**KOCAELİ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

**BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**

**YÜKSEK LİSANS TEZİ**

# YENİDEN YAPILANDIRILABİLİR HESAPLAMA TABANLI ÇOK KULLANICILI İŞLETİM SİSTEMİ

**TANER GÜVEN**

**KOCAELİ 2017**

**KOCAELİ ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

**BİLGİSAYAR MÜHENDİSLİĞİ**

**ANABİLİM DALI**

**YÜKSEK LİSANS TEZİ**

# YENİDEN YAPILANDIRILABİLİR HESAPLAMA TABANLI ÇOK KULLANICILI İŞLETİM SİSTEMİ

**TANER GÜVEN**

**Yrd. Doç. Dr. Suhap ŞAHİN**
**Danışman, Kocaeli Üniversitesi** .....

**Doç. Dr. Sevinç İlhan OMURCA**
**Jüri Üyesi, Kocaeli Üniversitesi** .....

**Yrd. Doç. Dr. Şafak KAYIKÇI**
**Jüri Üyesi, Abant İzzet Baysal Üniversitesi** .....

**Tezin Savunulduğu Tarih: 21.06.2017**

## ÖNSÖZ VE TEŞEKKÜR

Tez çalışmamda yardımlarını esirgemeyen ve bana her konuda destek olan değerli hocam Yrd. Doç. Dr. Suhap ŞAHİN'e, Doç. Dr. Sevinç İLHAN OMURCA'ya, Öğr. Gör. Dr. Onur GÖK'e, Bilgisayar Mühendisliği Bölümündeki tüm hocalarıma, Gömülü Sistemler Araştırma Laboratuvarı üyelerine, aileme ve bana yardımcı olan herkese teşekkür ederim.

Mayıs – 2017 Taner GÜVEN

## İÇİNDEKİLER

## ŞEKİLLER DİZİNİ

## TABLOLAR DİZİNİ

## SİMGELER VE KISALTMALAR DİZİNİ

| | | |
|---|---|---|
| GB | : | Gigabyte |
| MB | : | Megabyte |
| MB/s | : | Megabyte / saniye |
| MHz | : | Megahertz |
| ms | : | milisaniye |

### Kısaltmalar

| | | |
|---|---|---|
| ASIC | : | Application-specific Integrated Circuit (Uygulamaya Özel Tümleşik Devre) |
| AXI | : | Advanced eXtensible Interface (Gelişmiş Genişletilebilir Arayüz) |
| AXI_ACP | : | AXI Accelerator Coherency Port (AXI Hızlandırıcı Tutarlılık Portu) |
| AXI_HP | : | AXI High Performance (Yüksek Performanslı AXI) |
| CFS | : | Completely Fair Scheduler (Tamamen Adil Zamanlayıcı) |
| CPU | : | Central Processing Unit (Merkezi İşlemci Birimi) |
| DDR | : | Double Data Rate (Çift Veri Hızı) |
| DMA | : | Direct Memory Access (Doğrudan Bellek Erişimi) |
| DSP | : | Digital Signal Processor (Sayısal İşaret İşleyici) |
| FIFO | : | First In First Out (İlk Giren İlk Çıkar) |
| FPGA | : | Field-Programmable Gate Array (Sahada Programlanabilir Kapı Dizileri) |
| GID | : | Group Identifier (Grup Tanımlayıcı) |
| GPU | : | Graphics Processing Unit (Grafik İşlemci Birimi) |
| G/Ç | : | Giriş/Çıkış |
| HDMI | : | High-Definition Multimedia Interface (Yüksek-Çözünürlüklü Çokluortam Arayüzü) |
| HWT | : | Hardware Thread (Donanım İş Parçacığı) |
| HWTS | : | Hardware Thread Server (Donanım İş Parçacığı Sunucusu) |
| L1 | : | Seviye 1 Önbellek |
| L2 | : | Seviye 2 Önbellek |
| LUT | : | LookUp Table (Arama Tablosu |
| MEMIF | : | ReconOS Memory Interface (ReconOS bellek arayüzü) |
| MM2S | : | Memory Mapped To Stream Module (Bellek Eşlemeli Veri – Akış Veri Dönüşüm Modülü) |
| MMU | : | Memory Management Unit (Bellek Yönetim Birimi) |
| IPC | : | Inter-Process Communication (Görevler Arası Haberleşme) |
| OSIF | : | ReconOS Operating System Interface (ReconOS İşletim Sistemi Arayüzü) |
| PA | : | Physical Address (Fiziksel Adres) |
| PID | : | Process Identifier (Görev Tanımlayıcı) |
| PL | : | Programmable Logic (Programlanabilir Mantık) |
| PS | : | Processing System (İşlemci Sistemi) |

| | | |
|---|---|---|
| POSIX | : | Portable Operating System Interface for Unix (Unix için Taşınabilir İşletim Sistemi Arayüzü) |
| RAM | : | Random Access Memory (Rastgele Erişimli Bellek) |
| ROS | : | Reconfigurable Operating System (Yeniden Yapılandırılabilir İşletim Sistemi) |
| SoC | : | System On A Chip (Sistem Çip Üzerinde) |
| S2MM | : | Stream To Memory Mapped Module (Akış Veri – Bellek Eşlemeli Veri Dönüşüm Modülü) |
| S2MEM | : | Stream To Memory Module (Akış Veri – Bellek İletişim Modülü) |
| SSH | : | Secure Shell (Güvenli Kabuk) |
| SWT | : | Software Thread (Yazılım İş Parçacığı) |
| UID | : | User Identifier (Kullanıcı Tanımlayıcı) |
| USB | : | Universal Serial Bus (Evrensel Seri Veriyolu) |
| VA | : | Virtual Address (Sanal Adres) |
| YYH | : | Yeniden Yapılandırılabilir Hesaplama |

## YENİDEN YAPILANDIRILABİLİR HESAPLAMA TABANLI ÇOK KULLANICILI İŞLETİM SİSTEMİ


## ÖZET

Günümüzde FPGA tabanlı donanım ve yazılımın birlikte kullanıldığı uygulamalar, yüksek performanslı hesaplama, görüntü işleme, otomasyon, otomotiv, haberleşme gibi birçok alanda sıklıkla kullanılmaktadır. FPGA uygulamaları için kullanılan standartlaşmış bir işletim sistemi olmadığından genellikle hem yazılım hem de donanım konusunda sıfırdan geliştirmeye başlanılması gerekmektedir. Bu ihtiyaç "Yeniden Yapılandırılabilir İşletim Sistemi" çalışmaları sayesinde kısmen karşılanmaktadır. Bu alandaki işletim sistemi çalışmaları gerçek zamanlı sistemler veya yüksek performanslı hesaplama alanlarına odaklanmıştır.

Bu çalışmanın amacı, FPGA tabanlı yeniden yapılandırılabilir hesaplamanın genel kullanım amaçlı bilgisayar olarak kullanımını sağlayan işletim sistemi mimarisi geliştirmektir.

Öncelikle gerekli altyapıyı sağlayabilecek yeniden yapılandırılabilir işletim sistemleri incelenmiş ve çalışmada ReconOS'un temel alınmasına karar verilmiştir. ReconOS'un çok kullanıcılı işletim sistemi olarak kullanılabilmesi için yapılması gereken değişiklikler tespit edilmiştir.

ReconOS'un bellek erişimi performansının, çalışmada kullanılan Zynq-7000 platformunun sağlayabileceği performansa göre çok düşük olduğu tespit edilmiştir. Bu problemin çözümü için yeni bir bellek arayüzü geliştirilmiş ve ReconOS'a entegre edilmiştir.

Çok kullanıcılı işletim sistemlerinde bulunması zorunlu olan bellek koruması, görev yönetimi, görev soyutlaması ve zaman paylaşımı problemleri üzerinde çalışılmıştır. Bu problemlere pratikte uygulanabilecek çözümler geliştirilerek Zynq-7000 platformu üzerinde gerçeklemesi yapılmıştır.

Geliştirilen işletim sisteminin testi için video oynatma, bellekte veri kopyalama, basit görüntü işleme gibi alanlarda kullanıcı uygulamaları geliştirilmiştir. Kullanıcı uygulamaları ile işletim sistemi Xilinx ZedBoard kiti üzerinde test edilmiştir.



**Anahtar Kelimeler:** Çok Kullanıcılı İşletim Sistemi, FPGA, Kişisel Bilgisayar İşletim Sistemi, Yeniden Yapılandırılabilir Hesaplama, Yeniden Yapılandırılabilir İşletim Sistemi.

**RECONFIGURABLE COMPUTING BASED MULTI-USER OPERATING SYSTEM**

## ABSTRACT

Today, FPGA-based Hardware/Software Co-design applications are frequently used in many areas such as high performance computing, video processing, automation, automotive, and communications. Since there is not any standardized operating system used for FPGA applications, it is usually required to start developing from scratch for both software and hardware. This need is met partly by the Reconfigurable Operating System studies. The studies in this area is focused on real-time systems and high performance computing.

The purpose of this study is to develop an operating system architecture which supports FPGA based reconfigurable computing in general purpose computer model.

Firstly, reconfigurable operating systems which capable of providing necessary infrastructure was analyzed and decided to using ReconOS as basis for study. Changes needed to using ReconOS as multi-user operating system were identified.

It was detected that the memory access performance provided by the ReconOS is quite lower compared to the memory performance offered by the Zynq-7000 architecture. A new memory interface has been developed and integrated to ReconOS to solve this problem.

Work was made on memory protection, process management, process abstraction and time sharing issues which mandatory in multi-user operating systems. Practical solutions have been developed for these issues and implemented on Zynq-7000 platform.

User applications such as data copying, simple image processing, video playback were developed for the operating system test. Operating system was tested on Xilinx ZedBoard Development Board.

**Keywords:** Multi-User Operating System, FPGA, Personal Computer Operating System, Reconfigurable Computing, Reconfigurable Operating System.

# GİRİŞ

Geleneksel hesaplamada algoritmaların yürütülmesi için iki temel yöntem bulunmaktadır [1]. Birinci yöntem uygulamaya özel entegre devre (ASIC) geliştirilerek işlemlerin donanımda yapılmasıdır. İkinci yöntem ise işlemlerin merkezi işlemci (CPU), dijital sinyal işlemcileri (DSP), grafik işlemci (GPU) gibi mikroişlemci yapıları kullanılarak yürütülmesidir. ASIC'ler yapılacak hesaplamaya özel olarak tasarlandığı için işlemleri çok hızlı ve az kaynak kullanarak yapabilmektedir. Fakat üretimden sonra değişiklik yapılamamaktadır. Mikroişlemci yönteminde ise hesaplama bellekten komutlar okunarak yapıldığı için, donanım değiştirilmeden bellekteki komutlar (program) değiştirilerek farklı hesaplamalar yapılabilmektedir. Mikroişlemciler kullanımda esneklik sağlamasına karşın performans olarak ASIC'lerin çok gerisinde.

Yeniden yapılandırılabilir hesaplama (YYH), üçüncü hesaplama yöntemi olarak hızla gelişmektedir. YYH'nın ortaya çıkması Field Programmable Gate Array (FPGA)'lerin alan, hız ve güç olarak gelişmesiyle ortaya çıkmıştır [2]. FPGA'lerle uygulamaya özel donanım tasarlanabilmesi sayesinde verimli olması ve yeniden programlanma özelliği, YYH'nın diğer hesaplama yöntemlerine iyi bir alternatif olarak ortaya çıkmasını sağlamaktadır.

Günümüzde FPGA tabanlı donanımın ve yazılımın birlikte kullanıldığı uygulamalar yüksek performanslı hesaplama, görüntü işleme, otomasyon, otomotiv, haberleşme gibi birçok alanda sıklıkla kullanılmaktadır [3]. Bu uygulamalarda kullanılan platformlar genellikle CPU ve FPGA'in birlikte bulunduğu System-on-Chip (SoC) mimarilerini kullanmaktadır. Xilinx Zynq-7000 gibi güncel mimarilerde genellikle CPU için standart yazılım geliştirme yöntemleri, FPGA için standart donanım geliştirme yöntemleri kullanılmaktadır.

Yaygın olarak kullanılan FPGA geliştirme yöntemleri, gelişen yazılım teknolojisine göre epeyce geride kalmıştır. FPGA programlamada, işletim sisteminin sağladığı süreçler arası haberleşme, bellek yönetimi, Giriş/Çıkış yönetimi gibi konularda henüz

standartlaşmış ve yaygın kullanılan bir yöntem bulunmamaktadır. Yeniden kullanılabilirlik ve soyutlama konusunda yeterli çözüm bulunmadığından FPGA ile yeni bir uygulama geliştirmek için genellikle sıfırdan başlanılması gerekmektedir. Bu açığın giderilmesi için Reconfigurable Operating System (ROS) alanında [4] çalışma yapılmaktadır.

