

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ETHERNET SERİ YÖNETİM
PROTOKOLÜNÜN UART İLE
GÖRÜNTÜLENMESİNİN FPGA ÜZERİNDE
GERÇEKLENMESİ

UĞUR AKÇAY
YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE

2020

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ETHERNET SERİ YÖNETİM
PROTOKOLÜNÜN UART İLE
GÖRÜNTÜLENMESİNİN FPGA ÜZERİNDE
GERÇEKLENMESİ

UĞUR AKÇAY
YÜKSEK LİSANS TEZİ
ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
DR. ÖĞR. ÜYESİ ÖNDER ŞUVAK

GEBZE
2020

T.R.
GEBZE TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**IMPLEMENTATION OF VISUALIZATION
ETHERNET SERIAL MANAGEMENT
PROTOCOL WITH UART ON FPGA**

UĞUR AKÇAY
**A THESIS SUBMITTED FOR THE DEGREE OF
MASTER OF SCIENCE**
DEPARTMENT OF ELECTRONIC ENGINEERING

THESIS SUPERVISOR
ASSIST. PROF. DR. ÖNDER ŞUVAK

GEBZE
2020



YÜKSEK LİSANS JÜRİ ONAY FORMU

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 26/06/2020 tarih ve 2020/31 sayılı kararıyla oluşturulan jüri tarafından 20/07/2020 tarihinde tez savunma sınavı yapılan Uğur AKÇAY'ın tez çalışması Elektronik Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE

(TEZ DANIŞMANI) : DR. ÖĞR. ÜYESİ ÖNDER ŞUVAK

ÜYE

: DOÇ. DR. KORAY KAYABOL

ÜYE

: DOÇ. DR. MUSTAFA BERKE YELTEN

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun
...../...../..... tarih ve/..... sayılı kararı.

ÖZET

Bu çalışmada FPGA ile ETHERNET anahtar tümleşik devrelerini konfigüre edebilmek amacıyla kullanılan seri yönetim arayüzü (Serial management interface, SMI) protokolünün gerçekleştirilerek ilgili yönetim arayüzünün UART (Universal Asynchronous Receiver Transmitter) protokolü ile bilgisayar üzerinden bir terminal programı vasıtasıyla görsel olarak işlevselliğinin sağlanması incelenmiştir. FPGA ile gerçekleştirilen seri yönetim arayüzü protokolünün işlevselliğinin kontrolü amacıyla üzerinde ETHERNET anahtar tümleşik devresi bulunan bir elektronik kart kullanılmıştır. Seri yönetim arayüzü protokolünün bilgisayar üzerinde görsel olarak işlevselliğini sağlamak amacıyla da UART-USB dönüşümünü gerçekleştiren bir elektronik kart kullanılmıştır. Bu tezde, birinci aşamada seri yönetim arayüzü protokolü FPGA ile gerçekleştirilerek üzerinde ETHERNET anahtar tümleşik devresi bulunan elektronik karta gerekli bağlantılar yapılarak test işlemleri gerçekleştirilmiştir. Bu test işlemleri, ETHERNET anahtar tümleşik devresi registerlarını kontrol etme işlemleri üzerinedir. İkinci aşamada, FPGA ile gerçekleştirilen seri yönetim arayüzü ve UART arasında doğru iletişimin gerçekleştirilmesi için gerekli işlemler üzerinde durulmuştur. Son aşamada, bir bilgisayar kullanılarak ETHERNET anahtar tümleşik devresinin konfigürasyonlarının görsel olarak bir terminal ekranından gerçekleştirilmesi işlemi gerçekleştirilmiştir. Yapılan çalışmalar sonucunda, ETHERNET anahtar tümleşik devresinin izin verilen registerları aracılığıyla konfigürasyonunun bilgisayar terminal ekranından yapılabildiği görülmüştür.

Anahtar Kelimeler: ETHERNET Seri Yönetim Arayüzü (SMI, MDIO), ETHERNET Anahtar Konfigürasyon, Evrensel Asenkron Alıcı Gönderici (UART), ETHERNET Kontrol Tümleşik Devresi.

SUMMARY

In this work, implementation of Serial Management Interface (SMI) protocol used to configure ETHERNET switch integrated circuits using FPGA and visualization of the functionality of this management interface on a computer using UART (Universal Asynchronous Receiver Transmitter) and a serial console program were examined. An electronic card with ETHERNET switch integrated circuit on it was used to control of the functionality of SMI implemented using FPGA. For visualization of related management interface, an electronic card with a converter that converts UART to USB (Universal Serial Bus) interface was used. In the first step, test operations were examined, after implementing SMI using FPGA, making necessary connections to the electronic card with ETHERNET switch integrated circuit. These test operations are to control of ETHERNET switch integrated circuit's registers. In the second step, necessary operations to provide proper communication between SMI and UART were performed. In the final step, configuration of the ETHERNET switch integrated circuit visually was performed on a serial console screen with a computer. At the results of these examinations, it has been found that the configuration of the ETHERNET switch integrated circuit can be made via the computer serial console screen via allowable registers.

Key Words: ETHERNET Serial Management Interface (SMI, MDIO), ETHERNET Switch Configuration, UART (Universal Asynchronous Receiver Transmitter), ETHERNET Control Integrated Circuit.

TEŞEKKÜR

Başta, yüksek lisans eğitimimde ve akademik hayatımda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanım Dr. Öğr. Üyesi Önder ŞUVAK'a,

Bütün çalışmam boyunca yanımda olan, bilgi ve tecrübelerini benimle paylaşan değerli arkadaşlarım Kadir KÖSE ve Hayret Taha BENDERLİ' ye, en içten teşekkürlerimi sunarım.

İÇİNDEKİLER

	<u>Sayfa</u>
ÖZET	v
SUMMARY	vi
TEŞEKKÜR	vii
İÇİNDEKİLER	viii
SİMGELER ve KISALTMALAR DİZİNİ	ix
ŞEKİLLER DİZİNİ	xi
1. GİRİŞ	1
1.1. Literatür Araştırması	2
1.2. Tezin Amacı ve İçeriği	5
2. ETHERNET ANAHTARLAMA	7
2.1. ETHERNET Anahtar Tümlşik Devresi	7
3. ASENKRON VE SENKRON SERİ HABERLEŞME	10
3.1. Seri Yönetim Arayüzü (SMI, MDIO)	13
3.2. Evrensel Asenkron Alıcı Gönderici (UART)	16
4. FPGA	21
4.1. FPGA nedir, ne amaçla kullanılır?	21
5. SİSTEMİN DETAYLI ANLATIMI VE ÖLÇÜM SONUÇLARI	25
6. SONUÇLAR	36
KAYNAKLAR	38
ÖZGEÇMİŞ	40
EKLER	41

SİMGELER ve KISALTMALAR DİZİNİ

<u>Simgeler ve</u>	<u>Açıklamalar</u>
<u>Kısaltmalar</u>	
ASCII	: American Standard Code for Information Interchange
CTS	: Clear to Send
DSP	: Digital Signal Processor
EMAC	: ETHERNET Media Access Control
FPGA	: Field Programmable Gate Array
FPI	: Flexible Peripheral Interface
GMII	: Gigabit Media Independent Interface
GND	: Ground
GTÜ	: Gebze Teknik Üniversitesi
I2C	: Inter Integrated Circuits
ID	: Identification Number
IEEE	: Institute of Engineers and Everyone Else
I/O	: Input, Output
JTAG	: Joint Test Action Group
LSB	: Least Significant Bit
LUT	: Look Up Table
MAC	: Media Access Controller
MDC	: Management Data Clock
MDIO	: Management Data Input Output
MHz	: Mega Hertz
MII	: Media Independent Interface
MOSFET	: Metal Oxide Semiconductor Field Effect Transistor
OCDS	: On Chip Debug Support
PHY	: Physical Layer Transceiver
RS232	: Recommended Standard 232
RS422	: Recommended Standard 422
RTS	: Ready to Send
RX	: Receive
RTS	: Ready to Send
SMI	: Serial Management Interface

SPI : Serial Peripheral Interface
TX : Transmit
UART : Universal Asynchronous Receiver Transmitter
VHDL : Very High Speed Integrated Circuit Hardware Description Language

ŞEKİLLER DİZİNİ

<u>Sekil No:</u>	<u>Sayfa</u>
2.1: ETHERNET anahtarlama tümleşik devresi genel blok şeması.	8
3.1: Seri iletişim.	10
3.2: Örnek asenkron 8 bit veri gönderimi.	11
3.3: Örnek senkron 8 bit veri gönderimi.	12
3.4: SMI arayüzü temel gösterimi.	13
3.5: SMI hattı ile birden fazla tümleşik devrenin yönetilebilmesi.	14
3.6: SMI hattı örnek bir yazma işlemi.	15
3.7: SMI hattı örnek okuma işlemi.	16
3.8: UART veri hattı genel gösterimi.	17
3.9: UART 8 bit veri formatı gösterimi.	18
3.10: Mini USB-UART çevirici birimi.	19
4.1: Genel olarak FPGA iç yapısı (lojik bloklar, I/O bloklar ve bağlantı yolları) gösterimi.	21
4.2: Metal bağlantı yolları temel iç yapısı gösterimi.	22
4.3: D flip-flop genel gösterimi.	22
4.4: Örnek bir MUX gösterimi.	23
4.5: ALTERA MAX 10 FPGA ve üzerinde bulunduğu kart gösterimi.	24
5.1: Donanımsal yapıların birbirine entegre vaziyette genel gösterimi.	25
5.2: Konsol ekranından adres girilmesi.	26
5.3: Konsol ekranından register adres girilmesi.	27
5.4: Konsol ekranından yazma ya da okuma bilgisinin girilmesi.	29
5.5: Hatalı bir adres girilmesinin ardından konsol ekranı örnek görüntüsü.	31
5.6: Yazma işleminin osiloskop çıktısının gösterimi.	31
5.7: Şekil 5.6'da verilen görselin devamının gösterimi.	32
5.8: Doğru yazma işleminin gösterimi.	33
5.9: Okuma işleminin osiloskop çıktısının gösterimi.	33
5.10: Şekil 5.9'da verilen görselin devamının gösterimi.	34
5.11: FPGA ile çevirici arasındaki örnek UART haberleşmesi gösterimi.	35
5.12: UART üzerindeki herhangi bir verinin iletilme formatı.	35

1. GİRİŞ

Haberleşme ve veri iletiminde, seri ve paralel haberleşme türleri bulunmaktadır. Seri haberleşme belirli bir zamanda 1 bitlik veriyi gönderici taraftan alıcı tarafına göndermeye olanak sağlamaktadır. 1 bitlik durumdan daha fazla veri gönderme işlemi tek bir hat üzerinden ardışık bitler dizisi olarak yapılmaktadır. Paralel haberleşme türü ise belirli bir zamanda birden fazla hat üzerinde olmak kaydı ile birden fazla bitin gönderici taraftan alıcı tarafa gönderilebilmesine olanak tanımaktadır. Günümüzde birçok tümleşik devre diğer başka tümleşik devreler ile haberleşmek ya da başka elektronik kartlar ile haberleşmek için paralel haberleşmenin yanında seri tipte haberleşmeyi de destekleyen protokolleri kullanabilmektedir [10].

Seri tipte haberleşme ve veri iletimini sağlayan yapılar paralel yapıların aksine veri iletiminin sağlandığı hat sayısının az olmasından dolayı çevresel etmenlere karşı daha az duyarlı bir yapıya sahiptir. Hat sayısının az olması durumundan kaynaklı olarak da maliyet, veri iletim mesafesi, senkronizasyon kolaylığı gibi durumlara karşı kolaylık sağlayabilmektedir.

Seri haberleşme türü birçok tümleşik devre yapısında kullanılabildiği gibi ETHERNET kontrol ve anahtarlama tümleşik devrelerinde de kullanılmaktadır. ETHERNET kontrol ve anahtarlama tümleşik devreleri SPI (Serial Peripheral Interface), I2C (Inter Integrated Circuit) gibi seri haberleşmeye olanak sağlayan protokollerin yanında SMI (Serial Management Interface) arayüzüne de sahiptir. İlgili arayüz ETHERNET kontrol ve anahtarlama tümleşik devrelerinin yönetim ve konfigürasyonlarının seri bir veri hattı üzerinden yapılabilmesine olanak sağlamaktadır [8], [9].

SMI arayüzü ETHERNET üzerinden 1 Gbit/s' ye kadar veri iletimini sağlayan ETHERNET kontrol ve anahtarlama tümleşik devrelerinde ilgili tümleşik devre ile haberleşmeye ve iç register'larına ulaşarak yazma ve okuma işlemlerine olanak sunan bir arayüzdür. SMI IEEE 802.3 Clause 22 altında uygulanır.

1.1. Literatür Araştırması

Günümüzde farklı yaklaşımlar kullanarak konfigüre etme ya da hata ayıklama işlemini yerine getiren birçok seri haberleşme türü bulunmaktadır. 2 pinli veri hattı olan MDIO, ETHERNET anahtarlama ya da yönlendirme donanımları içerisinde bulunan ETHERNET anahtar ya da PHY (Physical Layer Transceiver) tümleşik devreleri ile haberleşmede ve bu tümleşik devrelerin iç yazmaçlarına erişerek veri yazma ya da hata ayıklama hususunda yüksek doğrulukta çalışmakta, yaygın olarak kullanılmaktadır [8], [25].

Elektronik tümleşik devrelerin iç yazmaçlarına yapılan veri yazma işlemi yazılımsal olarak ya da ilk açılış (reset) anında, tanımlı olan giriş pinlerine kontrollü olarak uygulanan gerilim seviyeleri ile donanımsal olarak gerçekleştirilebilmektedir. Bu sayede ilgili tümleşik devreler konfigüre edilip, fonksiyonel olarak davranışları ve görevleri değiştirilebilmektedir. Donanımsal olarak tümleşik devrelerde pin sayısı sınırlı olması gerektiğinden çoğu tasarımcı tarafından yazmaçlara veri yazma ya da yazmaçlardan veri okuma yaklaşımı benimsenmektedir. Diğer elektronik tümleşik devrelerde olduğu gibi ETHERNET üzerine geliştirilmiş olan tümleşik devrelerin de istenilen fonksiyonellikte kullanılabilmesi için tipik olarak fonksiyonel çalışma öncesinde ya da çalışma esnasında yönetilip, konfigüre edilmeleri gerekmektedir [6], [17].

ETHERNET anahtarlama tümleşik devreleri Fast ETHERNET, Gigabit ETHERNET özellikteki veri iletişimi için MII (Media Independent Interface), GMII (Gigabit Media Independent Interface) ve benzeri arayüzler ile işlemciler içerisinde bulunan MAC yapısına (Media Access Controller) bağlanarak, ilgili işlemci yapısı ve kendileri arasında arayüz oluşturmaktadırlar. Ancak, oluşturulan MAC arayüzü çoğunlukla durum yazmaçlarını okuma ya da bu yazmaçlara veri yazma amacıyla değil sadece veri iletimi amacıyla kullanılır. ETHERNET PHY ya da anahtarlama tümleşik devreleri için durum yazmaçlarının kontrolü ve erişimi amacıyla kullanılan arayüzler veri iletim arayüzlerinden ayrılmıştır [25].

Verilerin yazmaçların içerisine yazılabilmesi için farklı yöntemler bulunmaktadır. Bunlardan ilki, kontrol yazmacının elektronik eleman içerisinde belirli bir adrese konumlandırılması yaklaşımıdır. Bu yöntem flash bellek gibi birimlerde ortak olarak görülmektedir. Bu yaklaşımda, yazmaç içerisine veri yazmak

amacıyla, yönetici belirli bir hafıza konumunu işaret eder ve yazılacak olan veriyi gönderir. İkinci olarak, yazmaçların kontrolü için belirli bir veri iletim yolu yapısının kullanılması yaklaşımıdır. Elektronik tümleşik devre içerisinde belirli pinler bu amaca yönelik olarak atanmıştır. Yönetici eleman, yönetilecek olan elemana verileri yazmak için bu pinleri kullanır. Üçüncü yöntem ise, bir yol üzerinde birden fazla aynı görevi yerine getirmek üzere bulunan tümleşik devrelerden o anda iletişime geçilecek olan yapının seçiminin yapılarak aktif edilmesi diğerlerinin seçilmeyerek pasif durumda bırakılması yaklaşımıdır [14]-[16].

