuc_va_list) __attribute__ ((__format__
(__printf__, 2, 0))) ;
char * vasniprintf (char *, size_t *, const char *, __gnuc_va_list) __attribute__
((__format__ (__printf__, 3, 0))) ;
char * vasnprintf (char *, size_t *, const char *, __gnuc_va_list) __attribute__
((__format__ (__printf__, 3, 0))) ;
int vasprintf (char **, const char *, __gnuc_va_list) __attribute__ ((__format__
(__printf__, 2, 0))) ;
int vdiprintf (int, const char *, __gnuc_va_list) __attribute__ ((__format__
(__printf__, 2, 0)));
int vfiprintf (FILE *, const char *, __gnuc_va_list) __attribute__ ((__format__
(__printf__, 2, 0))) ;
int vfiscanf (FILE *, const char *, __gnuc_va_list) __attribute__ ((__format__
(__scanf__, 2, 0))) ;
int vfscanf (FILE *restrict, const char *restrict, __gnuc_va_list) __attribute__
((__format__ (__scanf__, 2, 0))) ;
int viprintf (const char *, __gnuc_va_list) __attribute__ ((__format__ (__printf__, 1,
0))) ;
int viscanf (const char *, __gnuc_va_list) __attribute__ ((__format__ (__scanf__, 1,
0))) ;
int vscanf (const char *, __gnuc_va_list) __attribute__ ((__format__ (__scanf__, 1,
0)));
int vsiprintf (char *, const char *, __gnuc_va_list) __attribute__ ((__format__
(__printf__, 2, 0))) ;
int vsiscanf (const char *, const char *, __gnuc_va_list) __attribute__ ((__format__
```

```
(__scanf__, 2, 0))) ;
int vsniprintf (char *, size_t, const char *, __gnuc_va_list) __attribute__
((__format__ (__printf__, 3, 0))) ;
int vsnprintf (char *restrict, size_t, const char *restrict, __gnuc_va_list)
__attribute__
((__format__ (__printf__, 3, 0))) ;
int vsscanf (const char *restrict, const char *restrict, __gnuc_va_list) __attribute__
((__format__ (__scanf__, 2, 0))) ;
# 313 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 3
FILE * fdopen (int, const char *);
int fileno (FILE *);
int getw (FILE *);
int pclose (FILE *);
FILE * popen (const char *, const char *);
int putw (int, FILE *);
void setbuffer (FILE *, char *, int);
int setlinebuf (FILE *);
int getc_unlocked (FILE *);
int getchar_unlocked (void);
void flockfile (FILE *);
int ftrylockfile (FILE *);
void funlockfile (FILE *);
int _vsscanf_r
(struct _reent *, const char *restrict, const char *restrict,
__gnuc_va_list) __attribute__ ((__format__ (__scanf__, 3, 0))) ;
int fpurge (FILE *);
ssize_t __getdelim (char **, size_t *, int, FILE *);
ssize_t __getline (char **, size_t *, FILE *);
void clearerr_unlocked (FILE *);int feof_unlocked (FILE *);
int ferror_unlocked (FILE *);
int fileno_unlocked (FILE *);
int fflush_unlocked (FILE *);
int fgetc_unlocked (FILE *);
int fputc_unlocked (int, FILE *);size_t fread_unlocked (void * restrict, size_t _size,
size_t _n, FILE *restrict);
typedef int cookie_seek_function_t(void *__cookie, off_t *__off, int __whence);
typedef int cookie_close_function_t(void *__cookie);
typedef struct{cookie_read_function_t *read;cookie_write_function_t *write;
cookie_seek_function_t *seek; cookie_close_function_t *close;
} cookie_io_functions_t;FILE *fopencookie (void *__cookie,
const char *__mode, cookie_io_functions_t__functions) ;FILE *_fopencookie_r
(struct _reent *, void *__cookie, const char *__mode,cookie_io_functions_t
__functions) ;
# 725 "/opt/ri5cy_gnu_toolchain/install/riscv32-unknown-elf/include/stdio.h" 3
# 8 "deneme.c" 2# 8 "deneme.c
"int main()
{printf("Merhaba Risc-V\n");
return 0;
}
```

Ek-B