ROS çalışmaları, yazılım ve donanımın ortak geliştirildiği projelerde tüm problemin yazılım soyutlama modeli ile çözülebilmesini amaçlamaktadır. ROS'lerde donanım modülleri (FPGA alanları) görev veya iş parçacığı olarak tanımlanmaktadır. Yazılıma benzer şekilde görevler arası haberleşme (IPC), yazılım - donanım arası haberleşme, Giriş/Çıkış yönetimi, zaman paylaşımı, bellek yönetimi gibi problemler bu alandaki güncel problemlerdir.

[5-8] çalışmalarında preemptive çoklu-görev yapılabilmesi için FPGA alanından içerik okuma ve geri yükleme üzerine çalışma yapılmıştır. Donanım modüllerinin yapısı FPGA mimarilerinde farklılık gösterdiği ve FPGA üreticileri tarafından standart bir içerik okuma ve geri yükleme yöntemi desteklenmediği için bu alandaki çözümler belirli FPGA modelleri ile sınırlıdır.

FPGA'de bir alan için oluşturulmuş uygulama (çalıştırılabilir dosya), diğer alanlarında doğrudan çalışamamaktadır. Bu problemin çözümü için [9] çalışmasında task relocation (görev taşıma) üzerinde çalışılmıştır. Geliştirilen yöntem belirli FPGA modellerinde çalışmaktadır.

Hthreads projesinde [10-13] FPGA modülleri Hardware Thread (donanım iş parçacığı) olarak tanımlanmış, FPGA modüllerinde POSIX pthreads programlama modeline benzer şekilde senkronizasyon ve zaman paylaşımı yönetimi yapılmıştır. Çalışmalar gerçek zamanlı işletim sistemi modeline göre yapılmış, düşük cevap zamanı için üzerinden çalışmıştır. Hthreads çalışmaları Xilinx Virtex-II pro üzerinde gerçekleştirilmiştir.

BORPH projesinde [14-17], bu alandaki diğer çalışmalardan farklı olarak donanım hızlandırıcılarının iş parçacığı yerine, Unix Görevi olarak modellenmiştir. Standart Linux'a yapılan eklentilerle FPGA kaynaklarına sağladığı çekirdek desteği ile donanım görevlerine yazılım görevlerindeki gibi dosya sistemi erişimi sağlamıştır.

Dosya sistemi desteği sayesinde donanım görevleri ve yazılım görevleri arasında Unix Pipe haberleşmesi kullanılmıştır. BORPH çalışmaları Xilinx Virtex-II Pro üzerinde gerçekleştirilmiştir.

R3TOS projesinde [18], gerçek-zamanlı sistemlere yönelik olarak performans ve hata izolasyonu üzerinde çalışılmıştır. Kendi kendine teşhis mekanizması ile arızalı kaynaklar tespit edilerek, görev taşıma yöntemi ile donanım görevleri dinamik olarak arızasız FPGA kaynaklarına taşınmaktadır. R3TOS çalışmalarında Xilinx Virtex-4 ile kullanılmıştır.

ReconOS projesinde [8,19-25], FPGA modüllerini donanım iş parçacığı (HWT) olarak değerlendirmiş, gömülü sistemlerde yazılımda kullanılan çok iş parçacıklı programlama modelinin HWT'lerde kullanılmasını sağlayan işletim sistemi desteği, geliştirme ortamı ve donanım detaylarından soyutlanmış kolay kullanılabilir programlama kütüphanesi geliştirilmiştir. [19] çalışmasında donanım ve yazılım iş parçacıkları arasında haberleşme ve senkronizasyon fonksiyonları geliştirilmiştir. [21] çalışmasında cooperative multitasking (işbirliği ile çoklu-görev) yöntemi kullanılmış, [24] çalışmasında preemptive çoklu-görev üzerinde çalışılmıştır. ReconOS, Xilinx Virtex-2 Pro, Virtex-4, Virtex-6 FPGA'leri ile ve Xilinx'in en güncel platformu olan Zynq-7000'i desteklemektedir. Ayrıca projenin kaynak kodları GNU GPLv2 lisansı ile özgür ve açık kaynak kodlu olarak [26,27] web sayfalarından yayınlanmaktadır.

Yeniden yapılandırılabilir işletim sistemi alanında yapılan çalışmalar gömülü sistemler, yüksek performanslı hesaplama, iletişim uygulamaları gibi alanlara odaklanmış olup kişisel bilgisayar modeli veya çok kullanıcılı kullanımı amaçlayan bir çalışma bulunmamaktadır.

Tez kapsamında yeniden yapılandırılabilir hesaplamanın genel kullanım amaçlı bilgisayar kişisel bilgisayar ve çok kullanıcılı sunucu şeklinde kullanımını sağlayan işletim sistemini modeli üzerinde çalışılmıştır. Genel kullanım amaçlı bilgisayar modelinde bulunması zorunlu olan bellek koruması, görevler arası soyutlama, görev yönetimi ve çok kullanıcı desteği problemleri üzerinde çalışılmıştır. Geliştirilen işletim sistemi Xilinx Zynq-7000 platformu üzerinde gerçeklenmiştir.

Bölüm 1'de çalışmanın amacı ve katkısı detaylı olarak açıklanmış, kullanılan donanım platformu ve temel alınan işletim sistemi ReconOS hakkında detaylı olarak bilgi verilmiştir. Bölüm 2'de bellek erişimi performansını arttırmak için geliştirilen bellek arayüzü açıklanmıştır. Bölüm 3'de geliştirilen işletim sistemi modeli ve görev yönetimi, bellek yönetimi, FPGA donanımında zaman paylaşımı konularında geliştirilen çözümler açıklanmıştır. Bölüm 4'de işletim sistemi modeline uygun olarak geliştirilen kullanıcı programları, test platformu ve uygulamaların performans verileri açıklanmıştır.

# 1. GENEL BİLGİLER

## 1.1. Tezin Katkısı

Görüntü işleme, işaret işleme, haberleşme uygulamaları gibi alanlarda yeniden yapılandırılabilir hesaplamayı destekleyen gömülü işletim sistemler kullanılabilmektedir. Genel kullanım amaçlı kişisel bilgisayar ve çok kullanıcılı sunucu bilgisayar modelinde yeniden yapılandırılabilir hesaplamayı destekleyen bir işletim sistemi çalışması bulunmamaktadır. Çalışmada yeniden yapılandırılabilir hesaplamanın genel kullanım amaçlı bilgisayar modelinde kullanılabilmesini sağlayan bir işletim sistemi modeli önerilmiş ve uygulanmıştır.

FPGA yönetimi için, gömülü sistem modelini destekleyen ReconOS işletim sistemi çalışmaya entegre edilmiştir. ReconOS'un bellek erişimi performansının kullanılan platformunun sağlayabileceği değerlere göre oldukça düşük olduğu ve çok kullanıcılı model için gerekli olan performansa ulaşamayacağı tespit edilmiştir. Bu problemin çözümü için yüksek performanslı bir bellek arayüzü (StreamIF) geliştirilmiş ve ReconOS'a entegre edilmiştir.

FPGA kaynaklarının birden fazla görev ve kullanıcı tarafından paylaşılabilmesini sağlamak amacıyla, FPGA kaynak yönetimini yapan "HWT Sunucu" isimli işletim sistemi servisi geliştirilmiştir. FPGA görevlerinin (HWT'lerin) yönetimi, HWT'lerde zaman paylaşımı, bellek yönetimi, bellek paylaşımı yöntemleri HWT Sunucusu içerisinde gerçeklenmiştir.

Geçmişte yapılan işletim sistemi çalışmaları [8,10-25] genel kullanım amaçlı bilgisayar modeline yönelik olmadığı için HWT'lerde bellek koruması ihtiyacı bulunmamaktadır. Tez çalışmasında, geçmişte yapılan çalışmalardan farklı olarak, genel kullanım amaçlı bilgisayar modelinde bulunması zorunlu olan bellek koruması ve görev soyutlama yöntemleri HWT'ler üzerinde uygulanmıştır. Bellek arayüzü içerisine HWT'e özel sayfa tablosu eklenerek HWT'lerde bellek sayfası paylaşımı sağlanmıştır.

Geçmişte yapılan çalışmalarda [5-8], FPGA alanlarında preemptive çoklu-görev konusunda sadece belirli FPGA modellerinde çalışabilen, platform bağımlı çözümler kullanılmaktadır. Çalışmada kullanılan FPGA platformda için içerik geri okuma yöntemi bulunmadığından dolayı donanımsal preemptive çoklu-görev çözümü geliştirilemektedir. Yapılan çalışmada, pratikte uygulanabilir bazı kısıtlar altında içerik geri okumayı gerektirmeyen bir preemptive çoklu-görev modeli geliştirilmiştir. Geliştirilen yöntem içerik geri okuma gerektirmediği için platformdan bağımsız olarak kullanılabilmektedir.

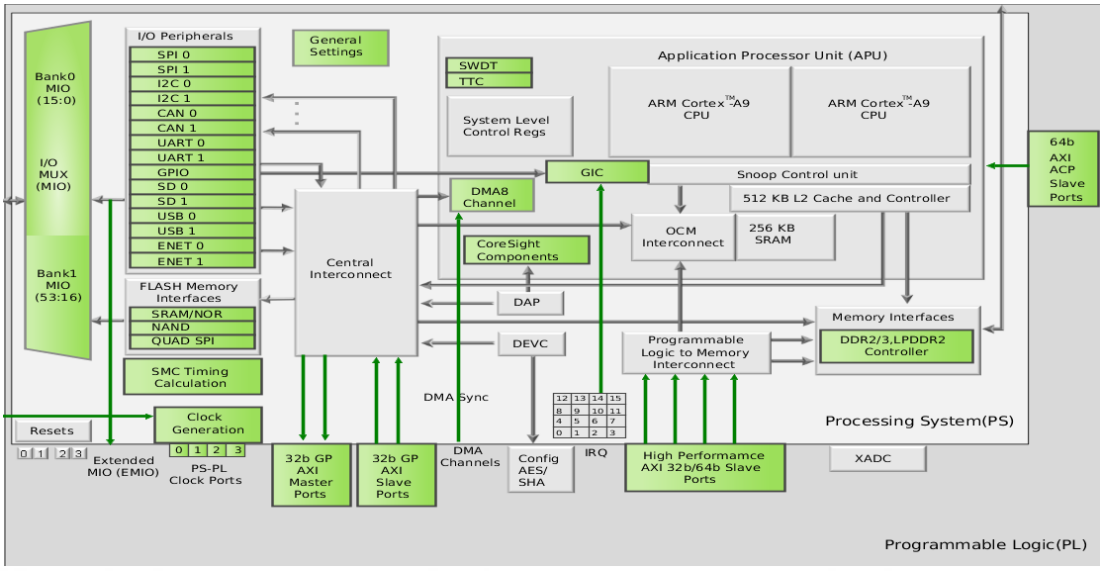
Test platformunda kişisel bilgisayar modelinde kullanımın için, önerilen bellek yüksek performanslı bellek arayüzü yöntemini kullanan HDMI destekli basit bir ekran kartı modülü geliştirilmiş ve ReconOS'a entegre edilmiştir. Xilinx ZedBoard üzerinde bir masaüstü GNU/Linux versiyonu çalıştırılmıştır. YYH destekli bir video oynatıcı uygulama üzerinden yapılan testler ile YYH'nın kişisel bilgisayarlarda kullanımının sağlayabileceği fayda gösterilmiştir.

### 1.2. Kullanılan Platform Ve İşletim Sistemi Altyapısı Hakkında Bilgiler

Geliştirilen çok kullanıcılı yeniden yapılandırılabilir işletim sistemi modelini açıklamak için öncelikle üzerinden geliştirme yapılan ZedBoard geliştirme kiti ve temel alınan ReconOS işletim sistemi açıklanmıştır.

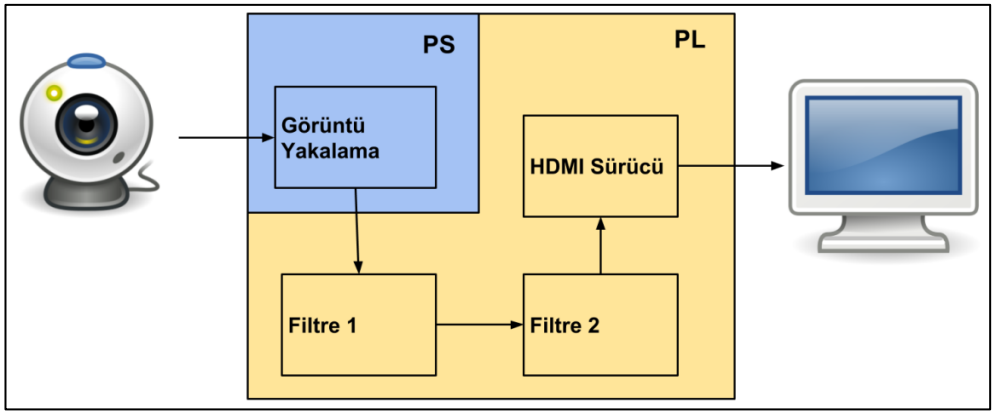
#### 1.2.1. ZedBoard geliştirme kiti

ZedBoard, üzerinde Zynq-7000 SoC, DDR3 RAM ile USB, Ethernet, SD Kart, HDMI gibi çevre birimlerini barındıran bir geliştirme kitidir. Zynq-7000 mimarisi, Processing System (PS) ve Programmable Logic (PL) olmak üzere iki kısımdan oluşmaktadır. PS, ARM işlemci ve DDR3, USB, Ethernet, SD Kart, UART gibi kontrolcüleri içerisinde barındırmaktadır. PL ise Artix-7 FPGA'den oluşmaktadır. Şekil 1.1'de Zynq-7000 mimarisinin içyapısı gösterilmiştir.