Wu ve arkadaşları tarafından gerçekleştirilen çalışmada ETHERNET arayüzünün RS232 (Recommended Standard 232) ya da RS422 (Recommended Standard 422) gibi seri bir arayüze dönüştürülmesi söz konusudur. Bir bilgisayara bağlanılarak bu bilgisayardan ETHERNET konektörü aracılığıyla analog olarak alınan veriler bir ETHERNET PHY yapısına götürülerek burada sayısallaştırma işlemi gerçekleştirilmiştir. İlgili sayısal sinyaller SPI arayüzü kullanılarak bir mikrokontrolör birimine gönderilerek burada ilgili dönüşüm işlemi yapılarak seri port üzerinden ETHERNET verilerinin elde edilmesi ya da tam tersi şekilde seri porttan alınan verilerin de tersi bir süreç izlenerek bilgisayara ETHERNET üzerinden gönderilmesi işlemi gerçekleştirilmiştir. Kullanılan bir program aracılığıyla da seri port üzerinden bilgisayara gönderilen ya da ETHERNET üzerinden bilgisayardan gelen verilerin görüntülenmesi işlemi gerçekleştirilmiştir. Bu sayede hata ayıklama işlemi gerçekleştirilmiştir. Normalde PHY ya da anahtar yapılarında veri iletimi ve kontrol, hata ayıklama işlemleri birbirinden bağımsız arayüzler vasıtasıyla gerçekleştirilmektedir. Bu sayede veri iletimi için kullanılan fiziksel port ETHERNET paketlerinin iletimi dışında meşgul edilmeyerek yüksek seviyede anahtarlama ve yönlendirme işlemi gerçekleştirilebilmektedir. Çalışmada önerilen yapı birden fazla tümleşik devre yapısının kontrol ve hata ayıklama işlemleri için kullanılması halinde her bir tümleşik devrenin sürdüğü fiziksel ETHERNET port gruplarından birine bağlanacak olan, çalışma kapsamında gerçekleştiren ve içerisinde mikrokontrolör, ETHERNET PHY gibi birimlerin bulunduğu yapı gerekmektedir. ETHERNET paketlerinin iletimi için kullanılan port üzerinden kontrol ve yönetim verilerinin de iletilmesinden dolayı ilgili portun iletebileceği ETHERNET veri paketi sayılarında düşüş olacaktır [15].

Verilen yaklaşımlardan belirli bir veri iletim yolu oluşturularak yazmaçların kontrolü, hata ayıklama gibi işlemler için kullanılan ve IEEE 1149.1 altında

tanımlanmış en eski yapılardan birisi de JTAG (Joint Test Action Group) arayüzüdür. İlgili yapıda tümleşik devrelerin JTAG arayüzüne ait veri çıkış pininin, diğerinin veri giriş pinine bağlanması suretiyle bir zincir yapısı oluşturulduğunda üreticilerinin hatta bulunan tümleşik devrelere sağladığı ID'ler (Identification Number) kullanılarak iletişim sağlanmakta ve bu sayede ilgili tümleşik devrelerin hata ayıklama, yazmaç kontrolü gibi işlemleri gerçekleştirilebilmektedir. Ek olarak, saat darbesi ve test modu seçimi çıkışı JTAG arayüzüne bağlanan tümleşik devrelere götürülmektedir. Minimum 3 çıkış, 1 girişi olan bir yapı olarak karşımıza çıkmaktadır. JTAG hattı üzerine birden fazla değişik üreticiye ait tümleşik devre bağlanabilir. Ancak JTAG daha karmaşık ETHERNET anahtarlar ya da işlemciler gibi büyük yapılarda genellikle karşımıza çıkmaktadır. Daha alt seviye özellikteki PHY ya da ETHERNET anahtarlar da çok sık rastlanmamaktadır [4], [5],[14], [16].

Maier tarafından yapılan çalışmada, belirli bir üreticiye ait 16 ve 32 bit mikroişlemci ve DSP (Digital Signal Processor) çekirdeklerinin konfigürasyon, hata ayıklama işlemlerine yönelik olarak yazılımsal bir arayüz gerçekleştirilmiştir. Çalışmada mevcut standart JTAG çekirdek yapısı korunmuştur. Bu yapıya ek olarak ilgili yapının işlemci içerisinde bulunan birden fazla çekirdekle haberleşmesini sağlayan ve bu çekirdeklere yapacağı okuma/yazma ile konfigüre etme, hata ayıklama işlemlerini yerine getiren birden fazla I/O (Input, Output) modülü tasarlanarak standart JTAG modülü genişletilmiştir. Bir I/O modülü ile birden fazla işlemci çekirdek yapısına erişim sağlanmaya çalışılmıştır. Erişim için 16 ve 32 bit veri iletimi yapısına uygun ortak FPI (Flexible Peripheral Interface) veri yolu kullanılmıştır. I/O modülü çekirdek yapılarının içerisindeki OCDS (On Chip Debug Support) modüllerinin yazmaçlarına erişim sağlayarak çekirdeklerin konfigürasyon ve iç durumlarına erişim işlemi sağlanmaktadır. FPI veri yolunun olmadığı durumlarda bir I/O modül ile bir işlemci çekirdek yapısı da direkt olarak bağlanarak kontrol edilebilmektedir. Yapısal olarak çoklu yöneticiye sahip olan bir veri yolu (Multi Master Bus) üzerinde durulmuştur. Burada birden fazla I/O modülün işlemci çekirdeklerine erişim sağlamak amacıyla kullanılmak istenmesi durumu da söz konusu olabilmektedir. Bu durumda belirli bir anda bir adet I/O modülünün veri hattı üzerinde söz sahibi olacak şekilde çalışma gerçekleştirilmiştir. Modüler bir yapı olarak birden fazla I/O modül standart JTAG modülüne bağlanmakta ve bu standart JTAG modül yapısı da tek bir fiziksel arayüz üzerinden dış dünya ya bağlantı sağlamaktadır. Çalışma belirli bir üreticiye yönelik olarak gerçekleştirildiğinden

retici bağımlı olan bir neri sunmaktadır. 16 ve 32 bitlik mikroişlemci yapıları için çalışmanın gerçekleştirilmesinden dolayı platform bağımlı bir durum söz konusudur. Ayrıca, birden fazla I/O modlnn 16 ya da 32 bitlik veri yoluna bağlanması ve bu veri yolunun da birden fazla işlemci çekirdeğine bağlanması veri yolu yükn de artırmaktadır [14].

Hu ve arkadaşlarının gerçekleştirmiş olduėu çalışmada, çok çekirdekli DSP'lerin çekirdeklerinin kontrol ve hata ayıklama işlemlerinin yazılımsal olarak gerçekleştirilmesi söz konusudur. Bir USB/JTAG simlatr kullanılarak DSP'yi zerinde bulunduran hedef kart ve bilgisayar arasında iletişim sağlanmış ve okuma/yazma işlemleri gerçekleştirilmiştir. Çalışma kapsamında standart JTAG yapısında herhangi bir donanımsal olarak farklılığa yol açmadan klasik bir JTAG zmlemesi ile bu işlem gerçekleştirilebilmiştir [16].

reticiler tarafından gereklilikler doėrultusunda fonksiyonellik aısından karmaşık yapıda ve büyük tmleşik devrelerin tasarımı ve retimi giderek artmaktadır. Bu karmaşık tmleşik devreleri yksek doėrulukta kontrol, konfigre etme ve hata ayıklama amacıyla gerekli mekanizmaların tasarımlarının yapılması ve mevcut tasarımlarında geliştirilmesi konuları olduka karmaşık olabilen konulardır. Bu amaçla tez kapsamında ETHERNET anahtarlama ya da PHY tmleşik devrelerine ynelik olarak grsel bir şekilde kontrol, konfigrasyon ve durum yazmalarının grntlenmesi zerine çalışma gerçekleştirilmiştir. İlgili çalışma gerçekleştirilirken tamamen fiziksel ETHERNET arayzlerinden bağımsız olarak tmleşik devre ierisindeki SMI modl ile haberleşme sağlayabilen modler bir adaptr tasarımı yapılmıştır. Hem yazılımsal hemde bu yazılımı destekleyen bir donanım yapısı oluşturulmuştur. Bu sayede veri iletim dzlemi ve kontrol dzlemi birbirinden bağımsız bir şekilde kullanılmıştır. Çalışma kapsamında retici ve platform farketmeksizin farklı karmaşıklık ve fonksiyonellikteki ETHERNET anahtarlama ya da PHY tmleşik devrelerine ynelik yazmalara erişim sağlanarak kontrol ve durum grntlenmesi işlemleri gerçekleştirilebilmektedir.

1.2. Tezin Amacı ve İeriėi

ETHERNET anahtarlama ve kontrol tmleşik devreleri gnmzde ynetilebilen ya da ynetimsiz ETHERNET anahtarlama, ynlendirici, IP telefon,

IPTV, kablosuz modem ve daha birçok cihazda güvenli, yüksek hızlı, yüksek veri kapasiteli kablolu ağ yaratmak amacıyla kullanımı açısından büyük önem taşımaktadır. İlgili tümleşik devreler ağ yapısı içerisinde yer alan tüm cihazların birbiri ile haberleşmesine ve veri transferine olanak sağlamaktadır.

Yapılan uygulamalar ETHERNET anahtarlama tümleşik devrelerinin üzerinde bulundukları elektronik karta SMI arayüzü için gerekli bağlantıların yapılarak ilgili tümleşik devrelerin platform bağımsız olarak bir adaptör vasıtasıyla yönetilebilme ve konfigüre edilebilme amacı doğrultusunda olanak sağlamaktadır. Ancak, ilgili arayüze ait saat (MDC, Management Data Clock) ve veri hattı (MDIO, Management Data Input Output) donanım üzerinde bir sıralı pin, konektör gibi dışarıdan bağlantı sağlayabilecek bir yapıya elektronik kart üzerinde götürülmüş olmalıdır.

Bu tez kapsamında, ikinci bölümde ETHERNET anahtarlama, ETHERNET anahtarlama tümleşik devreleri ve genel olarak yapıları hakkında bilgiler verilmiştir. Üçüncü bölümde, senkron ve asenkron seri haberleşme türleri, ETHERNET anahtarlama tümleşik devrelerini yönetmek ve konfigüre etmek amacıyla kullanılan SMI arayüzü ve UART anlatılmıştır. Dördüncü bölümde, FPGA hakkında genel bilgi ve tez kapsamındaki çalışmalarda kullanılan FPGA anlatılmaktadır. Beşinci ve son bölümde, FPGA ile gerçekleştirilen seri yönetim arayüzünden ETHERNET anahtar tümleşik devresinin iç registerlarına yapılan yazma ve registerlarından yapılan okuma (konfigürasyon ve yönetim) işlemleri, yapılan çalışmalara dair detaylı anlatımlarla, görseller ve blok diyagramlarla desteklenerek verilmiştir. Elde edilen sonuçlar ve geleceğe dair öneriler anlatılmıştır.

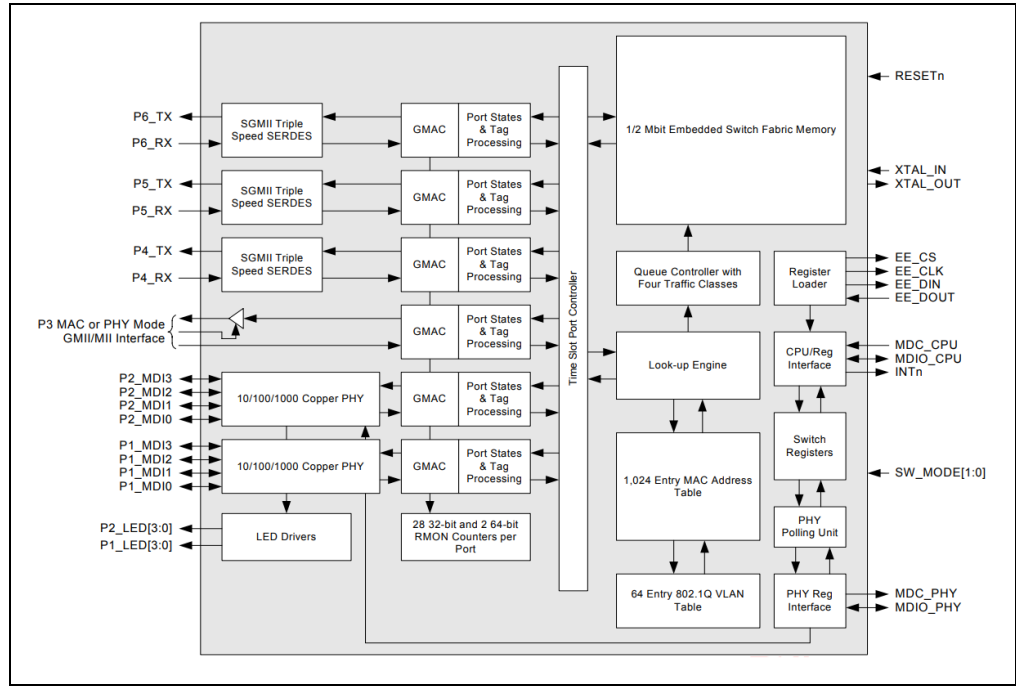
2. ETHERNET ANAHTARLAMA

ETHERNET ilk olarak tek eş eksenli bir kablo vasıtası ile birbirine bağlanan bilgisayarların haberleştirilmesi temeline dayanmaktaydı. ETHERNET bu ilk zamanlardaki durumundan günümüzdeki pek çok yerel ağ alt yapısını oluşturan karmaşık bükülü tel çifti ağ teknoloji yapısına dönüşmüştür. İletilecek verinin kapasitesi, ağ yapısındaki cihaz çeşitliliğinin artması sebebiyle de ETHERNET anahtarlama işlemi büyük önem kazanmıştır [7]-[9].

Genel bir tanım olarak, ETHERNET anahtarlama, bir girişten (porttan) gelen verinin bir veya birden fazla çıkışa (porta) iletilmesini sağlayan işlemler bütünü olarak nitelendirilebilir. ETHERNET anahtarlama işlemi, donanımsal olarak anahtarlama mekanizmaları ve bu anahtarlama yollarının kontrolü görevini gören yazılım birimleri ile sağlanmaktadır. Bu sayede, ETHERNET anahtarlama işlemi sayesinde bir yapıdan başka bir yapıya ETHERNET paketleri iletilerek tek bir ETHERNET yolu üzerinden daha geniş ağa ulaşım sağlanır. Son zamanlarda, hız, yüksek veri iletim kapasitesi, güvenlik gibi sebeplerden ötürü otomotiv, veri hızının önemli olduğu fabrika robot otomasyonu, ulaşım gibi alanlarda alt yapılar ETHERNET haberleşmesi ile revize edilmektedir.

2.1. ETHERNET Anahtar Tümleşik Devresi

ETHERNET anahtarlama işlemini gerçekleştiren cihaz türlerine ait donanımları incelediğimizde her bir cihaz elektronik kartı üzerinde anahtarlama ve ETHERNET haberleşmesini sağlamak amacıyla kullanılmış olan tümleşik devrelere sahiptir. Bu tümleşik devreler kullanım yeri ve ihtiyaçlar doğrultusunda ETHERNET hatları üzerinden farklı veri gönderme/alma hızlarına, farklı port sayılarına, dış veya iç ortamlara dayanım açısından farklı birçok özelliklere, kullanıcıya sunduğu müdahale imkanlarına göre yönetilebilen ya da yönetilemeyen konfigürasyona, donanım üzerinde koşacak protokol ve katmana göre farklı kabiliyetlere sahip olabilmektedir. Genel hatlarıyla bir ETHERNET anahtarlama tümleşik devresine ait en genel seviyeden gösterim sunan bir blok diyagram Şekil 2.1’de verilmektedir [21].



Şekil 2.1: ETHERNET anahtarlama tümleşik devresi genel blok şeması.

Şekil 2.1’de görüldüğü üzere temel olarak ETHERNET anahtarlama tümleşik devresinin iç yapısında, hafıza bloğu, MDIO modülü, ETHERNET ortam erişim kontrolörü (EMAC), fiziksel katmana çıkış için kullanılan PHY gibi yapılar bulunmaktadır. Bu yapılar ETHERNET anahtarlama ve benzeri bir işlem amacıyla kullanılan cihaza dışarıdan bağlanan herhangi başka bir cihaz ile ETHERNET protokolüne uygun olarak, iki yapı arasında meydana gelen trafiğin belirli hızlarda kontrollü olarak akıllı bir şekilde yönetilmesi işlemlerini yapan doğru konfigürasyonlar doğrultusunda yerine getirmektedir [7].

Tümleşik devre içerisinde bulunan temel yapılar incelendiğinde, bakır PHY ya da diğer bir ismi ile Transceiverlar donanımın ETHERNET framealarının gönderilip, alınması işlemini yerine getirmesini sağlamaktadır. Optik fiber veya bakır ETHERNET kablosu gibi fiziksel ortama Link katmanındaki bir yapının (EMAC) bağlanmasını sağlamaktadır. Paket sinyallerinin analog ortam (kablo ve bağlandığı port) ve dijital ortam arasında analog’tan dijitale dönüşümünü sağlamaktadır [8], [21]. İlgili yapı tümleşik devre üretim süreçlerindeki gelişmeler sonucunda transistör boyutlarının küçülmesi sayesinde artık çoğu ETHERNET anahtar tümleşik devresinin içerisinde gömülü olarak bulunabilmekle birlikte, sadece EMAC ve PHY arasındaki gerekli arayüz tümleşik devre dışına çıkılarak ayrı bir tümleşik devre

olarak da donanım tasarımı yapılırken ETHERNET anahtar tümleşik devresine bağlanabilmektedir.

MAC bloğu, paket sınırlandırma ve tanıma, varış noktalarının ya da düğümünün adreslenmesi, kaynak noktalarının ya da düğümünün adreslenmesi, frame sırasını kontrol ederek hatalara karşı koruma ve fiziksel iletim ortamına (transceiver veya PHY) erişimin kontrol edilmesi gibi görevleri yerine getirmektedir [8], [21].

