İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK KAYNAK KODLU OPENRISC TABANLI KIRMIKÜSTÜ SİSTEMLERİN GERÇEKLENMESİ VE UYGULAMALARI

YÜKSEK LİSANS TEZİ

Latif AKÇAY

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

ARALIK 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

AÇIK KAYNAK KODLU OPENRISC TABANLI KIRMIKÜSTÜ SİSTEMLERİN GERÇEKLENMESİ VE UYGULAMALARI

YÜKSEK LİSANS TEZİ

Latif AKÇAY
(504131231)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Sıddıka Berna Örs YALÇIN

ARALIK 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504131231 numaralı Yüksek Lisans Öğrencisi **Latif AKÇAY**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"AÇIK KAYNAK KODLU OPENRISC TABANLI KIRMIKÜSTÜ SİSTEMLERİN GERÇEKLENMESİ VE UYGULAMALARI"** başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Sıddıka Berna Örs YALÇIN** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Sıddıka Berna Örs YALÇIN** İstanbul Teknik Üniversitesi	.....
	Prof. Dr. Ece Olcay GÜNEŞ İstanbul Teknik Üniversitesi	.....
	Doç. Dr. Alper ŞEN Boğaziçi Üniversitesi	.....

Teslim Tarihi : **27 Kasım 2015**
Savunma Tarihi : **29 Aralık 2015**

Sevgili Aileme...

ÖNSÖZ

Mikroişlemci teknolojileri bilgi çağının en önemli aktörlerinden birisidir. Gömülü sistemler ve bu sistemlerle yapılan uygulamalar insan hayatının hemen her alanında yer almakta ve giderek daha da yaygın hale gelmektedir.

İşlemciler genel olarak mimarileri ve tasarım ayrıntıları sadece üreticisi tarafından bilinen ve bir kez üretildikten sonra iç yapısı bir daha asla değiştirilemeyen somut nesneler olarak bilinmektedir. Bu yaygın düşünceyi genelde yazılım dünyasında daha çok bilinir olan açık kaynak kodlu tasarım yaklaşımı değiştirmektedir. Dünyanın her yerinden donanım tasarımı ile ilgilenen insanlar herhangi bir işlevi yerine getirmek üzere tasarladıkları donanım kodlarını internet üzerinden paylaşmaktadır. FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri) yeniden yapılandırılabilme yeteneğine sahip elektronik devrelerdir. Bu sayede FPGA yongaları üzerinde çalışabilecek işlemciler tasarlanmıştır. Bu yapılar gerçeklenebilir işlemci olarak adlandırılmıştır.

OpenRISC 1000; açık kaynak kodlu, hiç bir ticari talebi olmayan bir komut kümesi mimarisi geliştirmek üzere başlatılmış bir projedir. İşlemci ve çevresel birimlere ait kaynak donanım kodları ve birçok yardımcı yazılımlara erişmek, incelemek, değiştirmek ve kullanmak tamamen ücretsizdir. Bu çalışmada ülkemizin oldukça yabancı olduğu bu teknolojileri tanınması, işlemci ve donanım tasarımına meraklı olanlara iyi bir referans sunulması ve açık kaynak kodlu projelere bir katkı yapılması hedeflenmiştir. Bununla birlikte OpenRISC tabanlı kırmıküstü sistemlerle yapılabilecek çeşitli uygulamalar gerçekleştirilerek bu sistemlerin endüstriyel alanda oldukça kullanışlı ve bir o kadar da avantajlı olduğu gösterilmiştir. Tüm uygulamalar tekrar edilebilir ve başkaları tarafından kolayca anlaşılabilir olması amacıyla adım adım anlatılmıştır.

Beni bu teknolojilerle tanıştırıp bu alana yönelmemi sağlayan çok kıymetli hocam sayın Doç. Dr. Sıddıka Berna Örs Yalçın'a şükranlarımı sunarım. Karşıma çıkan tüm zorlukları aşmamda bana tecrübeleri ile destek olan, çalışmamı adım adım takip eden ve her konuda yardımcı olan değerli arkadaşım Y. Müh. Mehmet Tükel'e özellikle teşekkür ederim. Hayatımın her döneminde iyi yetişmiş bir insan ve milletine faydalı bir birey olabilmem için her türlü fedakarlığı göze alan sevgili aileme saygı ve sevgilerimi sunarım.

Aralık 2015

Latif AKÇAY
(Elektrik-Elektronik Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

FPGA : Field Programmable Gate Array
ASIC : Application Specific Integrated Circuit
HDL : Hardware Description Language
VHDL : Very High speed Integrated Circuit Hardware Description Language
GCC : GNU Compiler Collection
RISC : Reduced Instruction Set Computing

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

AÇIK KAYNAK KODLU OPENRISC TABANLI KIRMIKÜSTÜ SİSTEMLERİN GERÇEKLENMESİ VE UYGULAMALARI

ÖZET

Günümüz teknolojisinin hemen her alanda karmaşık elektronik sistemler hayati rol oynamaktadır. Bilgi çağının en önemli alanlarından birisi de elektronik mühendisliği ve çeşitli uygulamalarıdır. Elektronik sistemler söz konusu olduğunda en önemli bileşenlerden birisinin de sistemlerin beynini oluşturan, çeşitli birimleri uyumlu halde çalıştıran, sistemin işlevsel görevlerini yerine getirmesini sağlayan ve tüm işleyişini kontrol edip yöneten mikroişlemci (yada mikrodenetleyici) olduğu görülmektedir.

Sistem tasarımcısı açısından en önemli parametrelerden biri tüm sistemi yönlendirecek olan işlemcinin doğru tasarlanması veya doğru seçilmesidir.İhtiyaca yönelik işlemci tasarımı yapılabilir ancak günümüz şartlarında bu hiç kolay bir iş değildir. İşlemci tasarımı ve üretiminin oldukça karmaşık ve yüksek maliyetli bir süreç olması ihtiyaca özel işlemci tasarımını zorlaştırmaktadır [4]. Ayrıca yalnızca bir sistem için tasarlanan bir işlemcinin başka bir iş için kullanımı söz konusu olmayacaksa sırf bir veya birkaç defaya mahsus işlemci tasarımı yapmak maliyet açısından mantıklı bir yaklaşım değildir.

Tüm bu durumlar göz önüne alındığında gerçeklenebilir işlemci çözümlerinin çok önemli faydalar sunduğu görülmektedir. Gerçeklenebilir işlemciler kaynak kodları bir donanım tanımlama dili ile yazılmış ve FPGA veya Uygulamaya Özel Tümleşik Devre (ASIC) platformlarda gerçeklenebilen, yapısı ve çevresel birimleri üzerinde değişiklikler yapamaya imkan veren tasarımlardır [5]. Kullanıcı lisans durumuna göre tasarımın kaynak kodlarını ya hiç göremez veya tamamen görebilir ve değiştirebilir. Ancak tüm gerçeklenebilir işlemciler ihtiyaca göre değişik kombinasyonlarda yeniden tasarlanıp kullanılabilir. Bu özellikleri ile yukarıda sözü edilen birçok sorunun aşılması adına iyi bir seçenek olarak değerlendirilebilir.

Dünya üzerinde gerçeklenebilir işlemci geliştiren firmalar ve tasarım grupları bulunmakta ve dolayısıyla birden fazla sayıda çözüm bulunmaktadır. Bu çalışmada özellikle bu çözümlerin lisans farklılıkları ve bundan kaynaklı avantajlar üzerinden hareket edilerek açık kaynak kodlu tasarımların avantajları aktarılmıştır. Zira gerçeklenebilir işlemcilerin bazıları Xilinx veya Altera gibi ticari şirketler tarafından tasarlanmıştır ve kaynak kodları gizlidir. Bu işlemcileri ticari anlamda kullanmanın bir takım kısıtlamaları vardır. Bazı gerçeklenebilir işlemciler ise kaynak kodları açık olmasına rağmen ticari anlamdaki kullanımları ücret ödemeyi gerektirmektedir.

OpenRISC gerçeklenebilir işlemci ailesi ise diğer gerçeklenebilir işlemcilerden farklı olarak tüm kaynak kodları açık ve aynı zamanda ticari olsun olmasın tüm kullanımların tamamen ücretsiz olduğu bir platformdur. Bu zamana kadar çeşitli gerçeklemeleri

FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri) ve ASIC (Application Specific Integrated Circuit - Uygulamaya Özel Tümleşik Devre) ortamlarında yapılmış ve geliştirilmeye devam edilmektedir. Tamamen gönüllülerden oluşan bir ekibin katkılarıyla geliştirilen bu sistemler gerçeklenebilir işlemciler içerisinde önemli bir yer bulmuştur. Ayrıca bu sistemler için yazılım desteği de sunulmaktadır. GNU GCC derleyicisi sisteme uyarlanmış ve böylece kolayca yazılım geliştirmek mümkün hale getirilmiştir.

Bu çalışmada gerçeklenebilir işlemci ve kırmıküstü sistem olarak OpenRISC tabanlı platformlar tercih edilmiştir. Yapılan çalışmalar ve teknik detayları FPGA üzerinde test edilmiş ve sonuçlar paylaşılmıştır.

Bu çalışma işlemci tasarımı ve bilgisayar mimarisi konusunda meraklı olan araştırmacılara iyi bir örnek olması, ülkemizde milli çip tasarımına olan ilgiyi artırması ve gerçeklenebilir işlemci çözümlerinin tanınması noktalarında bir katkı sunması ümidi ile yapılmıştır.

IMPLEMENTATION AND APPLICATIONS OF OPEN SOURCE OPENRISC BASED SOC's

SUMMARY

In todays world, almost in every part of technology microprocessors and microcontrollers are widely used. Because of complexity of information systems and advanced software uses all such systems need a brain part which consist of one or more integrated circuits for managing the whole system or a particular issue related to the system.

It is possible to make many types of classifications about the processor technology. In this work, we just deal with a basic diversity which is called as softcore and hardcore processors. Hardcore processors are actually the well known element of this basic classification. As one can easily understand, the hardcore processor is a device that had been implemented and produced and ready to use it with an application. Despite that the softcore processor is a package that includes rtl code files which were written by using a hardware description language. A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis tools. It can be implemented via different semiconductor devices containing programmable logic (e.g. ASIC, FPGA, CPLD), including both high-end and commodity variations. Only few years ago, most systems, if they were using a soft processor at all, could only use a single soft processor. However in recent years, designers manage to map as many softcores as possible onto an FPGA.

There are many advantages and fidelities of using a softcore processor for a system thanks to its natural properties. Maybe the most important feature of softcore processors is that the flexibility of the architecture. Designers can change, add or subtract some of peripherals or size of different units and more about the processor. In addition to this, softcore processors can be easily tested by using an FPGA. Every single part of design can be controlled by using an application and wherby all real life reactans of processor can be seen. This is something more than a simulation which is also the only way to understand the behaviors of a hardcore processor. Another important advantage of a softcore processor is the platform diversity. Many producers have softcore processor designs and advanced tools to implement and test designs.

Sometimes using of only one processor is not eanough especially for large systems such as a warplane. For this kind of situations desining a new processor or microcontroller is something so expensive and so much waste of time. Soft core processors and FPGA (Field Programmable Gate Array) provide a very effective solution to this issue. Because FPGA's are reconfigurable integrated circuits and can be used as a microcomputer by using a softcore processor on it. Security is one of the most critical parameter during a system design especially for defence industry. By risng of

advanced technology, there has been a new war concept which is called as electronic warfare. For this reason the security parameter has become the most impartan factor for military electronic systems. So that it is as least impartan as having these kinds of systems for countries and for armies. If imported integrated circuits were used in military electronic systems of a country, then the security concern would be a problem for everytime. Therefore armies are desired to have military systems which are wholly produced with their national devices and also designed and tested in their country by a national company.

Turkey, after Cyprus Peace Operation, decided to produce national defence systems. As an importer counrty, after the embargo applied by foreign countries, Turkey had realised that to be depentent to other countries in war technologies is very dangerous and risky. Then many defence technology companies were established and supported from Turkish state and foundations. Particularly in recent years, Turkish defence industry has been growing rapidly. Many types of electronic and mechanic weapons are designed and produced by national firms. Currently, there are billon dolars worth of projects which are on-going or at the planning stage such as national tank, national warplane or long rage missile defence system. In addition to this conditions, Turkey wants to become an important exporter country in the field defence industry on the world.

Like many other countries, Turkey is still also an imported country in the field of microcontroller and microprocessor. Because, there are few dominant companies which produce these devices and they have a very colossal knowledge and many precious patents rigths. Additionally, produce and investment costs are at extraordinary high levels. Competition diffuculty with dominat companies is also such a big problem for this industry. Perhaps the most important missing thing is that lack of trained engineer and technician. Because of all these reasons, we can say that the idea of self-design and self-produce is mandatory but not close to perform in near future for microprocessors and microcontrollers.

We hope that this work will become a small but an important step through to the national microprocessor target of our country. Even if the hdl codes don't belong to us, this work may be a good guide for a new design. Thanks to open source feature of OpenRisc project, one can examine and learn how to design a processor, how to prepare a toolchain, how to port it to a operating system etc. Therefore, we believe in that this work will be a guide for newbies of softcore processor design.

OpenRisc processor family is one of the most famous architecture in open source world. It is a project to develop and open source instruction set architecture (ISA) based on reduced instruction set computing (RISC) methodology.

So many advanced properties about the OpenRisc architecure were explained in this thesis work. But the key feature about the OpenRisc is related with the license status. OpenRisc implemantations are wholly open source and it is free to use, change, sell or distribute all of the details. That makes OpenRisc very useful acccording to the other softcore processor solutions. Nobody has to pay for any of the project tools or any type of usage. Therefore OpenRisc has become the prefered platform for this work. Because our aim is to learn about architecture of processor design and suggest an open

and free system-on-chip solution for national projects and also to be a guide for people who wants to improve their skills on processor desing and implementation.

There have been several implemantation of OpenRisc1000 architecture. The most popular ones are the or1200 cpu and the new mor1kx cpu. During this work, both or1200 and mor1kx processor are used to try different application on FPGA.

For this work, Digilent Atlys board was choosen for trials done by using Orpsoc-v2 and Orpsoc-v3. Orpsoc(OpenRisc Referece Platform System-on-Chip) is a paltform that contains an OpenRisc processor and many different peripherals such as ethernet, usb, spi, i2c, uart, gpio, debugger and so on. During the work, different applications are made for using these peripherals and for testing whether such a big open source project is usefull or not enough for an industrial usage.

For a microprocessor work, the first thing needed is a compiler. Because if we can not compile an application then it does not make sense to have a procossor. GNU GCC toolchain had been ported for OpenRisc 1000 family with different C libraries. There are different solutions shared for bare metal and for Linux systems. The first thing done was the installing the toochain which is for bare metal systems. After that, we have also installed a good simulator which was designed for OpenRisc processors. So, we were able to compile codes for used platforms.

After finishing works for toolchain issue, we have downloaded orpsoc-v2 project files and started to work on it. We have made many applications using orpsoc-v2 platform. These applications were sometimes as easy as a simple hello world code. But we have also tried complex applications like U-Boot and Linux. The hard challenge was the modification process. We have added new costum designed peripherals to the wishbone network of platform. The new peripherals were tested by using them during applications.

During the work, all issues about the hardware design and implemantation such as synthesis , place and route or bitstream file production, we have used Xilinx ISE Desing Suite programs. There are many extra tools provided with this package. For example, we have made some kind of simulations by using ISIM simulator which was installed together with Xilinx ISE webpack.

We have done similar steps also for orpsoc-v3 platform. The difference between these two platforms were explained in details in this thesis work. We have also added our future plans about the work. We have good plans about getting platforms more independent from any other companies tools. In addition to this, there is an intention for designing an ASIC model of OpenRisc 1000 family processor based system-on-chip. We will try to add encryption and decryption peripherals to the wishbone network so that the chip will be able to use in the field of security system designs.

1. GİRİŞ

Bu bölümde yapılan çalışmada kullanılan bazı önemli kavramlar açıklanmıştır. Konuya ilişkin ayrıntılara girilmeden çalışmanın yapıldığı alan hakkında temel bilgiler verilmiş ve amaçlardan söz edilmiştir.

1.1 Amaç

Gömülü sistemler günümüz dünyasında oldukça yaygın olarak kullanılmaktadır. Gerçeklenebilir işlemciler kullanarak gömülü sistem tasarımı yapılması çeşitli faydalar sağlamaktadır.

Gerçeklenebilir işlemcilerin ve kırmıküstü sistemlerin tasarımı genel olarak sadece tasarımcısı tarafından saklı tutulmaktadır. Son yıllarda açık kaynak kodlu donanım tasarımları internet ortamında paylaşılmaktadır.Bu çalışmanın amacı gömülü sistem tasarımlarında kullanılabilecek açık kaynak kodlu işlemci ve kırmıküstü sistemlerin tanıtılması ve bu sistemler kullanılarak yapılan uygulamaların anlatılmasıdır.

1.2 HDL ve FPGA Kavramları

HDL (Hardware Description Language) bir mantık devresinin tanımlanması amacıyla kullanılan bilgisayar dillerini ifade eden genel bir kavramdır [6]. Bu diller bir devrenin yapacağı işi özel bir anlamsallık ile metin tabanlı olarak ifade etmeye imkan sağlamaktadır. Bu diller sayesinde tasarımcı devresinin nasıl çalışacağına ilişkin bir model tanımlamaktadır. Donanım tanımlama dili (HDL) ile yazılmış bir kod, sentezleyici olarak adlandırılan bir program ile işlenirr. Bu program metin tabanlı bir kod dizisinden mantıksal donanım blokları ve bu bloklar arasındaki bağlantıyı tanımlayan bir başka dosya üretir.

Donanım tasarlama dillerinin tarihi gelişimi yetmişli yılların sonunda başlamış ve ortaya çok sayıda farklı dil çıkmıştır [6]. Ancak günümüzde neredeyse tüm

dünyada standart haline gelmiş olan Verilog ve VHDL dilleri kullanılmaka ve desteklenmektedir. Bu çalışmada kullanılan işlemci ve tasarlanan donanımlar da Verilog HDL kullanılarak tanımlanmıştır.

FPGA (Field Programmable Gate Array - Alanda Programlanabilir Kapı Dizileri) , programlanabilir mantık blokları ve bu bloklar arasındaki ara bağlantılardan oluşan ve geniş uygulama alanlarına sahip olan sayısal tümleşik devrelerdir [7].

FPGA, programlanabilir mantık blokları, bu blok dizisini çevreleyen giriş-çıkış blokları ve ara bağlantılar olmak üzere düzenlenebilir üç ana bölümden oluşur. Programlanabilir mantık blokları, ara bağlantılar içerisine gömülü şekilde bulunur. Programlanabilir mantık bloklarının yapılandırılması ve bu bloklar arasındaki iletişim ara bağlantılar sayesinde gerçekleşir. Giriş çıkış blokları, ara bağlantılar ile bütünleşmiş devrenin paket bacakları arasındaki ilişkiyi sağlar [7].

Düşük maliyetleri ve tasarım esnasında esneklik sunan doğası sayesinde FPGA'ler çok geniş bir alanda kullanılmaktadır. Özellikle paralel işaret işleme gerektiren uygulamalarda çokça kullanılmaktadır [7].

1.3 Gerçeklenebilir İşlemciler

İşlemci dendiği zaman genelde akla gelen milyonlarca ve hatta daha fazla transistorün bir araya getirilip çok küçük bir alanda büyük bir sistemi yönetebilecek şekilde organize edilmiş olan ve bir defa üretildikten sonra üzerindeki donanımın değiştirilmesi mümkün olmayan tümleşik devrelerdir. Gerçeklenebilir işlemci olarak adlandırılan tasarımlar ise bu tanımı değiştirmektedir. Bu işlemciler bir donanım tanımlama dilinde yazılmış, FPGA yada ASIC üzerinde gereçeklenebilen ve ihtiyaca göre yapıları değiştirilebilen sistemlerdir. Tıpkı bir yazılımın kodları üzerinde değişiklik yapar gibi bu işlemcilerinde kodları üzerinde de değişiklik yapılabilir ve kullanıcının arzusu doğrultusunda çevresel birimler istenildiği gibi düzenlenebilir ve hatta sisteme yeni çevresel birimler eklenebilir.

Gerçeklenebilir işlemcinin lisans durumuna göre tasarımın kaynak kodlarının paylaşımı ve kullanılabilirliği değişmektedir. Yani eğer işlemci ticari bir lisansla tasarlanmış ve kullanıcılara öyle sunulmuşsa kullanıcı işlemcinin kaynak kodlarına

erişemez, fakat kendisine sunulan bir arayüz vasıtasıyla işlemci tasarımcısının izin verdiği ölçüde değişiklikler yapabilir. Eğer tasarımcılar kaynak kodlarını paylaşıyorsalar, kullanıcı işlemcinin tüm kaynak kodlarını en ince ayrıntısına kadar görebilir ve bu kodları kendi ihtiyaç duyduğu biçimde kullanabilir. Bazı durumlarda ise gerçeklenebilir işlemcinin kaynak kodları tamamen olmasına rağmen ticari amaçlı kullanımlar tasarımcı firma tarafından ücrete tabi tutulmaktadır. Fakat bunun dışında eğitim amaçlı ve ticari olmayan durumlarda firma herhangi bir ücret talebinde bulunmayabilir. Bu çalışmada da tercih edilen OpenRISC tabanlı sistemler ise hem açık kaynak kodlu hemde ne amaçla olursa olsun kullanımı ücretsizdir [1].

Gerçeklenebilir işlemciler ve çevresel elemanlarla birlikte oluşturulan kırmıküstü sistemler gömülü sistem tasarımları için önemli avantajlar sağlamaktadır. Bu sistemler tasarım zamanını önemli ölçüde kısaltmakta, mümkün olan hataların tespit edilmesini kolaylaştırmakta ve oldukça esnek bir tasarım imkanı sunmaktadır [5]. Bu özellikleriyle sebebiyle sıklıkla tercih edilmektedirler.

2. OpenRISC 1000 AÇIK KAYNAK KODLU KOMUT KÜMESİ

2.1 OpenRISC 1000 Komut Kümesi Mimarisi

OpenRISC 1000 açık kaynak kodlu komut kümesi mimarileri geliştirmek üzere, indirgenmiş komut kümesi mimarisi, RISC (Reduced Instruction Set Computing), prensipleriyle tasarlanan ve tamamen gönüllü bireylerin katılımıyla yürütülen bir projedir.

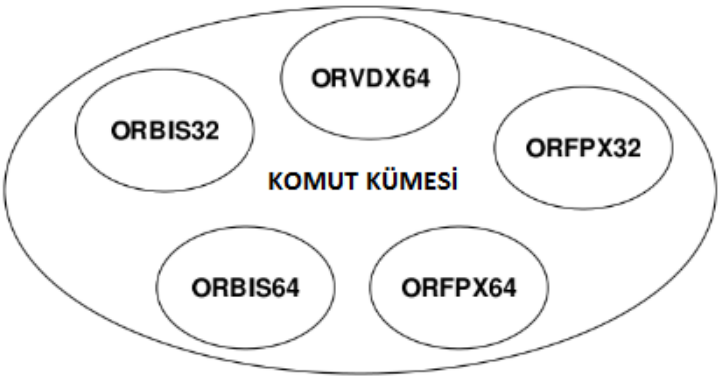
Bu kapsamda yazılan kodlar tamamen açıktır ve dileyen herkes bedel ödemeden bu tasarımları kullanma ve üzerinde değişiklik yapma hakkına sahiptir [8]. Bu kapsamda sunulan lisansın temel olarak sunduğu özgürlükleri şu şekilde sıralamak mümkündür;

- Kodları sınırsız kullanma hakkı
- Kaynak kodlarına erişebilme ve inceleme hakkı
- Kaynak kodları serbestçe dağıtma hakkı
- Kaynak kodları değiştirme hakkı

Bu mimaride geniş bir yelpazede (gömülü sistemler, otomotiv, taşınabilir bilgisayarlar vs) kullanılabilecek 32 ve 64 bit komutlar üretmeyi hedeflenmiştir [1].

Bilindiği gibi işlemciler komut kümelerinin gerçeklenmesiyle tasarlanmaktadır. Şekil 2.1'de OpenRISC 1000 komut kümesinin içerdiği 32 ve 64 bit destekli komut kümeleri gösterilmektedir. OpenRisc 1000 ailesinden bir işlemci gerçeklendiği zaman bu komut kümelerinden asgari olarak birisini desteklemek zorundadır [1].

Şekil 2.1'te görüldüğü gibi birden fazla sayıda komut kümesi mevcuttur. Bunlardan OpenRisc Temel Komut Kümesi (ORBIS32 - OpenRISC Basic Instruction Set) tüm gerçeklemelerde kullanılması zorunlu olan kümedir ve genel işlemci komutlarını içerir. Örneğin toplama, çıkarma, çarpma, kaydırma, sıçrama gibi temel algoritma işlemleri



Şekil 2.1: OpenRisc 1000 Komut Kümesi [1].

için tasarlanmış olan komutlar bu kümeye dahildir. Bu kümenin 64-bit olan versiyonu ise ORBIS64'tür. Bunun yanı sıra Şekil 2.1 görülen OpenRISC Vektör ve Sinyal İşleme Komut Kümesi (ORVDX64 - OpenRISC Vector/DSP Extension) kümesi 32-bit uzunluğunda komutlar içeren ve 64-bit uzunluğunda veri işlenen özel komut kümesidir. Ayrıca Kayan Noktalı Sayı İşlemleri Komut Kümesi (ORFPX32 - OpenRISC Floating Point Extension) ise kayan noktalı sayılar üzerine işlemler yapmak üzere tasarlanmış komut kümesidir [1].

Tüm bunlara ek olarak komut kümesinde kullanıcı da kendine özel olarak komutlar ekleyebilsin diye boş komutlar da tasarlanmıştır. Bu sayede kullanıcı kendi işine yarayacağının düşündüğü komutları bu boş komutlara koyarak gerçekleyebilir.

OpenRISC 1000 komut kümesi tasarımı yapılırken RISC prensipleri esas alınmıştır. Bu anlayışın temeli komut sayısını azaltarak işlemci devresini küçültmek ve böylece daha etkin bir gerçeklemeye imkan vermektir [9]. RISC prensiplerine sahip tasarımlar özellikle gömülü sistemlerde sıklıkla tercih edilir [10]. Son yıllarda akıllı telefon ve tablet gibi ürünlerin tüm dünya çapına yayılması ile birlikte RISC işlemcilerinin kullanımı daha da yaygınlaşmıştır [11].

OpenRisc 1000 komut kümesinde komutlar 32 veya 64 bit uzunlukta sabitlenmiştir. Eğer işlemci 32-bit olarak tasarlanmışsa komut kümesi 32-bit uzunlukta komutlardan, 64-bit tasarlanmış ise uzunluğu sabit 64-bit komutlardan oluşan bir komut kümesi kullanılır [1]. Komutlar birbirinden ayırt edilmesi açısından komutların baş harflerinde l., lv., lf., gibi karakterler kullanılmıştır. Bu adlandırma bir komutun kolayca hangi kümeye ait olduğunu şu şekilde gösterir:

l.add **İŞARETLİ SAYI TOPLA** **l.add**

31 26	25 . . . 21	20 . . . 16	15 . . . 11	10	9 8	7 . . 4	3 . . 0
K.KODU 0x38	D	A	B	AYRIK	K.KODU 0x0	AYRIK	K.KODU 0x0
6 bit.	5 bit	5 bit	5 bit.	1 bit.	2 bit	4 bit	4bit

Şekil 2.2: OpenRISC 1000 Komut Kümesine Ait Örnek Komut Tasarımı [1].

- l. : ORBIS32/64
- lv.: ORVDX64
- lf.: ORFPX32/64

Şekil 2.2 bir komutun yapısını göstermektedir. Diğer komutlar da benzer biçimde tasarlanmıştır.

Format:

```
l.add rD,rA,rB
```

32-bit Gerçekleme Modeli:

```
rD[31:0] <- rA[31:0] + rB[31:0]
SR[CY] <- carry
SR[OV] <- overflow
```

Komut Tanımı:

Genel amaçlı kullanım yazmacı rA'nın içeriği, genel amaçlı kullanım yazmacı rB'nin içeriği ile toplanarak, sonuç genel amaçlı kullanım yazmacı rD'ye yazılır.

Yukarıda da görüldüğü gibi komut kümesi oldukça basit ve anlaşılır bir biçimde tasarlanmış ve açıklanmıştır. Ayrıca hem bir model olması hem de ifadelerin kafa karışıklığına yol açması durumunda gösterimden anlaşılabilsin diye bir de gerçekleme

modeli verilmiştir. Mimariye ilişkin tüm bilgilere [1] numaralı kaynaktan erişmek mümkündür.