Şekil 1.1. Zynq-7000 mimarisinin iç yapısı [1]

ZedBoard ile geliştirilen uygulamalarda genellikle PS ve PL birlikte kullanılmaktadır. Şekil 1.2'de Zynq-7000'in görüntü işleme uygulamasındaki kullanımı gösterilmiştir. Kamera gibi giriş aygıtı sürücülerinin sadece FPGA ile geliştirilmesi zor olduğu için, görüntü yakalama PS'de Linux üzerinde çalışan yazılım ile gerçekleştirilmektedir[28]. Görüntü işleme yöntemleri FPGA'de gerçeklendiğinde, CPU'da gerçeklemeye göre daha hızlı ve daha az kaynak ile çalıştığı için PL'de çalıştırılmaktadır.




Şekil 1.2. Zynq-7000'in görüntü işleme uygulamasında kullanımı

ZedBoard ile yapılan uygulamalarda, PS için yapılan yazılım geliştirmede Linux ile yazılım geliştirme veya işletim sistemsiz yazılım geliştirme yöntemleri kullanılabilmektedir. PL üzerindeki FPGA uygulamalarında ise standart FPGA donanım geliştirme yöntemleri kullanılmaktadır. Yaygın olarak kullanılan yöntemde,

[1] Xilinx Vivado programından alınmış ekran görüntüsüdür

uygulama geliştiricisinin PS ile PL arasındaki haberleşmede veri aktarımı, kesmeler ve senkronizasyon problemleri için yazılım ve donanımda probleme özel çözüm üretmesi gerekmektedir. Yeniden yapılandırılabilir işletim sistemleri ile FPGA ve yazılım için standart bir arayüz sunarak uygulama geliştiricisine kolaylık sağlamayı amaçlanmaktadır.

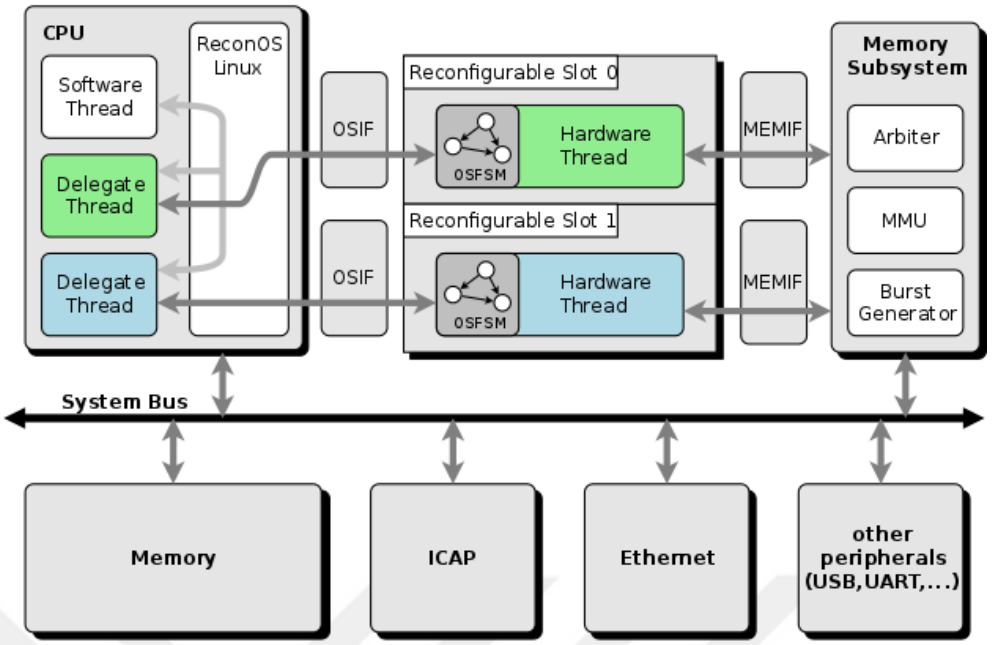
### 1.2.2. ReconOS

ReconOS [18-25] 2006 yılından bu yana aktif olarak geliştirilmekte olan açık kaynak kodlu bir yeniden yapılandırılabilir işletim sistemidir. İlk versiyonunda PowerPC üzerinde çalışan eCos çekirdeği kullanılmıştır. İlk versiyonu Xilinx Virtex-2 Pro ve Virtex-4 FPGA'lerini desteklemektedir. İkinci versiyonunda Linux çekirdeği desteği ile donanım modüllerine sanal adres uzayı desteği eklenmiştir. 2013'de yayınlanan 3. versiyonunda "Microblaze Soft Processor" mimarisi ile Virtex-6 FPGA'lerini desteklemektedir. 2014'de yayınlanan 3.1 versiyonu ile Xilinx'in en yeni platformu olan Zynq-7000 desteği eklenmiştir.

Tez çalışmasında kullanılan "Zynq-7000 ZedBoard Development Board" kiti ReconOS tarafından aktif olarak desteklendiği ve proje kaynak kodları ulaşılabilir olduğu için çalışmamız ReconOS işletim sistemi ile gerçekleştirilmiştir. Çalışmamızda karşılaştığımız problemleri çözmek için ReconOS üzerinde iyileştirme ve bazı mimarisel değişiklikler yapılmıştır.

#### 1.2.2.1. ReconOS mimarisi

ReconOS, FPGA üzerinde çalıştırılan donanım modüllerini Hardware Thread (HWT) olarak isimlendirmektedir. HWT'ler sistem tasarımı sırasında belirlenmiş olan yeniden yapılandırılabilir alanlar (Reconfigurable Slot) üzerine çalıştırılabilmektedir. HWT'ler, işletim sisteminin FPGA üzerindeki parçaları olan Operating System Interface (OSIF) ve Memory Interface (MEMIF) aracılığıyla Giriş/Çıkış işlemlerini gerçekleştirmektedir. Şekil 1.3'de ReconOS'un genel yapısı gösterilmiştir.




Şekil 1.3. ReconOS'un genel yapısı [23]

HWT'ler sistem çağrılarını OSIF aracılığıyla iletmektedir. Sistem çağrıları yazılım tarafında bulunan Delegate Thread (Temsilci İş Parçacığı) tarafından alınıp, işletim sistemine iletilmekte ve işlem sonuçları takip edilmektedir. Örneğin HWT tarafından mutex_lock işlemi şu adımlar ile gerçekleştirilmektedir:

1. HWT, OSIF üzerinden mutex_lock isteğinde bulunur ve beklemeye geçer
2. Linux çekirdeği kesme aracılığıyla isteği alır
3. Linux çekirdeği temsilci iş parçacığını uyandırır
4. Temsilci iş parçacığı işlemi normal yazılım olarak sistem çağrısı aracılığıyla gerçekleştirir
5. İşlem tamamlandıktan sonra sonuç OSIF aracılığıyla HWT'e iletilir
6. HWT uyandırılır

HWT çalışırken temsilci iş parçacığı bekleme durumunda bulunmaktadır. Temsilci iş parçacıkları, standart iş parçacıklarında olduğu gibi beklemeyi CPU zamanı kullanmadan gerçekleştirmektedir. HWT'lerde içerik değiştirme maliyeti yüksek olduğu için meşgul bekleme kullanılmaktadır.

HWT'ler bellek erişimini MEMIF aracılığıyla gerçekleştirmektedir. Bellek erişimleri ReconOS'un FPGA modülü olan Bellek Alt Sistemi (Arbiter, MMU, Burst

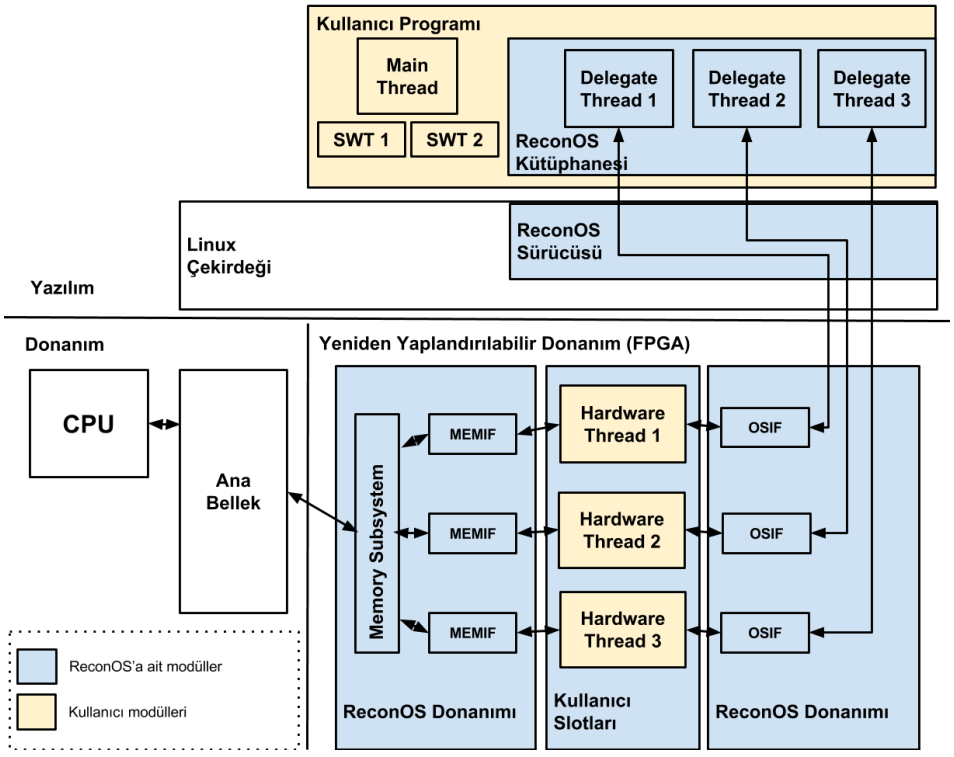
Generator) ile yönetilmektedir. Bellek Alt Sistemi 1.2.2.3. başlığı altında detaylı olarak açıklanmıştır.

HWT'lerin yüklendiği yeniden yapılandırılabilir alanlar, Xilinx'in Partial Reconfiguration özelliği ile kullanılarak oluşturulmaktadır. Standart dizaynda ZedBoard kiti ile en fazla 12 HWT alanı kullanılabilmektedir.

### 1.2.2.2. Kullanıcı programı modeli

ReconOS sisteminde çalıştırılacak kullanıcı programı, Linux üzerinde çalıştırılabilir program ve HWT alanlarına yüklenebilir FPGA Bitstreamleri olmak üzere iki parçadan oluşmaktadır. Şekil 1.4'te ReconOS'a ve kullanıcıya ait modüllerin şeması ile yazılım ve donanım katmanlarının ilişkisi gösterilmiştir.

Kullanıcı programı, içerisine ReconOS kütüphanesi eklenerek derlenmiş standart Linux çalıştırılabilir dosya formatındadır. Ana iş parçacığı, programın main fonksiyonundan çalıştırılan kodları kapsamaktadır. SWT'ler POSIX Threads [29] arayüzü ile oluşturulmuş standart Linux iş parçacıklarıdır. Kullanıcı ihtiyacına göre istediği kadar SWT oluşturabilmektedir.




Şekil 1.4. ReconOS modülleri, kullanıcı modülleri ve donanım ilişkisi

ReconOS kütüphanesi, sürücü ile haberleşme fonksiyonlarını ve temsilci iş parçacıklarını (delegate thread) içermektedir. Temsilci iş parçacıkları, kütüphanenin başlatıcı fonksiyonunda POSIX Threads kullanılarak HWT sayısı kadar, her biri bir HWT ile eşleşecek şekilde oluşturulmaktadır.

Ana iş parçacığı, SWT'ler ve temilci iş parçacıkları arasındaki iletişim üretici/tüketici problemi şeklinde semafor ve mutex yapılarını kullanarak gerçekleştirilmektedir. Temsilci iş parçacığı ile HWT'ler arasındaki iletişim ReconOS sürücüsü tarafından oluşturulan sanal dosya üzerinden yürütülmektedir. Temsilci iş parçacığı, gelen istekleri ReconOS sürücüsü tarafından oluşturulan sanal dosyadan okuyarak almakta, cevapları dosyaya yazarak iletmektedir.