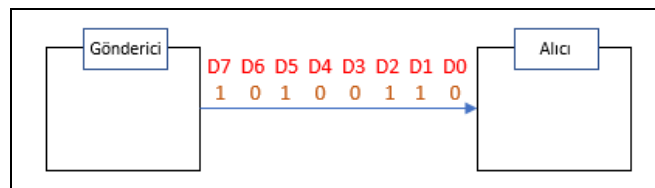
Genel olarak ETHERNET anahtar tümleşik devrelerinin içerisinde dahili olarak boyutları çok büyük olmayan genellikle Mbit veya daha az boyut mertebesinde olan hafıza bloğu bulunmaktadır. Hafıza bloğu, tümleşik devrenin konfigürasyonu ve benzeri amaçlarla iç registerlara yazılan bilgileri tutmaktadır. Tümleşik devre izin verilen iç registerlara bir kere bilgi yazıldığında her bir çalışma durumunda bu hafızadan register bilgilerini okuyarak kendisini konfigüre edebilmektedir [8], [21].

ETHERNET anahtarlama tümleşik devreleri, konfigürasyon ve iç registerlara kullanıcı erişiminin sağlanması amacıyla dışarıya çıkılmış olan seri yönetim arayüzüne sahiptir. İlgili arayüz 2 hatlı seri yönetim arayüzü (SMI) olarak adlandırılmaktadır. Bu çalışma kapsamında anlatılanlara uygun olması ve SMI protokolünün işlevliğinin uygulanması amacıyla MARVELL 88E6065 10/100 Mbit/s ETHERNET anahtar tümleşik devresi kullanılmıştır.

3. ASENKRON VE SENKRON SERİ HABERLEŞME

Veri iletimi ya da haberleşme aslen bir verinin bir yerden başka bir yere sağlıklı bir şekilde gönderilmesi demektir. Bu amaçla, elektronik cihazların birbiri ile veri transferi yapabilmeleri amacıyla birçok haberleşme protokolü ve bu protokollerin sağlıklı bir şekilde çalışması için çeşitli donanım yapıları ortaya koyulmuştur. Bu haberleşme türleri günden güne geliştirilmiş ve birbirlerinden belirli bir zaman aralığında gönderebildikleri veri sayısı (bit, bayt vb.), karşıya bağlanan donanım ile ne şekilde haberleşme yaptıkları, haberleşme işlemi sadece belirli bir elektronik kartın üzerindeki 2 veya daha fazla tümleşik devre arasında gerçekleşmeyip kablolar vasıtasıyla birbirinden uzak olan cihazların arasında gerçekleşiyorsa sağlıklı erişimin olduğu mesafe, maliyet ve benzeri konulardan dolayı bazı protokoller diğerlerine göre üstünlük sağlamıştır. Bu tür kullanışlı protokoller ve onlara ait donanımlar günümüzde birçok donanımın üzerindeki elektronik kartta koşturmakta ve ekipmanları kullanılmaktadır. Örnek verecek olursak, I2C, SPI, UART, SMI gibi seri haberleşme arayüzleri yakın zamanımızda çok yüksek hızda veri iletiminin gerekmediği yerlerde kullanılmakta olup, genel olarak çoğu tümleşik devrenin iç yapısında bu protokollerden en az bir veya birkaçı gömülü olarak bulunabilmekte ve diğer tümleşik devreler ile haberleşebilmek amacıyla kullanılmaktadır. Ancak bazı protokoller kullanım alanlarına ve amaçlarına göre belirli fonksiyonlar için özelleşmiş durumda olabilmektedir.

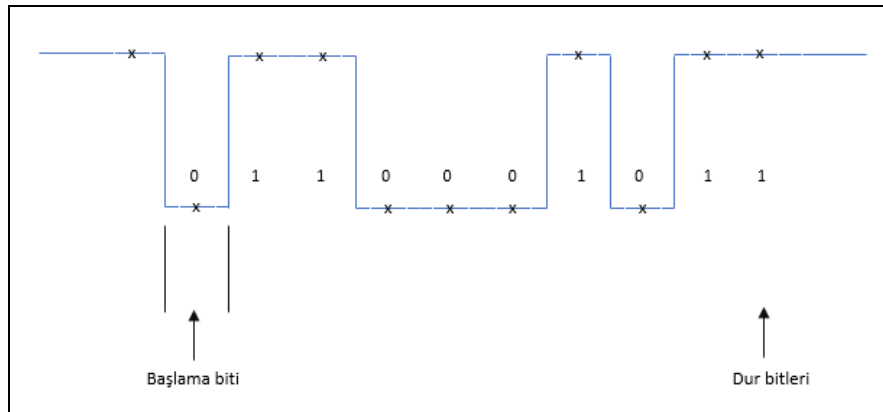
Seri haberleşme türünde belirli bir anda bir bitlik veri tek bir hat ya da kanal üzerinden ardışık bitler dizisi olarak gönderilmektedir. Bu ardışık bit dizileri varış noktalarına ulaştığında oradaki bir akıllı birim tarafından paralel hale getirilerek anlamlandırılmaktadırlar. Şekil 3.1’de temsili bir gösterim verilmektedir.



Şekil 3.1: Seri iletişim.

Seri haberleşmede veriler tek bir hat üzerinden gönderilip alınmayabilir. Veriler bir saat darbesine senkron bir şekilde gönderilip alınması halinde hat sayısı birden fazla olmaktadır. Seri haberleşmede bit dizileri bir hat üzerinden gönderildiğinden dolayı bu bitlerin hatta ne kadar süre kalacağı belirli olmalıdır. Çünkü iletişim hızı belirlenmesi işlemi bu sürelerle göre yapılmaktadır. Seri haberleşmede iletişim hızı baud yani bit(ler)/saniye olarak ifade edilmektedir. 1 saniyede hattan geçen bit sayısı demektir. Örnek verilecek olursa; 2400 baud = $1/2400 = 0.46$ milisaniye etmektedir. Sonuç olarak her bir bit hatta 0.46 milisaniye kalmalıdır. Bu sebeple, veri iletimi belirlenmiş standartlara göre gerçekleştirilmektedir. İki birim arasında sağlıklı bir haberleşmenin gerçekleştirilebilmesi için iki tarafta hız olarak birbirine uyumlu olmalıdır.

Seri haberleşme kendi içerisinde iki türe ayrılmaktadır. Bunlar asenkron ve senkron seri haberleşme türleridir. Asenkron haberleşme türünde isminden de anlaşılacağı gibi birbiri ile iletişim haline geçecek olan en az 2 yapı belirli bir saat darbesi yapısına bağımlı olmadan veri gönderip alma işlemini gerçekleştirirler. Her 2 yapının da saat darbeleri bağımsızdır. Bu sebeple, herhangi bir zamanda veri gönderilebilir ya da alınabilir. Saat darbesi hattının olmaması sebebiyle hat sayısı azalmıştır fakat hattın aynı anda çift yönlü olarak veri gönderip alabilmesi (full duplex) durumuna göre hat sayısı birden fazla olabilmektedir [1],[2]. Şekil 3.2’de asenkron olarak veri gönderimine ait 8 bitlik bir verinin gönderimi verilmektedir.

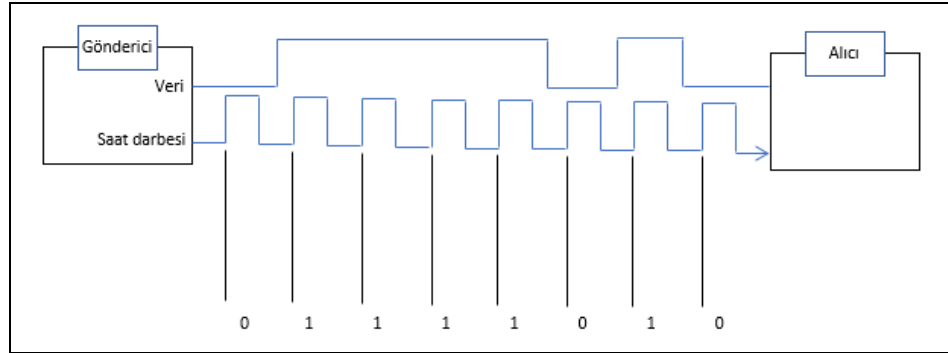


Şekil 3.2: Örnek asenkron 8 bit veri gönderimi.

Şekil 3.2’den de görüldüğü gibi herhangi bir haberleşme durumu yoksa asenkron seri haberleşmede hat lojik 1 seviyesinde tutulur. Bu durumdan sonra gelen ilk lojik 0 durumu yani 1 bitlik 0 durumu başlama bitidir. Veri transferi başlar. Başlama bitinin

ardından 8 bitlik veri gönderilmeye başlanır. Veri grubunun gönderilmesinin hemen ardından dur bitleri gönderilmektedir. Dur bit sayısı 1 veya 2 olabilmektedir. Sinyallerin genlik seviyeleri haberleşme arayüzüne göre farklılık gösterebilmektedir. Örnek verilecek olursa, UART için lojik seviyeler +3.3V ve 0V olabilirken RS-232 için bu seviyeler +15V ve -15V arasında olabilmektedir. Ayrıca, bazı arayüzler için düşük voltaj seviyesi lojik 1, yüksek voltaj seviyesi de lojik 0 olarak ters bir şekilde kabul edilerek haberleşme sağlanmaktadır. Asenkron seri haberleşme de bir saat yapısına senkron olarak veri gönderimi gerçekleştirmediği için daha yavaş veri iletimi ihtiyacı olan durumlarda kullanılmakla beraber veri kaybı meydana gelme olasılığı da mevcuttur.

Diğer seri haberleşme türü ise senkron haberleşme şeklidir. Senkron tipte, alıcı ve verici kısım aynı saat darbesini kullanmaktadır. Alıcı her bir saat darbesi geldiğinde hattı kontrol eder. Saat darbesi esnasında hatta lojik 1 ya da lojik 0 olması durumuna göre bunu almaktadır. Hem saat darbeleri hem de veri gönderici taraftan çıkarıldığı için alıcı kısım sadece verici kısımdan herhangi bir veri gönderilmesi durumunda hattı okumaya çalışır. Bir sonraki saat darbesi gelene kadar hattı okumaz [3]. Şekil 3.3’de temel düzeyde örnek bir senkron haberleşme diyagramı verilmektedir.



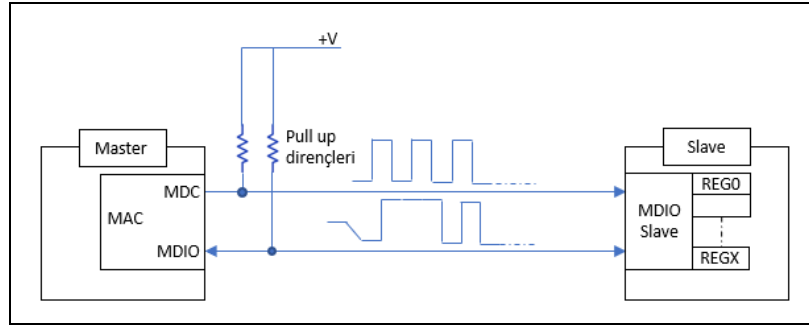
Şekil 3.3: Örnek senkron 8 bit veri gönderimi.

Veri gönderim işlemi saat darbesinin hızına bağlı bulunduğundan dolayı hızlı bir veri iletimi gerçekleştirilebilir. Hız ifadesi olarak her bir saat darbesinde veri gönderimi yapıldığı için saat darbesinin frekans değeri hız olarak ifade edilmektedir.

3.1. Seri Yönetim Arayüzü (SMI, MDIO)

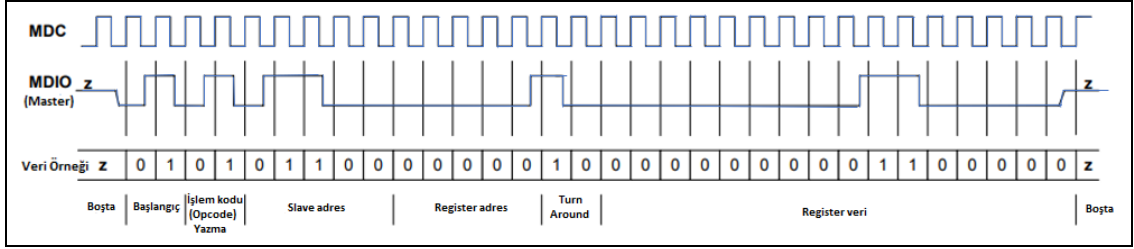
Seri Yönetim Arayüzü IEEE 802.3u Clause 22 standardı altında uygulanmaktadır. İlgili veri iletim arayüzü genellikle ETHERNET anahtarlama tümleşik devrelerinde fiziksel katman ve EMAC arasında yönetim açısından bir arayüz sağlamak üzere kullanılmaktadır. ETHERNET anahtar içindeki registerlara ulaşılıp bu registerların değiştirilmesini ya da mevcut durumlarının görülmesini sağlamaktadır. Bu sayede tümleşik devrenin çoğu çalışma modunun konfigüre edilmesini sağlamaktadır [7]-[9].

Donanım yapısı olarak 2 kablolu bir hat (pin) yapısına sahiptir. Bunlardan birisi veri hattı diğeri de saat darbesi hattıdır. Veri hattı genel olarak MDIO (Management Data Input Output), saat darbesine ait hat ise MDC (Management Data Clock) olarak literatürde adlandırılmaktadır. Şekil 3.4’de temel düzeyde olmakla beraber SMI hattının yapısı verilmektedir.



Şekil 3.4: SMI arayüzü temel gösterimi.

İlgili arayüz de hattın saat darbesini, kontrol bitlerini süren bir yönetici (Master) yapısı ve diğer tarafta da registerlarına yazma ve registerlarından okuma yapılacak olan yönetilen (Slave) yapısı bulunmaktadır. Yönetici yapısı bir işlemci, bu çalışma kapsamında kullanıldığı gibi bir FPGA, mikrokontrolör ve benzeri bir birim olabilmektedir. Yönetici olarak isimlendirilmesinin sebebi iki birim arasında veri iletimi gerçekleştirilmesi esnasında gönderilen kontrol bitleri ve saat yapısının belirli bir birimden gönderilmesidir. Veri hattı üzerinde çift yönlü (bidirectional) olarak veri akışı sağlanabilmektedir. Örneğin; yazma işlemi esnasında hatta yönetilen tarafa ait register içerisine yazılacak olan değerleri yöneten taraf göndermekte olup, okuma işlemi esnasında da registerdaki bilgiyi yönetilen taraf veri hattına göndermektedir.



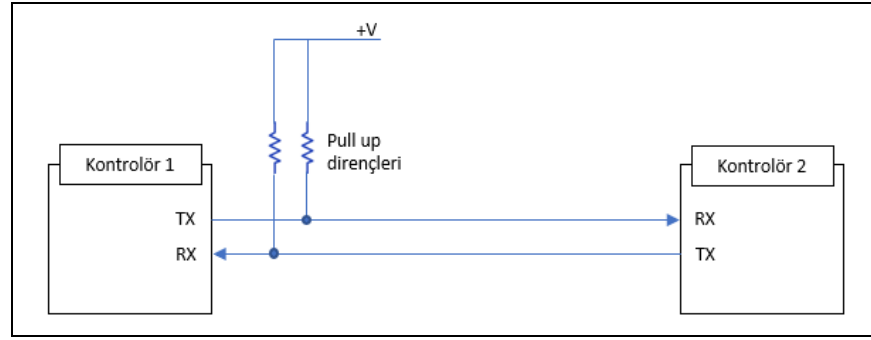
Şekil 3.6: SMI hattı örnek bir yazma işlemi.

SMI arayüzü veri formatında ilk olarak, herhangi bir veri iletimi yapılmıyorsa yönetici tarafın ilgili veri pini High-Z durumunda, hat ise lojik seviye olarak gerilime çekildikleri için sürekli lojik 1 seviyesindedir. MDIO hattında bulunan verilerin durumu saat darbelerinin lojik 1 olduğu esnada ne ise veri formatında o seviyeler geçerli olmaktadır. O sebeple saat hattı ve veri hattı arasındaki zamanlama farkları tümleşik devrenin müsaade ettiği ölçülerde olmak zorundadır. Aksi takdirde zamanlama hatasından dolayı istenilen yazma ve okuma işlemleri yapılamamaktadır.

Yazma işlemini başlatmak için ilk olarak yönetici (Master) taraftan hatta 2 bitlik başlangıç biti 01 olarak gönderilir. Bu bit dizini Clause 22’de standarttır. Hat lojik 1 seviyesinde iken gelen 2 bitlik dizin veri hattında bir işlem olacağını belirtmektedir. Bu işlemi takiben, yazma ya da okuma işlemlerinden hangisinin yapılacağını belirten 2 bitlik işlem kodu (Opcode) gönderilir. 01 yazma işlemi, 10 okuma işlemi ifade etmektedir. Kontrol bitlerinin gönderilmesinin ardından yönetilecek olan yapının bir anlamda kimliği olan 5 bitlik adres bilgisi hatta yer alır. Bu adres bilgisi yönetilecek olan tümleşik devrenin fonksiyonel olarak özelliklerini belirten bilgilendirici tasarım rehberlerinden elde edilmelidir. Veri gönderilecek yapının adresinin girilmesini takiben bu yapıya ait içerisine yazma işlemi yapılacak olan registerin adresinin veri hattında yer alması işlemi gerçekleştirilmektedir. Bu adres bilgisi de 5 bitlik bir dizidir. Sonraki durumda hatta yönetici tarafındaki MDIO sürücü tarafından turn around olarak bilinen 2 bitlik dizin gönderilir. 2 bit zamanı kadar hat bu turn around durumunda kalır. Hatta sırasıyla 10 bit dizini gönderilir. SMI veri iletimindeki veri formatında yazma işleminde en son adım ise daha önceden adresi verilen registera hangi bilgilerin yazılacağıdır. Bu bilgi 16 bitlik bir dizin halinde veri hattına gönderilir ve hat boşta bırakılır. Yönetici taraftaki MDIO sürücünün ilgili pini High-Z durumuna çekilir. Başka veri gönderilmemesi halinde hat sürekli lojik 1 düzeyinde kalır.