2.2 Adresleme Modları

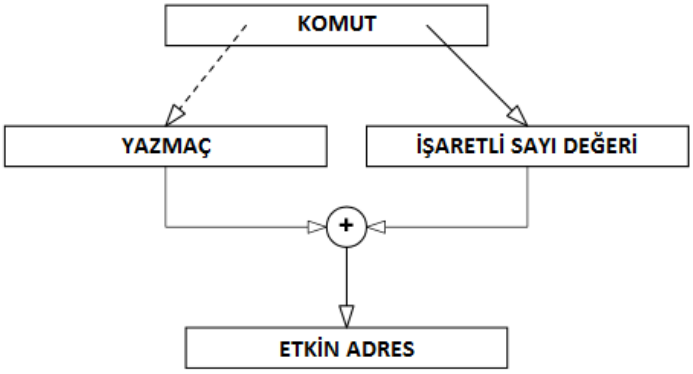
Adresleme modu bir işlemcinin hafıza birimlerine nasıl erişeceği, o birimleri ne şekilde kullanacağı, adres hesaplamalarını nasıl yapacağı gibi bellek yönetimine ilişkin konuları içeren bir kavramdır. OpenRISC işlemcilerinin adresleme modları daha ayrıntılı olarak [1] numaralı kaynaktan öğrenilebilir. Bu çalışma kapsamında çok derine inmeden genel olarak işlemcinin adres hesabını nasıl yaptığına ilişkin kısa bilgiler verilmesi uygun görülmüştür.

İşlemci bir sonraki ardışıl komuta geçerken veya bir bellek adresine ulaşmayla ilgili komutu çalıştırıken etkin bir adres hesaplar. Eğer hesaplanan adres maksimum mantıksal adresi aşıyorsa adres işleci maksimum etkin adresten minumum etkin adrese doğru düşürülür [1]. Yani böyle bir durumda maksimum adres aşımına uğranırsa fazlalık kadar sıfır adresinden yukarıya doğru gidilmiş olur.

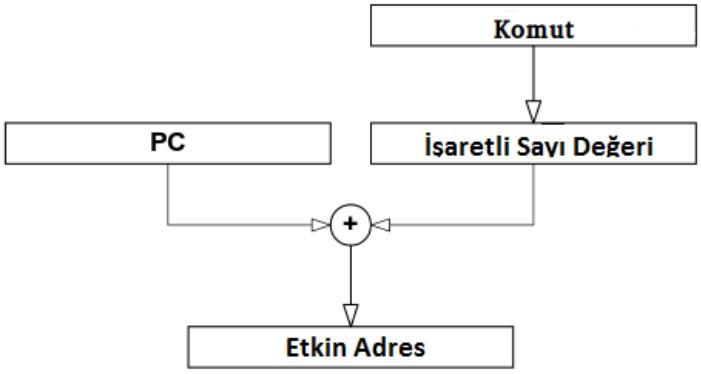
İşlemci iki farklı teknikle adres hesaplama yapmaktadır. Bazı komutlar bir tekniği bazıları ise diğer tekniği kullanmaktadır. Bu teknikler Yer Değiştirmeyle Dolaylı Adresleme (Register Indirect With Displacment Mode) ve Program Sayıcıya Bağlı Adresleme (Program Counter Relative Mode) olarak adlandırılmıştır [1].

İlk teknik (Yer Değiştirmeyle Dolaylı Adresleme) yükle/sakla komutları tarafından kullanılır. Bu modda 16-bit uzunluğunda bir değer, işaretli şekilde uzatılarak (32-bit) komuta işleç olarak girilmiş olan yazmacın içeriği ile toplanarak adres hesaplanır [1]. Şekil 2.3 bu modun işlevini daha açık bir biçimde göstermektedir.

Program Sayıcıya Bağlı Adresleme tekniği ise dallanma (branch) komutları tarafından kullanılmaktadır. Bu teknikte komuta işleç olarak girilen 26-bit uzunluğundaki değer, işaretli olarak uzatılır (32-bit) ve içeriği program sayıcı yazmacının içeriği ile toplanarak etkin adres oluşturulur [1]. Şekil 2.4 bu modun işlevini daha açık bir biçimde göstermektir.



Şekil 2.3: Yükle/sakla Komutlarının Kullandığı Adresleme Modu [1].



Şekil 2.4: Dallanma Komutlarının Kullandığı Adresleme Modu [1].

OpenRISC işlemcilerinde hafızada saklanan değerlerin uzunluklarına göre adlandırılması alışılagelmiş 32-bit işlemcilerde olduğu gibidir. Bunları Çizelge 2.1'deki gibi sıralamak mümkündür.

Çizelge 2.1: Veri Türleri Ve Uzunlukları.

Veri Türü	Bayt Türden Uzunluğu	Bit Türden Uzunluğu
Bayt	1	8
Yarım Kelime	2	16
Kelime	4	32
Çift Kelime	8	64

OpenRISC 1000 mimarisi sunduğu veri yerleşim tekniği (byte ordering) konusunda bir şart içermemektedir. Bu kavram bir bayt veriden daha uzun bir veri saklanacağı zaman verinin en anlamlı bitinin hangi bayttan başlayacağını bildirmek için kullanılan bir düzenlemedir. Büyük en sonda (big endian) ve küçük en sonda (little endian) olarak iki farklı teknik bulunmaktadır. OpenRISC işlemcileri gerçeklemeye göre iki çeşit sıralamayı da kullanabilir. Mimari buna yönelik bir şart içermese de varsayılan

Bit 31	Bit 24	Bit 23	Bit 16	Bit 15	Bit 8	Bit 7	Bit 0
YÜKSEK ANLAMLI						DÜŞÜK ANLAMLI	
BAYT ADRESİ 0		BAYT ADRESİ 1		BAYT ADRESİ 2		BAYT ADRESİ 3	

Şekil 2.5: Verilerin Big Endian Formatında Saklanması [1].

teknik büyük en sonda biçimdir [2]. Bu sıralamada en anlamlı veri biti sıfırıncı bayt içinde saklanır. Bu şekilde devam edilerek verinin uzunluğuna göre bitler yerleştirilir. Şekil 2.5'de bir kelime uzunluğundaki verinin bellekte hangi baytlarda saklanacağı gösterilmektedir.

2.3 Yazmaçlar

OpenRisc işlemcilerinde yazmaçlar genel olarak iki başlık altında incelenmektedir. Bunlar genel amançlı yazmaçlar ve özel amaçlı yazmaçlardır [2]. İsimlerinden de anlaşıldığı gibi genel amaçlı yazmaçlar programcının bir uygulama yazarken kullanması için sunulmuş yazmaçlar iken özel amaçlı yazmaçlar işlemcinin kendi çalışması esnasında bir takım değerleri sakladığı yazmaçlardır. Bu yazmaçlara ulaşmak için kullanılan l.mtspr ve l.mpsfr adlı iki komut tanımlanmıştır. Bu komutlar örneğin kesme (interrupt) geldiği zaman sürekli kesme fonksiyonuna dallanmayı önlemek için kesme bilgisinin saklandığı yazmacı silmek gibi işlemlerde kullanılır.

Genel amaçlı yazmaçlar 32 bit gerçeklemelerde 32-bit 64-bit gerçeklemelerde 64-bit bilgi tutacak uzunlukta ve 32 adettir. OpenRisc 1000 mimarisi bunlardan 16 yada 32 tanesi nin gerçeklenmesine olanak vermektedir [1]. Genel amaçlı yazmaçlar bir komuta hem kaynak hemde hedef yazmacı olarak kullanılabilmektedir. Yazmaç adlandırmaları ise R0 – R31 biçimindedir [1]. Fakat özel olarak R0 yazmacının uygulamalarda hedef yazmaç olarak kullanılmaması tavsiye edilmiştir [1]. Özel amaçlı yazmaçlar ise kullanıldıkları birimlere göre gruplandırılmış olup OpenRISC 1000 komut kümesi mimarisi dökümanlarında genişçe açıklanmıştır [1].

2.4 OR1200 İşlemci Mimarisi

OpenRISC 1000 gerçeklemelerinin ilki Damjen Lampret tarafından 2000 yılında OR1200 olarak adlandırılan ve sadece işlemci çekirdeği tasarımını içeren gerçek-

lemedir [2]. OpenRISC 1000 sadece komut kümesi mimarisidir. Yani gerçeklemeye yönelik donanım kodları içermez. Oysa OR1200 doğrudan bir donanım tasarımıdır ve yapısı OpenRISC 1000 komut kümesi mimarisine dayanmaktadır. Örneğin veri belleği boyutu veya zamanlayıcı birimi olup olmayacağı gerçeklemeyi yapan kişilere bağlıdır. OpenRISC 1000 komut kümesi mimarisi ile tasarlanan iki gerçeklenebilir işlemci bulunmaktadır: OR1200, mor1kx.

OR1200 CPU (Central Processing Unit)'nun tasarımı ilk yazıldığından bu yana tasarımcıların ve kullanıcıların katkı ve geri dönüşleri ile zaman içerisinde değiştirilmiştir. Ancak bu eklemeler işlemcinin temel özelliklerini ve mimarisini değiştirmek adına yapılan güncellemeler değil daha çok muhtemel hataların giderilmesi veya kodun daha etkin çalışması için yapılan değişikliklerdir. Daha radikal değişiklikler yapılacaksa yeni bir isimle aynı mimariye sahip başka bir işlemci tasarlanır ki buna örnek olarak ilk tasarımı 2012 yılında yapılan ve bu çalışmada da anlatılacak olan mor1kx işlemcisi verilebilir.

OR1200 işlemcisinin genel özelliklerini şöyle sıralamak mümkündür [2];

- İşlemcinin tüm özellikleri kullanıcı tarafından değiştirilebilir.
- Yüksek performanslı işlem yapabilme yeteneği
- Yüksek hızlarda çalışan veri bellek ve bellek yönetimi
- Wishbone arayüz uyumu
- İşlemci parametrelerinin kullanıcı tarafından kolay değiştirilebilir olması

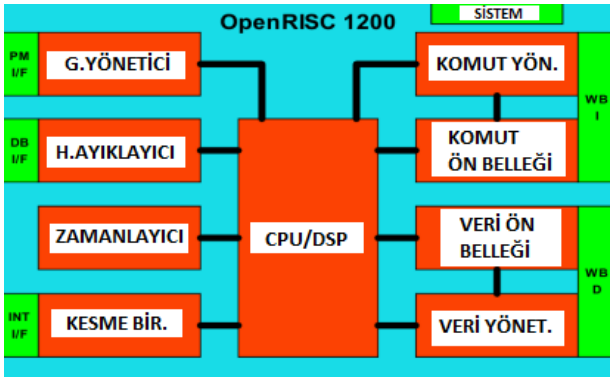
Bu genel özelliklerin ardından işlemciye biraz daha yakından bakıldığında bazı fonksiyonel özellikleri ise şu şekilde sıralanabilir [2];

- CPU - DSP (Digital Signal Processing Unit)
- Komut ve veri ön belleği
- Güç (enerji) yönetimi birimi ve arayüzü
- Zamanlayıcı

- Kesme kontrol birimi ve arayüzü

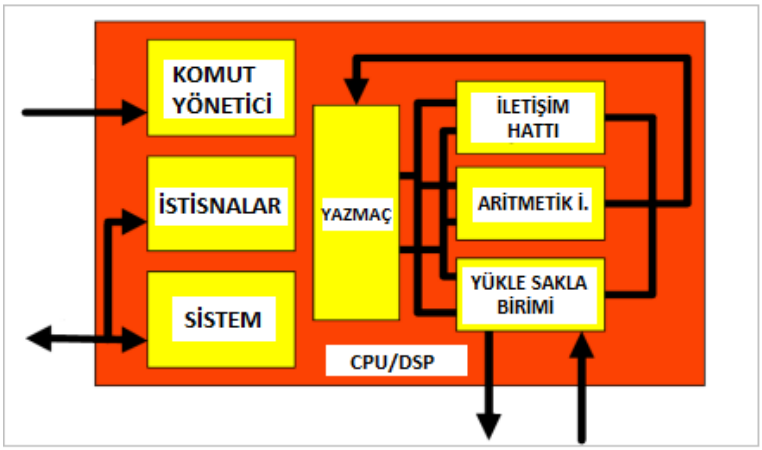
Bu özellikleri artırıp yeni birimler eklemek veya varolan birimlerden bazılarını çıkarmak mümküdür.

Şekil 2.6'dee OR1200 işlemcisinin yapısı gösterilmektedir. Bu yapı işlemcinin tasarımcılar tarafından sunulan varsayılan halidir. Ancak herhangi bir kimse bu formatı değiştirip kodları bu kendi değiştirdiği biçimde internet ortamından yayınlayabilir. Dolayısıyla başka kişilerin sunduğu kodların bu yapıyla örtüşmesi garanti edilemez.



Şekil 2.6: OR1200 İşlemci Yapısı [2].

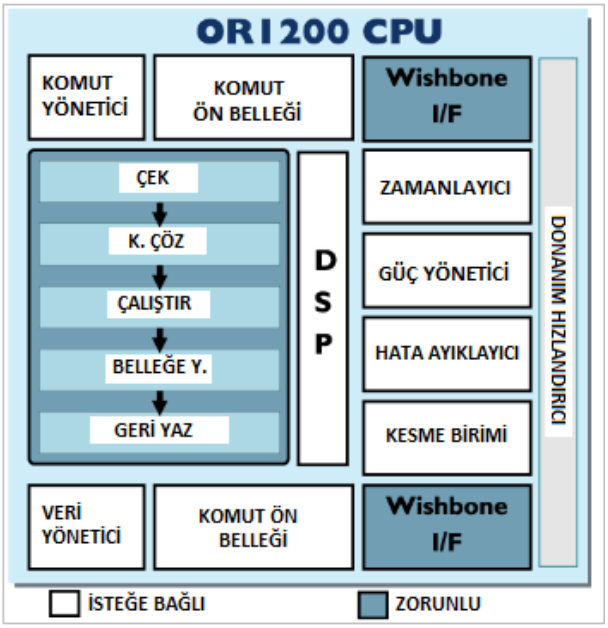
Şekil 2.6'de görünen işlemcinin merkezi işlem birimi ve sayısal sinyal işleme biriminin yapısı ise Şekil 2.7'den görülebilir.



Şekil 2.7: OR1200 Merkezi İşlem Birimi Bloğu [2].

İşlemcinin tasarımcılar tarafından sunulan formatında bazı birimleri devreden çıkarmak mümkündür. Ancak kodların çalışabilmesi için mecburen bazı birimlerin de bulunması gerekmektedir. Şekil 2.8'te zorunlu ve isteğe bağlı olarak çıkarılabilecek bölümler gösterilmektedir.

Bu yapıya ilişkin bir değişiklik yapılacağı zaman sadece değişiklik yapılacak birimin güncellenmesi yeterli olmayabilir veya bu işlem bir hataya da yol açabilir. Bunun sebebi birimlerin arasındaki bağımlıklıklardır. Bu nedenle değişiklik yapılmadan önce tüm bağlantılar gözden geçirilmelidir.



Şekil 2.8: OR1200 Gerçeklemesinde İsteğe Bağlı Ve Zorunlu Birimler [2].

2.5 mor1kx İşlemcisi

mor1kx işlemcisinin ilk versiyonu Julious Baxter başta olmak üzere bir grup OpenCores topluluğu üyeleri tarafından 2012 yılında açıklanmıştır. İlk yayınlandığı tarihten itibaren işlemciye ait kodlar güncellenmeltedir.

Tez çalışması döneminde mor1kx işlemcisi üzerinde de çalışmalar yapılmış ve hatalı çalıştığı tespit edilen bazı kısımlar düzeltilip test edildikten sonra projeden sorumlu gönüllülere bu hatalar bildirilerek düzeltilmesi sağlanmış ve böylece bu yeni tasarıma geliştirme anlamında da destek verilmiştir.

İşlemci tasarımında esas alınan en önemli iki madde, daha fazla esnek tasarım imkanı sunulması ve daha kaliteli kodlamadır. OR1200 işlemcisine göre kullanıcıya daha fazla esneklik sunulmuştur. Örneğin kesme birimi OR1200 işlemcisinde tek çeşit iken mor1kx işlemcisinde iki çeşit kesme biriminden istenilen kullanılabilir.

Bununla beraber işlemci üç farklı seviyede iletişim hattından birisi tercih edilerek gerçeklenebilir.

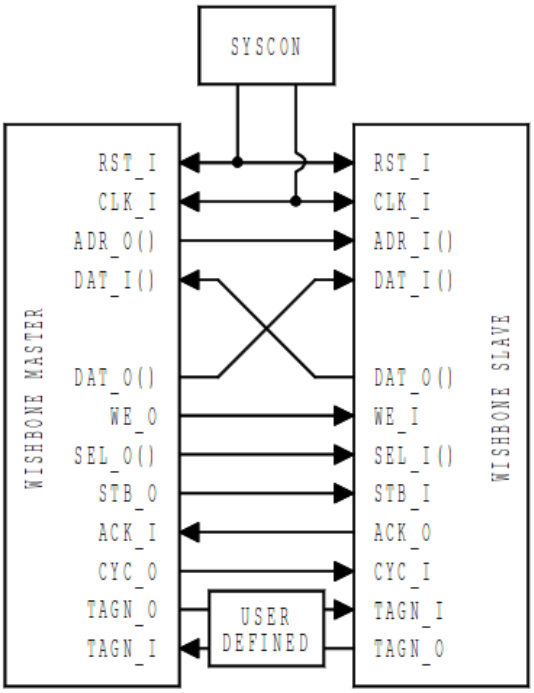
2.6 Wishbone Kırmık Üstü Sistem Arayüzü

Wishbone arayüzü bir sistemdeki donanımlar arasınndaki haberleşmenin sağlanması için açık kaynak kodlu ve esnek özellikleri olan bir veri yoludur [3]. Bu mimari birden fazla birimin aynı anda varolduğu bir çip içersinde belirli kurallar koyarak işleyişi, hiyerarşiyi ve haberleşmeyi sağlamak için tasarlanmıştır. Wishbone tasarımı da yine OpenCores topluluğu tarafından yapılmış ve tüm diğer projelere benzer özellikler taşımaktadır. Açık kaynak kodu ve ücretsiz kullanım başta olmak üzere kullanıcının ihtiyacına göre arttırıp azaltılabilecek esnek yapı da sunulmuştur.

OpenRISC işlemci gerçeklemeleri de bu arayüz kullanılarak yapılmıştır. İşlemci diğer birimlerle wishbone arayüzü üzerinden haberleşir ve tüm bilgi alış verişi bu hat üzerinden yapılır. Bir modül wishbone arayüzüne bağlanmak için o sistemdeki wishbone gerçeklemesinin durumuna göre bazı giriş ve çıkış portları içermek zorundadır. Çoğu zaman her modül için wishbone arayüzü ayrı olarak yazılır ve bu arayüz modüle bağlanır. Sistem modülle bilgi alış verişi yapacağı zaman modülün kendisi ile değil wishbone arayüzü ile bağlantı kurar. Bu arayüz modülle bağlantı halinde olduğu için modüle gelen bilgileri iletir veya modülden aldığı bilgiyi ilgili birimlere ulaştırır.

Wishbone arayüzünün özellikleri ve yetenekleri OpenCores tarafından yayınlanan anlaşılır ve kolay uygulanabilir dökümanlar aracılığı ile öğrenilebilir [3].

Wishbone arayüzü; bazı sinyalleri tanımlanması zorunlu, bazıları ise kullanıcıya bağlı olarak iki grup sinyalden oluşur. Eğer bir sinyal zorunlu gruptaysa kullanıcı o sinyalden aldığı bilgiyi kullanmıyor bile olsa muhakkak arayüzünde port olarak belirtmek zorundadır [3]. Örneğin kullanıcı modülü hatalı bir iş yaptığını bildiren bir sinyale sahip olmamış şekilde tasarlanmış olsun. Kullanıcı kendi modülünden bu yönde bir hata beklemiyor veya bu durumun oluşması halinde de bunun bilgisini almayı gereksiz görüyor olabilir. Ancak modülün bağlanacağı sistemde modüllerdeki hatalı işlemleri bildiren bir sinyal olduğunu varsayalım. Bu sinyal modül için



Şekil 2.9: Wishbone Master-Slave Sinyalleri [3].

slave(köle- yazılan), wishbone arayüzü için ise master (efendi) ise (yani modül arayüzünde output olarak tanımlanmışsa) kullanıcı bu sinyali doğrudan sıfır değerine bağlayıp işleyişi aksatmadan ve arayüzü bozmadan işlemi gerçekleştirmiş olur. Bunun dışında şayet ihtiyaç duyulmayan sinyal bir giriş sinyali ise bu durumda kullanıcı arayüzde port olarak sinyali açar fakat hiçbir şekilde kullanmak zorunda değildir. Sinyal sadece arayüzde açık bir port olarak durur. Genel olarak bir master ve bir slave arasındaki sinyaller Şekil 2.9'da gösterilmiştir.

Wishbone arayüzünde kullanılması zorunlu olan sinyallerin neler olduğunu ve ne anlamlar ifade ettiğini ifade etmek gerekmektedir. Bu sinyallerden en önemlileri aşağıda sırasıyla açıklanmıştır.

- wb_clk : Senkron saat sinyalini göstermektedir.
- wb_rst : Senkron reset sinyalini göstemektedir.
- wb_adr_i - wb_adr_o : Adres bilgisini taşıyan sinyali göstemektedir.
- wb_dat_i - wb_dat_o : Modüle gelen veya modülden çıkan bilgi sinyalini göstemektedir.

- wb_we_i - wb_we_o : Modüle yazmaya hazır olunduğu bilgisi verilmek için kullanılır. Bu sinyal yükseltilmeden modüle veri yazılamaz.
- wb_sel_i - wb_sel_o : word uzunluğunda (32-bit) bilginin hangi byte'larının seçildiğini ifade eder. Bu sinyal 4 bitliktir. Örneğin 0001 ise ilk byte seçildir.
- wb_stb_i - wb_stb_o : Aslında modül seçildi bilgisini verir. Bu sinyal yükseldiğinde adres yolunda geçerli adresin yüklendiğini gösterir.
- wb_ack_i - wb_ack_o : Okumaya hazır olunduğunun ya da yazmanın bittiğini gösterir.
- wb_cyc_i - wb_cyc_o : Bu işaret yükseldiği zaman modüle yazma veya modülden okuma işi sürmektedir bilgisi verilmiş olur. Örneğin 32-bit bir bilgi veri yoluna konulduktan itibaren yazma işi bitinceye kadar sinyal yükseltilir.
- wb_er_i - wb_err_o : Anormal bir durum olduğunu ifade eden sinyaldir. Ne şekilde davranılacağı gerçekleyen tarafından belirlenir. Seçime bağlı bir sinyaldir.
- wb_rty_i - wb_rty_o : Herhangi bir sebepten ötürü yazma veya okuma işi yapılamadığı ve bu sebeple tekrar deneniyor olduğunu ifade eder. Ne şekilde davranılacağı gerçekleyen tarafından belirlenir. Seçime bağlı bir sinyaldir.

Bunlara ek olarak başka sinyaller tanımlanmasına da izin verilmiştir. Bu gibi işlemlerin ve okuma yazma esnasında ayrıntılı olarak hangi sinyallerin aktif yada pasif edildiği gibi detaylar ve wishbone arayüz hakkında diğer bilgiler için resmi açıklama dökümanına bakılabilir [3].

3. DERLEYİCİ TAKIMI KURULUMU VE KULLANIMI

Bu bölümde ORPSoC-v2 (OpenRisc Reference System On Chip Platform) üzerinde yapılan çalışmalar anlatılmıştır. Bu gerçekleme or1200 işlemcisi ve çevresel birimlerini içeren bir platform olup Linux Mint 17.1 işletim sistemi ile çalışan bir bilgisayar üzerinde derleyici takımı kurulumu yapılmış ve bu şekilde uygulamalar için kod derlemeleri yapılabilmiştir. FPGA ile ilgili uygulamalar için Xilinx firmasına ait bazı yazılımlar kullanılmıştır [12]. Xilinx ISE Design Suite 14.7 programı kullanılarak kaynak kodlar derlenip FPGA'ya transfer edilmiştir [13]. FPGA üzerindeki uygulamalar için Digilent Atlys geliştirme kartı kullanılmıştır [14].

3.1 GNU Derleyici Takımı Kurulumu

Derleyici Takımı yazılım dünyasında metin editörü üzerinde yazılmış bir kodun makina komutları haline getirilebilmesi için gerekli tüm programların oluşturduğu bir grup yazılım anlamına gelir. Bir program metin editöründe yazılmış bir kodu örneğin ön işlem aşamasından geçirirken diğer bir program onun çıktısını alıp sıradaki işlemi yapar. Bu sayede istenilen formatta çalıştırılabilir dosyalar oluşturulur. Bununla birlikte derleyici takımı yazılım kodları üzerinde daha farklı amaçlar için de kullanılmaktadır.

Bu çalışmada kullanılan işlemcinin (or1200 CPU) kendine has bir komut kümesi olduğu için işletim sisteminde bulunan GNU GCC (GNU Compiler Collection) derleyicisi işlemci için kod derleyemez [15]. Çünkü kendisi kurulu bulunduğu bilgisayar için yani örneğin Intel marka bir işlemci için kod üretmek üzere kurulmuştur. Oysa yazılacak kodların OpenRISC işlemcisinin makina kodlarına dönüştürülmesi gerekmektedir. İşte bu işlem için GCC derleyicisi OpenRISC işlemcileri için de kod üretebilecek şekilde uyarlanmıştır. Dolayısıyla bu özelliği barındıran derleyici bilgisayara kurularak uygulamalar için kod derlenebilir. Bu işlem yine de bilgisayarın işlemcisi kullanarak yapılacaktır. Yani örneğin Intel işlemcili bir makina üzerinde

ve GNU GCC kullanılarak OpenRISC için makina kodu üretilecektir. Bu şekilde bir işlemci üzerinde başka bir aileye ait işlemcinin çalıştırılabileceği şekilde kod derlemeye çapraz derleme denilmektedir.

OpenRISC 1000 işlemcileri için GNU derleyici takımı desteği çeşitli versiyonlarda sağlanmış bulunmaktadır. Eğer OpenRisc ailesine mensup işlemci hiç bir işletim sisteminin olmadığı bir sistem için kullanılacaksa o halde GCC derleyici Newlib adlı C kütüphanesi ile derlenmelidir [16]. Eğer işlemcinin kullanılacağı sistemde bir işletim sistemi de çalışacaksa bu durumda GCC derleyici o işletim sisteminin fonksiyonlarını içeren kütüphalerle derlenmelidir. Bu çalışmada hiçbir işletim sistemi olmadan çalışan uygulamalar yapıldığı için Newlib ile derleme yapılan seçenek tercih edilmiştir. Fakat hali hazırda Linux işletim sistemi üzerinde çalışacak bir OpenRisc işlemcisi için kod derlemeyi sağlayan derleyici takımı desteği de mevcuttur. Bu kütüphaneler uClibc ve musl kütüphaneleridir [17] [18]. Bunlardan biri ile de GCC derlenebilmektedir. Derleyici takımı derlendiği kütüphanelere göre ayrı ayrı isimlendirilmiştir.

- **or1k-elf- :** İşletim sistemi olmadan çalışacak uygulamalar için (bare metal)
- **or1k-linux-uclibc :** Linux işletim sistemi üzerinde çalışacak uygulamalar için (uClibc kütüphanesi)
- **or1k-linux-musl :** Linux işletim sistemi üzerinde çalışacak uygulamalar için (musl kütüphanesi)

Bunlara ek olarak Windows ve Mac OS sistemler içinde derleyici takımı desteği sunulmaktadır. Fakat Windows desteği nisbeten daha zayıftır. Derleyici takımı ile alakalı daha geniş düzeyde ayrıntılar ve diğer konularla ilgili daha geniş versiyon bilgileri OpenCores web sitesinde projeye ilişkin sayfalardan elde edilebilir [19].

Bu çalışmada bu aşamadan itibaren derleyici takımı kurulumunun nasıl yapılacağı da anlatılacaktır. Eğer bu kurulum yapılmak istenmiyorsa doğrudan bir sanal makine programı (virtual box ücretisiz bir sanal makine yazılımıdır) kurulup önceden derleyici takımı derlenmiş olan bir disk dosyasını derleyici takımı (toolchain) web sayfasından indirmek suretiyle (web sayfasında “prebuilt versions” başlığı altında sunulmaktadır) kullanmak mümkündür. Sanal makine yazılımına hard disk olarak web

sayfasından indirilen dosya tanıtılır ve işletim sisteminin kurulması beklenir. İşlem tamamlandığında artık OpenRISC işlemcileri için kod derleyicisine sahip bir işletim sistemi temin edilmiş olur. Ancak en güncel versiyonlar her zaman bulunamayabilir. Yine de OpenRISC ekibinden talep edildiği takdirde güncel versiyonlar için de disk dosyaları temin edilebilir. İletişim için mail listesi ve bir web sitesi mevcuttur [19].