ReconOS'da kullanılan mevcut görev yapısına göre, ReconOS sistemi (FPGA modülleri) sadece bir kullanıcı görevi tarafından kullanabilmektedir. Kullanıcı görevi kendi içerisinde iş parçacıklarına bölünerek çok iş parçacıklılığı sağlanmaktadır. HWT'ler ve SWT'ler iş parçacığı şeklinde çalıştığı için, göreve ait sanal adres uzayına kısıtlamasız olarak erişebilmektedir. Koruma olmadığından dolayı HWT veya SWT'lerin herhangi birinde yanış bir bellek erişimi olduğunda tüm görev çalışamaz duruma gelmektedir.

Görev yapısı gömülü sistemler açısından değerlendirildiğinde, gömülü sistemler genellikle sadece bir işi yapmaya yönelik geliştirildiği için, görevler arasında bellek koruması gibi yöntemlere ihtiyaç duyulmamaktadır. ReconOS sistemi ile kameradan gelen görüntüyü işleyip ağdan aktaran bir uygulama üzerinden örnek verirsek, uygulama üç parçaya bölünebilmektedir:

- SWT 1: Kameradan görüntü alma ve HWT 1'e aktarma işlemi
- HWT 1: Görüntü üzerinde filtre uygulama ve SWT 2'ye aktarma işlemi
- SWT 2: Görüntünün ağ üzerinden aktarılması işlemi

Örnekteki uygulamada iş parçacıklarından herhangi biri hata ile karşılaştığında işlemin tümü geçersiz olduğu için bellek korumasına ihtiyaç duyulmamaktadır.

Çok kullanıcılı bilgisayarlar, gömülü sistemlerden farklı olarak birçok farklı işlemi bir arada yürütebilmektedir. Örneğin bir kullanıcı video sıkıştırma işlemi yaparken, başka bir kullanıcı fizik simülasyonu hesaplaması yapabilmektedir. Video sıkıştırma

ve fizik simülasyonu işlemleri FPGA kullanılarak hızlandırılabilmektedir. Mevcut ReconOS görev yapısında bu işlemlerin tek görev olarak yürütülmesi gerekmektedir. Çok kullanıcılı işletim sistemlerinin veri güvenliği sağlaması gerektiği için çok görevli modelin kullanılması gerekmektedir. Bu problemin çözümü için tez çalışmasında geliştirilen yöntem Bölüm 3'de açıklamıştır.

#### 1.2.2.3. Bellek yönetimi

ReconOS'da bellek yönetimi SWT ve HWT olmak üzere iki kısma ayrılmaktadır. SWT'ler standart Linux iş parçacıkları olduğu için programa ait tüm sanal adres uzayına erişebilmektedir. Programın ve SWT'lerin bellek yönetimi Linux çekirdeği tarafından gerçekleştirilmektedir.

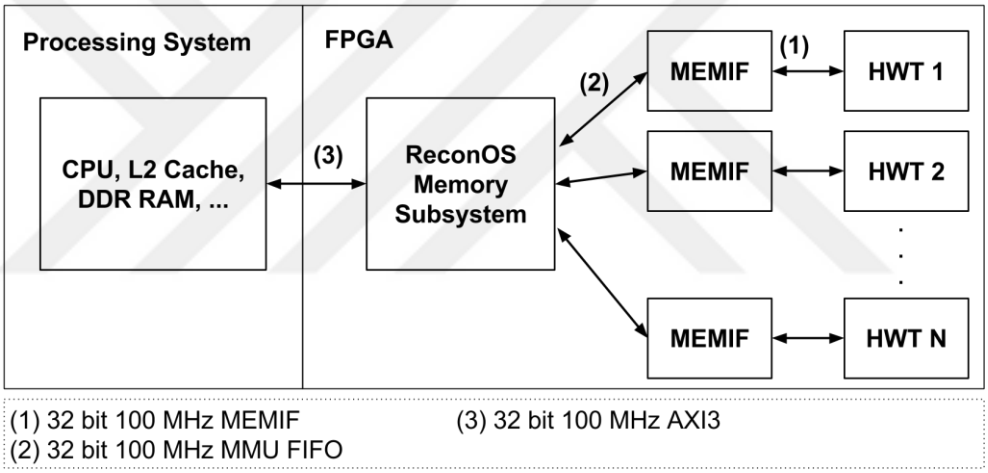
HWT'lerin bellek yönetimi ve belleğe erişimleri ReconOS Memory Subsystem (Bellek Alt Sistemi) tarafından yürütülmektedir. Bellek birimleri ve HWT arasındaki ilişki Şekil 1.5'de gösterilmektedir. Bellek işlemleri zaman paylaşımlı olarak sırayla gerçekleştirilmektedir. Bellek erişimi istekleri ve veri aktarımı MEMIF içerisindeki FIFO'lar aracılığıyla yürütülerek beklemeler azalmakta ve verimlilik arttırılmaktadır. Örneğin bir HWT bellekten okuma yaparken, diğer HWT'ler erişim isteklerini FIFO'ya kaydedebilmektedir.

Sanal adres - fiziksel adres dönüşümleri Bellek Yönetim Birimi (MMU) aracılığıyla gerçekleştirilmektedir. Bellek Alt Sistemi içerisinde bulunan MMU, CPU üzerinde bulunan standart MMU'lere benzer şekilde çalışmaktadır. Önbellekte bulunan adresler önbellek üzerinden dönüştürmektedir. Önbellekte bulunmayan adresler için ana bellekteki sayfa tablosuna erişilerek dönüşüm yapılmaktadır. Bellek Alt Sistemi'ndeki MMU, CPU'dakinden MMU'den farklı olarak, sayfa tablosuna erişimi çekirdek aracılığıyla gerçekleştirmektedir. Adres dönüşümün çalışma adımları şu şekildedir:

1. Adresi ön bellekte ara
2. Adres ön bellekte varsa adım 8'e git
3. İşletim sistemi aracılığıyla sayfa tablosundan L1 girdisini oku
4. L1 girdisi yoksa adım 9'a git
5. İşletim sistemi aracılığıyla sayfa tablosundan L2 girdisini oku
6. L2 girdisi yoksa adım 9'a git

7. Adres ve L2 girdisini ön belleğe kaydet
8. Bellek dönüşümünü gerçekleştir ve sonucu döndür
9. Page Fault hatası döndür

ReconOS'da HWT'lerin bellek erişim yöntemi Şekil 1.5'te gösterilmiştir. Bellek Alt Sistemi, AXI_ACP arayüzü üzerinden "Zynq Processing System"e bağlanmaktadır. AXI_ACP arayüzü, L2 önbellek üzerinden DDR RAM'e erişmektedir[30]. L2 önbellek kullanılarak yapılan bellek erişimi, CPU ile FPGA arasında küçük ve orta boyutlu verileri aktarımında gecikmenin azaltılması bakımından avantaj sağlamaktadır [3]. Fakat video işleme gibi yüksek bant genişliği gereken veya sürekli yeni veri aktarımı olan uygulamalarda cache trashing gibi problemlere sebep olabilmektedir.




Şekil 1.5. ReconOS'da HWT'lerin bellek erişim yöntemi

Bellek Alt Sistemi, half-duplex olarak 32 bit genişliğinde 100 MHz frekansında çalışmaktadır. Bu değerler ile teorik olarak okuma + yazma toplam 400 MB/s bant genişliğini desteklemektedir. Yaptığımız uygulama testlerinde bant genişliği 236 MB/s civarındadır. Bellek Alt Sistemi'nin sağladığı bant genişliği, Zynq-7000 mimarisinin sunduğu 4264 MB/s DDR RAM bant genişliğine göre oldukça düşüktür. Tablo 1.1'de Zynq-7000 platformu için FPGA ile PS iletişiminde kullanılabilen AXI_ACP ve AXI_HP arayüzleri ile DDR RAM'in özellikleri verilmiştir.

Tablo 1.1. Zynq-7000 mimarisinde bellek arayüzlerinin özellikleri [3]

| Özellik | AXI_ACP | AXI_HP | DDR |
|---|---|---|---|
| Veriyolu genişliği | 64 bit | 64 bit | 32 bit |
| En Yüksek Frekans | 150 MHz | 150 MHz | 1066 MHz |
| Okuma bant genişliği | 1200 MB/s | 1200 MB/s | 4264 MB/s |
| Yazma bant genişliği | 1200 MB/s | 1200 MB/s | 4264 MB/s |
| Okuma + yazma bant genişliği | 2400 MB/s | 2400 MB/s | 4264 MB/s |
| Arayüz sayısı | 1 | 4 | 1 |
| Toplam bant genişliği | 2400 MB/s | 9600 MB/s | 4264 MB/s |
| ReconOS Bellek Alt Sistemi'nin üst limiti* | 400 MB/s | kullanmıyor | 400 MB/s |

* 32 bit genişliğinde 100 MHz frekansta çalışan half-duplex aktarımın teorik olarak ulaşılabileceği en yüksek bant genişliği

ReconOS'un sunduğu Bellek Alt Sistemi, küçük ve orta boyutlu veriler için programcıya oldukça kolay ve hızlı bir geliştirme ortamı sağlamaktadır. Fakat yüksek bellek bant genişliği gerektiren problemlerde alternatif yöntemlere ihtiyaç duyulmaktadır.

Tez çalışması genel kullanım amaçlı çok kullanıcılı işletim sistemi modeline yönelik olduğu için, birden fazla HWT'in aynı anda büyük boyutlu verilere performans kaybı olmadan erişebilmesi gerekmektedir. Bu problemin çözümü için geliştirilen yüksek performanslı bellek arayüzü Bölüm 2'de açıklamıştır.

#### 1.2.2.4. Zaman paylaşımı

ReconOS görevler arası zaman paylaşımı yazılımda ve donanımda ayrı olarak gerçekleştirilmektedir. Yazılım içerisinde yer alan SWT ve temsilci iş parçacıkları pthread ile oluşturulduğu için standart Linux iş zamanlayıcı tarafından yönetilmektedir. Kullanılan versiyonda Linux standart olarak Completely Fair Scheduler (CFS) [31] algoritmasını kullanmaktadır.

HWT'ler arasındaki iş seçme yöntemi program geliştiricisine bırakılmıştır. HWT durdurulacağı zaman temsilci iş parçacığı program geliştiricisinin fonksiyonunu çalıştırmaktadır

HWT'ler arasındaki zaman paylaşımı işbirliği ile çoklu-görev yöntemi ile yapılmaktadır. Zynq-7000 platformunda preemptive çoklu-görev için gerekli olan

HWT'in durum bilgisinin donanımdan okunması desteklenmemektedir. Preemptive çoklu-görev problemi ve geliştirilen çözüm 3.4 başlığı altında açıklanmıştır.

ReconOS'daki işbirliği ile çoklu-görev, HWT'in "osif_set_yield" sistem çağrısı ile durdurulmaya uygun durumda olduğunu işletim sistemine bildirmesi şeklinde yapılmaktadır[21]. HWT durdurulmaya uygun olduğunu bildirdikten sonra alana sırada bekleyen HWT yüklenmekte ve çalıştırılmaktadır.

[24] çalışmasında ReconOS'da Virtex-6 modeli ile preemptive çoklu-görev gerçekleştirilmiş, saklama yapılarından Flip-Flop ve Block RAM'lerin içeriklerinin okunması ve geri yazılması başarılmıştır. Fakat LUT RAM ve DSP48 kaynakları desteklenmediğinden dolayı belirli HWT tasarımlarında çalışabilmektedir. Ayrıca yöntem tez çalışmasında kullanılan Zynq-7000 mimarisini desteklememektedir.

ReconOS'da kullanılan işbirliği ile çoklu-görev yöntemi HWT'lerin kontrolü ele aldıktan sonra işlemlerini tamamlayabildiği ve işletim sistemine durdurma isteği yaptığı durumlarda çalışabilmektedir. Uygulamaya özel geliştirilmiş gerçek zamanlı sistemlerde çalıştırılan tüm HWT'ler geliştirici tarafından eklendiği için güvenilirlik problemi bulunmamaktadır. Çok kullanıcılı bilgisayar modelinde HWT'ler güvenilmeyen kaynaktan gelebileceği için işbirliği ile çoklu-görev güvenilirlik problemi oluşturmaktadır. Bu problemin çözümü için tez çalışmasında geliştirilen yöntem Bölüm 3'de açıklamıştır.