Okuma işlemine ait veri iletim yapısı Şekil 3.7’de verilmektedir [21].

her değişimde kendi iç saatlerini tekrar kendilerine özgü senkronize ederler. Örneğin Gönderici UART kontrolör 1'den UART kontrolör 2'ye bir veri gönderildiğinde hattın ilk olarak lojik olarak 0'a çekilmesi veri iletişiminin başlangıcının habercisidir. Karşı alıcı taraf hatta bu değişikliği algıladığında kendi iç saat yapısını hemen tekrar senkronize eder ve başlangıç bitinin gelmesinin ardından gelecek olan verileri kendi iç saatinin her bir lojik 1 olduğu durumda anlamlandırıp mevcut durumlarını kaydeder bir anlamda hattı okur. UART veri iletişiminin sağlıklı bir şekilde yapılmasını sağlayan donanım yapısı Şekil 3.8'de verilmektedir.

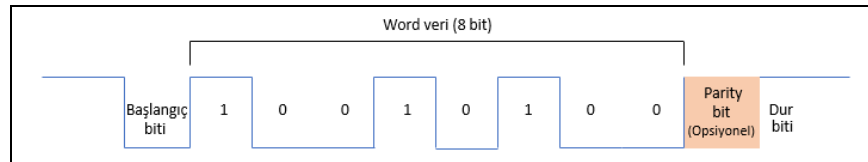


Şekil 3.8: UART veri hattı genel gösterimi.

Şekil 3.8'de verilen yapı incelendiğinde, hattı lojik 1 seviyesinde tutan pull up dirençleri görülmektedir. Aslen birçok tümleşik devre içerisinde iç yapıda UART kontrolörlerin gönderici ve alıcı pinleri lojik 1 seviyesinde tutulabilmektedir. Ancak dış etkiler, hattın uzunluğu ve benzeri durumlar da göz önüne alınarak veri hattının tasarlanan elektronik kart üzerinde de lojik 1 seviyesine belirtilen dirençler yardımıyla çekilmesi sağlıklı olacaktır. Kontrolör 1'in gönderici pini karşı taraftaki donanımda Kontrolör 2'nin alıcı pinine bağlanmalıdır. Aynı şekilde tam tersi Kontrolör 2'den Kontrolör 1'e giden hat içinde yapılmalıdır. Aksi durumda 2 yapı arasında haberleşme sağlanamayacaktır. 2 kablolu bir veri hattı söz konusudur. Bu durum veri haberleşmesi yapısına göre değişiklik gösterebilmektedir. Örneğin, tek yönlü bir yapı olarak sadece Kontrolör 1'den Kontrolör 2'ye veri gönderilecek ve Kontrolör 2'den herhangi bir veri beklenmeyecek olan bir yapının mümkün olması durumunda hat tek kablolu olacak ve sadece Kontrolör 1'e ait gönderici pin Kontrolör 2'de alıcı pine bağlanacak bir diğer deyişle simplex haberleşme sağlanacaktır. Ayrıca, UART kontrolörlerin iç yapısındaki buffer sayılarına göre aynı anda bir kontrolör 2 ayrı hatta hem gönderme hem de alma işlemi de

gerçekleştirebilir ya da aynı hat üzerinde hem gönderme hem de alma işlemi gerçekleştirilerek half duplex olarak karşılıklı veri iletimi yapılabilir. Bu tipte farklı veri iletimi durumları da oluşabilmektedir. Veri iletimi hat üzerinde ardışık bitler dizisi şeklinde karşı tarafa gönderilir karşı tarafta bu tekrardan paralel bir hale getirilerek anlamlandırılarak gerekli işlem yapılır. UART hızı saniyede gönderilen bit sayısı yani bit/s ya da baud olarak ifade edilmektedir. Önemli noktalardan biri de şudur ki, karşılıklı iki yapının sağlıklı haberleşmesi için, aynı bit hızına, aynı karakter uzunluğuna (veri hattında 8 bit veri alınıp gönderilmesi) ve stop biti sayısına ayarlama yapılmalıdır. Ek olarak UART birbirine Şekil 3.8’de gösterildiği şekilde donanımsal olarak bağlanmış olan 2 yapı arasında gerçekleşmektedir ve bir yapının yöneten diğer birden fazla yapının yönetilen olması gibi bir durumda söz konusu değildir. Çoklu bir şekilde veri iletimi ve haberleşme yapılamamaktadır. UART’ta hatta 5 ila 9 bit arasında veri tek bir paket halinde gönderilebilmektedir. Ancak genellikle 8 bitlik bir veri haberleşme paketleri kullanılır.

Şekil 3.9’da tipik olarak 8 bitlik bir verinin ne şekilde ve hangi kontrol bitleri kullanarak yapıldığı gösterilmektedir.

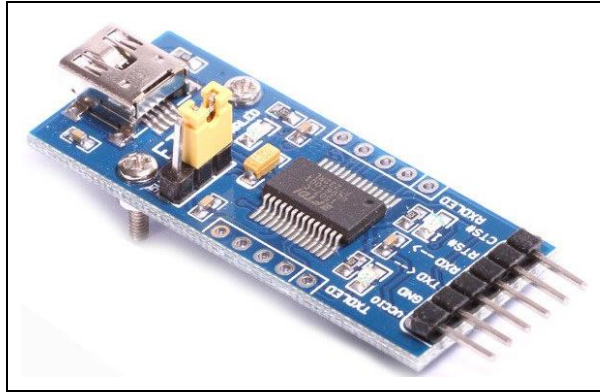


Şekil 3.9: UART 8 bit veri formatı gösterimi.

Veri iletimi yoksa hat sürekli olarak lojik 1 seviyesinde (idle state) tutulur. Sonraki süreçte ilk olarak, gönderici taraftan hatta standart gereği olarak lojik 0 bir bit gönderilir. Bu demektir ki gönderici taraf veriyi göndermeye hazırlanmakta ve alıcı tarafın da hattı dinlemesi gerektiğidir. Alıcı taraf hattı izler ve hattın lojik 1’den lojik 0’a düştüğünü algılayarak kendi iç devresindeki saatini senkron eder ve hatta veri gönderileceği bilgisini almış olur. Başlangıç bitini takiben hatta 8 bitlik veri ardışık olarak belirli bir bit zamanına göre gönderilmeye başlanır. Bu işlem en az önemli bittten (Least Significant Bit, LSB) başlanarak yapılır. Bu sırada alıcı taraf hattı izler ve yüksek olan gerilim seviyelerini lojik 1, 0 V olan gerilim seviyelerini de lojik 0 olarak algılar. Ardından gönderici tarafından hatta son olarak parity biti gönderilir. Gönderici taraf gönderdiği veri içindeki lojik 1’lerin sayısına göre hatta son olarak 0

veya 1 bitini hata tespiti amacıyla gönderir. Örneğin, çift parity'e göre gönderilen lojik 1'lerin sayısı çift ise gönderici taraf son olarak veri hattına parity bitini lojik 0 olarak tek parity'e göre de lojik 1 olarak eklemektedir. Bu veri paketi karşı tarafa gönderildiğinde eğer herhangi bir şekilde veride bir bozulma var ise alıcı taraf lojik 1'lerin sayısını yanlış sayacak ve sondaki parity bitinin lojik 1 değil lojik 0 olması gerektiğine karar verip hatalı bir verinin olduğunu anlayacak ve bu veri dizinini gönderen tarafa geri iletecektir. Bu sayede hatalı veri düzeltilmiş olacaktır. Ancak bu işlem opsiyonel olarak bulunmaktadır bir zorunluluğu yoktur. Genel olarak çoğu kullanımda bu parity biti gönderilmemektedir. Kullanılması durumunda parity bitinde gönderilmesini takiben gönderen taraf hattı kontrol etmeyi bırakır ve hat lojik 1 seviyesinde (idle state) kalmaya başlar. 1 veya 2 bit süresinde hat lojik 1'de kalması dur biti anlamına gelmektedir. Alıcı taraf bunu algılamaktadır.

Bu çalışma kapsamında, görsel arayüzden alınan register adresi, registera yazılacak olan veriler, yazma ya da okuma işleminin olacağını belirten bitler ve benzeri değerler bilgisayar vasıtasıyla seri konsol arayüzünden karakterler kullanılıp girilerek yazıldığı için karakterlerin bu değerlere ait ikilik sistemdeki (binary) karşılıkları USB'yi UART'a dönüştüren bir donanım üzerinden FPGA'ye transfer edilmiş ya da FPGA'den alınmıştır. Bu kısımda kullanılan bu donanıma yönelik özellik bilgileri ve Şekil 3.10'da genel görünümü verilmektedir [22].



Şekil 3.10: Mini USB-UART çevirici birimi.

İlgili çeviricinin genel olarak görevi ve çalışma kapsamında kullanım amacı incelendiğinde, bilgisayara USB arayüzünden bağlanarak kullanılan seri konsol programından bağlı bulunduğu portun seçilmesi ile birlikte yönetilmek istenen ETHERNET anahtar tümleşik devresine ait adresin, yazma ya da okuma yapılacaktır.

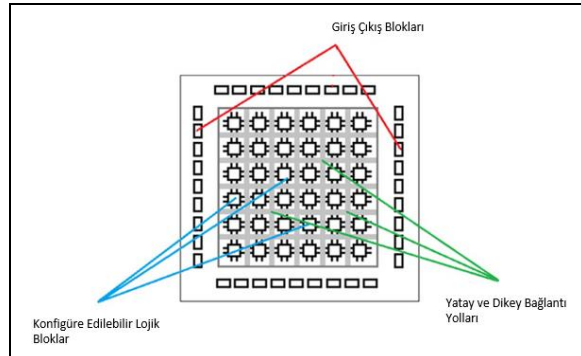
olan register adresinin ve eğer yazma yapılacaksa yazılacak olan veriler için konsol ekranında girilen karakterlerin ASCII değerlerinin USB arayüzünden alınarak UART'a dönüşümünü sağlayarak sonraki süreçte bu bilgilerin FPGA üzerinde değerlendirilmesini sağlamaktadır. Benzer şekilde ETHERNET anahtar tümleşik devresinden okuma işlemi gerçekleştirildiğinde de SMI arayüzü üzerinden gelen register verisinin UART olarak alınıp USB'ye dönüştürülerek ilgili verilerin ikilik sistemde gelen bitlerini bilgisayara iletmektedir. Bunun sonucunda da karakter karşılıklarının ekranda gösterimi sağlanmaktadır.

Çevirici, +3.3V ya da +5.0V'luk seviyelerde USB'yi TTL seviyesi sinyallere çevirmektedir. UART arayüzünden TX (transmit), RX (receive), RTS (Request to send), CTS (clear to send), GND (ground) sinyalleri dış dünyaya çıkmıştır [22]. Bunlardan çalışma kapsamında TX, RX ve GND sinyal çıkışları FPGA'e götürülmüştür. Bu amaçla FTDI FT232RL çevirici modül kullanılmıştır.

4. FPGA

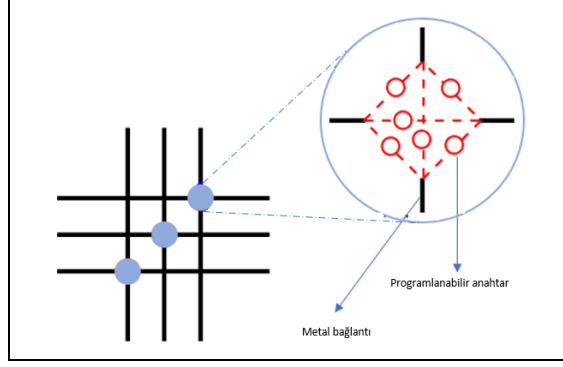
4.1. FPGA nedir, ne amaçla kullanılır?

FPGA (Field Programmable Gate Arrays) geliştiriciye çok spesifik ve belirlenmiş fonksiyonlar için devreler yaratmasını sağlayan tekrar tekrar konfigüre edilebilme özelliğine sahip mantıksal devreler bütünüdür. FPGA içerisinde bulunan donanım yapıları sabit değildir. FPGA içerisinde mantık birimleri sabit olmasına rağmen bu birimlerin birbiri arasında nasıl bağlanacağı, fonksiyonları kullanıcı tarafından değiştirilebilmektedir. Bu sebeple, FPGA'ler donanımsal olarak aslında sabit olmayan değiştirilebilir ve tekrar tekrar konfigüre edilebilir özellikte olan büyük, kompleks olabilen yapılardır [11]. Bu bahsetmiş olduğumuz en küçük lojik yapılar FPGA içerisinde genel olarak Şekil 4.1'de verildiği gibi bulunmaktadır.



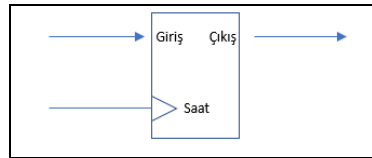
Şekil 4.1: Genel olarak FPGA iç yapısı (lojik bloklar, I/O bloklar ve bağlantı yolları) gösterimi.

FPGA iç yapısı incelendiğinde, yatay ve dikey olarak konumlandırılmış metal demetleri bulunmaktadır. Bu metal bağlantılar programlanabilir bağlantılardır. Yatay ve dikey metal bağlantı demetleri arasında anahtarlar bulunmaktadır. Bu anahtar yapıları MOSFET'ler sayesinde gerçekleştirilmektedir. Bu sayede istenilen bir metal bağlantı izin verilen herhangi bir metal bağlantıya ilgili transistörün sürülmesi ile bağlanabilmekte ve konfigüre edilebilir lojik bloklar birbirine bağlanabilmektedir. Metal bağlantı demetlerinin tek kullanım amacı bu olmamakla beraber saat sinyali dağıtımı ve benzeri gibi amaçlar için de kullanılabilir [12]. Şekil 4.2'de bu bağlantı demetlerinin temel düzeydeki yapısı verilmektedir.



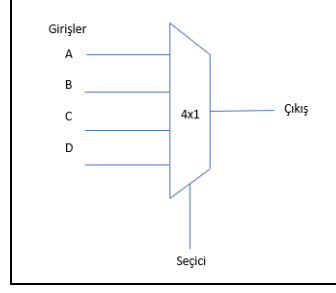
Şekil 4.2: Metal bağlantı yolları temel iç yapısı gösterimi.

Bir diğer temel yapı olarak konfigüre edilebilir lojik blok incelendiğinde, iç yapısında çok temel düzeyde LUT (Look Up Table), flip-flop ve multiplexerlardan (MUX) meydana gelmektedir. LUT aslında küçük bir hafıza olarak görev yapmaktadır. Genellikle 4 giriş 1 çıkışı bulunan ve FPGA'ye güç verildiğinde girişlerdeki olabilecek lojik durumlara göre AND, OR, XOR, NOT ya da bu fonksiyonların kombinasyonlarını üretebilen yapıdır. n girişli bir LUT, n girişinin herhangi bir fonksiyonunu uygulayabilir. LUT çıkışı başka lojik fonksiyona, flip-flop girişine ya da I/O yapısına bağlanabilir. Flip-flop yapısı kendine gelen saat darbelerine göre bu saat sinyalinin yükselen kenarlarında girişlerinde olan lojik durumları çıkışına verebilen yapıdır. Aslında bir hafıza birimi olarak davranır çünkü kendine gelen veriyi bir saat darbesi kadar geciktirir [19]. Şekil 4.3'de bu yapı temel düzeyde gösterilmiştir.



Şekil 4.3: D flip-flop genel gösterimi.

MUX yapısı temel olarak herhangi bir anda girişlerine gelen sinyalleri tek hat olarak çıkışına aktarmaktır. Fakat bu işlem iç yapısında bulunan hızlı bir anahtar ile yapılır ve bu sayede birden fazla girişindeki veri farklı zamanlarda çıkışına iletilir. Şekil 4.4'de temel düzeyde bir MUX yapısı gösterilmektedir.

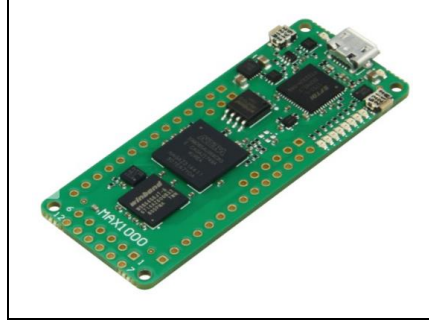


Şekil 4.4: Örnek bir MUX gösterimi.

Şekilde de belirtildiği gibi girişlerdeki veriler yapılan seçime göre A B C D ya da D C B A sıralamasıyla çıkışa verilir [12].