Kişinin kendisinin derleyici takımının kurulumunu doğrudan kaynak dosyalardan yapması için ilk önce yapması gereken Github adlı internet sitesine üye olması gerekir. Bu site OpenRISC gibi daha bir çok projenin tüm dosyalarının paylaşıldığı, indirildiği, güncellendiği ve incelendiği tanınmış bir web sayfasıdır. Github sayfasından bir başka bilgisayara dosya aktarımı yapılması için ilk önce bir RSA şifre tanımlaması yapılması gerekmektedir [20]. Bu adımlar sitenin uyguladığı doğal bir güvenlik prosedürü olarak görülmelidir. Bu işlemi yapmak için aşağıda linki verilen sayfada bulunan adımlar sırasıyla yapılmalıdır [21]:

https://help.github.com/articles/generating-ssh-keys/

Github sayfasına üyeliği tamamlandıktan sonra asıl iş olan derleyici takımı kurulumuna başlanabilir. Kurulumun yapılması için ilk önce birkaç yazılım paketinin yüklenmesi gerekmektedir. Bu paketler yüklenirken bunların da bağlı olduğu ve o sistemde daha önce olmayan paketler de kurulacaktır. Derleyici takımı kurulumu hata çıkma olasılığı nisbeten daha yüksek bir işlemdir. Şayet kurulum esnasında hata alınırsa bilinmesi gereken hataların çoğunun uyumsuz veya eksik paketlerden kaynaklandığıdır. Dolayısıyla doğrudan OpenRISC uyarlaması ile alakalı hata alma ihtimali daha azdır. Fakat bununla birlikte biraz daha ileride değinilecek olan ciddi bazı yanlışlıkların olduğunu da söylemek gerekmektedir. Fakat bu çalışma boyunca derleyici takımı ile alakalı başka bir probleme rastlanmamıştır. Kurulum yapılmaya başlanmadan önce yüklenecek paketler şunlardır:

- sudo apt-get install ssh (dosya transferi için)
- sudo apt-get install git (github repository için)
- sudo apt-get install svn-buildpackage (svn depo alanı için)
- sudo apt-get install automake (fusesoc kurulumu için)

- sudo apt-get install libusb-dev (digilent cable driverlar için)
- sudo apt-get install libmpc-dev
- sudo apt-get install libmpfr-dev
- sudo apt-get install libgmp-dev

Paketlerin kurulumu tamamlandıktan sonra işletim sistemini hazır hale getirmek için terminalden aşağıdaki kod çalıştırılarak ilgili paketlerin kurulması sağlanır.

```
sudo apt-get -y install build-essential
make gcc g++ flex bison patch
texinfo libncurses5-dev libmpfr-dev
libgmp3-dev libmpc-dev libzip-dev
python-dev libexpat1-dev
```

Hazırlık işlemleri tamamlandıktan sonra derleyici takımı kurulumuna başlanabilir. Kurulum dosyalarının tamamını kolaylık sağlaması açısından Linux'un "Ev Dizini" klasörüne indirmek mantıklı olacaktır. Bu aşamada önemli bir uyarı yapmak gerekmektedir. Bu çalışma yapıldığı sırada çok uzun bir süre boyunca derleyici takımından kaynaklanan bir hata ile uğraşmak durumunda kalınmıştır. Bu hata bazı C++ kodlarının çalışmasına engel teşkil eder ve kodların derlenmesine dahi engel olur. Daha sonra hata OpenRISC kanalına bildirilmiş ve karşılıklı uğraşlar sonucunda hatanın bizden kaynaklı değil de Newlib kütüphanesinde bir dosyada yazılı olan bir fonksiyonun başında bir "_" simgesinin olmamasından kaynaklandığı tespit edilmiştir. Bu hata güncel paketlerde giderilmiştir. Hatanın kaynaklandığı dosya /newlib/libgloss/or1k/syscalls.c dosyasıdır. Bu dosyayı açıp "open_r" adlı fonksiyon tanımına bakılmalıdır. Foksiyonun doğru hali şöyle olmalıdır:

```
_open_r(struct _reent *reent, const char *file,
int flags, int mode)
{
        reent->_errno = ENOSYS;
        return -1;
}
```

Şayet bu tanımlama da bir eksiklik veya yanlışlık var ise güncel olmayan bir paket indirilmiş demektir. Şayet durum buysa en güncel paketleri indirmek için

doğrudan Github web sayfasından "openrisc" hesabında bulunan newlib paketini indirmek mantıklı olacaktır. Ancak paketin ismi farklı ise kurulum kodlarındaki isimle değiştirilmelidir (newlib-2.2.0.20150225).

Bir başka hata ise Digilent Atlys geliştirme kartı ile ilgili bir takım tanımlamaların yapıldığı atlys.S dosyalarında yapılmıştır. Bu dosyalarda yapılan hata kurulum tamamlandıktan sonra bir uygulamanın derleneceği zaman derleyiciye geliştirme kartı parametresi olarak atlys geçildiğinde ortaya çıkması muhtemel bir yanlışlığa yol açabilmektedir. Çünkü atlys.S dosyalarında "_board_memsize" ifadesi ile tanımlanan değişken geliştirme kartı üzerinde bulunan RAM (Random Accessible Memory) belleğin değerini 32 MB (Mega Byte) olarak vermektedir. Oysa Atlys geliştirme kartı üzerinde;

- 128 MB DDR RAM (Double Data Rate Random Accessible Memory)
- Ethernet
- HDMI (High Definition Media Interface)
- GPIO (General Purpose Input Output)
- Led (8 adet) (Light Emitting Diode)
- Push Button (5 adet)
- Switch (8 adet)
- Ac 97
- USB (Universal Serial Bus)
- 16 MB SPI (Serial Peripheral Interface) FLASH

bulumaktadır [14].

Board ile ilgili bu oldukça kritik hatayı düzeltmek için /or1k-src/libgloss/or1k/atlys.S dosyasında ve /newlib/libgloss/or1k/boards/atlys.S dosyasında bulunan _board_memsize 0x2000000 olan değerini 128 mb değerine eşit olacak biçimde 0x8000000 olarak düzeltip kaydetmek gerekmektedir.

- Binutils kod paketinin indirilip açılması

```
wget http://ftp.gnu.org/gnu/binutils/
binutils-2.25.tar.bz2
tar xjvf binutils-2.25.tar.bz2
```

- Gcc kod paketinin indirilmesi

```
git clone https://github.com/openrisc/or1k-gcc
```

- Newlib kod paketinin indirlmesi

```
get ftp://sourceware.org/pub/newlib/
newlib-2.2.0.20150225.tar.gz
tar xzvf newlib-2.2.0.20150225.tar.gz
```

Kurulum işlemine başlarken çevresel değişkenler tanımlanmalı ve bunlar Ev Dizini klasöründe bulunan .bashrc dosyasına kaydedilmelidir. Bu değişken tanımı aşağıdaki gibi yapılabilir. Görüldüğü üzere PAHT adlı değişken Linux'un "root" dizininde bulunan "opt" klasörü içinde or1k-elf isminde bir yeri göstermektedir. Tüm derleyici takımı kurulumu buraya yapılacaktır.

```
export PREFIX=/opt/or1k-elf
export PATH=\$PATH:\$PREFIX/bin
```

Bu adımda ise kurulumun yapılacağı klasöre dosya açma ve açılan dosyanın okuma yazma izinlerinin nasıl verileceği gösterilmektedir.

```
sudo mkdir \$PREFIX
sudo chown <user>:<user> \$PREFIX
```

- Kurulum bu adımda başlamaktadır. Binutils kurulumu için aşağıdakiler yapılır.

```
mkdir build-binutils; cd build-binutils
../binutils-2.25/configure --target=or1k-elf
--prefix=\$PREFIX --enable-shared \
--disable-itcl --disable-tk --disable-tcl
--disable-winsup --disable-gdbtk \
--disable-libgui --disable-rda --disable-sid
--disable-sim --with-sysroot
make
make install
cd ..
```

- Newlib kurulumu için aşağıdakiler yapılır.

```
mkdir build-newlib; cd build-newlib
../newlib-2.2.0.20150225/configure  --target=or1k-elf
--prefix=\$PREFIX
make
make install
cd ..
```

- gcc stage 2 kurulumu için aşağıdakiler yapılır.

```
mkdir build-gcc-stage2; cd build-gcc-stage2
../or1k-gcc/configure --target=or1k-elf --prefix=\$PREFIX
--enable-languages=c,c++ --disable-shared
--disable-libssp --with-newlib
make
make install
cd ..
```

Gdb (gnu debuger) kurulumu için aşağıdakiler yapılır. Burada farklı olan gdb kurulumu için gerekli olan or1k-src dosyasının indirildiği yerdir. Bunun sebebi Gdb'nin henüz resmi bir binutils dağıtımı olmamasıdır [19]. Dolayısıyla github üzerinden çekilmesi gerekmektedir.

```
git clone git@github.com:openrisc/or1k-src
mkdir build-gdb; cd build-gdb
../or1k-src/configure --target=or1k-elf --prefix=\$PREFIX
--enable-shared --disable-itcl --disable-tk \
```

```
--disable-tcl --disable-winsup --disable-gdbtk
--disable-libgui --disable-rda --disable-sid
--enable-sim --disable-or1ksim --enable-gdb
--with-sysroot --disable-newlib --disable-libgloss
make
make install
cd ..
```

Kurulumun bu aşamaları tamamlandığında artık OpenRISC işlemcileri için kod derleyen ve ayrıca en önemli kod derleyicilerinden birisi olan GNU GCC hazırdır. C ve C++ kodları OpenRISC için derlemek artık oldukça kolay hale gelmiştir.

Derleyicilere ek olarak bir de simülatör programı bulunmaktadır. Or1ksim olarak adlandırılan bu simülatör ile hem yazılım kodlarının nasıl çalışacağı hem de o esnada işlemcinin yazmaçlarında hangi değerlerin oluştuğuna ilişkin bilgiler veren gelişmiş bir yazılımdır. Bunun için yukarıdakine benzer adımlarla bir takım işlemler yapılmalıdır. Simülatör kurulumu için aşağıdaki adımlar izlenmelidir.

```
git clone git://github.com/openrisc/or1ksim.git
mkdir builddir_or1ksim
cd builddir_or1ksim
../or1ksim/configure --target=or1k-elf
--prefix=/opt/or1ksim
make
make install
export PATH=/opt/or1ksim/bin:\$PATH
```

Bu kurulumun ardından PATH adlı değişkeni işletim sisteminin Ev Dizini klasöründe gizli dosya olarak bulunan ve terminal başlatılırken hangi kütüphane dosyaların bulunduğu yerlerin yollarına bakılacağını belirtmeye yarayan .bashrc adlı dosyaya kaydetmek gerekmektedir. Bu işlem yapılmadığı takdirde simülatör veya derleyici takımında bulunan araçlar çalıştırılmak istendiği zaman terminalden PATH değişkeni ile gösterilen dosyaların bulunduğu klasöre gitmek ve orada çalıştırma komutu vermek gibi zahmetli ve gereksiz bir işlem yapılmak durumunda kalınmaktadır.

3.2 GNU Derleyici Takımı Kullanımı

Simülatör de kurulumu ile birlikte kullanıcı artık oldukça geniş özelliklere sahip bir kod derleme ve test etme araçlarına sahip olmuş demektir. Bu araçların nasıl

kullanılacağı konusunda birkaç örnek verilmesi gerekirse basitçe şöyle bir örnek verilebilir: Önce tercih edilen herhangi bir kelime işlemcisi programı kullanarak standart çıkışa "merhaba" yazan bir C kodu yazılabilir. Kod dosyasının adının deneme.c olarak verilmiş oldğunu varsayalım. C kodumuz ise basit bir şekilde şu şekilde olabilir;

```
#include <stdio.h>
int main(){
  printf("merhaba\n");
  return 0;
}
```

Basit örnek yazılıp dosya kaydedilip kapatılabilir. Ardından terminal ekranını açıp bu kodu derlemek için şunlar yazılabir;

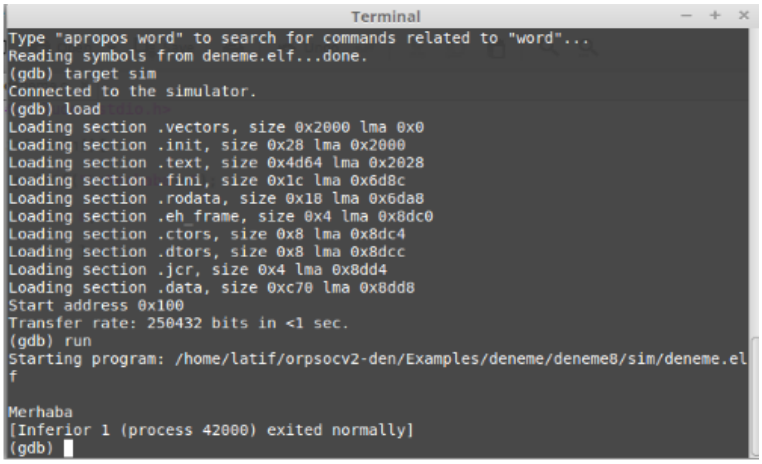
```
or1k-elf-gcc -mboard=atlys deneme.c -o deneme.elf
or1k-elf-gcc -g -mboard=or1ksim deneme.c -o deneme.elf
or1k-elf-g++ -mboard=atlys hello.cpp -o hello.elf
```

Görüldüğü üzere üç farklı ifade yazılmıştır. Kullanıcı hedeflediği işleme göre bunlardan birisini tercih edebilmektedir. Birinci satırdaki komutlar yazıldığı zaman derleyici deneme.elf adında ve atlys geliştirme kartı özeliklerini bilen bir çalıştırabilir kod üretir. Bu derleyici takımı ile üretilen kodlar işletim sistemi olmayan bir sistemde çalışabilir. Çünkü derleyici Newlib kütüphanesi ile derlenmiştir.

İkinci satırda bulunan komutlar çalıştırılırsa, bu sefer de simülator üzerinde çalışabilecek şekilde bir derleme yapılmış olur. Burada –g ve –mboard=atlys ifadelerine dikkat edilmelidir. Bu ifadeler yazılmadığı zaman üretilen .elf dosya simülator ile test edilemez. Oluşan .elf dosyayı simülatorde kullanmak için ise basitçe şunlar yapılmalıdır;

```
or1k-elf-gdb deneme.elf
target sim
load
run
```

Dikkat edilirse ilk önce Gbd çağırılmıştır. Bunun sebebi simülatörün debuger ile bağlantılı çalışmasından kaynaklanmaktadır. Eğer debuger bir şekilde yanlış kuruldu ise simülatörde ya hiç kurulamaz ya da yanlış çalışır. Yukarıdaki işlem yapıldığı zaman simülatör terminalinde "merhaba" yazısı Şekil 3.1 gibi görülecektir.



Şekil 3.1: Basit Bir Uygulamanın Simülatör Çıktısı.

Üçüncü satırdaki derleme işlemi ise kaynak kodu eğer C++ ile yazılmışsa yapılır. Dikkat edilirse derleme işlemi için or1k-elf-gcc değil de C++ kodlarını derleyen or1k-elf-g++ çağırılmıştır.

4. ORPSoC-v2

ORPSoC-v2 OpenRISC 1000 işlemci ailesinden OR1200 CPU üzerine kurulmuş olan ve çeşitli FPGA geliştirme kartları için desteklenen kırmıküstü sistem tasarımıdır. Bu sistem yavaş yavaş yerini ORPSoC-v3 sistemine bırakmak üzeredir. ORPSoC-v3 sistemi de bu çalışmada anlatılacaktır. Ancak henüz bazı hata olduğu için, geliştirme kartı desteği ORPSoC-v2'ye göre daha az olduğu için ve ayrıca ORPSoC-v2 daha kararlı bir halde olduğu için bu çalışmada ikinci plana atılmıştır. Esasen tasarım anlayışı iki platfromda da benzerdir. Dolayısyla birisini iyi anlayan diğeri içinde önemli mesafe almıştır denilebilir.

ORPSoC-v2 ve ORPSoC-v3 verilog dilinde yazılmış olan CPU ve çevresel birimlerin kod dosyalarını ve bir takım yardımcı yazılımları içeren sistemlerdir. Kullanıcılar bu dosyaları indirip kendi arzu ve ihtiyaçlarına göre bir işlemci ve o işlemciye bağlı çalışacak olan çevresel birimleri eksilterek ya da artırarak kullanabilmektedirler. Bu kodlar Xilinx yada Altera firmalarının sunduğu yazılımlar kullanılarak sentezlenebilir ve daha sonra hedef geliştirme kartı için FPGA'yı yapılandıracak kod üretilebilir.

4.1 Kodların İndirilmesi ve İlk İşlemler

Orpsoc-v2 kodlarını indirmek için Google'da kısa bir arama yapmak veya OpenCores web sayfasından faydalanmak mümkündür. Alternatif yollardan bir kaçı şöyledir:

- svn co http://opencores.org/ocsvn/openrisc/openrisc/trunk/orpsocv2
- https://github.com/lgeek/orpsoc (tarayıcı kullanılarak zip olarak indirilir)
- https://github.com/lab305itep/orpsocv2 (tarayıcı kullanılarak zip olarak indirilir)

Fakat burada dikkat edilmesi gereken şudur: Verilen ilk link projenin resmi lokasyonundan yapılan indirme yoludur. Diğer iki kaynak ise hâlihazırda bozulmamış şekilde durmuş olmasına rağmen kişisel kullanıcılara ait depolama alanları oldukları

için sahipleri tarafından değiştirilmiş olma ihtimalleri vardır. Dolayısıyla ilk linki kullanmak daha mantıklı olacaktır.

Kodları yukarıdaki yollardan birisi ile indirdikten sonra yapılması gereken çok önemli bir değişiklik vardır. İndirilen klasör açılıp sw/makefile.inc dosyası açıldığında dosyanın en başında şöyle bir ifade mevcut olduğu görülecektir;

```
OR32_TOOL_PREFIX=or32-elf-
```

Görüldüğü üzere bu parametre derleyici takımı'nın eski versiyonu olan or32-elf- içindir. Dolayısıyla bu klasörlerde işlem yapıalcağı zaman işletim sistem or32-elf- binary dosyaları arayacaktır. Oysa Bölüm 3'ten hatırlanacağı gibi kurulu olan sistem or1k-elf- derleyici takımıdır. Bunu düzeltmek oldukça basittir. Kod aşağıdaki hale getirilip dosya kaydedip kapatılmalıdır.

```
OR32_TOOL_PREFIX=or1k-elf-
```

Bu düzenleme sayesinde or1k-elf derleyici takımı artık indirilen kodlar için kullanılabilir hale getirlmiştir.

4.2 Xilinx ISE Desing Suite 14.7 Ve Kablo Sürücülerinin Yüklenmesi

Kısım 4.1'de ifade edildiği gibi kodlar üzerinde sentezleme ve diğer işlemleri yapabilmek için muhakkak surette bir yazılıma ihtiyaç duyulmaktadır. Bu çalışmada FPGA geliştirme kartı olarak daha önce de özelliklerine ait bilgiler bilgiler verilmiş olan Digilent Atlys geliştirme kartı kullanılmıştır. Xilinx ISE Design Suite Atlys geliştirme kartı desteği içermektedir. Zira geliştirme kartı üzerindeki FPGA aslında yine Xilinx firmasına ait olan spartan6 xc6slx45 cgs324 referans numaralu yongadır. Dolayısıyla muhakak surette Xilinx ISE Design Suit yazılımı indirilip kurulmalı ve ayrıca ek olarak geliştirme kartına bağlantı yapılmasını sağlayan kabloların da sürücülerinin kurulması zorunludur.

Xilinx ISE Design Suite yazılımı paralı bir yazılımdır. Ancak webpackage sürümü ücretsizdir ve sınırlı bir süre için IDE (Integrated Development Kit)'nin sunduğu birçok özelliği kullanabilecek şekilde programı kullanılmasına izin verir. Kurulumu yüklemek için firmanın web sayfasının indirme bölümü kullanılabilir [22]. Bu

çalışmada 14.7 sürümü kullanılmıştır. Kurulumun Linux'ta yapılması oldukça kolaydır. İndirilen dosyanın içindeki install.sh adlı dosya çalıştırılır ve kurulum başlatılır. Kurulum sihirbazı yönlendirmeleri yapacaktır. Tek dikkat edilecek nokta kuruluma başlamadan önce /opt/Xilinx şeklinde bir klasör açılması ve okuma yazma izinlerinin verilmesidir. Kurulum sihirbazına da bu yol gösterilmelidir. Kurulum ve yazılım hakkındaki diğer detaylara arzu edilirse firmanın web sayfasından bakılabilir [13].

Gerekli paketler (Bölüm 3) ve Xilinx ISE yüklendikten sonra terminalde aşağıdaki işlem yapılır:

```
cd /opt/Xilinx/14.7/ISE_DS/common/bin/lin64/digilent
(32 bit sistemler icin lin64 yerine lin)
sudo ./install_digilent.sh /opt/Xilinx/14.7/ISE_DS/ISE
```

Böylece kablo sürcüleri yüklemesi yapılmış olur. Artık Atlys geliştirme kartı ile bilgisayar arasında bağlantı kolay biçimde yapılabilir hale gelmiştir. Bu konuya ilişkin söylenebilecek en önemli detaylardan biri de şudur: Daha önce derleyici takımı kurulumunda yapıldığı gibi .bashrc dosyasına Xilinx ISE programının da derlenmiş kütüphane dosyalarının bulunduğu yol eklenmelidir.

```
export PATH=$PATH:...:/opt/or1ksim/bin:/opt/Xilinx/
14.7/ISE_DS/ISE/bin/lin64
```

4.3 ORPSoC-v2 Pratik Kullanımlar

ORPSoC-v2 dosyası indirildikten sonra dosya içeriğinin biraz incelenmesinde fayda vardır. Klasör içeriğinde Verilog kodları, yardımcı yazılımlar, desteklenen geliştirme kartları için yazılmış parametreler, bazı işlemleri otomatik olarak yapmak için gerekli olan makefile adındaki toplu komut listeleri bulunmaktadır. İlk olarak tüm sistem hakkında bilgi verilen kullanım kılavuzu niteliğinde bir belge üretilebilir. Bunun için terminalden orpsocv2 klasöründe bulunan doc klasörüne geçilir ve burada aşağıdaki kodları yazarak orpsoc.pdf user guide üretilir.

```
./configure
make pdf
```

Yukarıdaki işlemin ardından aynı klasörde orpsoc.pdf adında bilgilendirici bir dosya üretildiği görülecektir.

Biraz daha ileri bir işlem olarak bootrom.v dosyasının üretimi yapılabilir. Bu dosya işlemci ilk başlatıldığı zaman çalıştırılacak olan komutların bulunduğu Verilog dosyasıdır. Aslında program belleği bu Verilog dosyasının içeriği olacaktır. Bu dosya indirilen orpsoc-v2 dosyasının /sw/bootrom kısmında bulunan bootrom.S isimli bir makina kodunun geliştirme kartına özel olarak derlenmesi ile oluşturulur. Bu makina kodu işlemcinin ilk başladığından sonra spi flash içerisinde bulunan programın çalıştırılmasını sağlayan bir koddur. Ancak elbette isteniyorsa farklı işlemler yapan makina kodları da yazılabilir. Ancak bu çalışmada yapılan birçok işlem için de gerekli olan spi flash boot işlemi bu aşamada değiştirilmemelidir.

bootrom.v kodunun üretilmesi için terminalden ilk önce Atlys kodlarının bulunduğu alana gidilmeli ve daha sonra da bootrom kodunun üretileceği board klasörüne girilip üretim komutu verilmelidir. Bu işlemler terminalden şu şekilde yapılır:

```
cd orpsoc-v2/boards/xilinx/atlys/sw/bootrom
make all
```

Bu işlemin ardından bulunulan klasörde bootrom.v dosyası oluşmuştur. Bu dosya daha sonraki işlemlerde bir içerik dosyası gibi kullanılacağı için buradan kopyalanıp bir üst dizinde bulunan /rtl/verilog/include/ dosyasının içine yapıştırılması gerekmektedir.

4.4 Yardımcı Yazılımlarla Yapılabilen Gerçeklemeler

ORPSoC-v2 klasöründe kullanıcıların ihtiyaç duyabileceği birçok iş için pratik çözümler geliştirilmiştir. Verilog kodlarının sentezlenmesinden FPGA'nın yapılandırma dosyasının oluşturulmasına kadar her adım basit birer terminal komutuyla yapılabilmektedir. Bu sayede kullanıcı şayet Xilinx yada Altera gibi diğer firmaların yazılımlarına aşina değilse bile kolayca istediklerini yapabilmektedir.

Bu çalışmada en çok kullanılan dosyalar orpsoc-v2 klasöründe /boards/xilinx/atlys içersinde bulunmaktadır. Hiyerarşik olarak en üst seviye kod olan orpsoc_top.v dosyası açılıp incelendiği zaman or1200, RAM, GPIO (General Purpose Input Output), SPI (Serial Peripheral Interface), ETHERNET, UART (Universal Asynchronous

Receiver/Transmitter) ve diğer birimlerin ortak bir Wishbone veri hattına nasıl bağlandığı görülecektir.

Pratik işlemlere geçmeden önce yine bir düzeltme yapmak gerekmektedir. Sentez ve diğer işlemler için yazılan iki makefile içerisinde Atlys geliştirme kartına ait tanımlamalar yapılırken bir yanlışlık (veya yazan kişinin kendi elinde hız seviyesi -2 olan bir Atlys versiyonu bulunması) sonucu Atlys geliştirme kartına ait speed grade (hız seviyesi) değeri hatalı olarak verilmiştir. Bu hatayı düzeltmek için aşağıdaki iki makefile açılır.

```
/Orpsoc-v2/boards/xilinx/atlys/syn/xst/bin/makefile
/Orpsoc-v2/boards/xilinx/atlys/backend/par/bin/makefile
```

Dosya içersinde speed grade için değer atanan aşağıdaki kısım bulunur. Daha sonra ise -2 olan değer -3 olarak düzenlenip kaydedilir ve dosya kapatılır. Yani doğru biçim aşağıdaki gibi olmalıdır.

```
FPGA_PART ?=xc6slx45-3-csg324
OPT_MODE ?=Speed
OPT_LEVEL ?=3
```

Yukarıdaki hata farkedilmeden önceki dönemlerde de Atlys geliştirme kartı üzerinde uygulamalar çalıştırılmıştır. Fakat daha sonra bu değeri değiştirmeden yapılan işlemlerde zamanlama hatalarının olduğu farkedilmiştir. Hatalar işlemcinin çalışmasını engellememesine rağmen asla başka bir yerde sorun çıkarmayacağını garanti etmediğinden dolayı yukarıdaki değişikliklik yapmak gerekmektedir.

Yardımcı yazılımlarla yapılan pratik işlemler şu şekilde sıralanabilir: İlk olarak varsayılan ayarlar için Atlys geliştirme kartında çalışacak olan işlemci ve çevresel birimlere ait kodların sentezini ve diğer işlemleri yapabilmek için Xilinx kaynak dosyası terminalden çağrılarak başlatılır ve diğer işlemlere geçilir;

Kaynak dosyasını çağırmak için bulunulan klasörde terminal ekranı açılıp,

source /opt/Xilinix/14.7/ISE_DS/settings64.sh

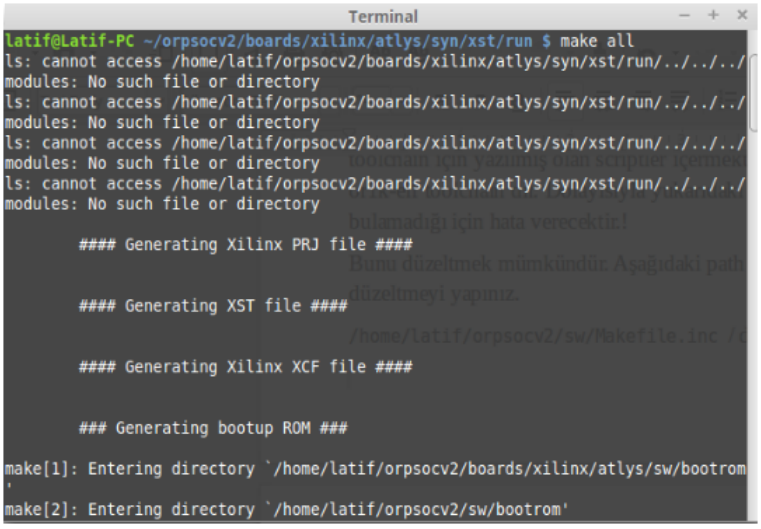
make all

komutu çalıştırılır. Sentez işlemi için;

cd /home/kullanıcı/orpsoc-v2/boards/xilinx/atlys/backend/par/run

make all

İşlemlerden sonra terminal ekranı Şekil 4.1 gibi görüntülenir.



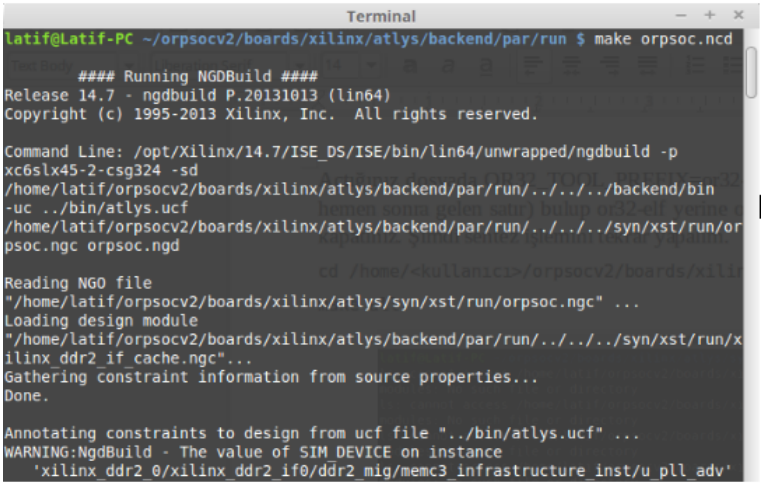
Şekil 4.1: Sentez İşlemi Esnasında Terminal Ekranı.