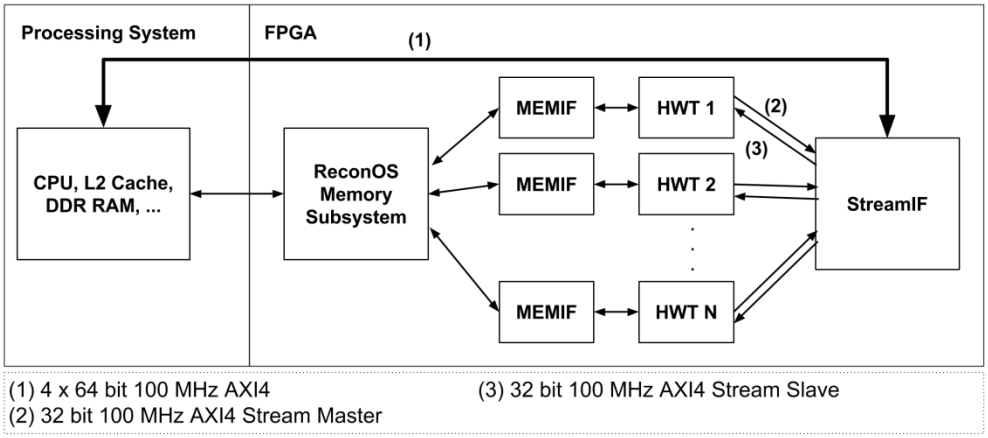
## 2. YÜKSEK PERFORMANSLI BELLEK ARAYÜZÜ

ReconOS'da kullanılan bellek erişim yöntemi, Zynq-7000 mimarisinin ana bellek bant genişliğini verimli kullanmamaktadır. ReconOS'un sağladığı bant genişliği çok kullanıcılı işletim sisteminde kullanım açısından yeterli performansa sahip değildir. Bu problemin çözümü için platformda ReconOS tarafından kullanılmayan portlar ile çalışan bir bellek arayüzü geliştirilmiştir. Geliştirilen bellek arayüzü proje içerisinde StreamIF şeklinde isimlendirilmiştir.

### 2.1. Bağlantı Şekli

StreamIF, akış şeklindeki ve yüksek boyutta veriler ile optimize çalışacak şekilde tasarlanmıştır. Zynq Processing System (PS) ile yapılan bağlantıda ana bellek bant genişliğini karşılayabilecek olan AXI_HP portu tercih edilmiştir. Dolayısıyla platformda bulunan 4 AXI_HP portunun da kullanıldığı bir tasarım ile ulaşılabilecek en yüksek bant genişliğinin elde edilmesi amaçlanmıştır.

Şekil 2.1'de StreamIF'in ReconOS'a eklenmesiyle oluşan bellek erişimi modeli gösterilmiştir. StreamIF, Şekil 1.6'daki ReconOS'un mevcut yapısını değiştirmeden ikinci bellek arayüzü olarak eklenmiştir. Bu modelle HWT'lerin ihtiyaca göre MEMIF'i ve StreamIF'i birlikte kullanabilmeleri amaçlanmıştır.




Şekil 2.1. StreamIF'in ReconOS'a eklenmesiyle oluşan bellek erişim modeli

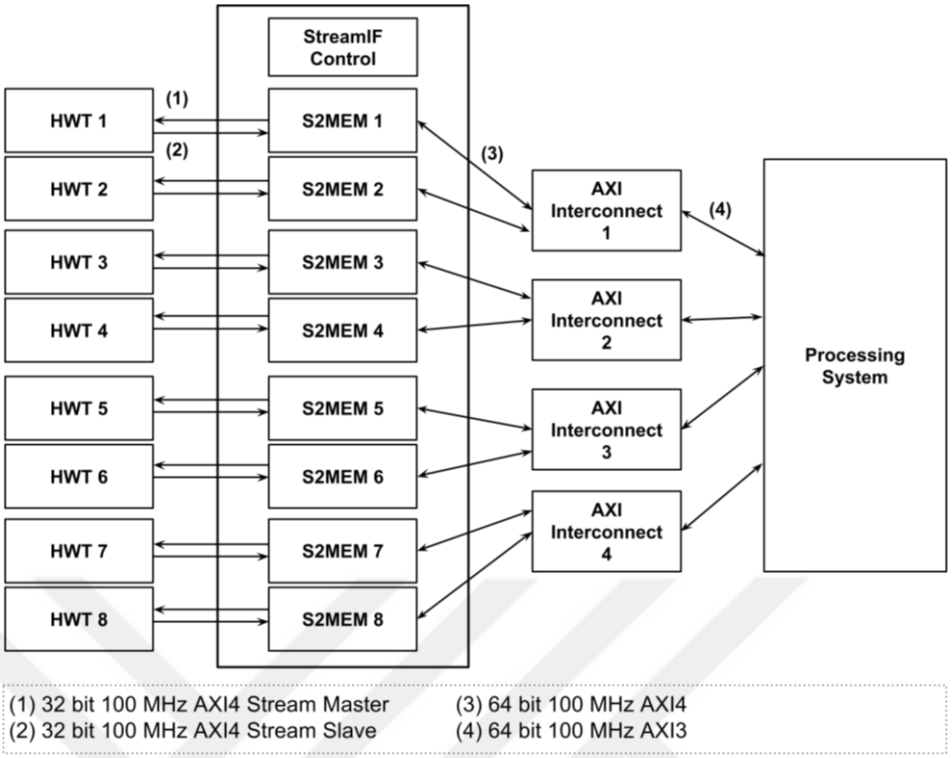
PS ile StreamIF bağlantısında 4 tane 64 bit genişliğinde 100 MHz frekansta çalışan veriyolu (1) kullanılmıştır. “AXI4 Memory Mapped” [32] protokolü kullanılan bu veriyolunda teorik olarak 3200 MB/s okuma ve 3200 MB/s yazma olmak üzere toplamda 6400 MB/s bant genişliği desteklenmektedir.

StreamIF ile HWT’ler arasında eş zamanlı olarak okuma ve yazma yapılabilmesi için iki veriyolu (2 ve 3) kullanılmıştır. Veri yolları 32 bit genişliğinde 100 MHz frekansta çalıştırılarak, her HWT için teorik olarak 400 MB/s okuma ve 400 MB/s yazma olmak üzere toplamda 800 MB/s bant genişliği desteklenmektedir.

StreamIF ile HWT'ler arasında “AXI4 Stream” [33] protokolü tercih edilmiştir. AXI Stream protokolünden, PS’e bağlantıda kullanılan “AXI4 Memory Mapped” protokolüne az kaynak kullanılarak dönüşüm yapılabilmektedir. Ayrıca “AXI Stream” protokolü, Xilinx High Level Synthesis [34] dahil olmak üzere birçok araç ve kütüphanenin sisteme entegre edilmesini kolaylaştırmaktadır.

### 2.2. StreamIF’in İçyapısı

StreamIF, HWT sayısına bağlı olarak bağlantı sayısı arttırılabilecek şekilde tasarlanmıştır. Şekil 2.2’de 8 HWT kullanılması durumunda StreamIF modülünün içyapısı ve bağlantıları gösterilmiştir. StreamIF, kontrol modülü ve HWT sayısı kadar S2MEM modülünden oluşmaktadır. S2MEM modülü bir HWT’in ana belleğe erişimini sağlamaktadır. StreamIF kontrol modülü, S2MEM modüllerinin yönetimini sağlamaktadır.

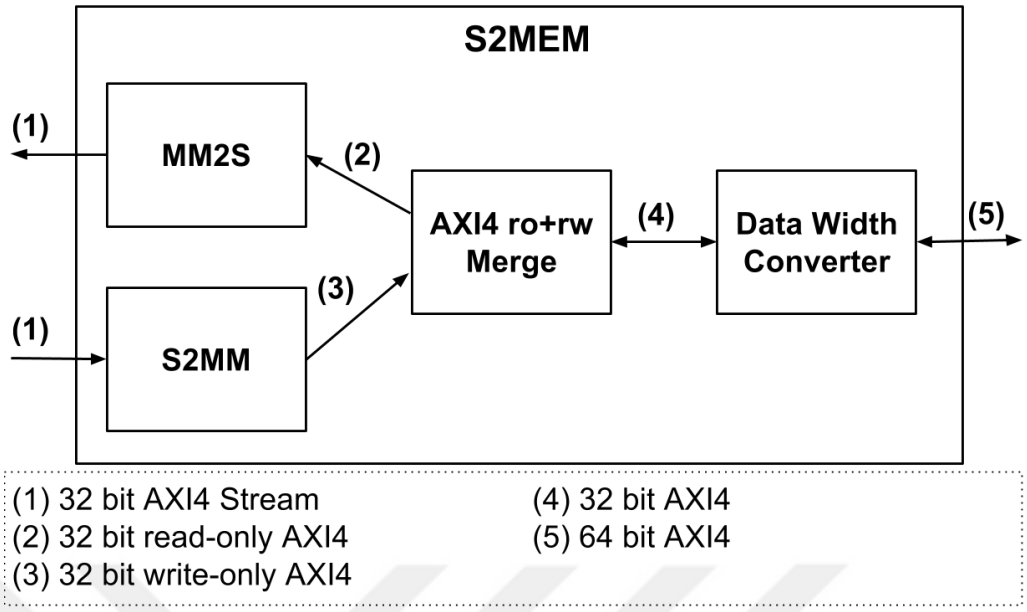



Şekil 2.2. 8 HWT kullanılması durumunda StreamIF modülünün içyapısı ve bağlantıları

S2MEM ile PS arasındaki bağlantı, 4 AXI_HP portu kullanıldığı için, iki S2MEM ile bir AXI_HP portu eşleşecek şekilde yapılmıştır. StreamIF ve PS arasındaki protokol dönüşümü ve çoklayıcı görevi için Xilinx'in geliştirme araçlarında (ISE ve Vivado) standart olarak bulunan Xilinx AXI Interconnect (AXI Ara Bağlantı) [35] modülü kullanılmıştır.

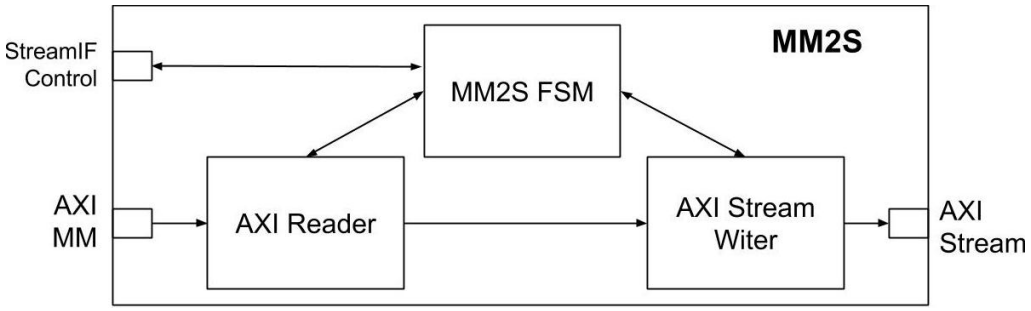
S2MEM ile AXI Ara Bağlantı arasında, AXI Ara Bağlantı'nın 64 bit modunda verimli çalıştırılabilmesi için 64 bitlik veriyolu kullanılmıştır. S2MEM ile HWT arasında 32 bitlik veriyolu kullanıldığından dolayı en az 2 HWT aktif olması durumunda AXI Ara Bağlantı tam verimle kullanılabilmektedir. Sistem genelinde tam performansa ulaşılabilmesi için en az 8 HWT'in aktif olması gerekmektedir.

S2MEM modülünün içyapısı Şekil 2.3'de gösterilmiştir. S2MEM, kontrol modülünden programlanabilen, MM2S ve S2MM isimli doğrudan bellek erişim (DMA) modüllerini içermektedir. MM2S veriyi bellekten okuyup HWT'e gönderme, S2MM ise HWT'den gelen veriyi belleğe yazma işlemini yapmaktadır.



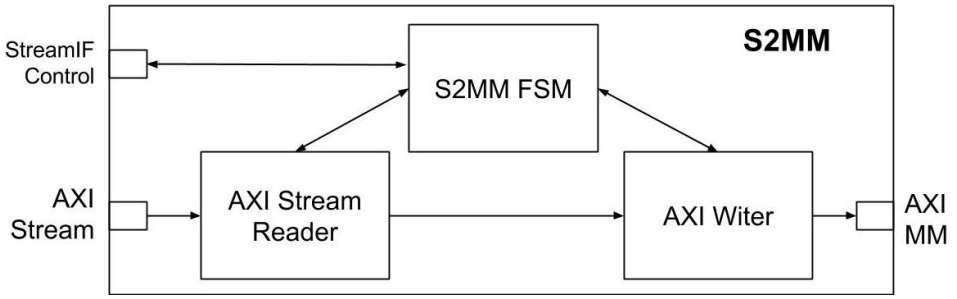

Şekil 2.3. S2MEM modülünün iç yapısı

MM2S modülü, veriyi AXI4-MM protokolü ile okuyup AXI4-Stream protokolü ile çıkış sağlamaktadır. Şekil 2.4'de MM2S modülünün içyapısı gösterilmiştir. "AXI4 Reader" ile "AXI4 Stream Writer" birbirlerine doğrudan bağlıdır. AXI4-MM portundan okunan veri aynı saat darbesinde (gecikmesiz olarak) AXI4 Stream portuna aktarılmaktadır. MM2S-FSM, okuma ve yazma modüllerinin koordine şekilde çalışmasını sağlamaktadır.




Şekil 2.4. MM2S modülünün iç yapısı

S2MM modülünün görevi, MM2S'in tam tersi, veriyi AXI4-Stream protokolü ile okuyup AXI4 protokolüne dönüştürmektedir. Şekil 2.5'de S2MM modülünün içyapısı gösterilmiştir. S2MM'in içyapısı MM2S ile benzer olup çalışma mantığı aynıdır.