FPGA kullanımı bazı yönleri ile avantaj sağlamaktadır. FPGA’i mikroişlemcilerden ayıran en önemli özelliği sabit bir iç donanım yapısına sahip olmamasıdır. Mikroişlemcilerde çevresel yapılar, iç donanımdaki bağlantılar, tüm transistör hafızası ve benzeri sabit ve değişmezdir. FPGA’de sadece en alt temel birim olan konfigüre edilebilir lojik birimler sabit bir donanımdır. Bu birimlerin çevresel birimlere bağlantı şekilleri değiştirilebilir. Ayrıca, mikroişlemcilerde ve mikrokontrolörlerde işlemler ardışıl bir şekilde yapılmakta olup yapılmaktaysa da sınırlı düzeyde paralel işlem yapılmaktadır. FPGA’de ise tamamen paralel işlem yapılabilmektedir. Belirli bir anda birden fazla hatta binlerce farklı işlem yapılabilmektedir. Bu özelliği sayesinde çok yoğun veri işlenmesi gereken durumlarda FPGA kullanılabilir. Birkaç saat darbesi boyunca çok büyük veriler işlenebilir. Ancak mikroişlemcilerde belirli bir anda veri akışı büyüklüğü işlemci veri yolu (16 bit ya da 32 bit veri yolu ve benzeri) tarafından sınırlanmıştır. Böylece veri işleme hızı da sınırlandırılmıştır. FPGA HDL, Verilog gibi donanım programlama dilleri sayesinde programlanabilmektedir. Programlama hususunda FPGA tekrar programlanabilir. Bu nedenle yeni bir tasarım çok fazla zamana ihtiyaç gereksinimi olmadan hemen yapılabilmektedir. Hatta FPGA kullanımda iken dahi programlayabilme olanağı bulunmaktadır.

Bu çalışma kapsamında VHDL donanım programlama dili kullanılmıştır. Şekil 4.5’de verilen Intel ALTERA MAX 10 model FPGA kullanılarak ETHERNET anahtar tümleşik devresinin konfigürasyon ve yönetim arayüzü için gerekli olan seri yönetim protokolü gerçekleştirilmiştir.

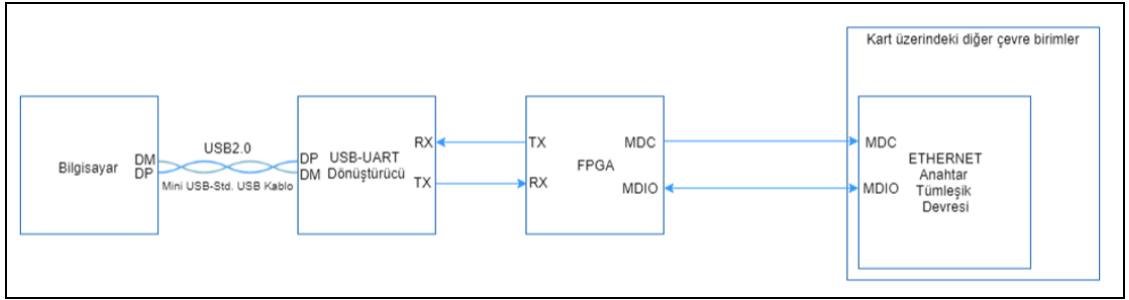


Şekil 4.5: ALTERA MAX 10 FPGA ve üzerinde bulunduğu kart gösterimi.

Çalışma kapsamında kullanılan FPGA'in özelliklerini belirtirsek; 378 Kb blok hafıza, 32-172 KB kullanıcı flash hafızası, harici hafıza arayüzü, 130 adet giriş çıkış (I/O), 8000 lojik element, 500 adet konfigüre edilebilir lojik blok, tek bir gerilimden beslenebilme gibi özellikleri bulunmaktadır [20].

5. SİSTEMİN DETAYLI ANLATIMI VE ÖLÇÜM SONUÇLARI

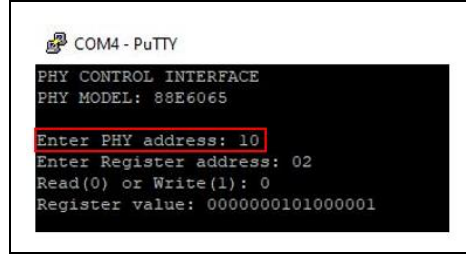
Bu bölümde SMI protokolünün gerçekleştirilmesi ve UART ile yapılacak olan veri alışverişinin gerçekleştirilmesi için yapılan işlemler detaylı ve görsel olarak verilmektedir. İlk olarak Şekil 5.1’de genel olarak birbirine entegre vaziyette FPGA, ETHERNET anahtar tümleşik devresi ve USB-UART dönüştürme yapının blok diyagramları ve donanımsal olarak anlatımları verilmektedir.



Şekil 5.1: Donanımsal yapıların birbirine entegre vaziyette genel gösterimi.

Yapı incelendiğinde, USB-UART çevirici ve FPGA bağlı bulunduğu bilgisayarın USB arayüzü üzerinden +5.0V’luk gerilim ile gücünü alıp çalışmaktadır. FPGA ve USB-UART çevirici arasındaki UART veri hattının hem gönderen hem de alan yolları +3.3V’luk gerilim ile pull-up’a çekilmiştir. Sinyalleme seviyesi +3.3V’luk genliktedir. FPGA ve USB-UART çevirici arasındaki veri hattı 9600 bit/s hızında kullanılmaktadır. Bu değer 115200 bit/s’e kadar çıkabilmekte olup ilgili çevirici bunu desteklemektedir. FPGA ile ETHERNET anahtar tümleşik devresi arasındaki SMI hattına ait MDC (saat) hattı +3.3V’luk gerilim ile pull-up durumundadır. Aynı şekilde veri hattı olan MDIO hattı da +3.3V’luk gerilim ile pull-up durumundadır. SMI hattı boyunca sinyalleme seviyesi +3.3V’tur. Saat hattı 1MHz’lik bir saat darbe sıklığı ile FPGA tarafından sürülmektedir. ETHERNET anahtar tümleşik devresi 8.3MHz’lik saat sıklığına kadar destekleyebilmektedir. Bu değer kullanılan ETHERNET anahtar tümleşik devresine özel bir değerdir. Bazı ETHERNET anahtarlama tümleşik devreleri 25MHz’lik saat sıklığına kadar destekleyebilmektedirler. Bu çalışma kapsamında saat hattı 1 MHz’lik bir saat darbe sıklığı ile sürülmesi tercih edilmiştir.

ETHERNET anahtar tümleşik devresinin kontrol edilmesi esnasında ilk aşamada bilgisayar üzerinden seri konsol arayüzü vasıtasıyla tümleşik devrenin adresi girilmektedir. Bu işlem Şekil 5.2’de gösterilmektedir.

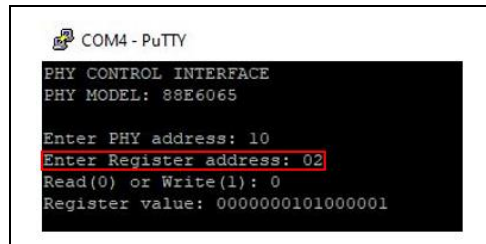


Şekil 5.2: Konsol ekranından adres girilmesi.

Örneğin; bu değer ekranda 10 olarak girilmiştir. Bu karakterleri temsil eden veriler bilgisayardan USB-UART çeviriciye gönderilir. USB-UART çeviricisinde girilen ETHERNET anahtar tümleşik devresi adres bilgisinin ASCII karşılığı sırası ile 8 bit olarak UART üzerinden FPGA’ye gönderilir. Örneğin, ASCII tablosu incelendiğinde, 1 karakterinin hexadecimal olarak karşılığı 0x31, decimal olarak karşılığı 49, 0 karakterinin hexadecimal olarak karşılığı 0x30, decimal olarak karşılığı 48’dir. UART çıkışından bu ASCII karşılığın 8 bitlik binary hali verildiğinden dolayı USB-UART çeviricinin gönderen hattında sırasıyla, 00110001 (decimal 49) ve 00110000 (decimal 48) bitleri görülecektir. Bu işlem için toplamda 2 kez 8 bitlik bir veri FPGA’ye gönderilmiş olacaktır. Çalışma kapsamında kullanılan ETHERNET anahtarlama tümleşik devresi adres olarak yapılan donanım konfigürasyon seçimine göre hexadecimal tabanda 0x00’dan 0x0F’e kadar ya da 0x10’dan 0x1F’e kadar olan adres değerini desteklemektedir. ETHERNET anahtar tümleşik devresi SMI protokolü gereği adres olarak 5 bitlik bir adres yapısı ile çalışmaktadır. Bu sebeple, konsol ekranından girilip UART çıkışından elde edilen 2 adet 8 bitten oluşan bitler dizisinin 5 bitlik formata dönüştürülmesi gerekmektedir. Bunun için mevcut kullanılan ETHERNET anahtar tümleşik devresinin adresi 0x00 ve 0x1F arasında olmak zorunda olduğundan dolayı ilk değer 0 ya da 1 olduğunun kontrolü yapılmaktadır. Bu işlem yapılırken de UART kısmından gelen ilk 8 bitin decimal olarak karşılığının 48 (karakter olarak 0) ya da 49 (karakter olarak 1) olup, olmama durumu kontrol edilir. İlk gelen 8 bitin ikili olarak karşılığı 00110000 (karakter 0) ise 0 olarak, 00110001 (karakter 1) ise 1 olarak 5 bitlik adres registerinin beşinci biti

alınmış olur. Adres registerinin sonraki 4 biti için ise UART kısmından gelen ikinci 8 bitlik veri okuması yapılarak alınır. İkinci gelen 8 bitlik veri FPGA’de 0 ile F arasındaki karakterlerin ASCII değerlerinin ikili sistemdeki karşılığı ile kontrolü yapılır. Bu amaçla, gelen 8 bitlik yapı bir kontrol mekanizmasına sokularak gelen verinin ASCII tablosundaki karakter karşılığı tespit edilir ve bu karakter karşılığı 4 bit olarak yazılır. Örneğin, adresin ikinci karakteri bilgisayar ekranından F olarak girilsin. Bu karakterin decimal karşılığı 70’tir. UART çıkışından FPGA’e gönderilen ikinci 8 bit dizisinde bu değer ikili olarak 01000110 olarak FPGA’e gönderilir. FPGA’de bu değer ASCII olarak F olduğu tespit edilir. Bu değer de 4 bitlik olarak 1111 şeklinde yazılarak adresin geri kalan 4 bitine yazılır. Bu şekilde ETHERNET anahtar tümleşik devresine ait adres 5 bit olarak register hale getirilmiş olur. Sonuç olarak bilgisayardan gelen karakterler ile SMI protokolüne uygun 5 bitlik bir veri elde edilmiş olur.

ETHERNET anahtar tümleşik devresinin adresinin geçerli bir adres olarak girilmesinin ardından bilgisayarın konsol ekranındaki ikinci işlem ise üzerinde değişim yapılacak ya da okunacak olan ETHERNET anahtar tümleşik devresindeki register adresinin girilmesidir. Bu aşamada tümleşik devre adresinin girilmesi ile aynı aşamalar takip edilmektedir. Kullanılan tümleşik devreye ait olan bilgilendirici rehber incelenerek değiştirilmek ya da okunmak istenen register adresi ekranda 2 adet karakter olarak girilir. Şekil 5.3’te örnek gösterim verilmektedir.

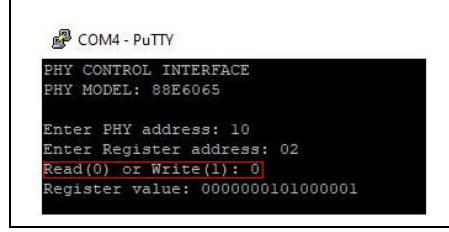


Şekil 5.3: Konsol ekranından register adres girilmesi.

Bu karakterlerin girilmesini takiben UART çeviriciden FPGA’e 2 kez 8 bitlik dizi gönderilir. Yine bu 8 bitlik diziler girilen karakterlerin ASCII tablosuna göre ikili halleridir. Çalışma kapsamında kullanılan ETHERNET anahtar tümleşik devresinin register adresi SMI protokolü gereği format olarak 5 bitlik bir yapıdadır. Bu yüzden FPGA’de ekrandan girilen karakterlerin bu 5 bitlik formata uygun olarak çevrilmesi işlemi gerçekleştirilmektedir. Çalışmalar kapsamında kullanılan ETHERNET

anahtarlama tümleşik devresinin konfigürasyon yapılabilecek olan registerları hexadecimal tabanda 0x00 ve 0x1F arasında değerler almaktadır. Bu sebeple ekrana girilen karakterlerin karşılıkları ikili olarak 8'er bitlik iki farklı veri halinde USB-UART çıkışından FPGA'ye gönderilir. FPGA'ye gelen ilk 8 bitlik dizi bilgisayar konsol ekranından gönderilen ilk karakterin ASCII tablosuna göre decimal 48 (ikili olarak 00110000) ya da decimal 49 (ikili olarak 00110001) yani karakter karşılığının 0 ya da 1 olup olmama durumu kontrol edilir ve 5 bitlik adres registerinin 5. bitine yazılır. İkinci 8 bitlik dizininde 0-F arasında kontrol edilme işlemi gerçekleştirilmektedir. İkinci olarak gelen 8 bitlik dizinin ASCII tablosuna göre karakter karşılığı 4 bit olarak ifade edilir. Bu 4 bitlik dizin de alınarak 5 bitlik registerin son 4 bitine yazılır. Sonuç olarak 5 bitlik adres dizini register edilmiş olur. Örneğin, ekrandan 1A girilmesini takiben UART çıkışından FPGA'ye sırasıyla 00110000 (ASCII tablosunda decimal 48), 01000001 (ASCII tablosunda decimal 65) bit dizinleri gönderilir. FPGA'de bu dizinlerin kontrolü yapılır ve karakter olarak 1A oldukların tespit edilir. 1 direkt olarak 5 bitlik adres registerinin beşinci bitine yazılır. A ise hexadecimal olarak 10'a karşılık gelmesinden dolayı 4 bitlik olarak 1010 kalan 4 bite yazılır. Sonuçta 5 bitlik adres 11010 olarak register edilmiş olur.

ETHERNET anahtar tümleşik devresinin registerlarından konfigüre edilecek olan register adresinin de konsol ekranından karakterler vasıtasıyla FPGA'ye iletilmesinin ardından ETHERNET anahtar tümleşik devresinin ilgili registerına yazma işleminin mi yoksa ilgili registerdan okuma işleminin yapılacağı hususunda gerekli bilgiyi girme işlemi gerçekleştirilir. Bu işlem için konsol ekranından okuma işlemi için 0, yazma işlemi için de 1 karakterlerinin yazılması istenmektedir. SMI protokolüne göre kontrol sinyali (opcode) 2 bit olarak gönderilmektedir. Okuma işleminin gerçekleştirilmesi için ekrana 0 karakteri girilir. Bu işlemi takiben çeviricinin UART çıkışına ASCII tablosundan 0 karakterine karşılık gelen değer 8 bitlik 00110000 olarak FPGA'ye gönderilmektedir. FPGA içerisinde gelen 8 bitlik verinin 00110000 ya da 00110001 olarak gelip gelmediğinin kontrolü yapılır. 00110000 yani ASCII karşılığı karakter 0 olan bilgi gelmesi halinde 2 bitlik yapıya SMI protokolü gereği okuma işlemi yapacağımızı belirten 10 bilgisi yazılır. 00110001 yani ASCII karşılığı karakter 1 olan bilgi gelmesi halinde 2 bitlik işlem kodu yapısına SMI protokolü gereği yazma işlemini yapacağımızı belirten 01 bilgisi yazılır. Örnek konsol ekranı görüntüsü Şekil 5.4'te verilmektedir.



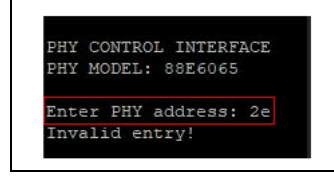
Şekil 5.4: Konsol ekranından yazma ya da okuma bilgisinin girilmesi.

ETHERNET anahtar tümleşik devresinin daha önceki adımlarda girilmiş olan ilgili register adresine yazma ya da bu register adresinden okuma işleminin yapılacağını belirtilmesini takiben okuma işlemini seçmiş olmamız durumunda bilgisayar konsol ekranında okuma, yazma seçim işleminin ardından direkt olarak ETHERNET anahtarlama tümleşik devresinin girilmiş olan registeri içerisinde tutulan verinin 16 bit olarak ekrana basılması işlemi gerçekleştirilmektedir. Bu işlem gerçekleştirilirken FPGA’de daha önceden girilen ETHERNET anahtar tümleşik devresinin 5 bitlik adresi, okuma ya da yazma yapılacak olan registerin 5 bitlik adresi, 2 bitlik işlem kodu, 2 bitlik start bit dizini SMI protokolüne belirlenmiş olan sıralama ile register edilerek MDIO hattı üzerinden ETHERNET anahtarlama tümleşik devresine gönderilmektedir. İlgili veri formatının ETHERNET anahtar tümleşik devresine gönderilmesi işlemi sırasında veri formatının en başına başlangıç bitlerini içeren 2 bitlik 01 bilgisi eklenerek gönderilmektedir. İlgili kontrol sinyallerini içeren veri yapısının son bitinin de ETHERNET anahtar tümleşik devresine gönderilmesinin hemen ardından FPGA’in MDIO hattına bağlı olan pini High-Z durumuna çekilir ve veri hattı pull up durumunda gerilime çekilmiş olur. Bu işlemi takiben veri hattı artık ETHERNET anahtar tümleşik devresine bırakılmış olur. ETHERNET anahtar tümleşik devresinin MDIO hattına bağlı olan pini register içerisindeki 16 bitlik veriyi göndermeden önce pinini High-Z durumundan lojik 0 durumuna çeker. Bu işleme turn around adı verilmektedir. Hattın kendisine bırakıldığının tespitinin ardından ilgili register içerisindeki veriyi 16 bit olarak MDIO hattına gönderir ve MDIO hattına bağlı bulunan pinini High-Z durumuna çeker ve hattı boşta bırakır. Hat pull up durumundan dolayı gerilime çekilmiş olur. Şekil 3.7’de SMI arayüzünden okuma işleminin nasıl gerçekleştirildiği görsel olarak gösterilmektedir. ETHERNET anahtar tümleşik devresi tarafından gönderilen 16 bitlik register verisi FPGA’e alınır. FPGA’de her bir gelen verinin 1 ya da 0 olduğunun kontrolü yapılır. Kontrolün ardından gelen 16 bitlik dizin içerisindeki her

bir 0 ve 1'in ASCII tablosundaki karşılıklarının ikili durumları 8'er bit olarak USB-UART çeviriciye gönderilir. 16 bit olarak gelen her bir 0 ve 1 biti 8'er bitlik dizinler halinde çeviriciye gönderilmektedir. Örneğin; ETHERNET anahtar tümleşik devresinden gelen register içerisindeki 16 bitlik veri 0000000101000001 olduğunu varsayalım. Bu verinin her bir biti kontrol işleminin ardından, ilk bit 1 olduğuna göre 00110001 (decimal 49), ikinci bit 0 olduğuna göre 00110000 (decimal 48), üçüncü bit 0 olduğuna göre 00110000 (decimal 48) şeklinde toplam 16 adet 8 bitlik veri çeviriciye FPGA'in gönderici pininden gönderilecektir. Bu veriler gerekli karakter dönüşümünün ardından bilgisayarın konsol ekranına basılmaktadır.