Aşağıdaki işlemler "map" ve "place and route" işlemlerininin nasıl yapılacağını gösterir.

cd /home/kullanıcı/orpsoc-v2/boards/xilinx/atlys/backend/par/run

make orpsoc.ncd

İşlemlerden sonra terminal ekranı Şekil 4.2 gibi görüntülenir.



Şekil 4.2: Map ve Place And Route Aşamaları Esnasında Terminal Ekranı.

Bu işlemi bittikten sonra orpsoc.par dosyasına bakılarak Place And Route sonuçları görülebilir. Yine aynı terminal ekranında eğer arzu ediliyorsa aşağıdaki kodlar kullanılarak Xilinix yapılandırmaları görülebilir.

```
cd /home/latif/orpsocv2/boards/xilinx/atlys/backend/par/run
make print-config
```

Bu işlem sonucunda aşağıdaki bilgiler görülür;

```
    ### Backend make configuration ###
FPGA_PART=xc6slx45-2-csg324
XILINX_FLAGS=-intstyle silent
XILINX_MAP_FLAGS=-logic_opt off
XILINX_AREA_TARGET=speed
TIMING_REPORT_OPTIONS=-u 1000 -e 1000
SPI_FLASH_SIZE_KBYTES=16384
SPI_BOOTLOADER_SW_OFFSET_HEX=1c0000
```

Zamanlama raporu (timing report) şöyle üretilir;

```
cd /home/latif/orpsocv2/boards/xilinx/atlys/backend/par/run
make timingreport
```

Ayrıntılı zamanlama bilgilerine üretilen orpsoc.twr dosyasından bakılabilir. FPGA yapılandırma dosyası üretme işlemi de yardımcı yazılımlar ile yapılabilir. Üretilen dosya (.bit dosyası) işlemciye ait tüm sayısal devre yapısını içerir. Bu dosya ile FPGA programlandığında OR1200 işlemcisi ve etrafında bulunan çevresel birimler, yani kırmıküstü sistemin tamamı, FPGA üzerine gömülmüş olur. Bu dosya şu şekilde üretilebilir;

```
cd /home/latif/orpsocv2/boards/xilinx/atlys/backend/par/run
make orpsoc.bit
```

Bu kısımda önemli bir ayrıntıya yer vermek gerekmektedir. Kullanıcı aslında ardı ardına bu işlemleri tekrarlamak zorunda değildir. Yukarıda en son yapılan işlem (make orpsoc.bit) işlemi yapılırsa daha önceki adımlar kendiliğinden yapılmaya başlanır ve en son .bit dosyası üretilir. Çünkü sentez ve place and route işlemleri olmaksızın .bit dosyası zaten üretilmeyecektir. İşlem bilgisayarın özelliklerine göre ve elbette kullanıcının kod dosyaları üzerindeki eklemelerine göre değişmekle birlikte yaklaşık 30 dakika kadar sürmektedir.

Son işlem tamamlandıktan sonra .bit dosyası ile FPGA'yı yapılandırmak için Xilinx ISE yazılımının sunduğu "impact" arayüzünü kullanmak gerekmektedir. Bu işleme başlamadan önce Atlys geliştirme kartı beslemesi yapılmalı ve programlama kablosu geliştirme kartı üzerindeki yerine ve diğer ucu da bilgisayardaki USB (Universal Serial Bus) portuna bağlanmalı ve geliştirme kartı açma kapama düğmesinden enerjilendirilmelidir. Daha sonrasında ise terminal ekranına "impact" yazılarak arayüz çalıştırılır. Ardından gelen pencelere "no" denilip sağ üst köşedeki menüden "Boundary Scan" seçeneğine tıklanır. Daha sonra ekranda sağ tıklama yapıp "initialize chain" diyerek Atlys geliştirme kartının okunup okunmadığını anlaşılır (Identify Succeed) . Ardından da edit menüsünden "Assign Configuration File" denilir ve gelen ekrandan orpsoc.bit dosyası gösterilir. Son olarak impact arayüzü bize "spi/bpi programlama dosyası eklemek ister misiniz?" anlamında bir soru sorar. Bu aşamada buna gerek olmadığı için "no" denilir ve daha sonra sağdaki menüden "program" seçeneğini tıklanıp .bit dosyasının yazılması beklenir. Yaklaşık 30 saniyede işlem tamamlanır ve "program succeeded" uyarısı görüldüğü zaman işlem tamamlanmış olur. Böylece FPGA üzerinde ORPSoC-v2 kırmıküstü sistem otomatik çalışmaya başlar. Bundan sonraki bölümlerde bu sistem üzerinde yapılan uygulamalar anlatılacaktır.

4.5 Xilinx Proje Dosyası Oluşturulması

Yardımcı yazılımlar kullanılarak yapılan işlemler oldukça pratiktir. Ancak bu yöntem Xilinx proje dosyası oluşturmadan işlemleri yapmaktadır. Bu daha hızlı biçimde işlemlerin bitmesi için avantajlı olsa da bir çok sebepten dolayı kullanıcılar Xilinx proje dosyası oluşturmayı ve projesine hangi dosyaların eklendiği gibi daha birçok detayı görmek isteyebilir. Ayrıca Xilinx ISE ayarlarını değiştirmek ancak bir proje dosyası olduğunda mümkün olmaktadır. Bu sebeple proje dosyası oluşturmak gerekmektedir. Aşağıdaki adımlarla bu işlem de kolayca yapılabilir:

- Terminal ekranından "ise" yazılarak Xilinx ISE başlatılır.
- "New Project" seçeneği ile proje açılır.

- Ayarlamalar yapılır. (Kart seçimi, verilog dili, proje adlandırması ve yer seçilmesi gibi)
- Proje açıldıktan sonra “Add Source” tıklanır.
- Açılan pencereden /. . . /atlys/rtl/verilog/orpsoc_top.v seçilir.
- orpsoc_top.v altında eksik bulunan dosyalar sırasıyla eklenir.
- İlk olarak /atlys/rtl içersindeki dosyalar ve /atlys/backend/par/bin deki atlys.ucf eklenir.
- Eksik kalan diğer dosyalar /orpsoc-v2/rtl/verilog içerisindedir. Bazı dosyalarda “include” dosyaları olduğundan /atlys/rtl/include ve /orpsoc-v2/rtl/verilog/include klasörleri “synthesis” kısmına sağ tıklanıp gelen menüden “advanced seçilerek gelen ekrandan “verilog include dir” kısmındaki ekle sembolüne tıklanıp “Automatic Includes” olarak eklenir.
- Tüm eksikler giderildikten sonra “Project” sekmesinden “Clean Up Project Files” seçeneği ve “Force Hierarcy Reparse” seçeneği birer kez yapılarak programın hiyerarşik yapılandırmaları algıladığından ve eski hata mesajlarını temizlediğinden emin olunur.

işlemlerin ardından istenirse sentez ya da diğer işlemler yapılabilir. FPGA yapılandırma dosyası (.bit) dosyası üretildikten sonra da impact arayüzü açılarak daha öce anlatıldığı şekilde FPGA programlanır.

4.6 ORPSoC-v2 İle Uygulama Çalıştırma

ORPSoC-v2 platformu ile uygulama geliştirmeye artık hazır hale gelinmiştir. Standart çıkışa (output) bir mesaj yazdıran uygulama iyi bir başlangıç olacaktır. Burada ilk düşünülmesi gereken işlemci ilk başladığı anda yapacağı işlemdir. Çalışma başlar başlamaz ilk olarak bootrom.v kodu çalıştırılır. Bu kod program belleğine gömülmüş olan komutlardan oluşur. Bilgisayar için düşünülürse işlemci ilk çalıştığında diskte bulunan işletim sistemini başlatan komutlar bootrom komutlarıdır. Böyle bir benzetme kurularak bootrom.v kodunun fonksiyonu anlaşılabilir.

Kısım 4.4'te bootrom.v kodunun nasıl üretildiği anlatılmış ve bu dosyanın /... /atlys/rtl / verilog/ include klasörüne kopyalanması gerektiği söylenmiştir. Bu düzenleme yapılmadıysa, bu bölümde anlatılacak uygulama için yapılması gerekmektedir.

/Orpsoc-v2/rtl/verilog/rom/rom.v dosyası açılır. Bu dosya sistemin ROM (Read-Only Memory) belleğidir. Dosya açıldığında görüleceği gibi beş satırlık bir kod aktif iken hemen üst kısımda bulunan bootrom.v kodunu rom.v modülüne bağlayan kısım yorum satırı şeklinde bırakılmıştır. Bu dosyada sadece ilgili kısmı düzenleyerek bootrom kodunun aktif hale getirilmesi gerekmektedir. Kod aşağıdaki gibi değiştirilip kaydedilerek kapatılır:

```
`include "bootrom.v"
      /*
       // Zero r0 and jump to 0x00000100
       0 : wb_dat_o <= 32'h18000000;
       1 : wb_dat_o <= 32'hA8200000;
       2 : wb_dat_o <= 32'hA8C00100;
       3 : wb_dat_o <= 32'h44003000;
       4 : wb_dat_o <= 32'h15000000;
       */
```

Aktif hale getirilen bootrom.v kodu daha öncede ifade edildiği gibi spi flash içerisinde yazılmış olan herhangi bir programı çalıştırmaya yarayan basit bir programdır. Bu program işlemci her başladığı zaman veya reset (yeniden başlatma) aldığı zaman yeniden çalıştırılır. Artık işlemci gerçeklenmeye hazır haldedir. Daha önce açıklanmış olan iki yoldan herhangi birisi kullanılarak sistemin yapılandırma dosyası üretilmelidir. En sade hali ile

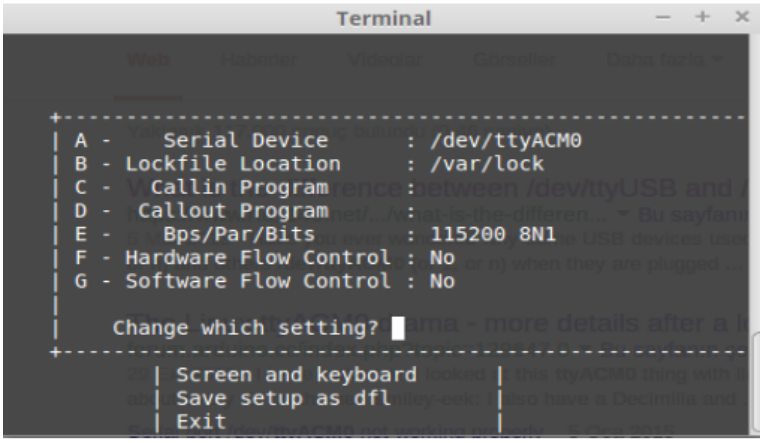
/home/latif/orpsocv2/boards/xilinx/atlys/backend/par/run

make orpsoc.bit

veya

make orpsoc_spiboot.bit

işlemi yapılabilir. İkinci işlem de yine aynı şekilde FPGA'yi yapılandıracak olan dosyayı üretir. İlk komutla arasındaki tek fark ikisinin farklı birer başlatma saat sinyali kullanmasıdır. Bu aşamada bizim için ikisi arasında bir fark olmayacaktır.



Şekil 4.3: Minicom Seri Terminal Bağlantı Ayarlamaları.

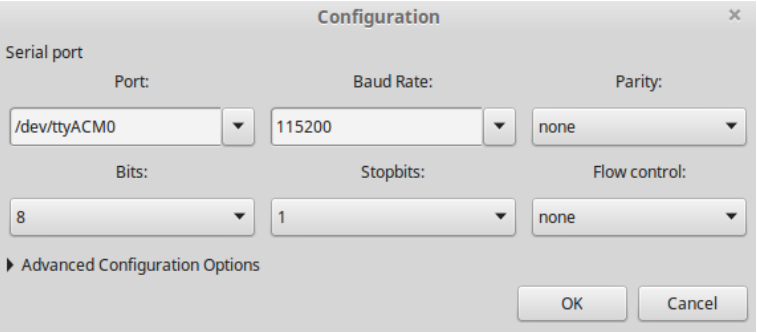
Donanım tarafındaki işler bittiğine göre yazılım kısmındaki yapılacaklara bakılabilir. İlk olarak seri terminal programının yüklenmesi gerekecektir. Atlys geliştirme kartı üzerinde sistem çalıştıktan sonra standart çıkış UART olduğu için işlemlerin sonuçları buraya yazdırılacaktır. Dolayısıyla bu çıkışa nelerin basıldığını görmek için böyle bir programa ihtiyaç duyulmaktadır. Bu işlem için mevcut birçok ücretsiz program vardır. Herhangi birisi kullanılabilir. Bu genel olarak "minicom" tercih edilmiştir. GtkTerm programı da alternatif olarak kullanılabilir. Bu programlar doğrudan terminalden yüklenebilmektedir.

sudo apt-get install minicom veya sudo apt-get install gtkterm

BU programlar her defasında çalışmak için yönetici iznine ihtiyaç duymaktadırlar. Bu gereksiz bir işlem olduğu için seri terminal programlarının izinlerini genele açmak kolaylık sağlayabilmektedir. Fakat bu yapılması zorunlu bir işlem de değildir. Bu işlem bir terminal ekranı açılarak şu şekilde yapılabilir.

sudo usermod -a -G dialout $USER

İzin işlemi de geçildikten sonra bağlantı ayarlarını yapmak gerekmektedir. Minicom veya GtkTerm için bu ayarı yapmak oldukça kolaydır. Bağlantı portu **/dev/ttyACM0** ve haberleşme hızı (yani bir saniyede kaç bit gönderileceği – Baud Rate) ise **115200** olarak ayarlanıp kaydedilmelidir. Bu işlemi Minicom için yapmak için terminalden **minicom** -s yazılarak programın ayar menüsüne girilmiş olur. Menüden **"Serial Port Setup"** bölümüne girilir. Burada haberleşme kanalları ile ilgili ayarlar yapılan aşağıdaki seçenekler gelecektir. Şekil 4.3'deki gibi gibi ayarlar yapılır;



Şekil 4.4: GtkTerm Seri Terminal Ayarlamaları.

Aynı işlem Gtkterm için de kolayca yapılabilir. Terminalden **gtkterm** yazılarak program başlatılır. Programın menüsünden **configuration** kısmına girilip **port configuration** seçilerek aşağıdaki ekran açılmış olur. Buradan yine aynı şekilde ayarlar yapılıp kaydedilir.İşlem Şekil 4.4'den de görülebilir

Seri terminal ayarları bitirildikten sonra OR200 işlemcisi tarafından çalıştırılacak olan uygulama belirlenmesine geçilebilir. Bu kısımda daha önce simülator anlatılırken örnek olarak verilen basit bir C kodu kullabildiği gibi daha karmaşık işlemler yapan bir kod da seçilebilir.

Bu kısımda 128-bit AES (Advanced Encryption Standart) şifreleme ve şifre çözme işlemleri yapan bir C kodu kullanılması tercih edilmiştir. Bu kodun benzerleri basit bir aramayla internette bulunabilir. Kod, basitçe ifade etmek gerekirse, kendisine verilen bir metni yine kendisine verilen bir anahtarı kullanarak şifreler, ardından ters işlem yaparak şifre çözer. Daha sonra ilk verilen metin ile şifresi çözülmüş metin karşılaştırır. Aynı metin olduğuna karar verilirse ekrana "doğru" yazması koda ilave edilmiştir. Ancak bu kod yerine sadece standart çıkışa "merhaba" yazan bir kod da kullanılabilir. Arada hiç bir fark yoktur.

Kod yazılıp kaydedildikten sonra, sıradaki işlem derleme ve dosya formatını ayarlama işlemleridir. Aşağıdaki işlemler terminalden sırasıyla yapılmalıdır.

or1k-elf-gcc -mboard=atlys aes.c -o aes.elf

or1k-elf-objcopy -O binary aes.elf aes.bin

bin2binsizeword aes.bin aes_size.bin

Yukarıdaki işlemlerin her biri derleyici takımının bir aracı ile yapılmaktadır. Sırasıyla ifade etmek gerekirse şunlar söylenebilir: İlk satırda normal bir C kodu derleme işlemi yapılmakta olup tek farklılık -mboard parametresidir. Bu parametre Atlys geliştirme kartına ait bir takım bilgilerin aktarılması içindir. İkinci satırda bulunan kod ise .elf veya başka bir çalıştırılabilir dosya formatından ikilik formata çevirme işlemi yapmaktadır. Son satırda bulunan kod ise binary formatlı dosyayı word (32-bit) formatında düzenlemeye yaramaktadır.

"bin2binsizeword" kodu indirmiş olduğumuz orpsoc-v2 klasöründe /sw/utils/ bölümünde mevcuttur. Bu uygulamayı yukarda yapıldığı gibi doğrudan terminalden çağırabilmek için ilk önce kodun derlenmiş halinin (çalıştırılabilir formattaki dosyanın) Linux terminalin baktığı yerlerden birisine kopyalanması gerekmektedir. Bu yer örneğin daha önce terminale gösterilmiş olan /opt/or1k-elf/bin klasörü olabileceği gibi terminalin otomatik olarak bildiği /usr/local/bin gibi bir yer de olabilir. Fakat ilk önce kodun derlenmesi gerekmektedir. Bunun için terminalden kodun bulunduğu klasöre gelip **make all** komutu çalıştırılmalıdır. İşlemin ardında birkaç dosya üretilecektir. Bunlardan bazıları da daha sonra kullanılacaktır. Bu aşamada **bin2binsizeword** dosyasının gerekli yerlerden birisine kopyalanması yeterlidir.

İşlemlerin bu aşamasına kadar donanım tarafında FPGA yapılandırma dosyası üretilmiştir. Yazılım tarafında ise uygulama derlenmiş ve gerekli formatlara dönüştürülmüştür. Ayrıca FPGA geliştirme kartı ile bağlantı kurulmasını sağlayan bir terminal programı da yüklenmiş ve bağlantı ayarları yapılmıştır. Geriye bu iki parçayı birleştirmek kalmıştır. Bu aşamada farklı tercihler yapılabilir. Bunları birbirinden ayırt etmek oldukça önemlidir.

- Uygulama kodu spi flash içerisine konacağı için uygulamanın en son üretilen ikilik format dosyası flash standardı olan .mcs dosyasına çevirilmek zorundadır. (prom formatı)

- FPGA yapılandırma dosyası istenirse uygulama ile birlikte spi flash'a yazılabilir. Ancak uygulamanın çalışması için şart değildir. Eğer yazılırsa geliştirme kartı enerjisi kesik olduğu zaman bile yapılandrıma dosyasının spi flash'ta saklayacak ve geliştirme kartı her açıldığında yapılandırma dosyası FPGA'yı

yapılandırcaktır. Eğer bu yol tercih edilmezse yapılandırma dosyası her seferinde tekrar bilgisayardan impact arayüzü ile FPGA'ya gönderilmek zorundadır. Çünkü enerji kesildiğinde bu yapılandırma silinecektir.

- Uygulama ve yapılandrıma dosyası tek bir dosya (mcs) halinde oluşturulup spi flash'a yazılabileceği gibi yapılandırma dosyası için ayrı, uygulama dosyası için ayrı mcs dosyaları oluşturulabilir.

- Yukarıda sayılan işlemler daha önce yapılandırma (bit) dosyası oluşturulan klasörde benzer "make ..." işlemleriyle yapılabilir. Ya da aleternatif bir yol olarak terminale doğrudan Xilinx araçlarının komutlarını yazarak da .mcs dosyaları üretilebilir. Aslında iki yöntemde de çalışan komut ve araç aynıdır. Tek farklılık yardımcı yazılım kullanılıyorsa mcs yapılacak kodun belli bir yerde durması gerekmektedir (/Oprsoc-v2/boards/xilinx/atlys). Bu da doğaldır. Çünkü otomatik yapılan bir işlemde dosyanın nerede olduğu bilinmek durumundadır. Ayrıca yine makefile ile çalışılıyorsa her üretimden önce mcs yapılmak istenen dosyanın ismi makefile' a yazılıp kaydedilmek zorundadır.

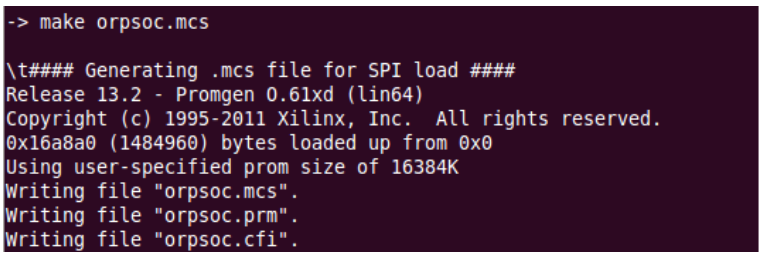
İlk olarak makefile ile çalışmak isteyenler için yapılması gerekenler anlatılacaktır. Bunun için yapılacak iki şey vardır.

- Uygulama_size.bin dosyası /Oprsoc-v2/boards/xilinx/atlys kısmına kopyalanır.

- /Oprsoc-v2/.../atlys/backend/par/bin/makefile dosyası açılarak aşağıdaki kısımda gösterilen satır eklenir. Bu satırda eklenen bilgi uygulama_size.bin dosyasının varlığını ve yerini belirtme amaçlıdır.

```
# Options for Xilinx PAR tools
FPGA_PART=xc6slx45-3-csg324
(...)
SPI_BOOTLOADER_SW_OFFSET_HEX ?=1c0000
BOOTLOADER_BIN ?=$(BOARD_ROOT)/uygulama_size.bin (eklenen satir)
```

Yukarıdaki son satır eklendikten sonra dosya kaydedilip kapatılır. Ardından daha önce FPGA yapılandırma dosyası üretilen klasöre gidilip terminal açılır ve aşağıdaki kod çalıştırılarak spi flash'a yazılacak dosya üretilmiş olur. İşlem esnasında terminal ekranı Şekil 4.5'deki gibidir.



Şekil 4.5: Spi Flash'a Yazılacak Dosya Üretimi.

make orpsoc.mcs

Spi flash'a yazılacak dosya üretimi için diğer yöntem ise daha önce de ifade edildiği gibi Xilinx araçlarının doğrudan çağırılmasıdır. Bunun için aşağıdaki şekillerde farklı amaçlara yönelik dosyalar üretilebilir. Bunların diğer yöntemden farkı kullanıcıyı herhangi bir konumda iken eğer komutun ihtiyaç duyduğu giriş dosyası da hazırsa her defasında makefile yeniden düzenlenmeden daha pratik bir şekilde amaçlanan dosya üretimini yapabilir kılmasıdır.

Aşağıdaki maddelerde farklı biçimlerde dosya üretimi için Xilinx araçlarını kullanan komutlar sırasıyla verilmiştir. Bunlar ihtiyaçları olan giriş dosyasını alıp çıkışta bizim istediğimiz formatta bir er dosyaya dönüştürler. Yukarıda yazılan makefile komutu aşağıda yazılan açık komutla aynı işi yapmaktadır.

Bitgen;

make orpsoc_spiboot.bit

Bu komut yazıldığında alt satırdaki kod çalışır

bitgen -w -intstyle silent -g StartUpClk:CClk orpsoc.ncd orpsoc_spiboot.bit

make orpsoc.bit

Bu komut yazıldığında alt satırdaki kod çalışır

bitgen -w -intstyle silent -g StartUpClk:Cclk orpsoc.ncd orpsoc_spiboot.bit

Promgen;

- Sadece bit file için .mcs dosyası üretmek (uygulama yok);

make orpsoc.mcs (makefile'da BOOTLADER_BIN satırı olmamalı)

Bu komut yazıldığında alt satırdaki kod çalışır

promgen -spi -p mcs -w -o orpsoc.mcs -s 16384 -u 0 orpsoc_spiboot.bit

promgen -spi -p mcs -w -o orpsoc.mcs -s 16384 -u 0 orpsoc.bit

- Bit file ve bir uygulama için tek bir .mcs dosyası üretmek

make orpsoc.mcs (makefile'da BOOTLADER_BIN satırı olmalı)

Bu komut yazıldığında alt satırdaki kod çalışır

promgen -spi -p mcs -w -o orpsoc.mcs -s 16384 -u 0 orpsoc.bit -data_file up 1c0000 uygulama_size.bin

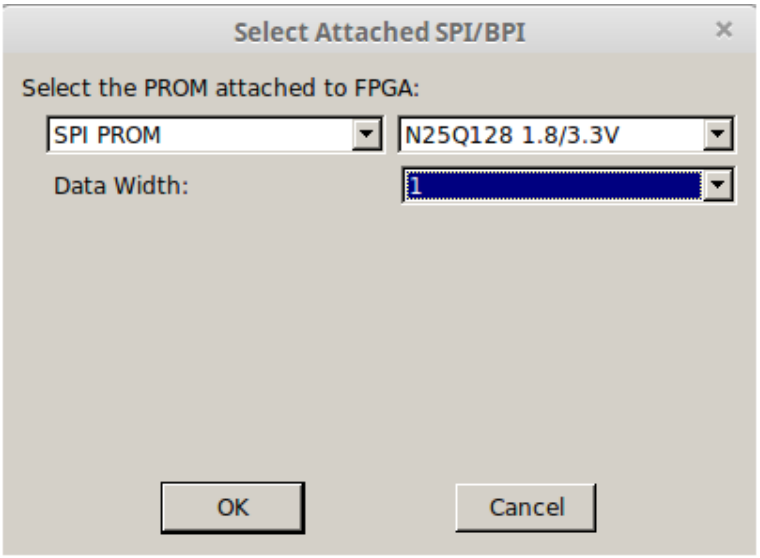
Sadece bir uygulma için (aes.c kodu için) .mcs dosyası oluşturma

promgen -spi -p mcs -w -o uygulama.mcs -s 16384 -data_file up 1c0000 uygulama_size.bin

Bundan sonraki aşama spi flash'a üretilen .mcs dosyasının yazılması ve uygulamanın çalışmasının görülmesidir. Bu noktada Atlys geliştirme kartı üzerinde yapılacak bir işlem söz konusudur. FPGA'in spi flash'tan yapılandırma dosyasını okuyarak başlaması için geliştirme kartı üzerinde bulunan JP11 numaralı soket çıkarılmalıdır. Bu soket takılı iken FPGA spi flash'tan başlayamaz. Bu işlem geliştirme kartına enerji verilmeden önce yapılmalıdır.

Geliştirme kartının programlama girişi ve UART girişi USB kabloları ile bilgisayara bağlandıktan sonra sıradaki işlem "impact" arayüzünün başlatılmasıdır. Program başlatılır ve bu sefer sağ tıklanıp "Add SPI/BPI prom" seçeneği tıklanır. Ardından gelen kısımda Atlys geliştime kartında bulunan spi flash'ın modelini belirtmmek gerekecektir. Kart üzerinde bulunan model N25Q128'dir. Seçim Şekil 4.6'deki gibi yapılır.

Bu sefer elimizde FPGA yapılandırma dosyası ve uygulama aynı .mcs dosyasında birleşik halde olduğu için FPGA doğrudan programlanmayacaktır. Elbette bu anlatım .mcs dosyasının bu şekilde oluşturulduğu varsayımı altında geçerlidir. Şayet sadece uygulama için .mcs oluşturulmuş ise bu sefer önce FPGA yapılandırma dosyası sonrasında ise spi flash'a uygulama dosyası yazılır. (Bu durumda FPGA'nın spi



Şekil 4.6: Spi Flash Model Seçimi.

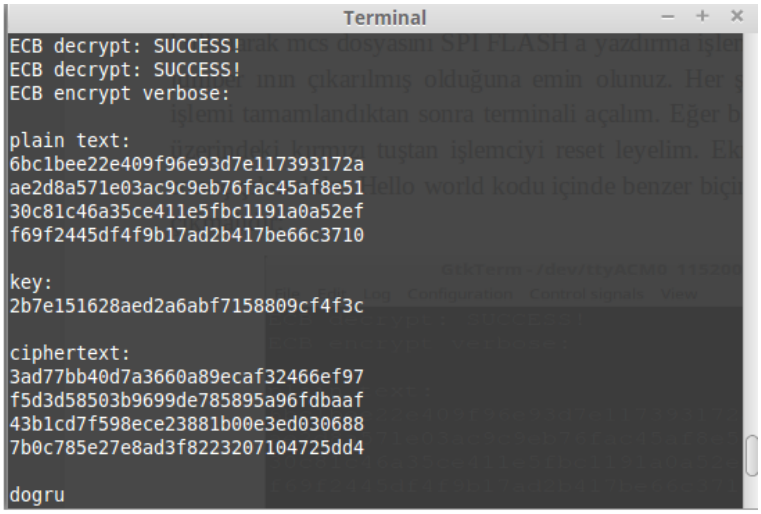
flashtan mı yoksa doğrudan kendisine gönderilen yapılandırma dosyasından mı başlatılacağı kullanıcı tarafından dosya yazılmadan hemen önce seçilebilir haldedir.)