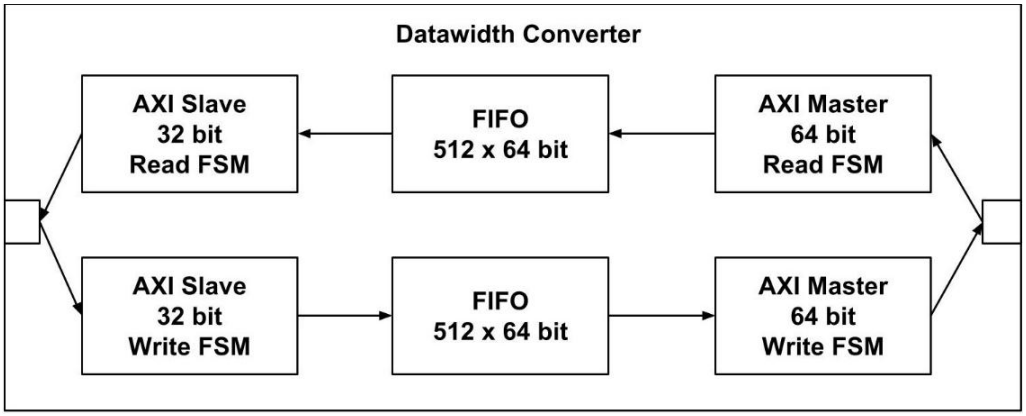

Şekil 2.5. S2MM modülünün iç yapısı

Şekil 2.3'deki Data Width Converter (Veri Genişliği Dönüştürücü) modülü, HWT ile AXI_HP portu arasındaki veriyolu genişlik farkı problemini çözmek için kullanılmıştır. Bu problem, Xilinx geliştirme araçlarında bulunan AXI Ara Bağlantı içerisindeki "Xilinx Veri Genişliği Dönüştürücü" kullanılarak da çözülebilmektedir. Fakat Xilinx'in veri genişliği dönüştürücüsünün kaynak kullanımının yüksek olması ve problemimize yeterince uyarlanabilir olmaması sebebiyle çalışmamızda probleme özel bir veri genişliği dönüştürücü geliştirilmiştir. Veri genişliği dönüştürücülerin kaynak kullanımı karşılaştırılması Tablo 2.1'de verilmiştir.

Tablo 2.1. Veri genişliği dönüştürücülerinin kaynak kullanımları

| Veri genişliği dönüştürücü | Slice | Slice Reg | LUT | BRAM |
|---|---|---|---|---|
| Xilinx Veri Genişliği Dönüştürücü | 344 | 887 | 738 | 3 |
| StreamIF Veri Genişliği Dönüştürücü | 46 | 116 | 115 | 2 |

Şekil 2.6'da geliştirilen veri genişliği dönüştürücünün içyapısı gösterilmiştir. Bu modül, "AXI Memory Mapped" protokolünde 32 bit genişliğinden 64 bit genişliğine dönüşüm yapmaktadır. Bit dönüşümü sebebiyle, 32 bit arayüzde bir işlemde 256 x 32 bit aktarım yapılırken, 64 bit arayüzde 128 x 64 bit aktarılmaktadır. Arayüzler arasında senkronizasyon için 512 derinliğinde 64 bitlik BRAM tabanlı FIFO kullanılmıştır.




Şekil 2.6. StreamIF Veri Genişliği Dönüştürücü modülünün iç yapısı

### 2.3. Performans Testleri

Performans testlerinde ReconOS Bellek Alt Sistemi ve StreamIF'in; sadece yazma, sadece okuma ve okuma + yazma kategorilerinde karşılaştırması yapılmıştır. Testlerde karmaşık hesaplama içermeyen, sadece okuma/yazma işlemlerine odaklanmış programlar kullanılarak işletim sisteminin yaklaşık kaynak kullanımın ölçülmesi amaçlanmıştır.

Okuma + yazma testi 4 MB'lık veri parçalarını RAM'in bir adresinden okuyup başka bir RAM adresine yazan HWT ile yapılmıştır. Tablo 2.2'de testin toplam bant genişliği kullanımı karşılaştırılması verilmiştir.

Tablo 2.2. Okuma + Yazma testinin bant genişliği kullanımı

| HWT Sayısı | MEMIF | StreamIF |
|---|---|---|
| 1 | 224,56 MB/s | 622,45 MB/s |
| 2 | 235,12 MB/s | 1241,06 MB/s |
| 3 | 234,92 MB/s | 1843,05 MB/s |
| 4 | 235,34 MB/s | 2432,96 MB/s |
| 5 | 235,31 MB/s | 2844,05 MB/s |
| 6 | 235,22 MB/s | 2908,33 MB/s |
| 7 | 235,23 MB/s | 2989,10 MB/s |
| 8 | 235,46 MB/s | 3039,14 MB/s |

Okuma + Yazma testinde ReconOS Bellek Alt Sistemi, HWT sayısından etkilenmeyerek en fazla 236 MB/s civarında hız değerlerine ulaşabilmiştir. StreamIF kullanıldığında HWT sayısına bağlı olarak 622 MB/s ile 3056 MB/s arasında değerler gözlemlenmiştir.

Xilinx tarafından Zynq-7000 platformu için DDR kontrolcüsü verimi değeri yaklaşık %75 olarak belirtilmiştir[30]. Xilinx'in "High-Performance Video Systems" [36] uygulamasında pratikte %70 bellek bant genişliği verimi değerine ulaşılmaktadır. Çalışmamızda 8 HWT'in aktif olduğu durumda 4264 MB/s ana bellek bant genişliğinden yaklaşık 3039 MB/s kullanılarak %71,2 verim elde edilmiştir.

Tablo 2.3'da sadece okuma testinin performans karşılaştırılması verilmiştir. Sadece okuma testi 32 bit tam sayı içeren 1048576 elemanlı bir dizinin (4 MB boyutunda) ana bellekten okunarak elemanlarında XOR işlemi yapan HWT kullanılarak yapılmıştır.

Tablo 2.3. Sadece okuma testinin bant genişliği kullanımı

| HWT Sayısı | MEMIF | StreamIF |
|---|---|---|
| 1 | 218,13 MB/s | 331,68 MB/s |
| 2 | 230,67 MB/s | 650,57 MB/s |
| 3 | 229,13 MB/s | 969,42 MB/s |
| 4 | 229,63 MB/s | 1282,69 MB/s |
| 5 | 229,43 MB/s | 1608,75 MB/s |
| 6 | 229,27 MB/s | 1905,71 MB/s |
| 7 | 228,89 MB/s | 2219,13 MB/s |
| 8 | 228,56 MB/s | 2524,19 MB/s |

Tablo 2.4'da sadece yazma testinin performans karşılaştırılması verilmiştir. Sadece yazma testi, 1048576'e kadar sayan bir sayıcının çıktısını ana belleğe kaydeden HWT kullanılarak yapılmıştır.

Tablo 2.4. Sadece yazma testinin bant genişliği kullanımı

| HWT Sayısı | MEMIF | StreamIF |
|---|---|---|
| 1 | 254,09 MB/s | 340,11 MB/s |
| 2 | 255,63 MB/s | 652,52 MB/s |
| 3 | 256,06 MB/s | 971,51 MB/s |
| 4 | 256,52 MB/s | 1290,32 MB/s |
| 5 | 257,10 MB/s | 1624,06 MB/s |
| 6 | 257,11 MB/s | 1925,54 MB/s |
| 7 | 257,09 MB/s | 2244,00 MB/s |
| 8 | 257,09 MB/s | 2547,58 MB/s |

Yazma + okuma testinde 8 HWT kullanılması durumundaki kaynak kullanımı Tablo 2.5'de verilmiştir. StreamIF'in ReconOS'a eklenmesiyle, Slice kategorisinde kaynak kullanımı yaklaşık %25 artarak %45'e çıkmıştır.

Tablo 2.5. Okuma + Yazma testinde 8 HWT kullanıldığı durumdaki kaynak kullanımı tablosu

| Kullanılan arayüz | Slice | Slice Reg | LUT | BRAM |
|---|---|---|---|---|
| Sadece MEMIF | 4832 (%36,3) | 6164 (%5,8) | 12276 (%23,1) | 8 (%5,7) |
| MEMIF + StreamIF | 5794 (%43,6) | 11129 (%10,5) | 13440 (%25,3) | 16 (%11,4) |

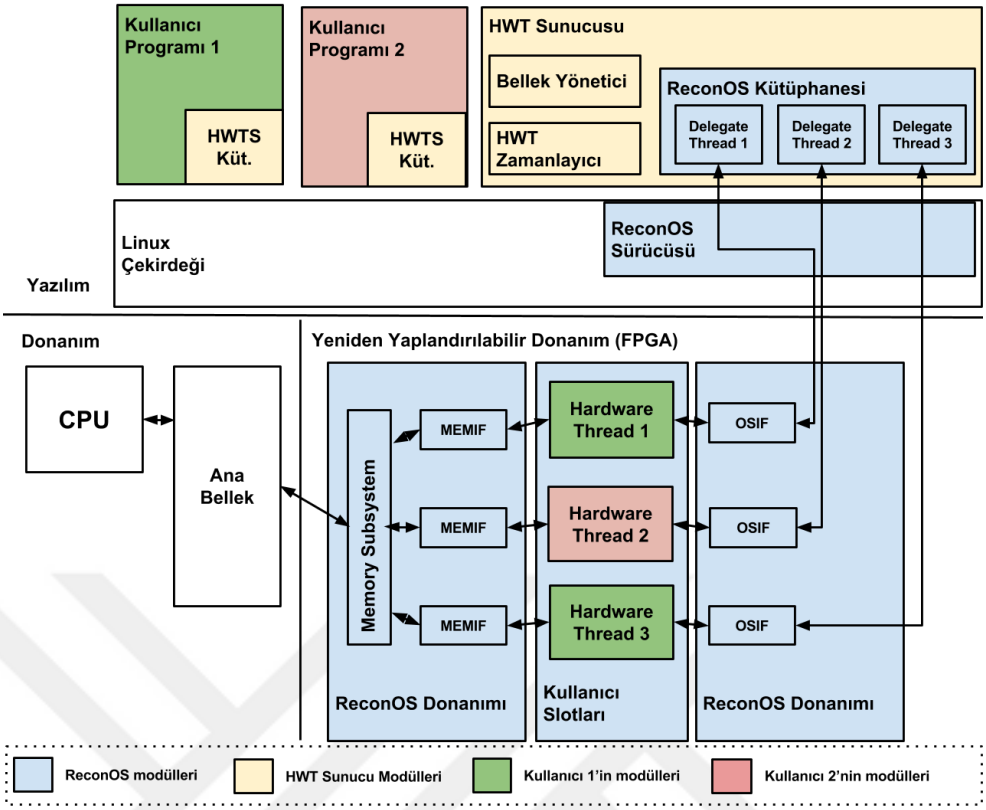
Testlerde elde edilen performans sonuçlarına göre, ReconOS'a geliştirilen bellek arayüzünün eklenmesiyle yaklaşık 12 kat yüksek bellek bant genişliği elde edilmiştir. Ana belleğin teorik bant genişliği değerinin yaklaşık %71'inin pratikte kullanılması başarılmıştır. Elde edilen 12 kat performans artışına karşın kaynak kullanımı sadece %20 artmıştır.

# 3. İŞLETİM SİSTEMİ MODELİ

Çok kullanıcılı işletim sistemlerinde birden fazla kullanıcı birçok farklı programı bir arada çalıştırabilmektedir. Kullanıcılar güvendiği ve güvenmediği kaynaklardan aldığı programları sistemde birlikte çalıştırmaktadır. Çalışan programların ve kullanıcıların birbirinden haberi olmadan, birbirinin çalışmasını bozmadan bir arada çalışabilmesi gerekmektedir. Modern işletim sistemlerinde programlar diğer programların bellek alanına veya kaynaklarına erişememesi için farklı adres uzaylarında çalıştırılmaktadır. İşletim sistemi ve donanım kaynaklarına erişim işletim sisteminin kontrolünde gerçekleşmektedir.

ReconOS'da kullanılan işletim sistemi modelinde FPGA kaynağı tek görev tarafından kullanılabilmektedir. Bu problemin temel sebebi HWT'lerin sanal adres – fiziksel adres dönüşümlerinde sadece donanımı başlatan göreve ait sayfa tablosunun kullanılabilmesidir. HWT'lerde adres dönüşümünde birden fazla gerçek sayfa tablosu kullanılması, donanım kaynağı ve geliştirme açısından maliyetli olacaktır. Bu yüzden problemin çözümü için sanal adres uzayını paylaşan bir görev (HWT Sunucu) kullanılmıştır.