```
module PULP_TOP (
exclk,
rst_n,
fetch_enable_sw,
uart_tx,
uart_rx,
gpio_in,
gpio_out,
spi_master_clk_o,
spi_master_csn0_o,
spi_master_dq);
input exclk;
input rst_n;
input fetch_enable_sw;
output uart_tx;
input uart_rx;
input [3:0] gpio_in;
output [3:0] gpio_out;
output spi_master_clk_o;
output spi_master_csn0_o;
inout [3:0] spi_master_dq;
wire
FE_ECO_HOLDN10_pinozed_pulpino_
platform_pulpino_i_peripherals_i_apb_spi_
master_i_spi_ctrl_data_rx_0;
wire
FE_ECO_HOLDN9_pinozed_pulpino_
platform_pulpino_
i_peripherals_i_apb_spi_
master_i_spi_ctrl_data_rx_3;
wire
FE_ECO_HOLDN8_pinozed_
pulpino_platform_pulpino_i_peripherals_i_apb_spi_
master_i_spi_ctrl_data_rx_2;
wire
FE_ECO_HOLDN7_pinozed_
pulpino_platform_pulpino_i_peripherals_i_apb_spi_
master_i_spi_ctrl_data_rx_1;
wire FE_ECO_HOLDN6_GPIO_IN_3;
wire FE_ECO_HOLDN5_GPIO_IN_2;
43
DEL02BWP7T
FE_ECO_HOLDC10_pinozed_pulpino
_platform_pulpino_i_peripherals_i_apb_spi_
master_i_spi_ctrl_data_rx_0
(.I(pinozed_pulpino_platform_pulpino_i_
peripherals_i_apb_spi_master_i_spi_ctrl_d
```

```
ata_rx[0]),
.Z(FE_ECO_HOLDN10_pinozed_pulpino
_platform_pulpino_i_peripherals_i
_apb_spi_master_i_spi_ctrl_data_rx_0));
DEL2BWP7T
FE_ECO_HOLDC9_pinozed_pulpino_platform_pulpino_i_
peripherals_i_apb_spi_master_i_spi_ctrl_data_rx_3
(.I(FE_ECO_HOLDN9_pinozed_pulpino
_platform_pulpino_i_peripherals_
i_apb_spi
_master_i_spi_ctrl_data_rx_3),
.Z(pinozed_pulpino_platform_pulpino_i
_peripherals_i_apb_spi_master_i_spi
_ctrl_data_rx[3]));
DEL2BWP7T
FE_ECO_HOLDC8_pinozed_pulpino_platform_pulpino_
i_peripherals_i_apb_spi_m
aster_i_spi_ctrl_data_rx_2
(.I(FE_ECO_HOLDN8_pinozed_pulpino_platform_pulpino_
i_peripherals_i_apb_spi
_master_i_spi_ctrl_data_rx_2),
.Z(pinozed_pulpino_platform_pulpino_i_peripherals_
i_apb_spi_master_i_spi
_ctrl_data_rx[2]));
DEL2BWP7T
FE_ECO_HOLDC7_pinozed_pulpino_platform_
pulpino_i_peripherals_i_apb_spi_m
aster_i_spi_ctrl_data_rx_1
(.I(FE_ECO_HOLDN7_pinozed_pulpino_platform_
pulpino_i_peripherals_i_apb_spi
_master_i_spi_ctrl_data_rx_1),
.Z(pinozed_pulpino_platform_pulpino_i_peripherals_
i_apb_spi_master_i_spi
_ctrl_data_rx[1]));
DEL2BWP7T FE_ECO_HOLDC6_GPIO_IN_3
(.I(GPIO_IN[3]),
.Z(FE_ECO_HOLDN6_GPIO_IN_3));
DEL2BWP7T FE_ECO_HOLDC5_GPIO_IN_2
 (.I(GPIO_IN[2]),
.Z(FE_ECO_HOLDN5_GPIO_IN_2));
DEL2BWP7T FE_ECO_HOLDC4_UART_RX (.I(UART_RX),
.Z(FE_ECO_HOLDN4_UART_RX));
DEL2BWP7T FE_ECO_HOLDC3_GPIO_IN_1 (.I(GPIO_IN[1]),
.Z(FE_ECO_HOLDN3_GPIO_IN_1));
DEL2BWP7T FE_ECO_HOLDC2_GPIO_IN_0
 (.I(GPIO_IN[0]),
.Z(FE_ECO_HOLDN2_GPIO_IN_0));
DEL2BWP7T FE_ECO_HOLDC1_
FETCH_EANBLE_SW
```

```
(.I(FETCH_EANBLE_SW),
.Z(FE_ECO_HOLDN1_FETCH_EANBLE_SW));
DEL2BWP7T FE_ECO_HOLDC0_RESET (
.I(FE_ECO_HOLDN0_RESET),
.Z(RESET));
```

KİŞİSEL YAYIN VE ESERLER

[1] **Emre G.**, Gökhan I., İnan E., Ali T., Risc-V Tabanlı AES Hızlandırılmış Kırmık Üstü Sistemin Fiziksel Tasarımı, *5.Uluslararası Marmara Fen Bilimleri Kongresi*, Kocaeli, Türkiye, 4-5 Aralık 2020.

ÖZGEÇMİŞ

İlk, orta ve lise öğrenimini Kocaeli'de tamamladı. 2010 yılında Kocaeli Üniversitesi Elektronik ve Haberleşme Mühendisliğine öğrenim görmeye hak kazandı. 2016 yılında Elektronik ve Haberleşme Mühendisi olarak öğrenimini tamamladı. 2017-2021 yılları arasında, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Elektronik ve Haberleşme Mühendisliği Anabilim Dalı'nda Yüksek Lisans öğrenimini tamamladı. Elektronik Mühendisi olarak görev yapmaktadır.