Varsayılan durum altında (uygulama ve yapılandırma dosyası tek bir .mcs dosyası) arayüzden programlama emri gönderildikten sonra uygulamanın durumuna bağlı olarak spi flash'a yazma süresi yaklaşık on dakika veya daha fazla sürmektedir. Burada asıl zaman kaybı FPGA yapılandırma dosyasından kaynaklanmaktadır. Çünkü sabit 1.5 MB yer kaplamaktadır. Genelde uygulamalar kB seviyesindedir. Elbette daha büyük bir uygulama da olabilir. Spi flash 16 MB boyutunda olduğu için epeyce büyük uygulamalar dahi saklanabilir durumdadır.

Tüm işlemlerin ardından nihayet uygulamanın sonucunu görmeye sıra gelmiştir. Spi flash'a yazma bittikten sonra daha önce yüklenmiş olan seri terminal açılır. Örnek bir AES kodu olduğu için Şekil 4.7'deki gibi bir çıktı seri terminalden iletilir.

4.7 LedTest Uygulaması

Şekil 4.7'de görüldüğü gibi işlemci kendisine verilen bir metin (plain text) üzerinde yine kendisine verilen bir anahtar (key) ile AES algoritmasına göre bir şifreleme yapmış ve şifreli metni (ciphertext) yazmıştır. Ardından bu işlemin tersini yaparak şifreli metinden AES şifre çözme algoritmasını kullanarak orijinal metni yeniden elde etmiştir. Son olarak eğer iki metin birbirinin aynı mı diye bir kontrol yapmış ve



Şekil 4.7: Aes Uygulamasının Çıktısı.

eşdeğer olduğunu teyit ettikten sonra en son doğru diye bastırmıştır. Bu işlem or1200 işlemcisi üzerinde elbette çok hızlı bir biçimde yapılmaktadır. Seri terminal açılır açılmaz sonuç doğrudan ekrana basılır. Hatta hızı burada sınırlayan uygulamadan ziyade ekrana basma işlemidir (Bunu da or1200 değil bilgisayar yapmaktadır. Or1200 sadece UART'a yazar).

Bu uygulamada Atlys geliştirme kartı üzerindeki ledleri belirli bir düzende yakan bir çalışmanın nasıl yapıldığı gösterilecektir. Bu sayede bir önceki uygulamadan farklı olarak geliştirme kartı üzerindeki bir birime nasıl erişildiğini anlamak mümkün olacaktır. İlk olarak /atlys/backend/par/bin/atlys.ucf dosyasını orpsoc_top.v ile birlikte incelemekte fayda vardır. Zira bu dosya platformun yonga dışına açılan birimlerinin geliştirme kartının neresinde olduğuna dair bilgileri içermektedir. Ayrıca işlemcinin istenen performanslarda çalışabilmesi için bazı sinyaller için zaman kısıtları yazılmıştır. Böylece gerçekleme aracı olan Xilinx bu dosyadaki bilgilere bakarak yonga üzerinde yerleşim stratejilerini uygulamaktadır.

İşlemciye bağlı bulunan 23-bitlik GPIO birimi led, switch, push buton gibi birimlere atanmıştır. GPIO birimi hem giriş hem de çıkış olarak kullanılabilen bir birim olduğu için C kodunda yapmamız gereken ilk şey GPIO birimlerini giriş veya çıkış olarak ayarlamak olmalıdır. Ayrıca bu uygulamada en önemli farklılıklardan birisi de yine yazılım tarafında bir include dosyası kullanma zorunluluğudur. Bu dosya ile ana programda erişmek istenilen birimleri adres bilgileri belirtilir. orpsoc-v2 içerisinde

böyle bir dosya kullanıcı için hazırlanmış durumdadır. Bu dosya xilinx/atlys/sw/board kısmında bulunmaktadır. Dosyanın açılıp incelenmesi tavsiye edilir. Zira burada yapılan GPIO_0_BASE gibi tanımlamalar uygulamalarda sıklıkla kullanılmaktadır.

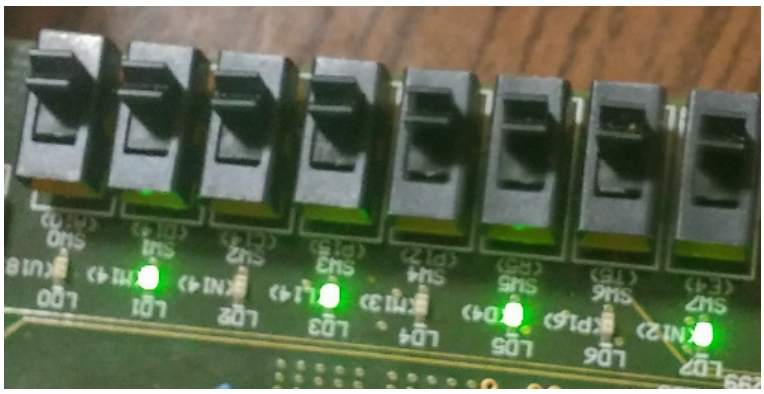
Ledleri belirli bir düzende yakan örnek bir uygulama için C kodu aşağıdaki gibi yazılabilir.

```
#include "board.h"
#define writeGPIO(addr,val) (*(unsigned char*) (addr) = (val))
#define readGPIO(addr)      (*(unsigned char*) (addr))
int main (int argc, char *argv[])
{

    writeGPIO((unsigned char *)(GPIO_0_BASE + 0x3), 0xff);
    writeGPIO((unsigned char *)(GPIO_0_BASE + 0x4), 0xff);
    writeGPIO((unsigned char *)(GPIO_0_BASE + 0x5), 0xff);

    writeGPIO((unsigned char *)GPIO_0_BASE, 0xaa);

    while(1);
 return 0;
}
```

C koduna bakıldığında dikkat çeken ilk şey makro tanımlarıdır. Bu uygulamada makro tanımları birer fonksiyon gibi kullanılmıştır. Elbette aynı işi yapan fonksiyonlar da kullanılabilir. İşlevsel olarak açıklamak gerekirse makro işaretsiz bir bayt değerinde içeriği olan bir adrese bir değer atamakta (yazma için kullanılan makro) veya o adresin içindeki işaretsiz bir bayt sayının değerini okumaktadır (okuma için kullanılan makro).

Kodda özellikle dikkat çeken diğer bir ayrıntı ise GPIO_0_BASE ile ifade edilmiş olan ön tanımlı değerlerdir. Bunların değeri board.h isimli include dosyasında tanımlanmıştır. Örneğin bu kodda kullanılan ifadeye karşılık gelen değer (adresin sayısal kısmı) 0x9100000 olarak belirlenmiştir.

ORPSoC-v2 de kullanılan GPIO birimi toplam altı adet yazmaç içermektedir. Bu yazmaçların ilk üç tanesi giriş veya çıkış verileri için sonraki üç tanesi ise GPIO biriminin kart üzerinde bağlı olduğu birimlerin giriş mi yoksa çıkış mı olduğunu belirten bilgi için kullanılmaktadır. Koda bakıldığında ilk olarak tüm birimler çıkış olarak ayarlanmış ve ardından ledlere karşılık gelen ilk yazmaca ledlere atanacak değer (0xaa) yazılmıştır. Böylece ledler sırasıyla bir yanık bir sönük şekilde kalacaklardır.



Şekil 4.8: LedTest Uygulamsı Sonrası Ledlerin Durumu.

Kod yazıldıktan sonra tıpkı bundan önceki uygulamada olduğu gibi önce derlenir. Ardından derlenmiş dosya binary formata çevrilir ve bin2binsizeword dönüştürücü programı ile gereken binary düzene dönüştürülür. Daha sonra önceki uygulamada anlatılan herhangi bir yöntemle spi flash'a yazılır. İşlem biter bitmez ledlerin (0xaa) şeklinde Şekil 4.8'deki gibi yandığı görülecektir.

Böylece Atlys geliştime kartı üzerindeki birimlere de erişilebilir olduğunu gösterir bir uygulama tamamlanmıştır.

4.8 U-Boot (Universial Boot Loader) Uygulaması

U-Boot (Universial Boot Loader) birçok platformu destekleyen, açık kaynak kodlu, boot (başlatma) formatlı kodları bir sabit diskten RAM belleğe transfer etmeye yarayan protokollere sahip bir programdır. Bu alanda oldukça sık kullanılan bir uygulamadır [23]. U-Boot ARM, MIPS, PowePC gibi birçok işlemci için destek vermektedir. Bu açıdan Linux işletim sistemini andırır ki aslında U-Boot bir çeşit Linux çekirdeğidir. Özellikle desteklediği transfer protokolleri sayesinde uygulamada önemli avantajlar sağlamaktadır. Bu çalışmada da özellikle uygulama çalıştırma noktasında ciddi kolaylıklar getirmiştir [23].

U-Boot bu çalışmada kolayca uygulama transfer etme ve çalıştırma amaçlı ve aynı zamanda ethernet biriminin test edilip çalıştığının gösterilmesi amaçları için kullanılmıştır. Bunun dışında U-Boot daha başka özellikleri için kullanılıyor olsa da bu çalışma kapsamında bu konuların üzerinde durulmamıştır.

U-Boot programının derlenmesi ve ardından spi flash'a gömülmesi gerekmektedir. Bunun için önce kodların indirilmesi gerekecektir. OpenRisc web sayfasının U-Boot ile alakalı bölümünde kısa tanımlama ve kullanım ayarları haricinde kaynak kodların bulunduğu linkler de paylaşılmıştır. Bu sayfayı incelemek faydalı olacaktır [23]. Sayfanın yönlendirdiği linklere tıklandığında aslında kaynak kodları o sayfada değil GitHub depo alanında olduğu anlaşılmaktadır. Dolayısyla kaynak kodları indirmek için aşağıdaki komutları terminale yazıp işletim sistemin Ev Dizinine indirebilir. Komutta kullanılan url adresi web sayfasından alınmıştır.

git clone git://git.denx.de/u-boot.git

Bunun haricinde kaynak kodlar başka kaynaklardan da indirilebilir. Bunlara da örnek olarak aşağıdaki adresler verilebilir.

https://github.com/mczerski/u-boot

https://github.com/sen1113/u-boot

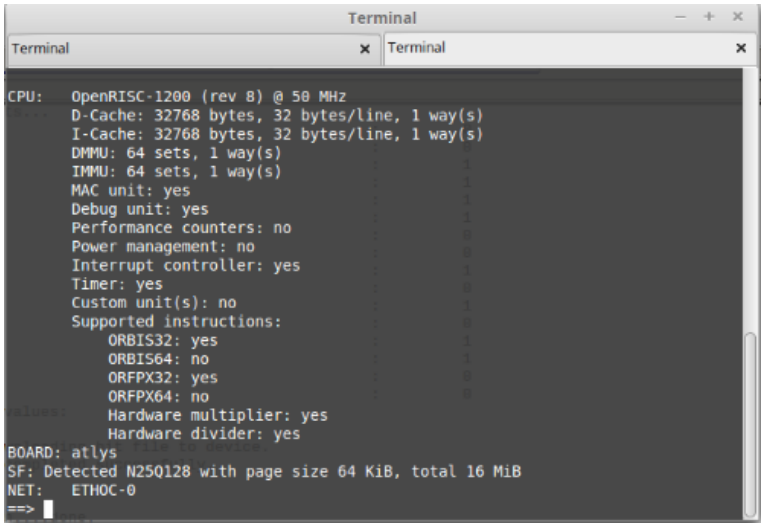
Kodlar indirdikten sonra bazı dosyalarda derleyici takımı versiyonundan dolaylı ufak değişiklikler yapılması gerekmektedir. Bu değişikliklerin sebebi bazı versiyonlarda halen daha eski derleyici takımı olan or32-elf ile ilgili parametreler olmasıdır. Güncel olarak kullanılan derleyici takımı or1k-elf olduğundan bir kaç dosyada or32-elf yerine or1k-elf yazılmalıdır. Aşağıdaki dosyaları kontrol edip bu değişikliklere iihtiyaç varsa yapılmalıdır.

/u-boot/arch/openrisc/config.mk

/u-boot/board/atlys/u-boot.lds

/u-boot/board/openrisc/openrisc-generic/u-boot.lds

İndirilen kodda OpenRISC ve Atlys tanımlamalarının olup olmadığına dikkat edilmelidir. Zira resmi kaynak kodların olduğu linkten indirme yapıldığında malesef Atlys kartına ilişkin dosyalar bulunmamaktadır. Bu eksiği kodların bulunduğu diğer adreslerden indirerek gidermek gerekecektir. Toplamda beş adet dosya barındıran Atlys klasörü indirilip board adlı dizine konulmalıdır. Bulunması gereken beş dosya ise şunlardır: makefile, atlys.c, u-boot.lds, or1ksim.cfg, config.mk. Bunların haricinde bir de yine resmi kaynak kodlarında olmayan ve alternatiflerden temin edilmesi gereken



Şekil 4.9: U-Boot Programı Açılış Ekranı.

bir dosya daha vardır. Eksik dosya temin edilip (atlys.h) /u-boot/include/configs/ klasörüne konulmalıdır.

Kurulum aşamasına geçildiğinde ilk yapılması gereken .bashrc dosyasına aşağıdaki parametreyi kopyalalıp kaydetmektir.

export ARCH=openrisc

export CROSS_COMPILE=or1k-elf-

Ardından terminalden U-Boot dosyasına gelip aşağıdaki işlemler yapılınca u-boot.bin adında dosya üretilecektir.

make distclean (daha önce derlenmiş bir kod varsa temizlemek için)

make atlys

U-Boot programı bilgisayarla bağlantı kurup dosya transferi yaparken ethernet birimini kullandığı için bize uygulamayı yaparken bir ethernet kablosu ve birde bağlantı çoğullayıcı gerekmektedir. Ana internet hattı bu çoğullayıcıya bağlanırken alınan iki ayrı bağlantının birisi bilgisayara birisi de Atlys geliştime kartında bulunan ethernet girişine bağlanır.

Kurulum işlemlerin ardından u-boot.bin dosyası da daha öcnceki örneklerde yapıldığı gibi bin2binsizeword ile dönüştürülür ve spi flash'a yazılır. İşlemci çalışmaya başladıktan sonra Şekil 4.9'deki gibi bir ekran elde edilir.

U-Boot ile uart üzerinden haberleşilebildiği için bu aşamada yapılacak olan U-Boot programını kullanabilmek için gerekli olan ayarları yapmaktır. Yapılacak iki ayarlama vardır denilebilir. Birincisi U-Boot programının kullanacağı IP ve diğer değişkenleri ayarlamak, ikincisi ise bilgisayara U-Boot tarafından desteklenen ve dosya transfer protokollerini sağlayan bir servis kurmak. İlk olarak yapılacak işlem (U-Boot bağlantı değişkenlerini ayarlamak) ekranda görülen terminale aşağıdakileri yazıp en son **saveenv** komutu ile kaydetmektir.

```
setenv gatewayip 192.168.255.254
setenv ipaddr 192.168.255.27
setenv netmask 255.255.0.0
setenv serverip 192.168.255.100
setenv ethaddr 00:12:34:56:78:9a
saveenv
```

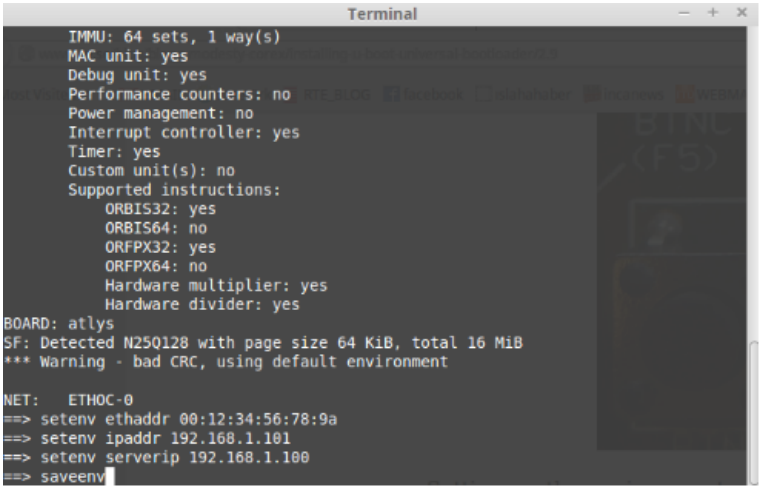
Yukarıdaki tanımlamalar adlarından da anlaşıldığı gibi programın ağ ayarlamaları için yapılmaktadır. "serverip" her bilgisayar için değişiktir. Uygulama yapılan bilgisayar server olarak görünecektir. "ipaddr" ayarı "serverip" olmayan başka bir şey de seçilebilir. Ayrıca "ethaddr" de rakamlardan anlaşılacağı üzere tamamen uydurulmuş bir atamadır. Bunların haricinde "netmask" ve "gatewayip" aslında bizim için zorunlu değişkenler değillerdir. İşlemden sonra bu bilgiler spi flash'a kaydedilir ve bundan sonraki uygulamlar için de kullanılanilir. Fakat işlemin geçerli olması için bazen geliştirme kartındaki reset (kırmızı buton) kullanılarak yeniden başlatmak veya kartı kapatıp yeniden açmak gerekebilir. Değişkenlerin durumundan emin olmak için U-Boot terminaline **printenv** yazılıp kontrol edilebilir. Ayrıca U-Boot ile yapılabilecek başka işlemleri öğrenmek için **help** yazılıp çıktılar okunabilir. İşlem esnasında terminal Şekil 4.10'daki gibi görüntülenecektir.

Bağlantı ayarlarının ardından şimdi de sırada bilgisayarza bir dosya transfer servisi kurmaya gelinmiştir. Bunun için bu çalışmada TFTP tercih edilmiştir. Bu protokole ait bilgi internette bolca mevcuttur. Burada kurulumdan bahsedilecektir. Aşağıdaki işlemler sırayla yapılır.

–> sudo apt-get install tftp

–> sudo apt-get install xinetd

–> sudo apt-get install tftpd



Şekil 4.10: U-Boot Ağ Ayarlarının Yapılışı.

Ardında yönetici dizininde bir klasör oluşturup izinlerini de açmak gerekmektedir;

–> sudo mkdir /tftpboot

–> sudo chmod -R 777 /tftpboot

–> sudo chown -R nobody /tftpboot

Daha sonra /etc/xinetd.d/tftp kısımda tftp adlı bir dosya açılmalıdır. Açılan dosyanın içine aşağıdakileri aynen yazıp kaydetmek gerekmektedir.

```
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /tftpboot
disable         = no
}
```

İşlemlerin ardından yüklenen programı çalıştırılıp kontrol edebilir duruma gelinmiştir. Bunun için aşağıdaki kodu terminale yazıp çalıştırmak gerekir;

sudo service xinetd restart

Programın çalışıp çalışmadığını anlamak için bir test yapılabilir. Terminalden şu komutlar girilir;

netstat -l -u | grep tftp

Şekil 4.11: TFTP Server Testi.

Eğer karşımıza Şekil 4.11 gibi bir sonuç çıkıyorsa program düzgün biçimde çalışıyor demektir.

Böylece artık dosya transferi yapabilecek hale gelinmiştir. Bundan sonraki işlemlerde U-Boot programı uygulama dosyalarını transfer edecek bir taşıyıcı ve kontrollü bir şekilde uygulama çalıştırmaya yarayan bir işlev görecektir.

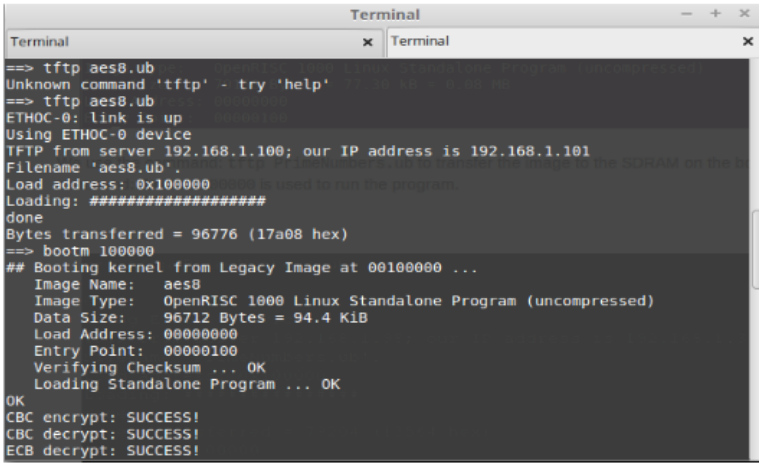
U-Boot programının dosya transferi yapıp transferi yapılan uygulamayı da işlemciyi kullanarak çalıştırması için uygulama dosyalarının bir format dönüşümünün yapılmasına ihtiyaç duyulmaktadır. Bu sefer de dosyqlar u-image formatına dönüştürülecektir. Dönüştürülen dosyalar daha sonra root dizininde daha önce oluşturulan tftpboot klasörüne taşınmalıdır. Uygulamaları uygun formata dönüştürmek için mkimage adlı bir araca ihtiyaç duyulmaktadır. Bu araç işletim sisteminde önceden yüklü olabileceği gibi aynı zamanda daha önce indirilen U-Boot klasörünün içinde tools klasöründe de bulunmaktadır. Bu klasörde bulunan mkimage.c ve mkimage.h dosyaları GCC ile derlenip üretilen mkimage dosyasının örneğin /usr/local/bin gibi bir yere kopyalanması gerekmektedir. Şayet işletim sisteminde yüklü değilse aşağıdaki gibi yüklemek mümkündür.

sudo apt-get install u-boot-tools

Bu program ile bir uygulama dosyası aşağıdaki gibi dönüştürülebilir.

mkimage -A or1k -T standalone -C none -a 0 -e 0x100 -n deneme -d deneme.bin /tftpboot/deneme.ub

Bu işlemin ardından deneme.ub dosyası root dizininde bulunan tftpboot klasörüne gönderilmiş olur. Bu aşamadan itibaren deneme.ub dosyasının U-Boot ile transfer edip çalıştırmak artık mümkün hale gelmiştir. Daha önceki bölümlerde anlatılan AES uygulaması bu kez de U-Boot üzerinden aşağıdaki şekilde çalıştırılır. AES kodu daha önce derlenmiş ve or1k-elf objcopy aracıyla binary formata çevrilmiştir. Bu dosyayı bin2binsizeword dönüşümü yapmadan doğrudan mkimage dönüşümüne



Şekil 4.12: Aes uygulamasının U-Boot ile Çalıştırılması.

tabi tuttuğumuzu ve üretilen aes.ub dosyasınında tftpboot klasörüne kopyalandığını varsayarak U-Boot terminalde aşağıdaki işlemler yapılırsa uygulamanın transfer edilip Şekil 4.12'deki gibi çalıştırıldığı görülecektir.

tftp deneme.ub

bootm 100000 (100000 adresinden çalıştır demektir)

Böylece hem U-boot gibi karmaşık bir uygulama gerçeklenmiş hem de sonraki uygulamaları hızlıca deneyebilecek bir transfer ağı oluşturulduğundan pratiklik kazanılmıştır.

4.9 Linux Kernel Uygulaması

OR1200, Linux Kernel gibi bir işletim sistemini rahat bir şekilde çalıştırabilen güçlü bir işlemcidir. Bu bölümde adım adım Linux Kernel'in or1k-elf- derleyici takımı ile derlenmesinden Atlys geliştirme kartı üzerinde çalıştırılmasına kadar yapılan işlemler anlatılmıştır.

Linux Kernel 3.1 sürümünden itibaren OpenRISC işlemcilerine destek vermektedir [24]. Dolayısıyla son sürümü veya ara sürümlerden birini kullanmak mümkündür. Ancak derleyici takımı ile ilgili problem yaşamamak için son sürümü kullanmak avantajlı olacaktır. Linux Kernel aşağıdaki gibi terminal aracılığıyla OpenRISC resmi GitHub depolama alanından indirilebilir. En güncel versiyon buradadır.

git clone https://github.com/openrisc/linux

Kuruluma geçmeden evvel ciddi bir değişiklik yapmak gerekmektedir. Bu çalışma yapılırken çokça üzerinde durulup çözülemeyen sorunlardan birisi budur. OR1200 işlemcinin veri ön bellek (32KB) özelliği kaldırılmadıkça Linux Kernel U-Boot kullanarak denenen çalışmalarda maalesef çalışmamıştır. Sorunun kaynağının ne olduğu bu çalışmanın yapıldığı dönemde tespit edilememiştir. Bununla ilgili çalışmalara devam edilecektir. Ancak Bölüm 5'te anlatılan olan mor1kx işlemcisinde bu sorunla karşılaşılmamıştır.

Veri ön belleğinin sistemden çıkarılması için basit ancak dikkatli davranılması gereken bir işlem yapılmalıdır. İşlemin yapılacağı dosyada aynı satırlardan iki adet olmasından dolayı bir kafa karışıklığı olmamalı ve dosyada satır sayısı kesinlikle artırılıp azaltılmamalıdır. Bu dosya indirilen Orpsoc-v2 klasöründe /.../atlys/rtl/verilog/include/or1200_defines.v olarak mevcuttur. Dosya açılıp aşağıda gösterildiği gibi, sadece varsayılan durumda yorum (comment) halde bulunan ve "data bellek olmasın" manasındaki ifadeyi aktif hale getirmek gerekmektedir. Dosya üzerinde başka hiç bir işlem yapılmadan kaydedilmelidir. Ayrıca aynı tanımlamaların olduğu ASIC için yazılmış bölümleri de içeren bir dosya olduğundan, bu dosya da ifadeler başlık başlık ayrılmıştır. Yapılması gereken tek şey aşağıdaki başlık altındaki ifadeyi bulup aktif hale getirmektir.

```
// Typical configuration for an FPGA
 /*.....................*/
// Do not implement Data bellek
`define OR1200_NO_DC
```

Linux Kernel'i Atlys üzerinde çalıştırmak için donanım tanımlama dosyalarının (veri ön belleği yukarıdaki gibi devreden çıkarılmış halde iken) bir kez daha sentez ve diğer aşamalardan geçirilip FPGA yapılandırma dosyası oluşturulmalıdır. Ayrıca Linux Kernel derlendikten sonra spi flash'a U-Boot aracılığı ile transfer edileceği için öcelikle donanım dosyalarında bootrom.v muhakkak olmalıdır (include içerisinde). Ayrıca bu dosya bundan önceki uygulamalarda da olduğu gibi rom.v içerisinde bağlanmış durumda olmalıdır.

Linux Kernel uygulaması ile ilgili donanım tarafında yapılması gerekenler işlemlerin yanı sıra bir de yazılım tarafında bazı işlemler yapılmalıdır. Geliştirme kartında kullanmadan önce Kernel'i bir kez simülatörde çalıştırmak faydalı olacaktır. Bunun

için ilk önce “xterm” adında bir servis kurulmalıdır. Bu program aslında X Window sistemler için bir terminal emülatörüdür [25].

sudo apt-get install xterm

Daha önce indirilmiş olan “linux” klasörünün içine girip Linux Kerneli or1ksim de çalışacak şekilde derlemek gerekmektedir. Bunun için şu komutları sırasıyla yaptırılmalıdır;

export CROSS_COMPILE=or1k-elf-

export ARCH=openrisc

make or1ksim_defconfig

make -j8

Son satırdaki -j8 parametresi zorunlu değildir. Sadece sistem kurulumu paralel olarak daha hızlı yapma imkanı varsa yapılması için yazılmıştır. Bu işlemlerden sonra **vmlinux** dosyası oluşmuş vaziyettedir.

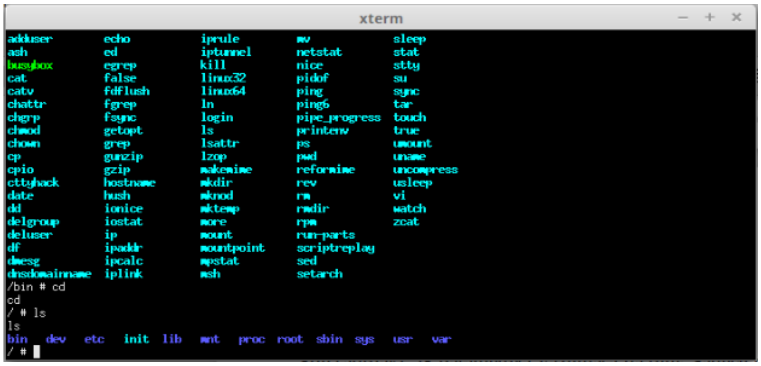
Linux Kernel’in simülator üzerinde çalışması için ve aynı zamanda daha önce kurulan xterm servisinin kullanması için bir işlem daha yapılmalıdır. /linux/arch/openrisc/or1ksim.cfg dosyası açılıp ve bu dosyada “uart section” kısmına gelinmeli ve bu kısım aşağıdaki hale getirilmelidir. Böylece Linux’un xterm üzerinde çalışması sağlanmış olur.

```
section uart
  enabled = 1
  baseaddr = 0x90000000
  irq = 2
  /* channel = "file:uart0.rx,uart0.tx" */
/*   channel = "tcp:10084" */
  channel = "xterm:"
  jitter = -1                   /* async behaviour */
  16550 = 1
end
```

Bu değişiklik yapılıp dosya kaydedildikten sonra terminal ekranında kalınan yere şu komut girilir:

or1k-elf-sim -f arch/openrisc/or1ksim.cfg vmlinux

Bu işlemin ardından bir süre sonra xterm ekranında “Enter to activate this consol” bildirimi alınacaktır. Enter tışuna basılınca Linux Kerne’in terminal ekranındaki çalışması görülecektir. Burada ls veya cd gibi basit komutlar çalıştırabilir ve Linux’un en sade hali test edilebilir. Bu durumda Linux dosya sistemi RAM bellekte bir bölgede tanımlanmaktadır. Ekran Şekil 4.13’deki gibi görünecektir.