HWT Sunucu, adres uzayı paylaşmanın yanında HWT alanlarını yönetmek, HWT alanlarında zaman paylaşımı yapmak ve kullanıcı programları ile HWT'ler arasındaki iletişimi yönetmek görevlerini içermektedir. Şekil 3.1'de HWT sunucusu çözümünü ve kullanıcı programları ile modüller arasındaki ilişkiler gösterilmiştir. Programlar HWT oluşturma, sonlandırma, bellek erişimi izinlerini değiştirme gibi yönetimsel işlemleri görevler arası haberleşme (IPC) ile HWT sunucusundan istekte bulunarak gerçekleştirmektedir.




Şekil 3.1. HWT Sunucusu, kullanıcı programları ve işletim sistemi modülleri arasındaki ilişki

### 3.1. HWT Sunucusu

HWT sunucusu, içerisine ReconOS kütüphanesi eklenmiş standart Linux programı olarak geliştirilmiştir. İşletim sistemi başlatıldığında Root kullanıcı yetkilerine sahip olarak yüksek öncelikli olarak çalıştırılmaktadır.

HWT Sunucusu ile kullanıcı programı arasındaki haberleşmede IPC yöntemi olarak Unix Alan Soketi [37] kullanılmıştır. Unix Alan Soketi üzerinden programın kullanıcı kimliği (UID), grup kimliği (GID), görev numarası (PID) gibi bilgileri kontrol edilebilmektedir. Ayrıca dosya tanımlayıcı paylaşma özelliği ile yetkili programdan yetkisi olmayan programa tekil dosya paylaşımı yapılabilmektedir.

HWT sunucusunun programlara sunduğu uygulama arayüzü fonksiyonları Tablo 3.1'de verilmiştir. İsimleri "hwt_" şablonuyla başlayan fonksiyonlar HWT yönetimini sağlamaktadır. Bu fonksiyonlara HWT Zamanlayıcı modülü tarafından cevap verilmektedir. İsimleri “hwtmem_" isim şablonuyla başlayan fonksiyonlar HWT ile program arasında bellek paylaşımını sağlamaktadır. Bu fonksiyonlara Bellek

Yöneticisi modülü cevap vermektedir. HWT sunucusunun arayüz fonksiyonları 3.3, 3.4 başlıkları altında detaylı olarak açıklanmıştır.

Tablo 3.1. HWT sunucusu programlama arayüzü fonksiyonları

| Fonksiyon | Açıklama |
|---|---|
| hwt_create | Bekleme durumunda HWT oluşturur |
| hwt_start | Bekleme durumundaki HWT'i başlatır |
| hwt_start_if | HWT'i zaman limitine göre başlatır |
| hwt_stop | HWT'i zorla bekleme durumuna alır |
| hwt_remove | HWT'i sonlandırır |
| hwt_timelimit | HWT'in çalışma zaman limitini ayarlar |
| hwtmem_malloc | HWT tarafından kullanılabilecek bellek alanı ayırır |
| hwtmem_free | Bellek alanını serbest bırakır |
| hwtmem_set_permission | Bellek alanının izinlerini değiştirir |
| hwtmem_shm_share_pid | Bellek alanını belirli bir görev ile paylaşır |
| hwtmem_shm_share_uid | Bellek alanını kullanıcı kimliğine göre diğer görevler ile paylaşır |
| hwtmem_shm_share_gid | Bellek alanını grup kimliğine göre diğer görevler ile paylaşır |
| hwtmem_shm_connect | Paylaşıma açılmış bellek alanını adres uzayına bağlar |

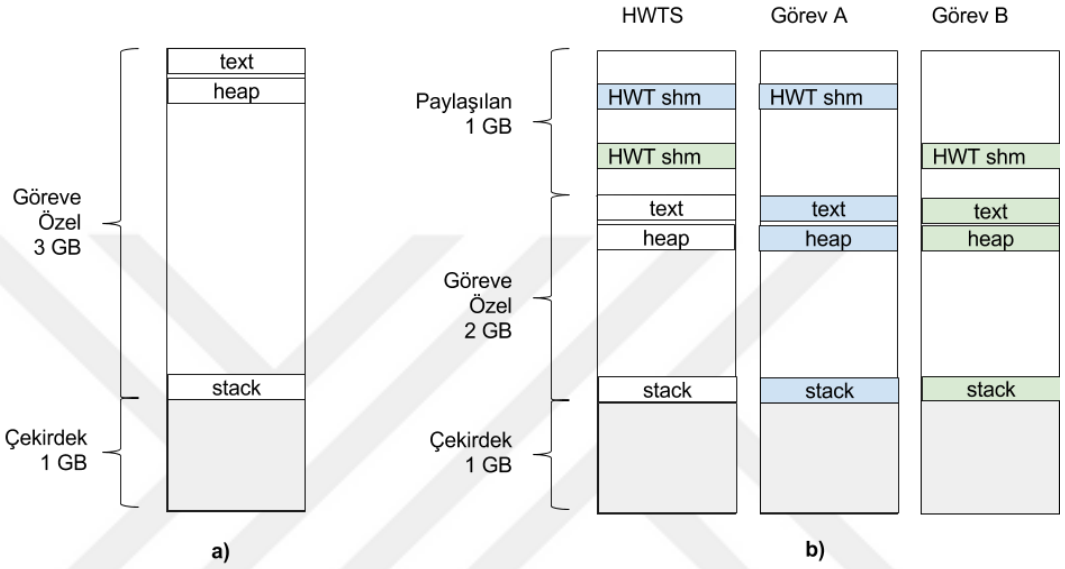
## 3.2. Program Modeli

Kullanıcı programları, HWT sunucusu ile haberleşme kütüphanesi eklenerek standart Linux programı olarak oluşturulmaktadır. Programda ihtiyaca göre Linux işletim sistemi ile çalışabilen kütüphane ve fonksiyonları kullanılabilmektedir.

Program ile sunucuda bulunan temsilci iş parçacıkları arasında haberleşme için görevler arası bellek paylaşımı kullanılmaktadır. Paylaşım yöntemi olarak Sistem V Paylaşılan Bellek [38] kullanılmıştır. ReconOS'un üretici/tüketici şeklinde haberleşen "message_box" fonksiyonları paylaşılan bellek üzerinden çalışacak şekilde düzenlenmiştir.

### 3.3. Bellek Yönetimi

ReconOS Bellek Alt Sistemi, tek göreve ait sanal adres uzayına ile çalışabildiği için programların bellek bölümlemesine standart Linux programlarından farklı olarak paylaşılan alan eklenmiştir. Şekil 3.2'de standart Linux bellek bölümlemesi ve kullanılan bellek bölümlemesi gösterilmiştir.




Şekil 3.2. a) Standart Linux bellek bölümlemesi b) Kullanılan bellek bölümlemesi

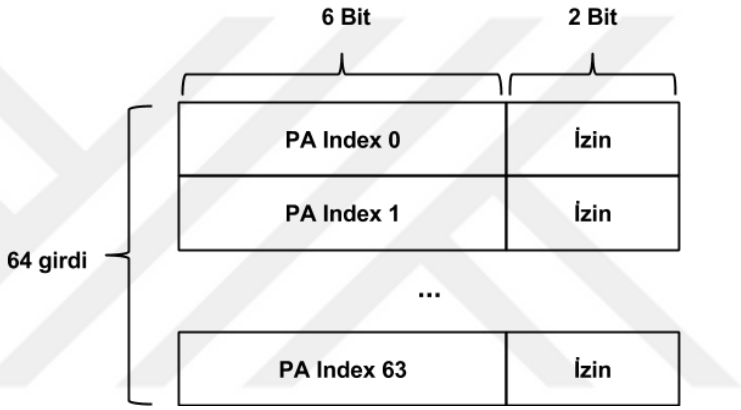
Paylaşılan alan HWT ve SWT'ler tarafından ortak olarak kullanılmaktadır. Sanal adres uzayının ilk 1 GB'lık kısmı bu alana ayrılmıştır. Paylaşılan alanın yönetimi HWT Sunucusu (HWTS) tarafından yapılmaktadır.

Programlar "hwtmem_malloc" fonksiyonu ile aldıkları bellek alanlarını HWT'ler ile paylaşabilmektedir. Ayrılan bellek alanına program ve HWT sunucusunda birebir aynı sanal adresten erişilmektedir. Bir göreve ait bellek alanı, diğer görevlerin sanal adres uzayında bulunmadığı için diğer görevler tarafından erişememektedir.

Bu modelde paylaşılan alanın sanal adresleri sunucu ve programda aynı olmak zorunda olduğu için sistem genelinde toplamda en fazla 1 GB sanal bellek paylaşılabilmektedir. Geliştirme yapılan ZedBoard kiti 512 MB fiziksel belleğe sahip olduğundan dolayı 1 GB'lık sanal adres paylaşım limiti herhangi bir kısıtlama getirmemektedir. 1 GB'dan daha fazla fiziksel belleğe sahip olan modellerde verimli kullanımı için 64 bit adres uzayını destekleyen işlemcilere ihtiyaç vardır.

Paylaşılan bellek alanlarında HWT'lerin izinlerini değiştirmek için "hwtmem_set_permission" fonksiyonu kullanılmaktadır. HWT'lere 4 MB'lık bellek blokları için sadece okuma, sadece yazma veya okuma + yazma izni verilebilmektedir[1].

Donanım tarafında bellek izinlerini gerçekleme için StreamIF modülüne her HWT için 64 girdiden oluşan adres, okuma ve yazma izinlerinden oluşan tablolar eklenmiştir. Bu tablolar aynı zamanda StreamIF içerisinde sanal adres (VA) – fiziksel adres (PA) dönüşümü için kullanılmaktadır. Şekil 3.3'de HWT'lerin bellek erişiminde kullanılan sayfa tablosu gösterilmiştir.



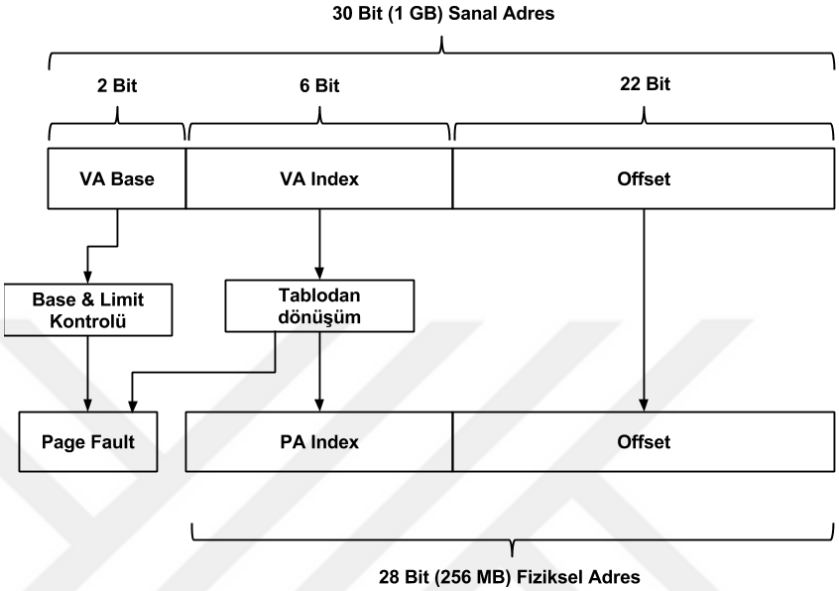

Şekil 3.3. HWT'lerin bellek erişiminde kullanılan sayfa tablosu

Kullanılan tablo modelinde 64 tane 4 MB'lık bellek alanı olmak üzere toplamda en fazla 256 MB fiziksel adrese erişebilmektedir. Geliştirme yapılan platformdaki 512 MB'lık fiziksel belleğin işletim sistemi, programlar ve 8 tane HWT arasında paylaşımı yapıldığı için HWT'lerin toplamda 256 MB'dan fazla fiziksel belleğe ihtiyacı olmayacağı varsayılmıştır. Farklı platformlarda ihtiyaca göre sayfa tablosunun girdi sayısı arttırılarak 256 MB'lık limit arttırılabilecektir.

Şekil 3.4'de sanal adres – fiziksel adres dönüşümü gösterilmiştir. 1 GB'lık sanal adres VA Base, VA Index ve Offset olmak üzere üç parçadan oluşmaktadır. VA Base ile görevin adres aralığı karşılaştırılarak adresin göreve ait olup olmadığı kontrol edilmekte olup göreve ait değilse Page Fault hatası verilmektedir. 6 bitlik VA Index

[1] Bellek alanı izinlerinin donanımdaki gerçeklemesi StreamIF'de uygulanmış olup, MEMIF'de henüz uygulanmamıştır.

değeri tablodan PA Index'e dönüştürülmektedir. Dönüşüm sırasında tablodaki izin alanı ile yapılan işlemin tipi (okuma veya yazma) kontrol edilmekte, izin yoksa Page Fault hatası üretilmektedir. PA Index'e Offset değeri eklenerek 28 bitlik fiziksel adres elde edilmektedir.