ETHERNET anahtar tümleşik devresinde ürün rehber dokümanlarında belirli izin verilen herhangi bir registerine veri yazmak için ise yazma işleminin konsol ekranından seçimini takiben okuma işleminde olduğu gibi 5 bitlik ETHERNET anahtar tümleşik devresinin adresi, içerisine yazma yapılacak olan 5 bitlik register adresi, 2 bitlik işlem kodu ile birlikte 01 olarak iki bitlik başlangıç bitlerinde SMI protokolü gereğince istenen formatta register edilmesinin ve ETHERNET anahtar tümleşik devresine MDIO hattı üzerinden gönderilmesinin ardından son bit de gönderildikten sonra FPGA tarafından hat lojik 1 ve lojik 0'a çekilmektedir. 10 olarak hat durumu değiştirilir. Buna turn around adı verilmektedir. Bu işlemin ardından registre yazılacak olan 16 bitlik veri hatta gönderilmeye başlanır. 16 bitlik verinin son bitinin gönderilmesinin ardından FPGA'in veri hattına bağlı olan ilgili pini High-Z durumuna çekilir ve hat pull up durumundan dolayı gerilime çekilmiş ve boşta kalmış olur. Bu sayede ETHERNET anahtar tümleşik devresinin ilgili bir registeri içerisine yazma işlemi gerçekleştirilmektedir. Yine 16 bitlik verinin konsol ekranından FPGA'a iletilmesi esnasında girilen karakterlerin ASCII tablosundaki karşılıklarının 8'er bitlik ikili halleri (0 decimal olarak 48, ikili olarak 00110000, 1 decimal olarak 49, ikili olarak 00110001) çeviricinin UART gönderici çıkışından FPGA'in alıcı pinine gönderilir. FPGA'de 0 ya da 1 olma durumlarının kontrolünün ardından ikili olarak 16 bitlik dizin halinde ETHERNET anahtarlama tümleşik devresine gönderilir. Şekil 3.6'da yazma işlemi gösterilmektedir.

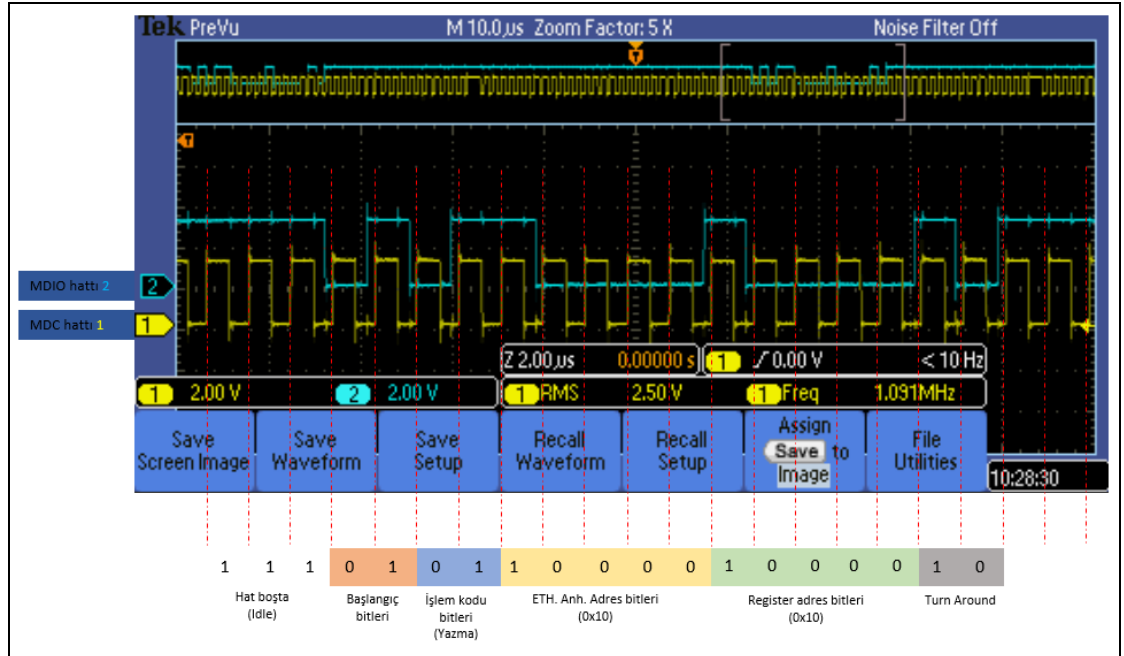
Bilgisayar konsol ekranından ETHERNET anahtar tümleşik devresinin adresi, register adresi, işlem kodu ve register içerisine yazılacak olan 16 bitlik verinin hatalı girilmesi durumunda FPGA'de kontrol işlemleri yapılarak ekrana Şekil 5.5'te gösterildiği gibi "Invalid entry!" ifadesi yazılmaktadır.



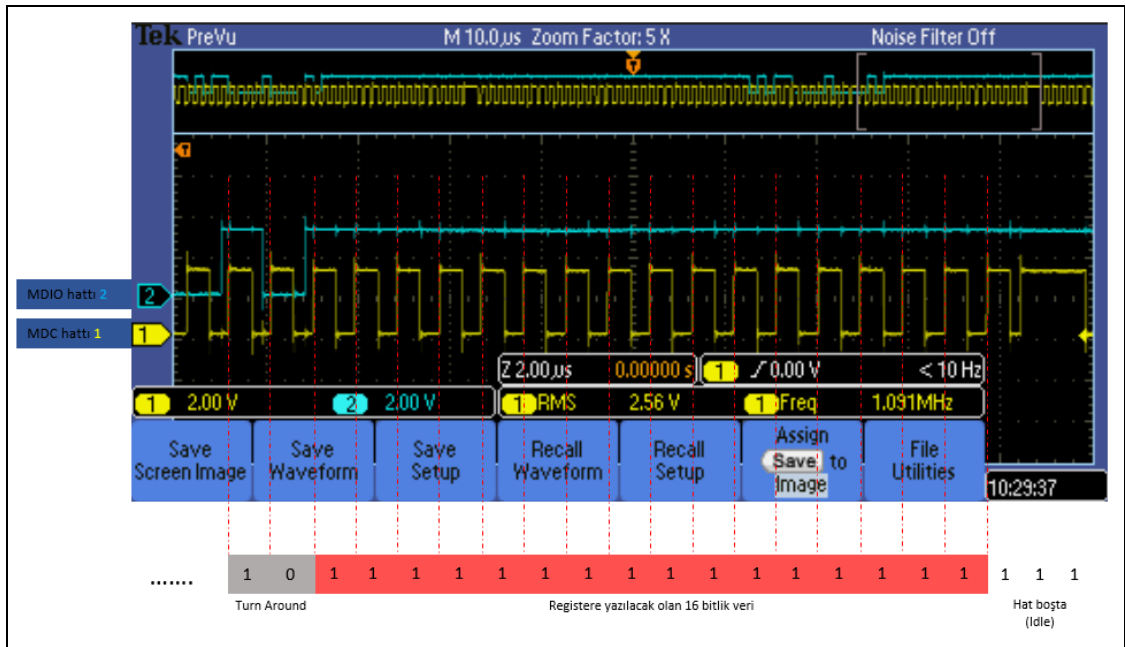
Şekil 5.5: Hatalı bir adres girilmesinin ardından konsol ekranı örnek görüntüsü.

Çalışma kapsamında FPGA’de gerçekleştirilen seri yönetim arayüzünün doğruluğunun sağlanması için görsel olarak sinyal seviyelerini, sıklıklarını, zamanlamalarını gösteren araç, gereçler kullanılmıştır. Bu cihazlardan alınan görsel çıktılara göre FPGA’de yazılan kod ve kullanılan donanımlar üzerinde düzenlemelere gidilmiş ve nihai doğru işlemlerin gerçekleştirilmesine kadar bu durum tekrar tekrar kontrol edilip kodta düzenlemeler yapılmıştır.

Şekil 5.6’da FPGA’den ETHERNET anahtar tümleşik devresinin herhangi bir registerına yazma işlemini gösteren osiloskop çıktıları ve bu çıktılara dair yorumlamalar ve açıklamalar verilmektedir.

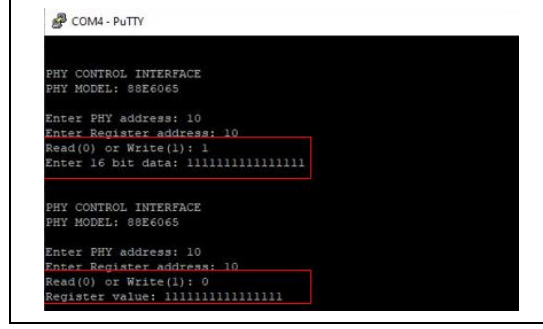


Şekil 5.6: Yazma işleminin osiloskop çıktısının gösterimi.



Şekil 5.7: Şekil 5.6’da verilen görselin devamının gösterimi.

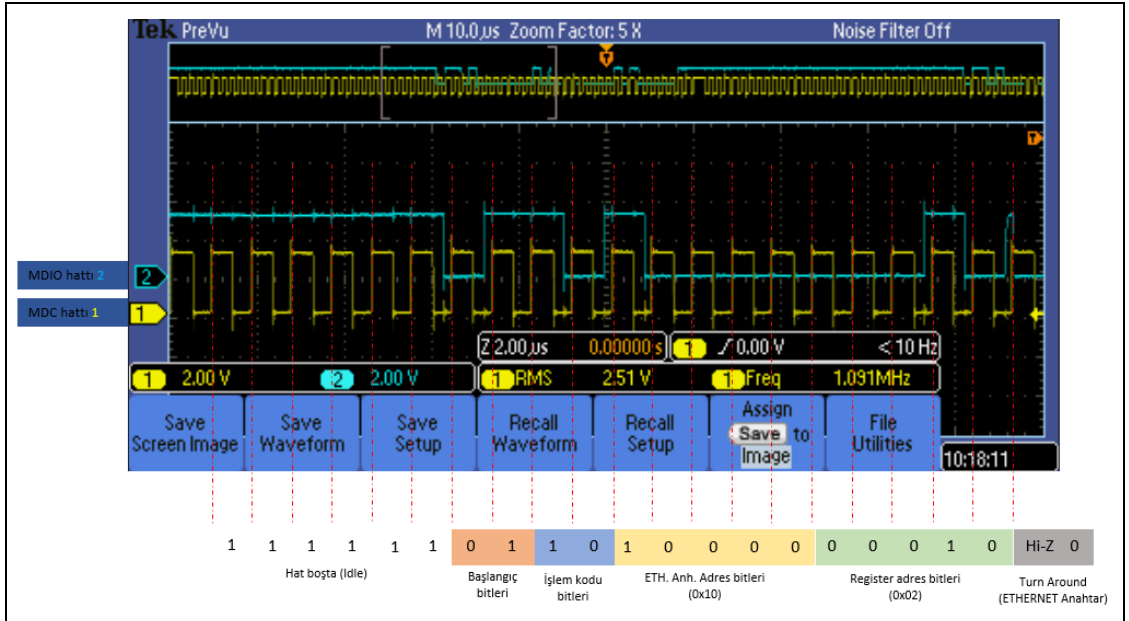
Osiloskop çıktılarında sarı renk ile gösterilen çıktı 1MHz'lık saat hattının görüntüsüdür. Mavi renk ile gösterilen veri hattının görsel çıktılarını göstermektedir. Çıktılar incelendiğinde, FPGA tarafından veri hattına gönderilen verinin bir bitinin genişliğinin saat darbesinin yükselen kenarından küçük bir zaman dilimi öncesine kadar, düşen kenarından da küçük bir zaman dilimi kadar sonraki sürece kadar devam ettirilerek genişliği sağlanmaktadır. Bu zamanlama sayesinde bir saat darbesinin lojik1 1 olduğu durumda MDIO hattındaki verinin yönetilen ETHERNET anahtar tümleşik devresi tarafından algılanması garanti edilmeye çalışılmaktadır. Örnek olarak, Şekil 5.6'da verilmiş olan çıktılara göre, ETHERNET anahtar tümleşik devresinin 0x10 registerine 16 bitlik lojik 1'lerden oluşan bilgi yazılmıştır. Yazılma işleminin kontrolünün sağlanması için ilgili register'dan okuma işlemi gerçekleştirilir ve ETHERNET anahtarlama tümleşik devresi tarafından hatta basılan register içeriğinin 16 bitlik lojik 1'lerden oluşan değer olması beklenmektedir. Aksi bir durumda registra yazma işlemi ya da yazma işlemi başarılı olmuş olup okuma işleminde bir problem olup olmadığı tespit edilir. Şekil 5.8'de konsol ekranından ETHERNET anahtarlama tümleşik devresinin 0x10 registerine 16 bitlik 1111111111111111 verisinin yazılma işlemi yapılmış hemen ardından da bu registerin okuma işlemi yapılmış ve register içerisine verinin doğru bir şekilde yazılıp yazılmama durumu kontrol edilmiştir.



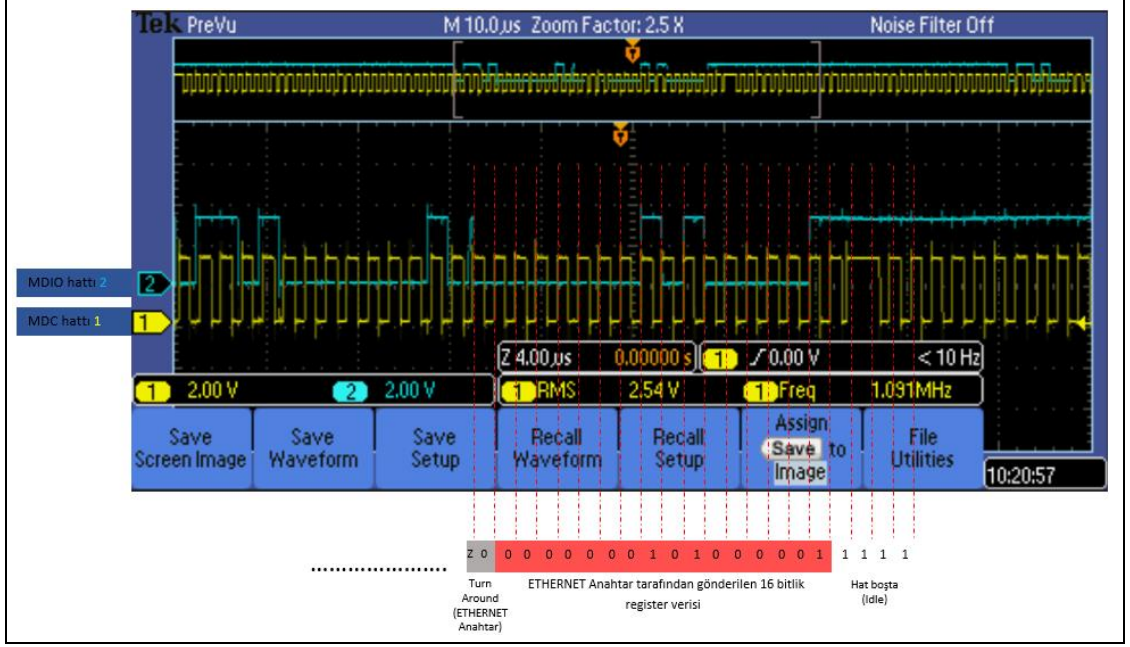
Şekil 5.8: Doğru yazma işleminin gösterimi.

Çalışma kapsamında ilgili tümleşik devrenin bilgilendirici dokümanı incelenerek yazma yapılacak olan registerlara veriler yazılmış ve tekrar tümleşik devreden okunarak doğru yazma işleminin yapıldığı teyit edilmiştir. Mevcut olarak ETHERNET anahtarlama tümleşik devresinin üzerinde bulunduğu elektronik kart üzerinde yapılan register değişikliklerinin tümleşik devre tarafından uygulamalı çevresel arayüzlere aktarılması durumu da incelenerek gerekli teyit etme işlemleri yapılmıştır.