Şekil 4.13: Linux Kernel xterm Servis Üzerinde Çalışması.

Sistemi simülasyon ortamında denedikten sonra, şimdi de Atlys geliştirme kartı üzerinde deneme aşamasına geçilebilir. Bunun için ilk önce yukarıda xterm için yapılmış olan “uart section” ayarları eski haline getirilmelidir. Ayrıca klasör içinde “make distclean” yapılarak önceki üretilmiş dosyalar silinmelidir. Ancak bunu yapmak yerine daha akılcı olan tekrar bir “linux” dosyası indirmek ve mevcut olanı simülasyon için kullanmaktır.

Atlys geliştirme kartınının özelliklerini içeren bir dosyaya daha ihtiyaç duyulmaktadır. Bu dosya indirilen Linux dosyasında olmasa da internette rahatça bulunabilir. Ayrıca orpsoc-v3 (sonraki bölümde anlatılacak) içerisinde de standart olarak bulunmaktadır. Dosyanın ismi atlys.dts olup bu dosya bulunup indirildikten sonra **linux/arch/openrisc/boot/dts/** kısmına kopyalanmalıdır.

```
{
  compatible = "digilent,atlys";
  #address-cells = <1>;
  #size-cells = <1>;
  interrupt-parent = <&pic>;
  chosen {
    bootargs = "console=uart,mmio,0x90000000,115200";
  };

  memory@0 {
    device_type = "memory";
    reg = <0x00000000 0x08000000>;
```

```
  };

  cpus {
    #address-cells = <1>;
    #size-cells = <0>;
    cpu@0 {
      compatible = "opencores,or1200-rtlsvn481";
      reg = <0>;
      clock-frequency = <50000000>;
    };
  };

  pic: pic {
    compatible = "opencores,or1k-pic";
    #interrupt-cells = <1>;
    interrupt-controller;
  };

  serial0: serial@90000000 {
    compatible = "opencores,uart16550-rtlsvn105", "ns16550a";
    reg = <0x90000000 0x100>;
    interrupts = <2>;
    clock-frequency = <50000000>;
  };

  enet0: ethoc@92000000 {
    compatible = "opencores,ethmac-rtlsvn338";
    reg = <0x92000000 0x100>;
    device_type = "network";
    local-mac-address = [ 00 12 34 56 78 9a ];
    interrupts = <4>;
  };
};
```

Sözü edilen atlys.dts dosyası yukarıda görüldüğü gibi geliştirme kartına ait Linux'ta tanımlanması gereken birçok parametreyi tanımlamıştır. Bu eksik de giderildikten sonra, derleme safhasına geçilebilir. Terminalden linux klasörüne girip aşağıdaki işlemleri yapmak gerekmektedir.

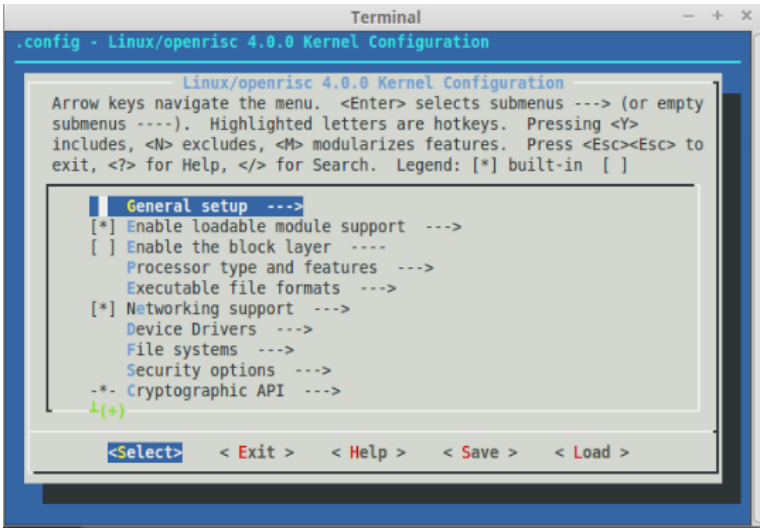
export CROSS_COMPILE=or1k-elf-

export ARCH=openrisc

make defconfig

make menuconfig

Bu işlem yapıldığında Linux Kernel'in çeşitli özelliklerinin ayarlanabileceği bir ekranla karşılaşılacaktır. Bu arayüzden bu uygulama için kritik olan iki değişiklik yapılacaktır. Bu değişiklikler derleyici takımı seçimi ve atlys.dts dosyasının



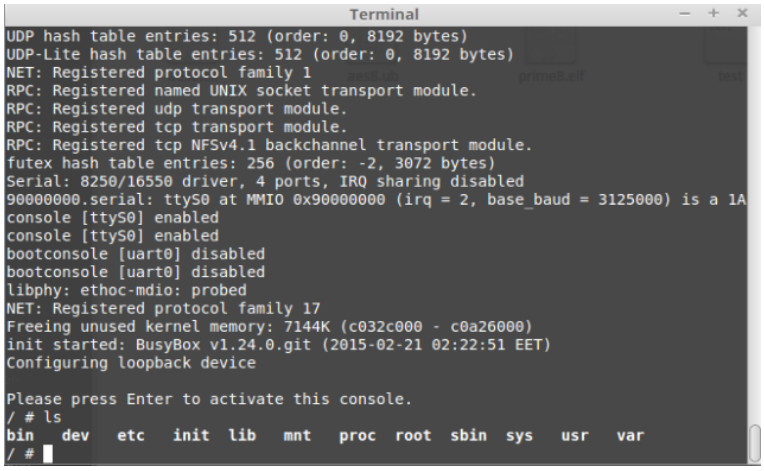
Şekil 4.14: Linux Kernel Menü Ayarları.

gösterilmesi için yapılan seçimlerdir. Böylece sistem bu parametrelere göre derlenecektir. Menü Şekil 4.14'deki gibi görüntülenecektir.

Şekil 4.14'deki menüde görüldüğü gibi çok geniş ayarlamalar yapmak mümkündür. Bu uygulama için yapılması gerekli ayarlar için ise ilk önce **General Setup** başlığına girilip ardından gelen kısımda **cross compile tool prefix** parametresine tıklanıp **or1k-elf-** olarak düzenlenmesi gerekir.

Diğer ayarlama ise atlys.dts dosyasının tanıtılması içindir. Bunun için biraz önce yapılan ayar penceresinden exit komutu ile çıkıp (üst menüye) burada bu kez "Processor type and features" başlığına girilip burada **Built in DTB** başlığına tıklayıp bu kısmın parametresini "atlys" olarak değiştirmek gerekmektedir. Tüm ayarları yaparken **save** komutunu kullanarak işlemleri eksiksiz yapıp kaydedildiğine emin olunmalı ve en son menüden çıkmadan save komutu ile değişiklikler kaydedilmelidir. Ardından **exit** komutu ile pencere kapatılıp yeniden terminal ekranına dönülür. Terminalde **make** komutu ile derleme başlatılır ve uzun bir müddet kurulumun tamamlanması için beklenir.

Kurulumun bitmesi ile **vmlinux** dosyası üretilmiş olur. Buradan sonra yapılacak işlem tıpkı önceki uygulamalarda yapıldığı gibi vmlinux dosyasını ikilik formata çevirme işlemi ve mkimage programı ile transfer edilebilir formata dönüştürülmesidir. Bunu da aşağıdaki gibi yapmak gerekmektedir.



Şekil 4.15: Linux'un Atlys Geliştirme Kartından Çalıştırılması.

or1k-elf-objcopy -O binary vmlinux vmlinux.bin

mkimage -n 'Linux for OpenRISC' -A or1k -O linux -T kernel -C none -a 0 -e 0x100 -d vmlinux.bin uImage

Son adımda üretilen uImage adlı dosya yönetici dizininde bulunan tftpboot klasörüne taşınmalıdır. Böylece Linux Kernel'i Atlys üzerinde çalıştırmak için hazır hale gelinir. Şimdi yapılacak işlem U-Boot terminali açıp aşağıdaki gibi **uImage** dosyasını transfer etmek ve başlatmaktır.

tftp uImage

bootm 100000

İşlemin ardından yaklaşık on saniye evvel Linux'un başladığı ve "Enter to Activate Console" mesajını verdiği görülür. Enter tuşuna basılarak Linux Terminal'e geçilir. Ekran Şekil 4.15'deki gibi olacaktır.

Böylece işlemcisi, işletim sistemi, RAM belleği, flash belleği ve daha birçok çevresel elemanları ile birlikte epeyce karmaşık bir sistem gerçeklenmiştir. Bu durumda işletim sistemi olarak Linux devreye girdiği için sahip olduğu dosya sistemi RAM bellekte tanımlı haldedir. Ancak bu geçici olduğu için Linux'a dosya sisteminin bir bilgisayara veya başka bir geniş belleğe koyabilmesini sağlayan **"Build Root"** adlı bir paketin derlenip kurulması gerekmektedir. Ancak bu çalışmada bu işlemin nasıl yapılacağı anlatılmayacaktır.

4.10 ORPSoC-v2 Platformuna Kullanıcı Tasarımı Donanımlar Ekleme

4.10.1 Sayıcı Modülünün Eklenmesi

OpenRISC 1000 ailesine mensup işlemciler açık kaynak kodlu oldukları için iç yapıları modüllerin incelenmesiyle anlaşılabilir. Bu noktadan hareketle sisteme çevresel bir birim eklemek veya mevcut birimleri değiştirmek veya tamamen devreden çıkarmak mümkün hale gelmektedir. Bu esneklik gerçeklenebilir işlemcilerin sunduğu büyük bir avantajdır [5]. OpenRISC ise buna ek olarak açık kaynak kodlu bir yapı sunduğu için daha da avantajlı durumdadır.

Birçok projede bazı birimlere sahip denetleyici veya işlemcilere ihtiyaç duyulur. Öyleki bazen sırf bir çevresel elemanı olmadığı için ihtiyaçlara en uygun olan işlemci yerine başka bir çözüm aranmak zorunda kalınır. İşte bu sorun açık kaynak kodlu işlemcilerin en işe yarar kılındığı noktalardan birisidir. Tasarımcılar her ihtiyaç duyulan (bazen bir kaç adetten fazla yongaya ihtiyaç olmaz) durumda yeni bir işlemci tasarlamak (mevcut durumdakiler ihtiyaca cevap veremiyorsa) veya sırf bazı birimleri eksik diye başka çözümlere yönelmek yerine açık kaynak kodlu işlemcileri tercih edip istedikleri gibi düzenleyerek kullanabilir.

Bu bölümde basit sayılabilecek bir modülün işlemcinin Wishbone arayüzüne nasıl eklenebileceği adım adım anlatılmıştır. Fakat burada asıl dikkat edilmesi gereken modülün basit veya karmaşık bir yapı olmasından ziyade Wishbone arayüzüne doğru bir şekilde bağlanmasıdır. Çünkü işlemci modülün ne yaptığı ile ilgilenmez ve sadece modüle bir giriş verip sonucunu alır. Dolayısyla önemli olan wishbone arayüzünün arkasında ne olduğundan çok Wishbone arayüzünün doğru tasarlanmasıdır. Fakat bu tabii ki modülün karmaşık yapısının arayüz yazımını değiştirmediği anlamına gelmez!

Uygulamaya geçilmeden önce bilinmesi gereken önemli başka bir bilgi de platformun arayüz tasarımı ile ilglilidir. ORPSoC-v2 platformunda üç ayrı arayüz bulunmaktadır. Bunlar komut hattı (instruction bus), veri hattı (data bus) ve bayt hattı (byte bus)'dır. Komut hattı adından anlaşılacağı gibi işlenecek olan komutları taşımaktadır. Burada dikkat çeken kısım 32-bit genişliğinde bir veri hattına ve buna ilaveten 8-bit genişliğinde ve byte bus olarak adlandırılan ikinci bir veri hattı tanımlanmış olmasıdır. Bu yapıyla ilgili kodlar dikkatlice incelendiği zaman şöyle bir tasarım

olduğu anlaşılmaktadır: RAM, Ethernet, Veri Ön Belleği gibi hem daha önemli hem de daha hızlı çalışması gereken birimler 32-bitlik veri hattına bağlanmıştır. Bununla birlikte Gpio, Spi, Uart ve diğer birimler ise bayt genişliğinde bilgi taşınan hatta bağlanmıştır. Bu iki hat birbirinden tamamen ayrı da değildir. 32-bitlik veri hattında gerçeklenmiş olan bir köprü (bridge) iki hattı birbirine bağlar. Şayet 32 bitlik veri hattına bağlı modüllere bir bilgi gelmiyor veya onlardan bir bilgi gitmiyorsa (yani bu modüller o an hattı kullanmıyorsa) slave (köle) olarak köprü seçilir ve bu köprü bayt hattından gelen verileri (veya o hattaki modüllere giden verileri) yine asıl 32-bit hat üzerinden iletir. Yani aslında herşey 32-bitlik veri hattına bir şekilde bağlıdır. Ancak bir hiyerarşik düzenleme yapılmıştır. Bu çalışmada hem byte genişliğindeki hatta, hem de 32-bit genişliğindeki hiyerarşik olarak daha üstte olan hatta modül eklenmiştir. Elbette 32-bitlik hatta modül eklemek biraz daha zordur ancak iki durumda da yapılan işlemler biribirine oldukça benzerdir.

- İlk olarak sayıcı modülünün byte genişliğindeki hatta (byte bus) nasıl bağlandığına bakılmalıdır. Bunun için yapılan basit bir sayıcı tasarımı EK A.1'de paylaşılmıştır. Bu tasarımda arayüz ve sayıcı aynı modüldedir. Yapılan işlem çok basit olduğu için arayüzü ayrı modülü ayrı yazmayı gerektirecek bir durum yoktur. Yapılan işlemler GPIO gibi basit bir modülden benzetilerek yapılabilir.

- EK A.1'de verilen sayıcı kodu verilog dosyası olarak kaydedilir (mycounter.v) ve /atlys/rtl/verilog/ klasöründe aynı adla bir klasör açılarak bu klasörün içerisine kopyalanır.

- Daha sonra /atlys/rtl/verilog/include/ klasöründe bulunan orpsoc-defines.v dosyası açılır ve burada topkı diğer modüllerin (örneğin GPIO) tanımlandığı gibi eklenenecek modülün de tanımlaması yapılır. (Bu çalışmadaki adlandırmaya göre **'define MYCOUNTER0**)

- Ardından yine aynı klasörde bulunan orpsoc-params.v adlı dosyayı açıp eklenecek modül için aşağıdaki gibi adres ve veri parametreleri eklenmelidir. Tıpkı diğer modüller için yapıldığı gibi eklenecek modül için de bir başlık açıp düzenli bir şekilde bildirim yapmak önemlidir.

```
/MYCOUNTER0 params ----LATIF AKCAY
parameter wbs_d_mycounter0_data_width = 8;
parameter mycounter0_wb_adr_width = 3;
parameter mycounter0_wb_adr = 8'ha2;
```

- Yine aynı dosyada bulunan ve hatta bağlanan modül sayısını ifade eden tanımı bir artırmak gereklidir.

```
parameter bbus_arb_wb_num_slaves = 6;
```

- Yine aynı dosyada alt kısımda bulunan ve mevcut modüllerin adres bağlantısını ifade eden kısma yeni eklenen modül için de benzer bağıntı yazılmalıdır. Bu kısımda zaten kullanıcılar modül eklerse diye diğer bir çok kısımda olduğu gibi hazır ifadeler de yazılmış ve yorum halinde bırakılmıştır.

```
parameter bbus_arb_slave5_adr  = mycounter0_wb_adr;
```

Bu işlemlerin ardından geriye değiştirilmesi gereken iki dosya kalmıştır. Bunlar orpsoc_top.v ve hat gerçeklemesinin yapıldığı arbiter_bytebus.v dosyalarıdır. Bunlarda yapılan düzenlemeler bir nebze daha zordur ve dikkatli olmak gerekir.

Şimdi de /atlys/rtl/verilog/arbiter/ klasörüne gidip buradaki arbiter_bytebus.v dosyası üzerindeki değişiklikleri yapmak gerekmektedir.

- Arbiter modülünde portların tanımlandığı yere aşağıdaki gibi ekleme yapılır.

```
// Slave six  ----MYCOUNTER0
   // Wishbone Slave interface
   wbs5_adr_i,
   wbs5_dat_i,
   wbs5_we_i,
   wbs5_cyc_i,
   wbs5_stb_i,
   wbs5_cti_i,
   wbs5_bte_i,
   wbs5_dat_o,
   wbs5_ack_o,
   wbs5_err_o,
   wbs5_rty_o,
```

- “Slave interface” tanımlarının yapıldığı yere aşağıdakiler eklenir.

```
// Wishbone Slave interface
  output [wb_adr_width-1:0]   wbs5_adr_i;
  output [wbs_dat_width-1:0]  wbs5_dat_i;
  output                      wbs5_we_i;
  output                      wbs5_cyc_i;
  output                      wbs5_stb_i;
  output [2:0]                wbs5_cti_i;
  output [1:0]                wbs5_bte_i;
  input [wbs_dat_width-1:0]   wbs5_dat_o;
  input                       wbs5_ack_o;
  input                       wbs5_err_o;
  input                       wbs5_rty_o;
```

- “Slave select” atamalarının altına;

```
assign wb_slave_sel[5] =
wbm_adr_o[`WB_ARB_ADDR_MATCH_SEL] == slave5_adr;
```

- “Slave input” tanımlamalarının yapıldığı yere;

```
// Slave 5 inputs
  assign wbs5_adr_i = wbm_adr_o;
  assign wbs5_dat_i = wbm_dat_o;
  assign wbs5_cyc_i = wbm_cyc_o & wb_slave_sel[5];
  assign wbs5_stb_i = wbm_stb_o & wb_slave_sel[5];
  assign  wbs5_we_i =  wbm_we_o;
  assign wbs5_cti_i = wbm_cti_o;
  assign wbs5_bte_i = wbm_bte_o;
  assign wbs_dat_o_mux_i[5] = wbs5_dat_o;
  assign wbs_ack_o_mux_i[5] = wbs5_ack_o & wb_slave_sel[5];
  assign wbs_err_o_mux_i[5] = wbs5_err_o & wb_slave_sel[5];
  assign wbs_rty_o_mux_i[5] = wbs5_rty_o & wb_slave_sel[5];
```

- Master out mux from slave in data başlığının altına;

```
wb_slave_sel[5] ? wbs_dat_o_mux_i[5] :
```

- Master out acks, or together baslığının altına;

```
wbs_ack_o_mux_i[5]
```

- Hemen ardından yapılan atamaların altına;

```
wbs_err_o_mux_i[5] |
```

- Hemen ardından yapılan atamaların altına;

```
wbs_rty_o_mux_i[5]
```

İşlemlerin ardından dosya kaydedilip kapatılır. Ayrıca değişiklik yapılan yerlere birer yorum yazıp (örneğin isim) belirtilmesi iyi bir tekniktir. Herhangi bir geri dönüş durumunda kafa karışıklığını gidermek için bu yöntemin uygulanabilir.

Son olarak /atlys/rtl/verilog/orpsoc_top/orpsoc_top.v dosyasını açıp aşağıdaki gibi değişiklikleri yapmak gerekir.

- Instruction bus slave wires başlığındaki tanımların altına;

```
// mycoounter0 wires ----LATIF AKCAY
  wire [31:0] wbs_d_mycounter0_adr_i;
  wire [wbs_d_mycounter0_data_width-1:0]  wbs_d_mycounter0_dat_i;
  wire [3:0]  wbs_d_mycounter0_sel_i;
  wire       wbs_d_mycounter0_we_i;
  wire         wbs_d_mycounter0_cyc_i;
  wire         wbs_d_mycounter0_stb_i;
  wire [2:0]  wbs_d_mycounter0_cti_i;
  wire [1:0]  wbs_d_mycounter0_bte_i;
  wire [wbs_d_mycounter0_data_width-1:0]  wbs_d_mycounter0_dat_o;
  wire  wbs_d_mycounter0_ack_o;
  wire  wbs_d_mycounter0_err_o;
  wire  wbs_d_mycounter0_rty_o;
```

- Byte Bus slaves başlığının altındaki tanımların altına;

```
  .wbs5_adr_i     (wbs_d_mycounter0_adr_i),
  .wbs5_dat_i     (wbs_d_mycounter0_dat_i),
  .wbs5_we_i      (wbs_d_mycounter0_we_i),
  .wbs5_cyc_i     (wbs_d_mycounter0_cyc_i),
  .wbs5_stb_i     (wbs_d_mycounter0_stb_i),
  .wbs5_cti_i     (wbs_d_mycounter0_cti_i),
  .wbs5_bte_i     (wbs_d_mycounter0_bte_i),
  .wbs5_dat_o     (wbs_d_mycounter0_dat_o),
  .wbs5_ack_o     (wbs_d_mycounter0_ack_o),
  .wbs5_err_o     (wbs_d_mycounter0_err_o),
  .wbs5_rty_o     (wbs_d_mycounter0_rty_o),
```

- Clock reset inputs başlığının altına;

```
defparam arbiter_bytebus0.slave5_adr = bbus_arb_slave5_adr;
```

- Wires başlığının alttında gpio modülünün altına;

```
 `ifdef MYCOUNTER0
     wire counter_interrutp;
   // MYCOUNTER 0 ---LATIF AKCAY
   mycounter mycounter0
   (
       .wb_adr_i        (wbs_d_mycounter0_adr_i[mycounter0_wb_adr_width-1:0]),
       .wb_dat_i        (wbs_d_mycounter0_dat_i),
       .wb_we_i         (wbs_d_mycounter0_we_i),
       .wb_cyc_i        (wbs_d_mycounter0_cyc_i),
       .wb_stb_i        (wbs_d_mycounter0_stb_i),
       .wb_cti_i        (wbs_d_mycounter0_cti_i),
       .wb_bte_i        (wbs_d_mycounter0_bte_i),
       .wb_dat_o        (wbs_d_mycounter0_dat_o),
       .wb_ack_o        (wbs_d_mycounter0_ack_o),
       .wb_err_o        (wbs_d_mycounter0_err_o),
       .wb_rty_o        (wbs_d_mycounter0_rty_o),
       .wb_clk        (wb_clk),
       .wb_rst        (wb_rst),
    .int_o                (counter_interrupt)
   );
 ////////////////////////////////////////////////////
`else
   assign wbs_d_mycounter0_dat_o = 0;
   assign wbs_d_mycounter0_ack_o = 0;
   assign wbs_d_mycounter0_err_o = 0;
   assign wbs_d_mycounter0_rty_o = 0;
`endif        /// ifndef MYCOUNTER0
```

- Or1200 Interrupt Assigments başlığının altına (Kesme hattı bağlantıları)

```
`ifdef DESWBC0 // LATIF
     assign or1200_pic_ints[6] = counter_interrupt;
   `else
     assign or1200_pic_ints[6] = 0;
   `endif
```

İşlemlerin ardından dosya kaydedip kapatılır. Böylece işlemler sona erer ve bu hali ile dosyalar yeniden sentez ve diğer aşamlarıdan geçirilerek FPGA yapılandırma dosyası tekrar üretilir.

Yukarıdaki düzenlemeler ve EK A.1'daki sayıcı modülü incelendiğinde özellikle dikkat edilmesi gereken bir konu da kesme (interrupt) işleminin nasıl yapıldığıdır. Dikkatlice incelendiğinde sayıcı modülünde işlemin tamamlandığı durumda (sayma işlemi bittiğinde) diğer durumlarda hep lojik sıfır değerinde tutulan int_o (kesme sinyali) değeri bu durumda lojik 1 değerine atanmıştır. Bu sinyal daha sonra orpsoc_top.v dosyasında bir başka sinyale (counter_interrupt) bağlanarak kesme sinyalinin ana modüle ulaştırılması sağlanmıştır. Ardından bu sinyal de tüm kesme sinyallerinin bağlı bulunduğu birime (or1200_pic, en son yapılan atama) bağlanmış ve böylece eklenen yeni birime bir kesme sinyali de konmuştur. Esasen bu işlem olmazsa olmaz bir işlem değil ekstra bir özelliktir. Örneğin bu platfordaki modüllerden bazılarının kesme sinyali yoktur (örneğin GPIO).

Donanım tarafında yapılaması gereken işlemler bitirildikten sonra yazılım kısmına geçilip burada da bir örnek uygulama yazılarak hem yeni eklenen modülü test etmek hem de kesme sinyalinin ve kesme fonksiyonlarının nasıl kullanılacağı anlaşılabilir.

C kodu ile yapılan bir uygulama örneği EK A.2'de verilmiştir. Bu kod derlenmeden önce board.h dosyası ile aynı klasörde olmalı ve board.h dosyası içersinde daha önce orpsoc-params.v de belirlenmiş olan adres tanımlanmalıdır. Bu dosya açılıp içerisine;

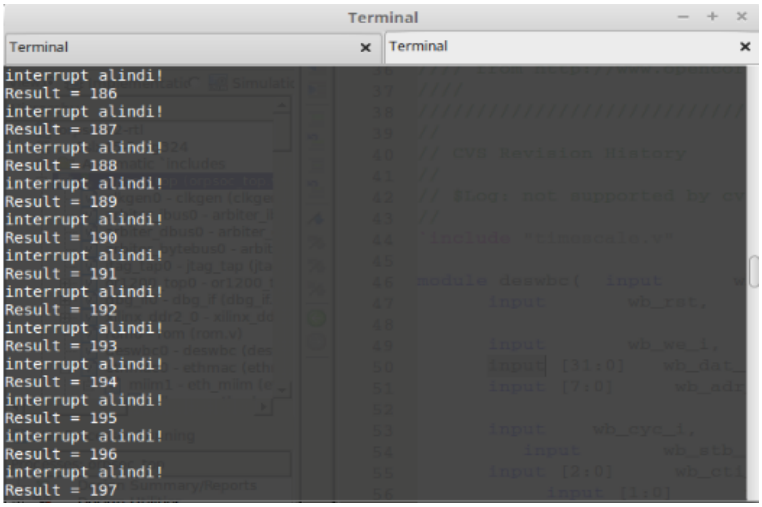
```
#define MYCOUNTER_0_BASE      0xa2000000
```

ifadesi eklenmelidir. Bu yapılmazsa kod geçersiz sayılır ve tabii derlenmesi de mümkün olmaz. EK A.2 ' de verilen kod incelendiğinde yazma işleminin nasıl yapıldığı aslında kolayca anlaşılacaktır. İşlem daha önce yapılan ledlerin testindeki ile oldukça benzerdir. Fakat dikkat çekici olan kesme fonksiyonu ve kullanımıdır. Kesme ile ilgili bu fonksiyon ve buna benzer olarak zamanlayıcı (timer), istisna (exception) gibi önemli ve sık kullanılan fomksiyonlar OpenRıSC ve dolayısıyla or1k-elf- derleyici takımı için newlib kütüphanesine dahil edilmiştir. Bunlar hakkında bilgi web sayfasından alınabilir [16].

Genel olarak fonksiyon şöyle özetlenebilir: OR1200 işlemcisinde latch level (seviyeye göre) kesme yöntemi kullanılmıştır. Bir diğer kesme yöntem ise triger level (yükseliş anına göre) yöntemidir. Bunların farkı ise ilk yöntemde kesme sinyali yükselip lojik 1 olduktan sonra ve 1 kaldığı sürece kesme uygulanır. İkinci yöntemde ise kesme

sinyali yükseldiği anda 1 saat işareti müddetince kesme uygulanır. Bu yöntemlerin ikisi de Bölüm 5 'te anlatılan mor1kx işlemcisinde gerçeklenmiştir. Böylece kullanıcı tarafından istenilen şekilde seçilebilmektedir. EK A.2'de verilen kodda ise programın başında tüm kesmeler aktif edilmiş ve daha sonra altı numaralı kesme de aktif edilmiştir. Bu numara bu çalışmada eklenen modülün kesme numarasıdır ve orpsoc_top.v dosyasında yapılan düzenlemelerde en son adımda belirlenmiştir. Daha sonra ise kesme fonksiyonu yazılmıştır. Program akışı içerisinde sayıcı modülden kesme işareti alındığı zaman program akışı koşulsuz olarak kesme fonksiyonuna gidecektir. Burada ilk yapılması gereken kesme sinyalini tekrar lojik sıfıra çekmektir. Aksi halde tekrar tekrar kesmeye girme durumu gerçekleşir. Daha sonraki satırda ise modüle ait kesme devre dışı bırakılmıştır. İlk yazılan or1k_mtspr fonksiyonuna ait bilgi daha önce verilen kaynaktan elde edilebilir [16]. Bu sayede oluşabilecek bir hata yüzünden tekrar kesme gelmesi engellenmiş olmaktadır. Ancak bu devre dışı bırakılma daha sonra yine kesme fonksiyonu içerisinde istenilenler yapıldıktan sonra tekrar aktif edilmelidir. Aksi halde tekrar kesme gelmeyecektir.