Şekil 5.9’da örnek olarak ETHERNET anahtar tümleşik devresinin izin verilen herhangi bir registerından veri okuma işlemi gerçekleştirilmiştir. FPGA ile gönderilen kontrol bitleri ve ETHERNET anahtarlama tümleşik devresi tarafından hatta verilen 16 bitlik register içeriğinin doğru veri formatı verilmektedir.

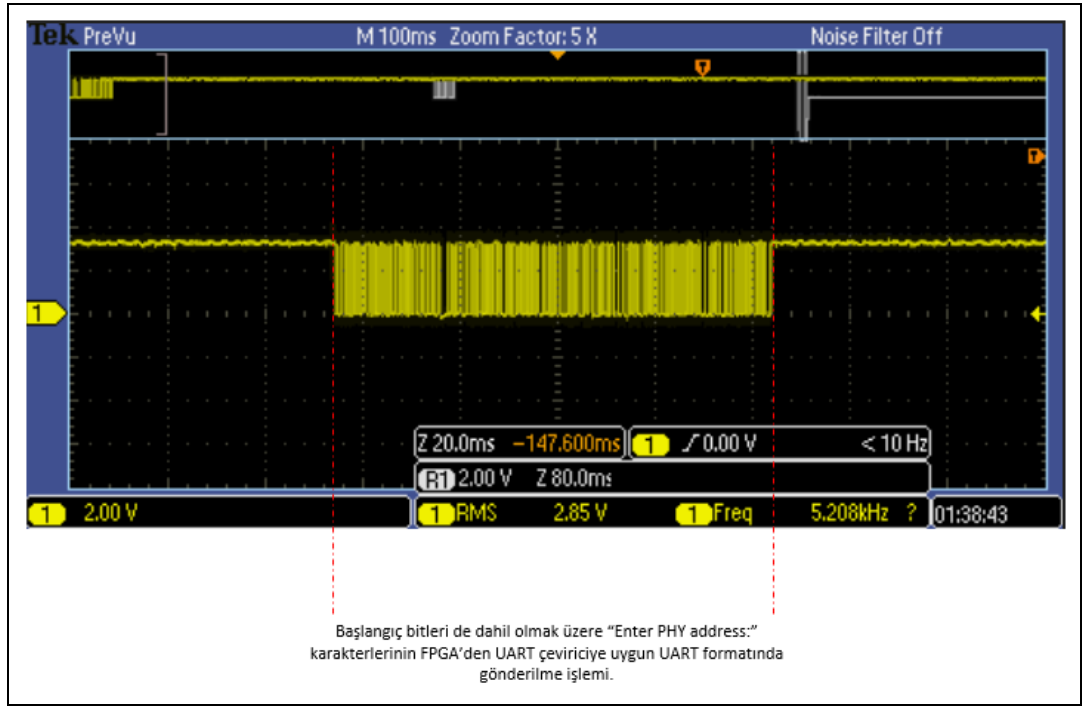


Şekil 5.9: Okuma işleminin osiloskop çıktısının gösterimi.



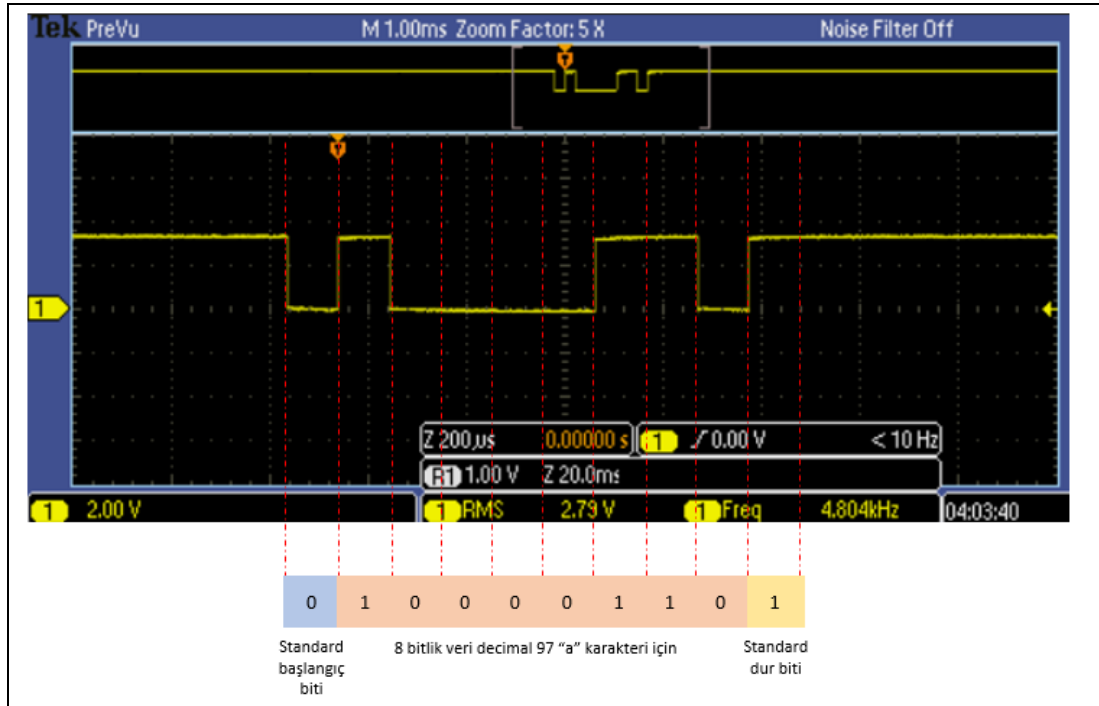
Şekil 5.10: Şekil 5.9’da verilen görselin devamının gösterimi.

FPGA ile USB-UART çevirici arasındaki iletişime dair görsel osiloskop çıktısı Şekil 5.11’de verilmektedir. Bu iletişimde FPGA ve çevirici arasında bilgisayardan FPGA’ye gönderilen ya da FPGA’dan bilgisayar ekranına gönderilen verilerin ASCII tablosundaki karşılıklarının ikili sistem olarak 8’er bitlik dizinleri kullanılarak haberleşme sağlanmaktadır. Enter, Backspace tuşları da dahil olmak üzere konsol ekranındaki her bir işlem bu şekilde FPGA ve çevirici arasında veri iletimi ile gerçekleştirilmektedir. Şekil 5.11’de FPGA’dan konsol ekranına gönderilen “Enter PHY adress:” ibaresinin çeviricinin alıcı pinine FPGA’in gönderici pininden ne şekilde gönderildiği yer almaktadır.



Şekil 5.11: FPGA ile çevirici arasındaki örnek UART haberleşmesi gösterimi.

Daha detaylı bir gösterim amacıyla konsol ekranından örnek olarak basılan "a" karakterinin UART üzerinden nasıl iletildiğinin gösterimi Şekil 5.12'de verilmektedir.



Şekil 5.12: UART üzerindeki herhangi bir verinin iletirme formatı.

6. SONUÇLAR

Yapılan çalışma farklı ETHERNET anahtarlama tümleşik devrelerinin kontrol, hata ayıklama ve içerisindeki yazmaçlara erişilmesini sağlayan platform bağımsız bir yapıdır. İlgili çalışmanın meydana getirilmesi aşamasında standart kontrol devrelerinde kullanılan ve tümleşik devrelerin test, hata ayıklama ve kontrol yazmaçlarına ulaşmayı sağlayan tümleşik devre üreticisi bağımsız bir protokol olan JTAG ve ST Microelectronics firmasına ait kendi geliştirdiği, kontrol birimlerinin programlama, test ve hata ayıklama arayüzü olan ST LINK arayüzü incelenmiştir. JTAG bütün tümleşik devre sağlayıcıları için standart bir arayüz olup, ST LINK ise ST Microelectronics firmasına özel bir arayüzdür. Yapılan çalışma JTAG arayüzüne benzer bir yapıda platform bağımsız olarak tüm ETHERNET anahtarlama tümleşik devrelerinin tek bir yerden kontrol edilmesine olanak sağlayan bir arayüzdür.

Yapılan çalışma ana olarak iki basamaktan oluşmaktadır. Birincisi FPGA üzerinde yapılan çalışmalardır. İkincisi ise bir bilgisayar üzerinde görüntüleme yaparak sistemin kullanıcıya belirli çıktılar vermesi ya da kullanıcıdan belirli girdiler istemesi üzerine kurulu olan konsol arayüzüdür. Bu konsol arayüzünde hem bilgisayar ve FPGA arasında bir haberleşme yapısı hem de ETHERNET tümleşik devreleri ve FPGA arasında olan bir haberleşme yapısı bulunmaktadır. Bu haberleşme yapıları üzerinde de ETHERNET anahtarlama tümleşik devreleri ile haberleşme yapmamızı sağlayan SMI arayüzü bulunmaktadır. Bu arayüz bir adet saat darbesi hattı (MDC), bir adet de çift yönlü veri hattından (MDIO) oluşmaktadır. Bu arayüz gerçekleşirken Clause 22 olarak adlandırılan standarta ve haberleşme yapısına uygun olarak bütün zamanlamalar, paket ve adresleme yapıları gerçekleştirilmiştir. Çalışma ETHERNET anahtarlama tümleşik devrelerinden alınan ve FPGA üzerinden tam çift yönlü, asenkron yapıdaki UART vasıtasıyla bir bilgisayara gönderilen verilerin ya da bilgisayar üzerinden alınan ve UART üzerinden FPGA vasıtasıyla ETHERNET anahtarlama tümleşik devresine gönderilen verilerin konsol ekranında görüntülenmesini sağlayan bir çalışmadır. Bu işlemler gerçekleştirilirken FPGA üzerinde durum mekanizmaları yapıları ile kodlar yazılmış olup, ilgili kodun test ve analiz edilmesi işlemi için görsel kontrol ve görüntüleme olanağı sunan araç ve gereçler kullanılarak test etme işlemi gerçekleştirilmiştir.

Yapılan çalışmada platformun test edilmesi ve güvenilirliğinin sağlanması amacıyla MARVELL Semiconductor, QUALCOMM gibi tümleşik devre sağlayıcılarına ait ETHERNET anahtarlama tümleşik devreleri ile çalışılmıştır.

Genel olarak ETHERNET anahtarlama tümleşik devrelerinin kontrol, hata ayıklama ve içerisindeki yazmaçlara erişim hususunda hali hazırda standartlaşmış bir zincir yapısı bulunmamaktadır. Yapılan çalışmanın ileriki süreçte özellikle ETHERNET anahtarlama alanı üzerine yönelik uzmanlaşmış kimselere standart bir zincir yapısı sunan bir protokole doğru evrilme olanağı bulunmaktadır.

KAYNAKLAR

- [1] Zhou J., Mason A., (2002), "Communication Buses and Protocols for Sensor Networks", Sensors 2002, 245-257, East Lansing, Michigan, USA, 4 July.
- [2] Wakhle G. B., Aggarwal I., Gaba S., (2012), "Synthesis and Implementation of UART using VHDL Codes", 2012 International Symposium on Computer, Consumer and Control, 1-3, Taichung, Taiwan, 4-6 June.
- [3] PENG H., NIE Y., (2011), "Design of Serial Communication Interface Based on FPGA", 2011 IEEE International Conference on Computer Science and Automation Engineering, 410-414, Shanghai, China, 14 July.
- [4] Ngo B. V., Law P., Sparks A., (2008), "Use of JTAG Boundary-Scan for Testing Electronic Circuit Boards and Systems", 2008 IEEE AUTOTESTCON, Salt Lake City, UT, USA, 8-11 September.
- [5] Shashidhara H. B., Yellampalii S., Goudanavar V., (2014), "Board Level JTAG/Boundary Scan Test Solution", International Conferences on Circuits, Communication, Control and Computing, 73-76, Bangalore, India, 21-22 November.
- [6] Grecu C., Iordache C., (2015), "Portable I2C Monitor and Debugger", 2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME), 131-134, Brasov, Romania, 22-25 October.
- [7] Giladi R., (2008), "Network Processors: Architecture, Programming, and Implementation", 1st Edition, Morgan Kaufmann.
- [8] Spurgeon C., (2014), Zimmerman J., "Ethernet: The Definitive Guide, 2nd Edition: Designing and Managing Local Area Networks", 2nd Edition, O'Reilly Media.
- [9] Chowdhury D. D., (2000), "High Speed LAN Technology Handbook", 1st Edition, Spring-Verlag Berlin Heidelberg.
- [10] Frenzel L., (2015), "Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/Output I/O Standards", 1st Edition, Newnes.
- [11] Chu P. P., (2008), "FPGA Prototyping by VHDL Examples", 2nd Edition, Wiley & Sons, Inc.
- [12] Sarıtaş E., Karataş S., (2013), "Her Yönüyle FPGA ve VHDL", 1.Baskı, Palme Yayıncılık

- [13]Tahir M., Javed K., (2017), “ARM Microprocessor Systems: Cortex-M architecture, programming and interfacing”, 1st Edition, CRC Press, Taylor & Francis Group.
- [14]Maier K., (2003), “On-Chip Debug Support for Embedded Systems-On-Chip”, Proceedings of 2003 International International Symposium on Circuits and Systems ,2003. ISCAS ‘03, 565-568, Bangkok, Thailand, 25-28 May.
- [15]Wu D., Likun L., Yeli L., (2012), “The Design of Data Mutual Conversion System Between Serial Port and Ethernet Based on W5100”, 2012 International Conference on Image Analysis and Signal Processing, Hangzhou, China, 9-11 November.
- [16]Hu X., Jin Y., Li Z., (2018), “A Parallel JTAG-based Debugging and Selection Scheme for Multi-core Digital Signal Processors ”, 2018 IEEE International Conference of Safety Produce Informatization (IICSPI), 527-530, Chongqing, China, 10-12 December.
- [17]Torroja Y., Lopez A., Portilla J., Riesgo T., “A Serial Port Based Debugging Tool To Improve Learning With Arduino”, 2015 Conference on Design of Circuits and Integrated Systems (DCIS), Estoril, Portugal, 25-27 November.
- [18]Zhuxinkai, Xulixin, Jiayonghong, “USB Interface Data Acquisition System Hardware Design”, 2006 Chinese Control Conference, 1405-1410, Harbin, China, 711 August.
- [19] Web 1, (2019), <https://www.furkan.space/course/elec457/> (Eriřim Tarihi: 11/03/2019)
- [20] Web 2, (2019), https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_datasheet.pdf (Eriřim Tarihi: 15/05/2019)
- [21]Web 3, (2019), <https://extranet.marvell.com/extranet/index.do> (Eriřim Tarihi: 20/06/2019)
- [22]Web 4, (2019), https://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf, (Eriřim Tarihi: 23/08/2019)
- [23]Web 5, (2019), <http://www.ti.com/lit/an/spracc8.pdf>,(Eriřim Tarihi:01/09/2019)
- [24]Web 6, (2019), <http://www.gtu.edu.tr/icerik/1478/591/lisansustu.aspx> ,(Eriřim Tarihi: 24/11/2019)
- [25]Web 7, (2020), http://ww1.microchip.com/downloads/en/DeviceDoc/ANLAN207_Management_Interfaces.pdf , (Eriřim tarihi : 08/07/2020)

ÖZGEÇMİŞ

Uğur AKÇAY 1991 yılında Kütahya’da doğdu. 2009 yılında başladığı Gazi Üniversitesi Mühendislik Fakültesi Elektrik-Elektronik Mühendisliği Bölümünü 2014 yılında başarıyla tamamlamıştır. 2017 yılında yüksek lisans eğitimine Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik Mühendisliği Anabilim Dalında başladı. 2014 yılından bu yana sırasıyla NETAŞ Telekomünikasyon A.Ş., Andasis Elektronik Sanayi ve Ticaret A.Ş. firmalarında Donanım Tasarım Mühendisi olarak çalışmaktadır.

EKLER

Ek A: Tez Çalışması Kapsamında Yazılan VHDL Kod Blokları

```
12 -- en üst bloğun port ve generic tanımlaması
13 -- generic kısmında uart baudrate ayarı ve kodun düzgün çalışması için referans clock hızını ayarlanması yapılabilir.
14 entity UART_TO_SMI_TEST is
15   generic (
16     CLK_FREQ      : integer := 12e6; -- set system clock frequency in Hz
17     BAUD_RATE     : integer := 9600; -- baud rate value
18     PARITY_BIT    : string  := "none"; -- legal values: "none", "even", "odd", "mark", "space"
19     USE_DEBOUNCER : boolean := FALSE; -- enable/disable debouncer
20   );
21   port (
22     CLK : in std_logic; -- system clock
23     RST : in std_logic; -- high active synchronous reset
24     -- UART INTERFACE
25     UART_TXD : out std_logic;
26     UART_RXD : in std_logic;
27     mdc_t    : inout std_logic;
28     mdio_t   : inout std_logic;
29   );
30 end UART_TO_SMI_TEST;
```

Şekil A1.1 : En üst bloğun generic ve portlarının tanımlanması.

```
39 type texts is array (0 to 63) of std_logic_vector(7 downto 0);
40 signal info_phy_msg : texts:= ("01010000", "01001000", "01011001", "00100000", "01000011", "01001111", "01001110", "01010100", "01010010", "01001111",
41 signal reg_msg : texts:= ("01000101", "01101110", "01101000", "01100101", "01100100", "00100000", "01010010", "01100101", "01100111", "01101001", "011
42 signal op_msg : texts:= ("01010010", "01100101", "01100001", "01100100", "00101000", "00110000", "00101001", "00100000", "01101111", "01100100", "0010
43 signal error_msg : texts:= ("00001101", "00001010", "01001001", "01101110", "01101110", "01100001", "01101100", "01101001", "01101000", "00100000", "0
44 signal wr_msg : texts:= ("01000101", "01101110", "01101000", "01100101", "01100100", "00100000", "00110001", "00110110", "00100000", "01100010", "0110
45 signal data_rd_msg : texts:= ("01010010", "01100101", "01100111", "01101001", "01110011", "01101000", "01100100", "01110010", "00100000", "01110110",
46 -- Yukarıda bulunan 6 adet TEXTS tipindeki dizi konsol ekranına basılacak olan mesajlardaki karakterlerin ASCII karşılığını 8 bit olarak içermektedir.
```

Şekil A1.2 : Mesaj dizileri.

```
176 process(all) -- uarttan gelen dataların smi hattına, smi hattından gelen dataların uart hattına uygun bir şekilde iletilmesi için oluşturulan state machine.
177 -- aynı zamanda her durum içerisinde hatalı giriş yapıp, yapılmadığı kontrol edilir.
178 begin
179   -- kritik sinyaller için default değer ataması. böylelikle bu sinyaller için latch oluşması önlenir.
180   next_state <= state;
181   next_count <= count;
182   next_cnt <= cnt;
183   next_valid_in <= valid_in;
184   next_data_in <= data_in;
185   next_phy_adr <= phy_adr;
186   next_reg_adr <= reg_adr;
187   next_op <= op;
188   next_err <= err;
189   next_data_161 <= data_161;
190   next_data_162 <= data_162;
191   next_phy_data_out <= phy_data_out;
192   next_reg_adr_temp <= reg_adr_temp;
193   next_phy_adr_temp <= phy_adr_temp;
194   next_op_temp <= op_temp;
195   next_strt_temp <= strt_temp;
```

Şekil A1.3 : State machine prosesi.

```
200 when info=> -- "PHY CONTROL INTERFACE" "PHY MODEL: 88E6065" ve "ENTER PHY ADDRESS:" mesajını içeren info_phy_msg dizisinin
201 -- uart modülünün tx hattı üzerinden konsol ekranına gönderilmesi.

210 -- bu if bloğu ile "PHY CONTROL INTERFACE" "PHY MODEL: 88E6065" ve "ENTER PHY ADDRESS:" mesajı konsol ekranına yazdırılır.
211 |
212   if(count < 64) then
213     next_data_in <= info_phy_msg(count);
214     next_valid_in <= '1';
215   elsif(count = 64) then
216     next_valid_in <= '0';
217     next_state <= phy;
218     next_count <= 0;
219     next_cnt <= 0;
220   end if;
```

Şekil A1.4 : Info state.

```

223 | when phy=> -- kullanıcının konsol ekranına yazdığı phy adrese karşılık gelen karakterlerin ASCII karşılığı alındığı için
224 | -- hangi karaktere basılmış kontrol edilmesi, bu verinin 5 bite map edilmesi.

239 |
240 |
241 |
242 | /* 1 */ -- 1 ve 2 numaralı if bloğu ile konsoldan gelen PHY adres verisi 5 bite map edilir.
243 | if(count = 0) then --
244 | next_data_in <= data_out;
245 | if(data_out = "00110000") then --ASCII 0
246 | next_phy_adr(4) <= '0';
247 | elsif(data_out = "00110001") then --ASCII 1
248 | next_phy_adr(4) <= '1';
249 | elsif(data_out = "00001000") then --Backspace
250 | if(err > 0) then
251 | next_err <= (err - 1);
252 | end if;
253 | else
254 | next_err <= (err + 1);
255 | end if;
256 | end if;
257 |
258 | /* 2 */
259 | if(count = 1) then
260 | next_data_in <= data_out;
261 | if(data_out = "00110000") then --ASCII 0
262 | next_phy_adr(3 downto 0) <= "0000";
263 | elsif(data_out = "00110001") then --ASCII 1
264 | next_phy_adr(3 downto 0) <= "0001";
265 | elsif(data_out = "00110010") then --ASCII 2
266 | next_phy_adr(3 downto 0) <= "0010";
267 | elsif(data_out = "00110011") then --ASCII 3
268 | next_phy_adr(3 downto 0) <= "0011";
269 | elsif(data_out = "00110100") then --ASCII 4
270 | next_phy_adr(3 downto 0) <= "0100";
271 | elsif(data_out = "00110101") then --ASCII 5
272 | next_phy_adr(3 downto 0) <= "0101";
273 | elsif(data_out = "00110110") then --ASCII 6
274 | next_phy_adr(3 downto 0) <= "0110";
275 | elsif(data_out = "00110111") then --ASCII 7
276 | next_phy_adr(3 downto 0) <= "0111";
277 | elsif(data_out = "00111000") then --ASCII 8
278 | next_phy_adr(3 downto 0) <= "1000";
279 | elsif(data_out = "00111001") then --ASCII 9
280 | next_phy_adr(3 downto 0) <= "1001";
281 | elsif(data_out = "01000001" or data_out = "01100001") then --ASCII A
282 | next_phy_adr(3 downto 0) <= "1010";
283 | elsif(data_out = "01000010" or data_out = "01100010") then --ASCII B
284 | next_phy_adr(3 downto 0) <= "1011";
285 | elsif(data_out = "01000011" or data_out = "01100011") then --ASCII C
286 | next_phy_adr(3 downto 0) <= "1100";
287 | elsif(data_out = "01000100" or data_out = "01100100") then --ASCII D
288 | next_phy_adr(3 downto 0) <= "1101";
289 | elsif(data_out = "01000101" or data_out = "01100101") then --ASCII E
290 | next_phy_adr(3 downto 0) <= "1110";
291 | elsif(data_out = "01000110" or data_out = "01100110") then --ASCII F
292 | next_phy_adr(3 downto 0) <= "1111";
293 | elsif(data_out = "00001000") then --Backspace
294 | if(err > 0) then
295 | next_err <= (err - 1);
296 | end if;
297 | else
298 | next_err <= (err + 1);
299 | end if;
300 | end if;

```

Şekil A1.5 : Phy state.

```

320 | when reg=> -- "ENTER REGISTER ADDRESS:" mesajını içeren reg_msg dizinin konsol ekranına iletilmesi ve kullanıcının
321 |           -- konsol ekranına yazdığı register adrese karşılık gelen karakterlerin
322 |           -- ASCII karşılığı alındığı için hangi karaktere basılmış kontrol edilmesi, bu datanın 5 bite map edilmesi.

338 | -- "Enter Register Address :" mesajı konsol ekranına yazdırılır.
339 | if(count > 1 and count < 26) then
340 |   next_valid_in <= '1';
341 |   next_data_in <= reg_msg(count - 2);
342 | end if;
343 |
-----
362 | -- 1 ve 2 numaralı if blokları ile kullanıcıdan gelen register adres verisi 5 bite map edilir.
363 | /* 1 */
364 | if(cnt = 0) then
365 |   next_data_in <= data_out;
366 |   if(data_out = "00110000") then --ASCII 0
367 |     next_reg_adr(4) <= '0';
368 |   elsif(data_out = "00110001") then --ASCII 1
369 |     next_reg_adr(4) <= '1';
370 |   elsif(data_out = "00001000") then --Backspace
371 |     if(err > 0) then
372 |       next_err <= (err - 1);
373 |     end if;
374 |   else
375 |     next_err <= (err + 1);
376 |   end if;
377 | end if;
378 |
379 | /* 2 */
380 | if(cnt = 1) then
381 |   next_data_in <= data_out;
382 |   if(data_out = "00110000") then --ASCII 0
383 |     next_reg_adr(3 downto 0) <= "0000";
384 |   elsif(data_out = "00110001") then --ASCII 1
385 |     next_reg_adr(3 downto 0) <= "0001";
386 |   elsif(data_out = "00110010") then --ASCII 2
387 |     next_reg_adr(3 downto 0) <= "0010";
388 |   elsif(data_out = "00110011") then --ASCII 3
389 |     next_reg_adr(3 downto 0) <= "0011";
390 |   elsif(data_out = "00110100") then --ASCII 4
391 |     next_reg_adr(3 downto 0) <= "0100";
392 |   elsif(data_out = "00110101") then --ASCII 5
393 |     next_reg_adr(3 downto 0) <= "0101";
394 |   elsif(data_out = "00110110") then --ASCII 6
395 |     next_reg_adr(3 downto 0) <= "0110";
396 |   elsif(data_out = "00110111") then --ASCII 7
397 |     next_reg_adr(3 downto 0) <= "0111";
398 |   elsif(data_out = "00111000") then --ASCII 8
399 |     next_reg_adr(3 downto 0) <= "1000";
400 |   elsif(data_out = "00111001") then --ASCII 9
401 |     next_reg_adr(3 downto 0) <= "1001";
402 |   elsif(data_out = "01000001" or data_out = "01100001") then --ASCII A
403 |     next_reg_adr(3 downto 0) <= "1010";
404 |   elsif(data_out = "01000010" or data_out = "01100010") then --ASCII B
405 |     next_reg_adr(3 downto 0) <= "1011";
406 |   elsif(data_out = "01000011" or data_out = "01100011") then --ASCII C
407 |     next_reg_adr(3 downto 0) <= "1100";
408 |   elsif(data_out = "01000100" or data_out = "01100100") then --ASCII D
409 |     next_reg_adr(3 downto 0) <= "1101";
410 |   elsif(data_out = "01000101" or data_out = "01100101") then --ASCII E
411 |     next_reg_adr(3 downto 0) <= "1110";
412 |   elsif(data_out = "01000110" or data_out = "01100110") then --ASCII F
413 |     next_reg_adr(3 downto 0) <= "1111";
414 |   elsif(data_out = "00001000") then --Backspace
415 |     if(err > 0) then
416 |       next_err <= (err - 1);
417 |     end if;
418 |   else
419 |     next_err <= (err + 1);
420 |   end if;
421 | end if;

```

Şekil A1.6 : Reg state.

```

442 -- "READ(0) or WRITE(1):" mesajını içeren op_msg dizisinin konsol ekranına iletilmesi ve
443 -- kullanıcının konsol ekranına yazdığı "0" ya da "1" karakterinin
444 -- ASCII karşılığı alındığı için hangi karaktere basılmış kontrol edilmesi, bu datanın 2 bite map edilmesi.
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462 -- " Read (0) or write (1): " mesajı konsol ekranına iletilir.
463 if(count > 1 and count < 23) then
464     next_valid_in <= '1';
465     next_data_in <= op_msg(count - 2);
466 end if;
467
468 -----
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484 -- konsol ekranından gelen yazma ya da okuma verisi (işlem kodu) karşılık gelen 2 bite map edilir. okuma : 10, yazma : 01
485 if(cnt = 0) then
486     next_data_in <= data_out;
487     if(data_out = "00110000") then --ASCII 0
488         next_op <= "10";
489     elseif(data_out = "00110001") then --ASCII 1
490         next_op <= "01";
491     elseif(data_out = "00001000") then --Backspace
492         if(err > 0) then
493             next_err <= (err - 1);
494         end if;
495     else
496         next_err <= (err + 1);
497     end if;
498 end if;
499
500 -----

```

Şekil A1.7 : Opcode state.

```

520 when to_smi=> -- kullanıcın alınan phy adres, register adres ve opcode datalarının,
521               -- istat bitleri olan "01" ile beraber smi_22_control ile biğuna bağı olan sinyallere atanması
522               -- ayrıca kullanıcın girdiğı opcode kontrol edilerek read ya da write durumlarına geçilir.
523               next_valid_in <= '0';
524               next_reg_addr_temp <= reg_addr;
525               next_phy_addr_temp <= phy_addr;
526               next_op_temp <= op;
527               next_strt_temp <= "01";
528               if(op = "10") then
529                 next_state <= rd;
530               elsif(op = "01") then
531                 next_state <= wr;
532               end if;
533

```

Şekil A1.8 : To_smi state.

```

535 -- eğer kullanıcı read yapmak istiyorsa, "register value:" mesajını içeren data_rd_msg dizisinin konsol ekranına iletilmesi ve
536 -- slave'den okunan datanın 1 ya da 0 olduğunun kontrol edilmesi. okunan data konsol ekranına "1" ve "0" karakterlerinin ASCII
537 -- karşılığı 8 bit olarak yollanarak konsol ekranına yazdırılması.
538
539
540
541
542
543
544 -- "register value:" mesajı konsol ekranına iletilir.
545 if(count > 1 and count < 16) then
546   next_valid_in <= '1';
547   next_data_in <= data_rd_msg(count - 2);
548 end if;
549 -----
550
551 -- Okunan register verisinin 0 ya da 1 olduğu kontrol edilir ve kontrol sonucunda 0 ya da 1 karakterinin ASCII karşılığı konsol ekranına iletilir.
552 if(count > 17 and count < 34) then
553   next_valid_in <= '1';
554   if(phy_data_in(33 - count) = '0') then
555     next_data_in <= "00100001"; --ASCII 0
556   elseif(phy_data_in(33 - count) = '1') then
557     next_data_in <= "00110001"; --ASCII 1
558   end if;
559 end if;
560 -----
561

```

Şekil A1.9 : Rd state.

```

600 when wr => -- eğer kullanıcı write yapmak istiyorsa "Enter 16 bit data:" mesajını içeren wr_msg dizisinin
601 -- konsol ekranına iletilmesi ve kullanıcının konsol ekranına
602 -- girdiği 16 bit 1 ve 0 karakterlerinden oluşan datanın her birinin "1" ya da "0" karakterlerinin
603 -- ASCII karşılığına göre kontrol edilmesi. Alınan datanın
604 -- smi_22_controllier bloğundaki ilgili sinyalle atanarak, slave'de ilgili register'a yazılması.
605
623
624 -- "Enter 16 bit data:" mesajı konsol ekranına iletilir.
625 if(count > 1 and count < 21) then
626     next_valid_in <= '1';
627     next_data_in <= wr_msg(count - 2);
628 end if;
629 -----
648
649 -- 1 ve 2 numaralı if blokları ile konsol ekranından gelen 16 x 8 bitlik veri 0 ya da 1'in ASCII karşılıkları ile kontrol edilir.
650 -- kontrol işlemi yapıldıktan sonra 8'er bitlik 2 sinyale map edilir.
651 /* 1 */
652 if(cnt < 8) then
653     next_data_in <= data_out;
654     if(data_out = "00110000") then --ASCII 0
655         next_data_161(7 - cnt) <= '0';
656     elseif(data_out = "00110001") then --ASCII 1
657         next_data_161(7 - cnt) <= '1';
658     elseif(data_out = "00001000") then --Backspace
659         if(err > 0) then
660             next_err <= (err - 1);
661         end if;
662     else
663         next_err <= (err + 1);
664     end if;
665 end if;
666 /* 2 */
667 if(cnt > 7 and cnt < 16) then
668     next_data_in <= data_out;
669     if(data_out = "00110000") then --ASCII 0
670         next_data_162(15 - cnt) <= '0';
671     elseif(data_out = "00110001") then --ASCII 1
672         next_data_162(15 - cnt) <= '1';
673     elseif(data_out = "00001000") then --Backspace
674         if(err > 0) then
675             next_err <= (err - 1);
676         end if;
677     else
678         next_err <= (err + 1);
679     end if;
680 end if;
681 if(data_out = "00001000" and cnt = 16) then --Backspace
682     if(err > 0) then
683         next_err <= (err - 1);
684     end if;
685 end if;
686
687 -- Aşağıdaki if bloğunda gelen 8 bitlik 2 sinyal eğer hata yok ise yönetilen (slave) cihaza gönderilecek 16 bitlik sinyale map edilir.
688 if(data_out /= "00001000" and cnt > 15) then
689     next_valid_in <= '0';
690     if(data_out = "00001101") then
691         next_state <= wr_info;
692         next_cnt <= 0;
693         next_count <= 0;
694         if(err > 0) then
695             next_err <= 0;
696             next_state <= error;
697         else
698             next_phy_data_out(15 downto 8) <= data_161;
699             next_phy_data_out(7 downto 0) <= data_162;
700         end if;
701     end if;
702 end if;
703 -----

```

Şekil A1.10 : Wr state.

```

729
730 -- Aşağıdaki state'te konsol ekranından alınan verilerde hatalı giriş yapılması durumunda "Invalid entry !" mesajı basılarak
731 -- başlangıç durumuna dönlür.
732 when error=>
733
734 if(data_in_ready = '1' and data_in_ready_prev = '0') then
735     next_count <= (count + 1);
736 end if;
737
738 if(count < 21) then
739     next_valid_in <= '1';
740     next_data_in <= error_msg(count);
741 elseif(count = 21) then
742     next_valid_in <= '0';
743     next_state <= info;
744     next_count <= 0;
745     next_cnt <= 0;
746 end if;

```

Şekil A1.11 : Error state.

Ek B: FPGA Kaynak Kullanımı

Yazılan koda ilişkin kaynak kullanım bilgileri verilmektedir.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Tue Jun 09 22:05:17 2020
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	UART_TO_SMI_TEST
Top-level Entity Name	UART_TO_SMI_TEST
Family	MAX 10
Device	10M08SAU169C8G
Timing Models	Final
Total logic elements	814 / 8,064 (10 %)
Total registers	299
Total pins	6 / 130 (5 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 1 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

Şekil B1.1 : FPGA kaynak kullanım verileri.