Daha sonraki adımlarda ise sayıcı modülüne başka değerler atanmış ve bu sayede modülün aralıksız bir şekilde çalışması ve işi bittiğinde gelip tekrar kesmeye girmesi ve tekrar çalışması şeklinde tekrar eden bir çalışma arzu edilmiştir. Kod bu şekilde yazılıp derlendikten sonra tıpkı diğer uygulamalarda olduğu gibi çeşitli dönüşümlerle doğrudan spi flash'a yazdırılabilir yada U-Boot ile transfer edilerek hızlı bir biçimde test edilebilir. Terminal ekranında Şekil 4.16'daki gibi bir sonuç elde edilmiştir.



Şekil 4.16: Sayıcı Modülünün Kesme Fonksiyonu İle Kullanımı.

4.10.2 DES Şifreleme Modülünün Eklenmesi

Bu bölümde DES (Data Encryption Standart) yöntemi ile 64-bit şifreleme ve şifre çözme işlemi yapan bir donanımının Orpsoc-v2 platformunda daha önce bahsedilen 32-bitlik veri hattına bağlanması ve yine önceki uygulamalara benzer olarak bir C kodu ile test edilmesi anlatılacaktır.

DES algoritması hakkında çok geniş kaynaklar bulunabilir. Oldukça önemli ve çok kullanılan bir şifreleme yöntemdir [26]. Yapılacak olan işlemler önceki bölümde anlatılan sayıcı modülüne benzer olmakla birlikte elbette farklılıklar içermektedir. Kullanılacak olan DES modülüne ait kodlar OpenCores web sayfasından temin edilebilir [27]. Ayrıca yine aynı sayfadan birçok açık kaynak kodlu modül alınıp kullanılabilir. Yüzlerce açık kaynak kodlu proje yapılmış veya yapılmaya devam etmektedir. Bizim kullandığımız kodlarda ise ayrıca wishbone arayüzü de yazılmıştır. Hemen anlaşılacağı gibi modülün wishbone arayüzü asıl şifre çözme işlemi yapılan DES algoritmalarının gerçeklendiği ana modüle bağlanmıştır.

Modülün Wishbone arayüzüne bakıldığında ise oldukça açık ve sade bir biçimde okuma ve yazmanın nasıl olacağı anlaşılacaktır. Modüle ait adresler 24'e kadar çıkmaktadır. İlk olarak sıfır adresi kontrol yazmacına ayrılmıştır ki bu yazmaca yazılan bilgilerle yeniden başlatma, yükleme veya şifre çözme gibi işlemler tercih edilmektedir. Sonrasında dördüncü adresten başlayarak giriş değeri ve anahtar için ayrılan toplam 20 adet 8-bitlik yazmaç ve sonuçların yazılacağı yine 8-bitlik 8 yazmaç kullanılmıştır. Dolayısıyla toplamda 28 adet yazmacı adreslemek için gerekli olan adres genişliği en az 5-bit olmalıdır. Uygulamayı etkileyen bir değişiklik olduğu için bu modülün arayüzünde verilen değer (8-bit adres genişliği) değiştirilmelidir. Sebebi daha sonraki işlemlerden anlaşılacaktır. Ayrıca modül arayüzüne bir de kesme sinyali eklemek iyi bir test imkanı verecektir. Modül arayüzü bu eklenti sebebiyle varolan haline göre biraz değiştirilmiştir. Bu yeni arayüz EK A.3'te verilmiştir.

Arayüz düzenlendikten sonra modülün bağlanmasına geçilebilir.

- Öncelikli olarak yapılması gereken modül arayüzü ve diğer verilog dosyalarının aynı isim altında bir klasörde tutulması ve bu klasörün /atlys/rtl/verilog/ dizinine konulmasıdır.

- Ardından orpsoc-defines.v içerisinde modül tanımlanmalıdır.

```
#define DESWBC0
```

- Daha sonra orpsoc-params.v içersinde modüle ait parametreler tanımlanmalıdır.

```
// DESWBC 0 params ----LATIF
  parameter wbs_d_deswbc0_data_width = 32;
  parameter deswbc0_wb_adr_width = 5;
  parameter deswbc0_wb_adr = 8'ha2;
    parameter dbus_arb_slave2_adr = deswbc0_wb_adr;
  parameter dbus_arb_wb_num_slaves = 6;
```

Bu aşamadan sonra üzerinde değişiklik yapılacak olan iki dosya kalmıştır. Özellikle arbiter_dbus.v modülünde yapılacak değişiklikler dikkat edilmesi gereken işlemlerdir. Zira bu dosyada yapılacak işlem sadece yeni modülü eklemek değil aynı zamanda köprü (bridge) modülünü bir sonraya ötelemektir. Yani eklenecek modül aslında bu hatta zaten bağlı olan bir birimin yerine geçecek ve o birimde bağlantı numaraları birer artırılarak en sona alınacaktır. Bu işlemin sebebi ise hiyerarşinin sağlanmasıdır. Bu yapıya göre 32-bitlik hatta aktif olan bir işlem yoksa köprü (bridge) seçilir ve bu köprü üzerinden 8-bitlik byte veri hattı (modüller 8-bitten fazla olabilir ancak okuma yazma 8-bit) hatta bağlı olan diğer modüller sisteme dahil edilir. Bu köprü sıralamada en sonda yer almaktadır. Dolayısıyla 32-bitlik hatta veri eklemek istediğinde bu köprünün yine sonda kalması için eklenilen modülü köprü bağlantılarının yerine alınmalı ve köprünün bağlantı numaraları birer artırılarak sonda kalması sağlanmalıdır. Örneğin arbiter_dbus.v dosyasında portların yazıldığı ilk düzenlemenin yapılacağı yerde şu şekilde bir işlem yapılacaktır;

```
// Slave three
   // Wishbone Slave interface
   wbs2_adr_i,
   wbs2_dat_i,
   wbs2_sel_i,
   wbs2_we_i,
```

```
    wbs2_cyc_i,
    wbs2_stb_i,
    wbs2_cti_i,
    wbs2_bte_i,
    wbs2_dat_o,
    wbs2_ack_o,
    wbs2_err_o,
    wbs2_rty_o,

    // Slave four
    // Wishbone Slave interface
    wbs3_adr_i,
    wbs3_dat_i,
    wbs3_sel_i,
    wbs3_we_i,
    wbs3_cyc_i,
    wbs3_stb_i,
    wbs3_cti_i,
    wbs3_bte_i,
    wbs3_dat_o,
    wbs3_ack_o,
    wbs3_err_o,
    wbs3_rty_o,
```

- Bundan sonra aynı dosyada bulunan aşağıdaki parametre bulunup 4 yapılmalıdır;

```
parameter wb_num_slaves = 4;
```

- Wishbone Slave Interface başlıklarının olduğu yerde yorum durumda bulunan bir grup sinyal aktif hale getirilir. Böylece yeni açılan bu yer köprü için kullanılır. Çünkü köprünün tanımları yeni eklenen modül için kullanılacaktır.

Aşağıdaki bölüm kodun 870 numaralı satırlarına denk gelmektedir. Bu bölümü bulup aşağıdaki hale getirmek gerekmektedir. Burada ifade edilen kısaca "veri hattına bağlı modüller seçilmediyse köprü seçilsin" demektir. Yeni eklenen modül de hiyerarşide yüksek bir noktaya alınmış olur.

```
// Slave selects
    assign wb_slave_sel[0] =
    wbm_adr_o[31:28] == slave0_adr | wbm_adr_o[31:28] == 4'hf;
    assign wb_slave_sel[1] =
    wbm_adr_o[`WB_ARB_ADDR_MATCH_SEL] == slave1_adr;
    assign wb_slave_sel[2] =
    wbm_adr_o[`WB_ARB_ADDR_MATCH_SEL] == slave2_adr;
  // Auto select last slave when others are not selected.
    assign wb_slave_sel[3] =
    !(wb_slave_sel_r[0] | wb_slave_sel_r[1] | wb_slave_sel_r[2]);
```

- Daha sonra 973 numaralı satıra denk gelen “Slave Inputs” kısmında yorum halinde olan bir grup sinyal benzer biçimde aktif hale getirilir.

- Daha sonra 1187 numuralı satırdan itibaren modülün sonuna kadar tanımlanmış olan her grup halindeki sinyale alt kısımlarında yorum halinde bırakılmış olan bir satır sinyal eklenir. Bu artırımların sebebi açıktır. Dolayısıyla daha önce yorum olan aşağıdaki sinyaller artık modülde aktif hale getirilmiştir.

```
wb_slave_sel_r[3] ? wbs_dat_o_mux_i[3] :
wbs_ack_o_mux_i[3]
wbs_err_o_mux_i[3] |
wbs_rty_o_mux_i[3]
```

Bu hali ile arbiter_dbus.v dosyası kaydedilip kapatılır. Son olarak bir de orpsoc_top.v dosyası düzenlenmek üzere açılır. Bu dosyada da tıpkı daha önceki sayıcı modülünde yapıldığı gibi benzer biçimde değiştirilir.

- Arbiter ana başlığının altına;

```
// deswbc0 wires  LATIF AKCAY
  wire [31:0]  wbs_d_deswbc0_adr_i;
  wire [wbs_d_deswbc0_data_width-1:0]   wbs_d_deswbc0_dat_i;
  wire [3:0]  wbs_d_deswbc0_sel_i;
  wire       wbs_d_deswbc0_we_i;
  wire       wbs_d_deswbc0_cyc_i;
  wire       wbs_d_deswbc0_stb_i;
  wire [2:0]     wbs_d_deswbc0_cti_i;
  wire [1:0]     wbs_d_deswbc0_bte_i;
  wire [wbs_d_deswbc0_data_width-1:0]   wbs_d_deswbc0_dat_o;
  wire    wbs_d_deswbc0_ack_o;
  wire    wbs_d_deswbc0_err_o;
  wire    wbs_d_deswbc0_rty_o;
```

- Wishbone Data Bus Arbiter başlığı altına aşağıdaki bölüm eklenir. Bu bölüm aslında köprü yerine açılmış olacaktır.

```
  .wbs3_adr_i       (wbm_b_d_adr_o),
  .wbs3_dat_i       (wbm_b_d_dat_o),
  .wbs3_sel_i       (wbm_b_d_sel_o),
  .wbs3_we_i        (wbm_b_d_we_o),
  .wbs3_cyc_i       (wbm_b_d_cyc_o),
  .wbs3_stb_i       (wbm_b_d_stb_o),
```

```
      .wbs3_cti_i      (wbm_b_d_cti_o),
    .wbs3_bte_i      (wbm_b_d_bte_o),
    .wbs3_dat_o      (wbm_b_d_dat_i),
    .wbs3_ack_o      (wbm_b_d_ack_i),
    .wbs3_err_o      (wbm_b_d_err_i),
    .wbs3_rty_o      (wbm_b_d_rty_i),
```

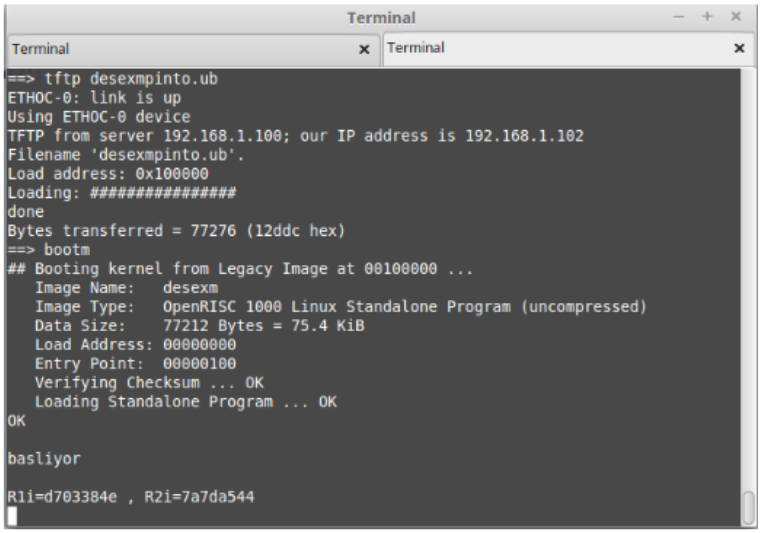
- Aşağıdaki başlığın bulunduğu yere şu satır eklenir;

```
defparam arbiter_dbus0.slave2_adr = dbus_arb_slave2_adr;   // DES Adress
```

- Daha sonra ise modül diğer birimlerin bağlandığı yerde herhangi bir araya bağlanır. Örneğin Generic Ram ve ETH0 arasına şu şekilde bağlanabilir;

```
`ifdef DESWBC0
   // DESWBC0 --- LATIF AKCAY
  wire deswbc0_irq;
   deswbc deswbc0
     (
      // Wishbone slave interface
      .wb_adr_i      (wbs_d_deswbc0_adr_i[deswbc0_wb_adr_width-1:2]),
      .wb_dat_i      (wbs_d_deswbc0_dat_i),
      .wb_we_i       (wbs_d_deswbc0_we_i),
      .wb_cyc_i      (wbs_d_deswbc0_cyc_i),
      .wb_stb_i      (wbs_d_deswbc0_stb_i),
      .wb_cti_i      (wbs_d_deswbc0_cti_i),
      .wb_bte_i      (wbs_d_deswbc0_bte_i),
      .wb_dat_o      (wbs_d_deswbc0_dat_o),
      .wb_ack_o      (wbs_d_deswbc0_ack_o),
      .wb_err_o      (wbs_d_deswbc0_err_o),
      .wb_rty_o      (wbs_d_deswbc0_rty_o),
      .int_o       (deswbc0_irq),
      .wb_clk      (wb_clk),
      .wb_rst      (wb_rst)
      );
   //////////////////////////////////////////////
`else // !`ifndef DESWBC0
   assign wbs_d_deswbc0_dat_o = 0;
   assign wbs_d_deswbc0_ack_o = 0;
   assign wbs_d_deswbc0_err_o = 0;
   assign wbs_d_deswbc0_rty_o = 0;
   //////////////////////////////////////////////
`endif // !`ifdef DESWBC0
```

Yukarıdaki wb_addr_i bağlantısına özellikle dikkat edilmelidir. Görüldüğü gibi bağlanan modülün adres girişinin son iki biti bağlanmamaktadır. Bunun sebebi 32-bitlik bir veri yazma hattında adresler dörder dörder artırılıp azaltılmaktadır. Çünkü



Şekil 4.17: DES Modülünün Eklenip Atlys Üzerinde Test Edilmesi.

bir defada dört yazmaca birden erişilmektedir. Dolayısıyla sadece dördün katları olan adresler geçerli olacağı için son iki bite bakılmaz. Bu sebeple son iki bit bağlanmaz ve bunun yerine üçüncü bitin değişimine bakılarak işlem yapılır.

- Son olarak tıpkı sayıcı modülünde olduğu gibi kesme sinyali modülün sonunda bulunan birimde boş bir kesme hattına atanır. (Örneğin 12 numara)

Böylece donanım tarafında işler bitirilmiş olur. Yazılım tarafında ise benzer bir şekilde kesme sinyalinin de kullanıldığı bir uygulama yapılabilir. Örnek bir kod EK A.4'te verilmiştir. Bu kod derlenir ve tıpkı diğer uygulamalarda olduğu gibi dosya formatları ve dönüşümleri yapılıp FPGA yapılandırma dosyası ile beraber veya ayrı ayrı spi flash'a yazdırılır ya da U-Boot üzerinden transfer edilebilir.

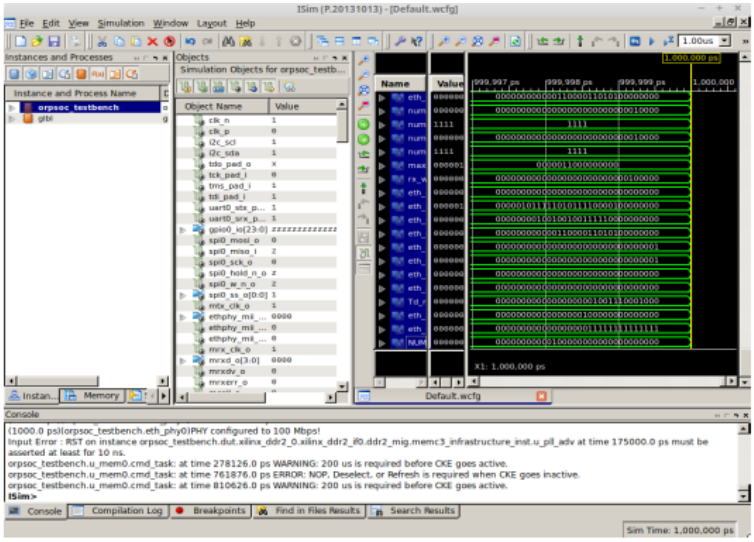
Uygulama çalıştığında Şekil 4.17 gibi bir ekran görülecektir. Bu uygulamada da kesme mantığı tıpkı sayıcı modülündeki gibi ayarlanmıştır. Modül bir şifreleme donanımı olmasına rağmen çok hızlı çalışmaktadır(400 saat işaretinde işlemi bitirmektedir). Modül tasarımı Javier Castillo tarafından yapılmıştır [27].

4.11 ORPSoC-v2 Platformunun ISIM Simülatör Üzerinde Denenmesi

ISIM Simülatör Xilinx ISE kurulumu ile beraber sunulan ve gelişmiş özelliklere sahip donanımın simülatörüdür. Kullanıcı tasarladığı donanımı bir test donanımı yazarak (girişleri belirleyerek) test edebilir ve böylece devrenin davranışı izlenebilir.

Bu çalışmada kullanılan Oprsoc-v2 platformunda hazır olarak sunulan bir test dosyası ile platform ISIM üzerinde test edilebilmektedir. Bunun için yapılması gereken test dosyasını projeye eklemek ve daha sonra gerekli olan include dosyalarını da tıpkı platformu sentezlemeden önce yapıldığı gibi Automatic Includes içerisine eklemektir.

Test dosyası /orpsocv2-den/bench/verilog kısmında bulunan orpsoc_testbench.v'dir. Bu dosya Xilinx projesi oluşturulup tüm modüller eklendikten sonra kaynak dosya olarak eklenir. Ardından da include dosyalarını eklemek için Xilinx aracından simülasyon kısmına geçilir. Gerekli olan tüm include dosyaları Orpsoc-v2 içerisinde mevcuttur. Bu dosyaların bulunduğu dizinler verilog include directory olarak Xilinx aracına tanıtılır. Daha sonra simülasyon başlatılır ve ISIM ekranının sol kısmında bulunan sinyallerden istenilenler izleme penceresine alınıp durumları gözlenebilir. Bu işlemler esnasında ISIM ekranı Şekil 4.18 gibidir.



Şekil 4.18: ORPSoC-v2 Platfromunun ISIM ile Denenmesi.

Ayrıca ISIM simülatör ile biraz kısıtlı bir yöntem olsa da yazılım-donamım ortak simülasyon yapmak mümkündür. Bunun için yazılan uygulamanın binary formattan .vmem formatına dönüştürülmesi gerekmektedir. Bu dönüşümü yapan program ise yine /orpsoc-v2/sw/utils/ kısmında bulunmaktadır. **bin2vmem** adlı bu program kullanılarak uygulama tıpkı RAM bellekte olması gerektiği gibi formatlanır. Böylece simülatör oluşan uygulama.vmem dosyasını sanki RAM bellekmiş gibi algılayacaktır. Testbench içerisinde adlandırma sram.vmem olarak yapıldığı için denenilecek uygulamaya da isim olarak sram.vmem vermek gerekecektir. Üretilen

dosya ana proje klasörünün içerisine konulmalıdır (.bit dosyasının üretildiği yere). Bunun haricinde test-defines.v adlı bir dosya oluşturup bu dosyayı da include dosyası olarak tanıtmak gerekmektedir. Dosya içeriği ise aşağıdaki gibi olmalıdır.

```
`define RTL_SIM
`define SIMULATOR_ISIM
`define TEST_NAME_STRING "or1200-simple"
`define PROCESSOR_MONITOR_ENABLE_LOGS
`define PRELOAD_RAM
```

Ayrıca rom.v dosyasını açıp bootrom modülünün olduğu satırı yorum haline getirmek gerekecektir. Çünkü simülasyonda yapılacak olan uygulama doğrudan sram.vmem içeriğini RAM bellek olarak görmelidir. Oysa bootrom durumunda RAM içeriği spi flash içeriğine göre değişecektir. Eğer spi flash'tan başlanacak şekilde bir konfigürasyon test edilmek istenirse bu durumda RAM modeline ek olarak bir de flash modeli oluşturulmalıdır. Bu durumda bootrom satırı yorum yapılmaz. Ayrıca test-defines.v den preload RAM satırı silinir. Flash modeli olarak orpsoc-v2 içinde bulunan AT26DFxxx.v dosyası kullanılır. Flash içeriği olarak ise flash.in adında bir dosya oluşturulup içerisine uygulama bayt'lar halinde yazılmalıdır.

Uygulamanın UART'a "merhaba dünya" yazan bir kod olduğunu düşünelim. İşlemler tamamlanıp simülasyon başlatıldıktan sonra yaklaşık 3.5 milisaniye sonra ISIM konsolunda "merhaba dünya" görüntülenecektir. Ancak 3.5 milisaniye bile donanım simülatörü için çok uzun bir süredir. Sadece birkaç sinyal dahi izleme penceresinde alındığında bu aşamaya gelinmesi yaklaşık 45 dakika sürmektedir. Ancak yine de yazılım-donanım ortak simülasyonu olduğundan oldukça kullanışlı bir yöntemdir.

5. ORPSoC-v3

Bu bölümde OpenRisc 1000 ailesinden ve henüz çok yeni sayılabilecek bir işlemci çekirdeği olan mor1kx CPU ile birlikte sunulan ORPSoC-V3 (OpenRisc Reference Platform System-on-Chip) için yapılan çalışmalar anlatılmıştır.

5.1 mor1kx İşlemcisinin Getirdiği Yenilikler

İşlemci tıpkı selefi OR1200 işlemcisi gibi OpenRISC gönüllüleri tarafından geliştirilmiştir. İlk kez 2012 yılında yayınlanan kaynak kodlar bugün büyük oranda geliştirilmiştir ve bu geliştirme halen de sürdürülmektedir.

mor1kx işlemcisi OpenRisc 1000 komut kümesi mimarisinden ORBIS32 desteği sunan bir çekirdektir. İşlemci tasarlanırken ana parametre olarak "esneklik" göz önüne alınmıştır. Bununla birliktre geçmiş gerçeklemelere göre daha kaliteli bir kodlama tekniği de kullanılmıştır.

Farklı seviyelerde iletişim hattı (pipeline) sunabiliyor olması gerçeklemenin en çok öne çıkan taraflarından birisidir. Zira bu gerçekleme altı seviyeli bir adet ve iki seviyeli iki adet olmak üzere toplam üç farklı seçenek sunmaktadır. Bu sayede ihtiyaca yöenelik farklı gerçeklemeler yapma imkanı sunulmuştur. Böylece örneğin çok fazla hız talebi olmayan sistemlerde kullanıcılar iletişim hattı seviyesi daha düşük olan modeli tercih edip yer konusunda önemli derecede kazanç sağlayabilirler. İşlemci esneklik özelliği sunmasının yanı sıra bu esnekliği kullanmayı da oldukça kolay hale getirmiştir. İşlemciye ait birçok özellik verilog kodunda parametrelerin değiştirimesi ile seçilebilmektedir.İletişim hattı seviyesi dahi bu parametreler kullanılarak kolayca belirlenebilmektdir.

Cappuccino adı verilen ve altı seviyeli iletişim hattı sunan en gelişmiş versiyon hata ayıklama ünitesi, zamanlayıcı, kesme, geciktirme slotu, veri ön belleği, hafıza kontrol

birimi ve ORBIS32 desteği gibi birçok özelliği sunmaktadır. Ayrıca bu özelliklerden çoğu da esnek bir yapıda tasarlandığı için eklenip çıkarılabilir şekilde ayarlanmıştır.

Espresso ise iki seviyeli iletişim hattına sahip ve Cappuccino versiyonunun birçok özelliğini aynen barından diğer bir gerçeklemedir. Tıpkı Cappuccino gibi ORBIS32, zamanlayıcı, kesme gecikme slotu, hata ayıklama ünitesi sunulmuştur.

Pronto Espresso gerçeklemesi ise daha mütevazı fakat yine de en çok kullanılan birimleri barındıran ve yine iki seviyeli bir iletişim hattının sunulduğu bir seçenektir. Bu versiyonda hata ayıklama ünitesi, zamanlayıcı, kesme birimleri ve ORBIS32 desteği sunulmuştur.

İşlemci hakkında henüz ciddi bir dokümantasyon çalışması yapılmamıştır. Açık kaynak kodlu bir proje olması ve sürekli geliştirilmesi sebebiyle standart bir seviyeye çıkılmadan bu çalışmanın yapılamamış olması da normaldir. Henüz tamamlanmamış olmasına rağmen genel olarak işlemci hakkında bilgiler içeren küçük bir dokümana projenin depolama alanı olan ve birçok açık kaynak kodlu projenin de bulunduğu Github internet sayfasından erişilebilmektedir [28].

5.2 FuseSoc

FuseSoc donamım kodları ve donanım geliştirme araçları için geliştirilmiş olan bir paket yönetim programıdır [29]. Açık kaynak kodlu olarak sunulan bu program donanımların sentezlenmesi ve diğer aşamalardan geçirilmesi veya simülasyona yapılması gibi en çok ihtiyaç duyulan işlemler konusunda kullanıcılara ciddi kolaylıklar sağlamaktadır. Aslında ilk olarak ORPSoC-v3 projesi içerisinde tasarlanmış fakat sonradan daha geniş bir alanda da kullanılabileceği için projeden çıkarılarak mevcut isimlendirme ile sunulmuştur.

Bu program işleyiş olarak bir üst dosya içerisinde hangi modüllerin projede kullanılacağı gibi bilgileri tutmaktadır. Bu dosya <platform>system olarak adlandırılmıştır.

ORPSoC-v3 platformu için bu çalışmada FuseSoc programından faydalanılmıştır. Programın kurulumu ise oldukça kolaydır. Bunun için ilk önce yapılması gereken proje dosyalarının indirilmesidir.

```
git clone https://github.com/olofk/fusesoc.git
```

Daha sonra ise ORPSoC-v3 dosyalarının bulunduğu orpsoc-cores klasörü indirilmedir. Bu klasör aşağıdaki gibi indirileblir.

```
git clone htpps://github.com/openrisc/orpsoc-cores.git
```

Klasör bu bölümde bahse konu olan işlemci kodlarını ve diğer çevresel birimlere ait olan tüm donanım kodlarını içermektedir. Ayrı ayrı klasörlenmiş olan proje bu özelliği ile karmaşa yaşanmasına da engel olur.

Bu iki klasör indirildikten sonra FuseSoc kurulumu yapılmalıdır. Bunun için terminalden FuseSoc klasörüne girilirek aşağıdaki işlemler yapılır.

```
autoreconf -i
./configure
make
make install
```

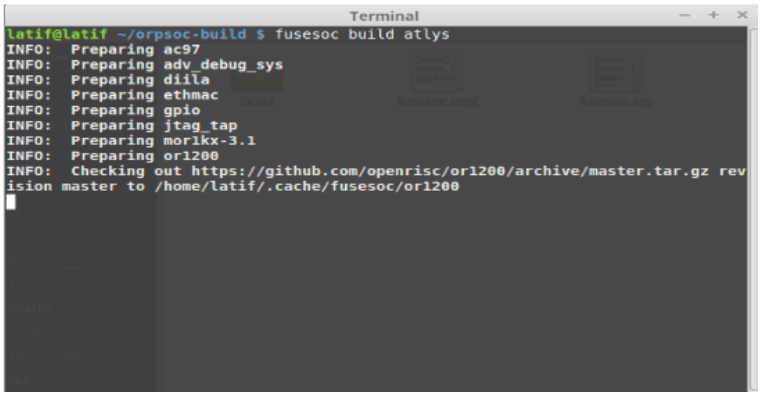
Bu işlemlerden sonra Linux işletim sisteminin Ev dizininde orpsoc-build adında yeni bir klasör açılır. Ardından bu klasörün içerisine bir metin dosyası açılarak fusesoc.conf olarak adlandırılır. Bu dosya açılarak içersine aşağıdaki metin aynen yazılır ve dosya kaydedilip kapatılır.

```
[main]
cores_root    = ../orpsoc-cores/cores
systems_root = ../orpsoc-cores/systems
```

Böylece FuseSoc kurulumu tamamlanmış olur. Bu andan itibaren terminal açıp aşağıdaki kodlar yazılarak FuseSoc ve yetenekleri hakkında daha geniş bilgi edinilebilir.

```
fusesoc --help
```

FuseSoc donanım kodları için sentez ve diğer işlemleri Xilinx ve Altera firmalarının araçlarını kullanarak ve aynı zamanda yine donanım kodları için simülasyon işlemlerini Icarus verilog, Verilator, Modelsim gibi programları kullanarak gerçekleştirebilmektedir.



Şekil 5.1: FuseSoc Çalışma Ekranı

FuseSoc ORPSoC-v2 için hazırlanan yardımcı yazılımlardan farklı olarak Xilinx proje dosyası oluşturmaktadır. Böylece FuseSoc işlemlerinden sonra eğer istenirse Xilinx ISE aracıyla proje açılabilir ve istenilen işlemler uygulanabilir.

FuseSoc ile bu çalışmada kullanılan Atlys geliştirme kartı için hazırlanmış olan mor1kx işlemcili ve diğer çevresel birimleri içeren platformdan FPGA yapılandırma dosyasını elde etmek için terminalden sadece aşağıdaki kodu yazmak yeterli olacaktır.

```
cd orpsoc-build
fusesoc build atlys
```

Bu işlemin ardından FuseSoc çalışmaya başlar ve ilk olarak dosyaların son versiyonlarını internete bağlanarak kontrol eder. Ardından Xilinx araçlarını kullanrak sentezleme ve diğer işlemleri yapmaya başlar. Böylece kullanıcı tek bir komut ile tüm platformu bir yapılandırma dosyasına sığacak şekilde üretmiş olur. Bu işlem ciddi bir kolaylık sağlamaktadır.

FuseSoc çalışmaya başladıktan sonra Şekil 5.1'daki gibi bir terminal ekranı görülecektir.

İşlemlerin tamamlanmasının ardından FPGA yapılandırma dosyası orpsoc-build klasörü içerisinde build klasörü ve onunda içindeki build-ise klasöründe bulunabilir.

Donanım kodları üzerinde önceki bölümde yapılan düzenlemeler yapılarak istenildiği gibi yapılandırma dosyası üretmek mümkündür. Bunun için daha önce indirilen orpsoc-cores klasörüne girilip burada bulunan donanım kodlarından arzu edilenler bulunup açılır ve gerekli düzenlemeler yapılıp kaydedilir. (örneğin bootrom kodunun

aktif olması için rom.v dosyasında yapılması gereken düzenleme yapılır). Ardından da orpsoc-build klasörüne gelinip FuseSoc çalıştırılır. Böylece istenilen formatta yapılandırma dosyası üretilmiş olur. Bu dosya ile FPGA'nın programlanması için yine impact programı kullanılmalıdır.

Geçen bölümde yapılan uygulama çalışmaları bu bölümde de mor1kx için tekrar edilebilir. Dolayısıyla bunları yeniden anlatmak gerekli görülmemiştir. Çünkü yapılandırma dosyası üretildikten sonra yapılan işlemler de herhangi bir farklılık yoktur. Ancak donnımdan gelen farklılıklara dikkat edilmelidir. Örneğin GPIO modülü bu platformda biraz daha farklıdır. Bu platform üzerinde çalışmaya başlanmadan evvel donanım kodlarının dikkatli bir incelemeye tabi tutulması faydalı olacaktır.

Çok önemli bir yenilik Linux Uygulaması için söylenmelidir. Orpsoc-v2 platformunda yapılan uygulama da hatırlanacağı gibi Linux Kernel'in çalışması için data belleğin devre dışı edilmesi gerekli idi. Bu oldukça can sıkıcı bir durumdu. Sevindirici olan ise bu platformda ise böyle bir problem bulunmamasıdır. Böylece donanımdan hiç bir kırpma olmadan Linux Kernel'i kullanmak mümkün hale gelmiştir.

5.3 Ramtest Uygulaması

Ramtest uygulaması Atlys geliştirme kartı üzerinde bulunan ve ORPSoC-v2 ve ORPSoC-v3 platformlarında Wishbone arayüzü ile haberleşilen 128 MB DDR RAM'in sistemle uyum içinde ve düzgün bir biçimde çalışıp çalışmadığını anlamak amacıyla yazılmış bir makina kodudur. Bu çalışmada yer almasının özel bir sebebi de OpenRISC ailesine ait makina komutlarıyla yazılmış bir uygulama olmasıdır.

Ramtest uygulaması için yazılmış olan kodlar EK A.5'da paylaşılmıştır. Kodların teker teker anlamlarını ve işlevlerini anlamak için OpenRISC dökümanlarından faydalınılabilir [1]. Genel olarak yazılan algoritma şöyledir: Program ilk olarak GPIO birimini çıkış olarak belirler. Ardından led'lerin belirli bir düzenle yakılmasını sağlar. Konulmuş olan bir gecikme (delay) mekanizması ile bu halde bir süre beklenir. Daha sonra led'ler farklı bir düzende yakılır. Asıl test edilecek olan RAM ile ilgili kısımlar ise buradan itabaren başlar. RAM belleğin sıfır adresinden başlayarak belli

bir adresine kadar tüm yazmaçlarına o adreslerin sayısal değeri içerik olarak yazdırılır. (sıfır adresine sıfır, dört adresine 4... vs). Yazma aşaması bittikten sonra ise sıra bu değerleri geri okumaya gelir. Aynı şekilde sıfır adresinden başlayarak belli bir adrese kadar tüm yazmaçlar okunur ve biraz önce yazılan verilerle şimdi okunan verilerin aynı olup olmadığını karşılaştırır. Şayet bir adreste beklenen sonuç alınmadıysa o halde hata sayacı bir artırılır ve bu şekilde diğer sıradaki adreslere bakılır. İşlem sonunda hata sayısı led'lere yazılır. Eğer hiç hata yoksa led'ler biri başta biri sonda olmak üzere (ikilik düzende 10000001) yakılır ve böylece işlemin başarıyla bittiği ve Ram belleğin sistemle uyum içerisinde çalıştığı anlaşılır. Kodlarda bazı bölümlerde gereksiz görülebilecek ifadeler olabilir fakat bu kısımlar bu çalışma yapılırken farklı bir takım gözlemler yapılması için yazılmıştır.

Kodlar yazıldıktan sonrası derleme işlemi yapılır. Ancak bu kez kodumuz makina kodu formatında olduğu için derleme işlemi biraz farklı olacaktır. Kod aşağıdaki şekilde derlenebilir.

```
or1k-elf-cpp -P ramtest.S ramtest.asm
or1k-elf-as -o ramtest.elf ramtest.asm
or1k-elf-objcopy -O binary ramtest.elf ramtest.bin
```

İlk satırda işlem C dili formatından da bilinen ön işlem (pre-processor) aşamasıdır. Burada yorum veya tanımlama gibi işlemler yapılır ve çıkış olarak saf makina kodu üretilir (ramtest.asm). İkinci satırda bulunan araç ise saf makina kodunu çalıştırılabilir .elf formatına dönüştür. Ardında da kod ikilik formata çevrilmiştir.

Daha önceki uygulamalarda yapıldığı gibi ikilik halden ister U-Boot araçları kullanarak ya da istenirse spi flash'a doğrudan yazdırılarak bu uygulama denenebilir. Uygulamaları gerçekleştirmenin bir diğer yolu da uygulamayı işlemcinin donanım kodlarına bootrom.v olarak gömmektir. Bu uygulama çalıştırmak için oldukça farklı bir yoldur. Ancak tabi her kodu böyle gömmenin olanağı yoktur. Çünkü bazı uygulamaların makina kodları epeyce uzundur ve bootrom.v kodu olarak donanıma gömülmesi mümkün değildir.

İkilik formatta bilgi bulunan bir dosyadan OpenRISC komut setine göre verilog kodu üretecek olan bir program daha önce indirilmiş olan ORPSoC-v2 klasöründe /sw/utils içerisinde bin2vlogarray.c adındaki dosyada bulunmaktadır. Bu kod GCC ile derlenip

Şekil 5.2: Ramtest Uygulması Sonucu.

üretilen çalıştırıbilir dosya bin2binsizeword kodunda da yapıldığı gibi /user/local/bin veya otomatik bilinen bir yere konmalıdır. Bu program daha sonra şu şekilde kullanılır.

```
bin2vlogarray < bootrom.bin > bootrom.v
```

Bu uygulama ORPSoC-v2 ve ORPSoC-v3 platformu için yapılmıştır. Sonuç olarak hiçbir hata alınmamış ve işlem beklendiği gibi devam edip sonlanmıştır. En son durumda led'lerin durumu Şekil 5.2'daki gibidir ve bize hiç hata oluşmadığını söylemektedir.

6. SONUÇ

6.1 Sonuçlar ve Gelecek Çalışmalar

OpenRISC 1000 mimarisine sahip iki farklı işlemci çekirdeği üzerine kurulmuş olan iki farklı kırmıküstü sistem üzerinde yapılan bu çalışmada işlemci tasarımı ve donanım tasarımı konularında iyi bir tecrübe kazanılmıştır. Bahse konu olan kırmıküstü sistem platformlarında çalışabilecek uygulamalar ve bu uygulamaların derlenip çalışabilir hale getirilebilmesi için gereken derleyici ve dönüştürücü araçlar, üzerinde çalışılmaya başlanan ilk konulardır. OpenRISC için farklı C kütüphaneleri ile derlenmiş olan GNU derleyici takımı bu noktada projenin en büyük avantajlarından biridir. Böylece hem işletim sistemi olan hem de böyle bir sisteme sahip olmayan projelerde ihtiyca göre kullanılabilecek olan üç farklı derleyici ve dönüştürücü araç zinciri desteği sunulmuştur. Çok bilinen ve çok kullanılan bir derleyicinin açık kaynak kodlu bir işlemciye de destek verir hale getirilmiş olması da açık kaynak kodlu donanım tasarımı projelerinin daha da geliştiğini göstermektedir.

Açık kaynak kodlu proje geliştirme çalışmaları uzunca bir süredir sürdürmektedir. Öyle ki bu çalışmalar zaman zaman eşdeğeri olan diğer lisanslı çözümlere göre çok daha başarılı sonuçlar verebilmektedir [30]. Dolayısıyla ülkemizde faaliyet gösteren endüstriyel firmalarda veya akademik alanda çalışan mühendislerin bu projeleri takip etmesi gerekmektedir. Bu anlamda açık kaynak kodlu donanım projeleri içerisinde önemli bir çalışma olan OpenRISC projesi üzerinde çalışılmış ve elde edilen sonuçlar paylaşılarak konuya dikkat çekilmiştir.

Platformlar üzerinde yapılan işlemler ve uygulamaların nasıl yapıldığı adım adım anlatılmıştır. Böyle bir anlayışın tercih edilmiş olmasının sebebi ise bu dökümanın sadece prosedürel bir zorunluluk olarak değil, ülkemizde bu konulara ilgi duyan ve öğrenmek isteyenlere yönelik bir rehber ve uygulama kitapçığı şeklinde düşünülmüş olmasıdır. Zira bu dökümanı okuyan ve bu alanda azda olsa bilgi birikimi olan birisi

uygulamaları adım adım yapabilir ve böylece bilgi ve tecrübelerini önemli ölçüde artırabilir.

Üzerinde çalışılan ORPSoC-v2 ve ORPSoC-v3 platformlarının tasarım anlayışı olarak esnek ve kullanıcı tercihlerine göre kolayca adapte edilebilir bir yapıda olması endüstriyel alanda OpenRISC çözümlerine ciddi bir avantaj sağlamaktadır. Bu çalışmada da işlemcinin ve çevresel birimlerinin nasıl düzenlendiği ve kullanıcı tasarımı çevresel birimlerin platformlara nasıl eklenip kullanıldığı anlatılmıştır.

Bu çalışmalara ek olarak daha birçok uygulama yapılabilir ve platformlar daha ileri seviyelere doğru taşınabilir. Genel olarak bakılırsa sistemlerin sürekli geliştirilmesi, performans verilerinin iyileştirilmesi, daha geniş platformlara da destek sunabilir hale getirilmesi ileriye dönük yapılmak istenenlerdir. Bunlara ek olarak bu çalışmanın hemen ardından yapılması planlanan birçok faaliyet vardır. Bunlardan en önemlisi sistemin bağımlılığını iyice azaltmalaya yönelik olarak düşünülen açık kaynak kodlu bir RAM arayüzü çalışmasıdır. Mevcut durumda Xilinx aracının otomatik olarak ürettiği ancak içeriği bilinmeyen bir arayüzle kullanılan RAM bellek için uyarlanabilecek bir arayüz tasarlama veya bu şekilde yürütülen projeleri öğrenip destek olma düşüncesi gelecek planları arasındadır. Ayrıca bu paltformları Atlys gibi bir geliştirme kartında değil de kendi tasarımımız olacak bir kart üzerinde kullanmak için yapılması gereken çalışmalar planlanmaktadır. Tüm bunların yanı sıra bu platformların ASIC olarak tasarlanması yönünde yapılan çalışmalara katılmak ve bu çalışmaları kendi imkanlarımızla yapabilecek noktaya ulaşmak hedeflenmiştir.

KAYNAKLAR

[1] **Lampret, D.**, **Chen, C.M.**, **Mlinar, M.**, **Rydberg, J.**, **Ziv-Av, M.**, **Ziomkowski, C.**, **McGary, G.**, **Gardner, B.**, **Mathur, R. ve Bolado, M.**, 2007. Openrisc 1000 architecture manual, *Rev*, **1**, 15.

[2] **Lampret, D.**, 2001. OpenRISC 1200 IP core specification, *September June*.

[3] **Specification, O.W.**, 2010. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, *OpenCores. rev. B*, **4**.

[4] **Foster, M.J. ve Kung, H.**, 1980. The design of special-purpose VLSI chips, *Computer*, **(1)**, 26–40.

[5] **Tong, J.G.**, **Anderson, I.D. ve Khalid, M.A.**, 2006. Soft-core processors for embedded systems, Microelectronics, 2006. ICM'06. International Conference on, IEEE, s.170–173.

[6] **Hartenstein, R.W.**, 1987. Hardware description languages, North-Holland.

[7] **Brown, S.D.**, **Francis, R.J.**, **Rose, J. ve Vranesic, Z.G.**, 2012. Field-programmable gate arrays, cilt180, Springer Science & Business Media.

[8] **OpenCores**, or1k Toolchain Main Page, `http://opencores.org/or1k/Main_Page`, alındığı tarih: 24.11.2015.

[9] **Patterson, D.A.**, 1985. Reduced instruction set computers, *Communications of the ACM*, **28(1)**, 8–21.

[10] **Goodacre, J. ve Sloss, A.N.**, 2005. Parallelism and the ARM instruction set architecture, *Computer*, **38(7)**, 42–50.

[11] **Zheng, P. ve Ni, L.M.**, 2006. Spotlight: the rise of the smart phone, *Distributed systems online, IEEE*, **7(3)**.

[12] **Rodriguez-Andina, J.J.**, **Moure, M.J. ve Valdes, M.D.**, 2007. Features, design tools, and application domains of FPGAs, *Industrial Electronics, IEEE Transactions on*, **54(4)**, 1810–1823.

[13] **Xilinx, I.**, 2013. Design Suite 14: Release Notes, *Installation, and Licensing*.

[14] **Digilent, A.S.F.**, Development Board Documentation.

[15] **Stallman, R.**, 2001. Using and porting the GNU compiler collection, MIT Artificial Intelligence Laboratory, Citeseer.

[16] **Johnston, J. ve Fitzsimmons, T.**, 2011. The newlib homepage, *URL http://sourceware. org/newlib*.

[17] **Erik, A.**, 2005, uClibc: AC library for embedded Linux, Home Page, `http://www.uclibc.org/`.

[18] musl-libc: a new libc for linux, Home Page, `http://www.musl-libc.org/`.

[19] **OpenCores**, OpenRisc GNU Toolchain, `http://opencores.org/or1k/OpenRISC_GNU_tool_chain`, alındığı tarih: 24.11.2015.

[20] **Robles, S.**, 2006. THE RSA CRYPTOSYSTEM, Citeseer.

[21] **GitHub**, GitHub Help Webpage, `https://help.github.com/articles/generating-ssh-keys/`, alındığı tarih: 24.11.2015.

[22] **Xilinx**, Downloads, `http://www.xilinx.com/support/download.html`, alındığı tarih: 24.11.2015.

[23] **Das, U.**, 2002. Boot–the Universal Boot Loader, *Verfügbar unte r http://www. denx. de/wiki/U-Boot [23.09. 12]*.

[24] **OpenCores**, or1k Linux, `http://opencores.org/or1k/Linux`, alındığı tarih: 24.11.2015.

[25] **Quercia, V. ve O'Reilly, T.**, 1990. Volume Three: X Window System User's Guide, O'Reilly & Associates.

[26] **Coppersmith, D.**, 1994. The Data Encryption Standard (DES) and its strength against attacks, *IBM journal of research and development*, **38(3)**, 243–250.

[27] **Castillo J.**, OpenCores Project, SystemC DES, `http://opencores.org/project,systemcdes`, alındığı tarih: 24.11.2015.

[28] **Baxter J.**, mor1kx, `https://github.com/openrisc/mor1kx/blob/master/doc/mor1kx.asciidoc`, alındığı tarih: 24.11.2015.

[29] **Kindgren O.**, FuseSoc, `http://or1k.debian.net/tmp/opencores/opencores.org/or1k/ORPSoC.html#FuseSoC`, alındığı tarih: 24.11.2015.

[30] **Hars, A. ve Ou, S.**, 2001. Working for free? Motivations of participating in open source projects, System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on, IEEE, s.9–pp.

EKLER

EK A.1

```
`include "timescale.v"
module deswbc(
  input         wb_clk,
  input         wb_rst,
  input         wb_we_i,
  input [31:0]   wb_dat_i,
  input [7:0]    wb_adr_i,
  input       wb_cyc_i,
    input         wb_stb_i,
  input [2:0]    wb_cti_i,
  input [1:0]    wb_bte_i,
  output reg     wb_ack_o,
  output reg [31:0]       wb_dat_o,
  output        wb_err_o,
    output        wb_rty_o,
    output reg     int_o // interrupt
);
integer i = 0;
integer sum = 0;
reg start = 0;
reg done = 0;
always @(posedge wb_clk)
begin:COUNTER
  if (wb_rst) begin
    wb_dat_o <= 0;
    sum <= 0;
    start <= 1'b0;
    done <= 1'b0;
    i <= 0;
    int_o <= 1'b0;
  end
  else if (wb_we_i & wb_stb_i) begin // master writes
    sum <= wb_dat_i;
    start <= 1'b1;
    done <= 1'b0;
    int_o <= 1'b0;
  end
  else if (!wb_we_i & wb_stb_i & done) begin //master reads
    wb_dat_o <= sum;
    done <= 1'b0;
    int_o <= 1'b0;
   end
  if (start == 1'b1) begin  // operation
    if (i < 50) begin
      sum <= sum + 1;
      i <= i+1;
    end
    else if (i == 50) begin
      i <= 0;
      start <= 1'b0;
```

```
      int_o <= 1'b1;
      done <= 1'b1;
    end
  end //start==1
end
//ack generation
always @(posedge wb_clk)
 if (wb_rst)
   wb_ack_o <= 0;
 else if (wb_ack_o)
   wb_ack_o <= 0;
 else if (wb_stb_i & !wb_ack_o)
   wb_ack_o <= 1;
assign wb_err_o = 0;
assign wb_rty_o = 0;
endmodule
```

EK A.2

```
#include <stdio.h>
#include "board.h"
#include <or1k-support.h>
#include <or1k-sprs.h>
#define writeMYCOUNTER(addr,val)  (*(unsigned char*)(addr) = (val))
#define readMYCOUNTER(addr)      (*(unsigned char*)(addr))
unsigned int counter = 0;
const unsigned int hwirq = 6;

void myinterrupt ( void *abc) {
  unsigned int result;
or1k_mtspr(OR1K_SPR_PIC_PICSR_ADDR,
or1k_mfspr(OR1K_SPR_PIC_PICSR_ADDR) & ~(1U << hwirq));
  or1k_interrupt_disable(hwirq);
  printf("interrupt alindi!\n");
  result = readMYCOUNTER(MYCOUNTER_0_BASE);
  printf("Result = %u\n", result);
  fflush(stdout);
  writeMYCOUNTER(MYCOUNTER_0_BASE ,counter++);
  or1k_interrupt_enable(hwirq);
}
int main() {
  or1k_interrupts_enable();
  or1k_interrupt_handler_add(hwirq, myinterrupt, NULL);
  or1k_interrupt_enable(hwirq);
  printf("\nbasliyor\n");
  writeMYCOUNTER(MYCOUNTER_0_BASE ,50);
  while(1);
return 0;
}
```

EK A.3

```
`include "timescale.v"
module deswbc(wb_clk, wb_rst, wb_stb_i, wb_dat_o, wb_dat_i, wb_ack_o,
              wb_adr_i, wb_we_i, wb_cyc_i, wb_sel_i, int_o, wb_cti_i,
          wb_bte_i, wb_err_o, wb_rty_o);

input           wb_clk;
input           wb_rst;
input           wb_stb_i;
output [31:0]   wb_dat_o;
input  [31:0]   wb_dat_i;
output          wb_ack_o;
input  [4:0]    wb_adr_i; //onceki [7:0]   wb_adr_i
input           wb_we_i;
input           wb_cyc_i;
input  [3:0]    wb_sel_i;
input [2:0]     wb_cti_i;
input [1:0]     wb_bte_i;

output int_o;
output    wb_err_o;
output    wb_rty_o;

reg   [31:0]   wb_dat_o;
reg            wb_ack_o;
wire  [63:0]   data_i;
reg   [63:0]   data_o;
wire           ready_i;
reg   [63:0]   key_o;
reg   [31:0]   control_reg;
reg   [63:0]   cypher_data_reg;

des  des(.clk(wb_clk),
         .reset(~control_reg[0]),
     .load_i(control_reg[1]),
     .decrypt_i(control_reg[3]),
     .ready_o(ready_i),
     .data_o(data_i),
     .data_i(data_o),
     .key_i(key_o)
      );
always @(posedge wb_clk or posedge wb_rst)
begin
      if(wb_rst==1)
      begin
        wb_ack_o<=#1 0;
        wb_dat_o<=#1 0;
        control_reg <= #1 32'h61;
        cypher_data_reg <= #1 64'h0;
        key_o <= #1 64'h0;
        data_o <= #1 64'h0;
      end
      else
      begin
```

```
 control_reg[31:4]<= #1 28'h6;
 control_reg[0]<= 1'b0;

if(ready_i)
begin
 control_reg[2] <= #1 1'b1;
 cypher_data_reg <= #1 data_i;
end

if(wb_stb_i && wb_cyc_i && wb_we_i && ~wb_ack_o)
begin
  wb_ack_o<=#1 1;
  case(wb_adr_i)
       8'h0:
       begin
            //Writing control register
            control_reg[3:0]<= #1 wb_dat_i[3:0];
       end
       8'h4:
        begin
            data_o[63:32]<= #1 wb_dat_i;
       end
       8'h8:
       begin
            data_o[31:0]<= #1 wb_dat_i;
       end
       8'hC:
       begin
            key_o[63:32]<= #1 wb_dat_i;
       end
       8'h10:
       begin
            key_o[31:0]<= #1 wb_dat_i;
       end
  endcase
end
else if(wb_stb_i && wb_cyc_i && ~wb_we_i && ~wb_ack_o)
begin
    wb_ack_o<=#1 1;
    case(wb_adr_i)
      8'h0:
      begin
            wb_dat_o<= #1 control_reg;
            control_reg[2]<=1'b0;
      end
      8'h14:
      begin
            wb_dat_o<= #1 cypher_data_reg[63:32];
      end
      8'h18:
      begin
            wb_dat_o<= #1 cypher_data_reg[31:0];
      end
    endcaseTimes New Roman
end
else
```

```
        begin
            wb_ack_o<=#1 0;
            control_reg[1]<= #1 1'b0;
        end

    end
end

assign int_o = ready_i;
assign wb_err_o = 0;
assign wb_rty_o = 0;

endmodule
```

EK A.4

```
#include <stdio.h>
#include "board.h"
#include <or1k-support.h>
#include <or1k-sprs.h>

#define writeDATA32(addr,val)    (*(unsigned int*)(addr) = (val))
#define readDATA32(addr)     (*(unsigned int*)(addr))

#define writeDATA8(addr,val) (*(unsigned char*) (addr) = (val))
#define readDATA8(addr)       (*(unsigned char*) (addr))

const unsigned int hwirq = 12;

void myinterrupt ( void *abc) {
  unsigned int result_int1, result_int2;
or1k_mtspr(OR1K_SPR_PIC_PICSR_ADDR,
or1k_mfspr(OR1K_SPR_PIC_PICSR_ADDR) & ~(1U << hwirq));
  or1k_interrupt_disable(hwirq);
  result_int1 = readDATA32(DESWBC_0_BASE+0x14);
  result_int2 = readDATA32(DESWBC_0_BASE+0x18);
  printf("\nR1i=%u , R2i=%u\n", result_int1, result_int1);
  fflush(stdout);

  writeDATA32(DESWBC_0_BASE+0x4, 0xabcd);
  writeDATA32(DESWBC_0_BASE+0x8, 0x1234);
  writeDATA32(DESWBC_0_BASE+0xC, 0x1234);
  writeDATA32(DESWBC_0_BASE+0x10, 0x5678);
  writeDATA32(DESWBC_0_BASE, 0x2);
  or1k_interrupt_enable(hwirq);
}
int main()
{
  unsigned int result1, result2;
  or1k_interrupt_handler_add(hwirq, myinterrupt, NULL);
  or1k_interrupts_enable();
  or1k_interrupt_enable(hwirq);
  printf("\nbasliyor\n");
  fflush(stdout);
  writeDATA32(DESWBC_0_BASE+0x4, 0xabcd);
  writeDATA32(DESWBC_0_BASE+0x8, 0x1234);
  writeDATA32(DESWBC_0_BASE+0xC, 0x1234);
  writeDATA32(DESWBC_0_BASE+0x10, 0x5678);
  writeDATA32(DESWBC_0_BASE, 0x2); // load(start)
  result1 = readDATA32(DESWBC_0_BASE+0x14);
  result2 = readDATA32(DESWBC_0_BASE+0x18);
  printf("R1=%u , R2=%u\n", result1, result2);
  while(1)
  ;
return 0;
}
```

EK A.5

```
#include "board.h"

#define GPIO_BASE GPIO0_BASE

boot_init:
  l.movhi r0, 0

  l.ori r15, r0, 0xff
  l.movhi r16, hi(GPIO_BASE)
  l.sb 0x1(r16), r15     // set the direction reg to output!
  //l.sb 0x4(r16), r15    //for testing orpsoc-v2
  //l.sb 0x5(r16), r15    //for testing orpsoc-v2
  l.ori r14, r0, 0xaa
  l.sb 0x0(r16), r14      // write the value to the leds!

  l.and r22, r0, r0
  l.and r21, r0, r0
delay:
  l.addi r21, r21, 1
  l.sfeqi r21, 25000      // set the flag if r21 == 25000
  l.bnf delay             // if the flag is not set go to delay
  l.nop
  l.bf loop0             // if the flag set go to loop0
  l.nop

loop0:
  l.addi r22, r22, 1
  l.sfeqi r22, 2000     // set the flag if r21 == 2000
  l.and r21, r0, r0
  l.bnf delay
  l.nop

  l.ori r14, r0, 0x0f
  l.sb 0x0(r16), r14     // write the value to the leds!

  l.and r21, r0, r0


  l.and r19, r0, r0 // make r19 zero
  l.and r17, r0, r0      // make r17 zero
loop1:
  l.sw 0x0(r17), r17    // store value of r17 to adress pointed by r17
  l.addi r17, r17, 4    // increase content of r17 by 4
  l.sfeqi r17, 32000    // set flag if r17 == 10000
  l.bnf loop1            // if flag is not set go to loop1

  l.ori r14, r0, 0x03
  l.sb 0x0(r16), r14

  l.and r17, r0, r0      // make r17 zero
loop2:
  l.sfeqi r17, 32000     // set flag if r17 == 10000
  l.bf finish             // if flag is set go to bitti
  l.nop
```

```
  l.lws r18, 0x0(r17) // load content of adress pointed by r17 to r18
  l.sfeq r17, r18     // set flag if r17=r18
  l.addi r17, r17, 4 // increase r17 by 1
  l.bf loop2
  l.nop
  l.bnf error
  l.nop

error:
  l.addi r19, r19, 1    // increase r19 by 1
  l.j loop2
  l.nop
finish:
  l.sb 0x0(r16), r19  // // write the value to the leds!
  l.sfeqi r19, 0      //set flag if r19 == 0
  l.bf noerror      // go to noerror
  l.nop
  l.bnf ledset      // go to ledset
  l.nop

noerror:
  l.and r20, r0, r0    //make r20 zero
  l.ori r20, r0, 129    //store 120 in r20
  l.sb 0x0(r16), r20    //store byte value in r20 to r16

ledset:
  l.nop     // wait on this state
  l.j ledset
  l.nop
```

ÖZGEÇMİŞ

Ad Soyad: Latif AKÇAY

Doğum Yeri ve Tarihi: OLTU/ERZURUM 05.01.1990

Adres: İstanbul Teknik Üniversitesi EEF-1111

E-Posta: akcayl@itu.edu.tr

Lisans: Zonguldak Karaelmas Üniversitesi

Y. Lisans: İstanbul Teknik Üniversitesi