Şekil 3.4. HWT'lerin bellek erişimde yapılan adres dönüşümü

### 3.4. Zaman Paylaşımı

HWT'ler kullanıcı tarafından geliştirilen ve çalışma anında değiştirilen donanımlar olduğundan dolayı içerisindeki saklama yapıları (Register, Block Ram, Lut Ram), işlemci yapısı ve yapılar arasındaki ilişki işletim sistemi tarafından bilinmemektedir. Zynq-7000 platformunda donanım desteği ile çalışan preemptive çoklu-görev çözümü bulunmadığı için alternatif preemptive çoklu-görev yöntemlerine ihtiyaç duyulmaktadır.

HWT içerisine eklenen işletim sistemine yardımcı fonksiyonlar ile içeriğin okunması mümkündür. Fakat çok kullanıcılı bilgisayarda HWT içerisinde işletim sistemi fonksiyonu bulundurmak güvenilirlik ve hata soyutlaması açısından zaafiyet oluşturacaktır.

Çok kullanıcılı işletim sistemlerinde programlar güvenilmeyen kaynaklardan gelebileceği için HWT'leri zamanlarken preemptive çoklu-görev kullanılması zorunludur. Çalışmamızda HWT'lerde zaman paylaşımı için preemtive çoklu-göreve

benzeyen alternatif bir yöntem geliştirilmiştir. Yöntem şu kısıtlar altında çalışabilmektedir:

- Programlar çalıştırdığı HWT'lerin yaklaşık ne kadar sürede işlemini tamamlayacağını bilmelidir[1]
- Zaman paylaşımını kullanacak HWT'lerin durdurulmadan çalışması gereken işlem süreleri içerik değiştirme aralığından küçük olmalıdır

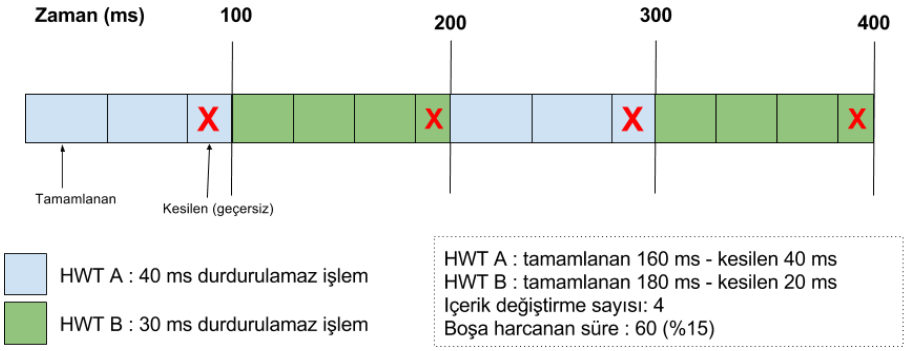
Geliştirilen zaman paylaşımı modelinde içerik değiştirme aralığı 100 milisaniye olarak belirlenmiştir. Kullandığımız platform yapılandırmasına göre sistem tam yükte çalışırken 100 milisaniye sürede bir HWT bellekte yaklaşık 18 MB okuma, 18 MB yazma ve HWT alanında yaklaşık 10 milyon saat darbesi süresi kadar işlem yapılabilmektedir.

HWT'lerin durdurulmadan çalışması gereken işlemlerini 100 milisaniyeden kısa sürede tamamlaması gerekmektedir. Bir işlemi bu süreden fazla olan HWT'lerin durdurulabilmesi için 100 milisaniyeden kısa sürelerde HWT durumunu kaydedecek şekilde tasarlanması gerekmektedir.

Programlar oluşturdukları HWT'lerin FPGA zamanını verimli şekilde kullanabilmesi için yaklaşık olarak ne kadar süre çalışacağının sunucuya bildirmesi gerekmektedir. Çalışma zamanını bildirilmediğinde veya hatalı bildirildiğinde HWT'ler 100 ms sonunda sıfırlanarak zorla kesilmektedir. Zorla kesilme sonucunda programa HWT'in kesildiği ve son işlemin geçersiz olduğunu bildiren bir hata mesajı iletilmektedir.

Sistemin işleyiş biçimini açıklamak için öncelikle çalışma süresi bildirilmediği durumundaki zaman paylaşımı örneği Şekil 3.5'de verilmiştir. Örnekte "HWT A" 40 milisaniyelik, "HWT B" 30 milisaniyelik işlemler şeklinde çalışmaktadır. HWT A'nın ilk iki işlemi düzgün bir şekilde tamamlanmıştır. 3. işlem tamamlanamadan ayrılan zaman dolduğu için işlem zorla kesilmiş ve HWT B yüklenmiştir. HWT B çalışırken 3 tane işlemi tamamlanmış, 4. işlemde süre dolduğu için zorla kesilerek HWT A yüklenmiştir.

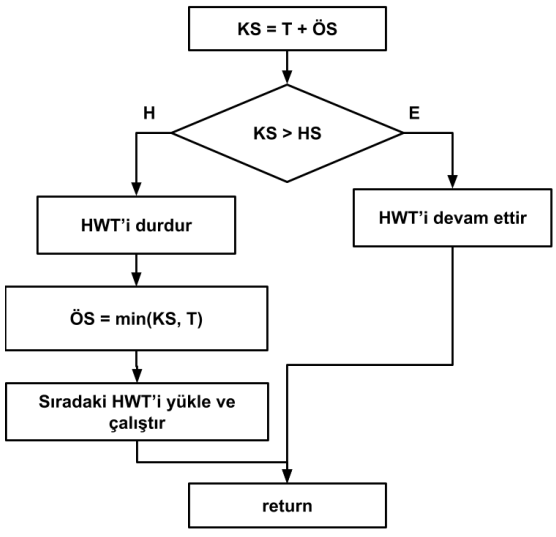
[1] HWT'ler donanım tasarımı olarak geliştirildiği için işlemin süresi geliştirici tarafından hesaplanabilmektedir.




Şekil 3.5. Çalışma süresi bildirilmediğindeki zaman paylaşımı

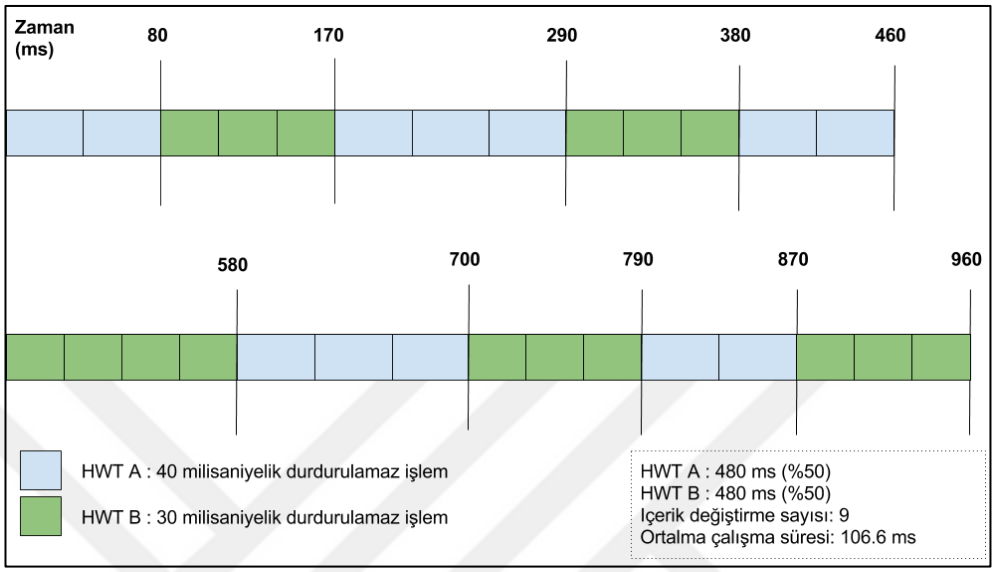
HWT'ler zorla kesildiğinde son işlemdeki verileri geçersiz olduğu için verilen örnekte 400 milisaniyelik zaman dilimi içinde 60 milisaniyelik süre boşa harcanmıştır.

Geliştirilen yöntemde HWT'lerin kesilmesini önlemek için içerik değiştirme zaman aralığı sabit 100 milisaniye yerine, ortalama 100 milisaniye olacak şekilde kullanılmaktadır. İçerik değiştirmenin 100 milisaniyeden önce veya sonra yapılması gerektiği kararı programın bildirdiği süreye göre yapılmaktadır. 100 milisaniyeden önce kesilen HWT'lerin kullanmadığı süre saklanmaktadır. Sonraki çalışmada kullanmadığı süre eklenerek çalıştırılmaktadır. Yöntemin akış şeması Şekil 3.6'da verilmiştir.




Şekil 3.6. HWT'lerde zaman paylaşımı algoritması

Geliştirilen yöntem 50-200 milisaniye arasında içerik değiştirme yapmakta olup ortalamada 100 milisaniyeye yaklaşarak çalışmaktadır. Şekil 3.5.'de verilen örneğin çalışma süresi bildirildiğindeki zaman paylaşımı grafiği Şekil 3.7'de verilmiştir.




Şekil 3.7. Çalışma süresi bildirildiğindeki zaman paylaşımı

Örnekte HWT A ilk çalışmasında iki işlem tamamlayıp (80 ms), kalan 20 ms süre bir işlem daha yapmaya yetmediği için kalan süresi kaydedilip HWT B çalıştırılmaya başlanmıştır. HWT B üç işlem tamamlamış (90 ms), kalan 10 ms işlem yapmaya yetmediği için süre kaydedilip HWT A çalıştırılmıştır. HWT A ikinci çalışmasında 20 ms ek süresi olduğu için üç işlem tamamlamıştır. İki HWT arasındaki zaman paylaşımı bu şekilde devam ederek 960 ms sonunda ikisi HWT de 480 ms çalıştırılmıştır. Örnekteki zaman aralığında ortalama 106,6 milisaniyelik zaman aralıklarıyla içerik değiştirme yapılmıştır. Bu örneğin C dilinde IPC altyapısı kullanılarak yazılmış simülasyonda yaklaşık 1 dakika çalıştırılmasından sonra elde edilen sonuçlar Tablo 3.2'de verilmiştir. Çalışma süresi arttığı için ortalama 100 milisaniyeye yaklaşmıştır.

Tablo 3.2. Zaman paylaşımı örneğinin çalışma süresi karşılaştırması

| Özellik | HWT A | HWT B |
|---|---|---|
| Durdurulamayacak süre | 40 ms | 30 ms |
| Çalışma sayısı | 300 | 300 |
| Çalışma süresi | 30,04 sn | 30,05 sn |
| Ortalama çalışma süresi | 100,130 ms | 100,173 ms |
| En uzun çalışma süresi | 124,809 ms | 125,952 ms |
| En kısa çalışma süresi | 81,195 ms | 91,109 ms |
| IPC gecikmesine harcanan süre | ~0,04 sn | ~0,05 sn |

Şekil 3.8'de aynı örneğin, 1 HWT alanının aktif olduğu sistemde yaklaşık 1 dk çalıştırılması sonrasında elde edilen ekran görüntüsü verilmiştir. hwt:0 40 milisaniyelik işlem, hwt:1 ise 30 milisaniyelik işlem yapmaktadır. Ekran görüntüsü alınan ana kadar 30,21 saniye hwt:0, 30,19 saniye hwt:1 çalışmıştır.

```
pid:8727 uid:1000 mem:1152kb
  hwt:0    RUNNING(0)   test-1       30.21  %45
  hwt:1    WAITING      test-2       30.19  %50
----------------------
 FPGA usage: %96 - CPU usage: %11
```

Şekil 3.8. 1 HWT alanının aktif olduğu sistemde 2 HWT çalıştırılması

Ekran çıktısında 2. sütun HWT'in o andaki çalışma durumunu göstermektedir. "RUNNING(0)" değeri, HWT'in 0 numaralı alanda çalıştığını, "WAITING" ise bekleme durumunda olduğunu göstermektedir. 3. sütunda HWT'in program içindeki etiketi, 5. sütunda ise son 4 saniyedeki ortalama çalışma zamanı göstermektedir.

Ekran çıktısının son satırında FPGA ve CPU kullanım oranları gösterilmektedir. İçerik değiştirme sırasında işlem yapılamamasından dolayı FPGA zamanından %4 kayıp yaşanmıştır. Yapılan işlem CPU kaynağı kullanmadığı için IPC, HWT Sunucu ve işletim sistemi servislerinden kaynaklanan %11'lik CPU kullanımı gözlemlenmektedir.

İdeal olmayan durumun testi için 50 ms, 96 ms ve 30 ms işlem yapabilen üç HWT kullanılmıştır. Bu durum simülasyonda 1 dakika çalıştırıldıktan sonra elde edilen sonuçlar Tablo 3.3'de verilmiştir. İdeal olmayan durumda en kısa ve un uzun çalışma

sürelerinde 50-200 ms sınırlarına yakın değerler elde edilmiştir. HWT'ler eşit sayıda ortalama 100 milisaniyelik süreyle çalışmıştır.

Tablo 3.3. İdeal olmayan zaman paylaşımı örneğinin çalışma süresi karşılaştırması

| Özellik | HWT A | HWT B | HWT C |
|---|---|---|---|
| Durdurulamayacak süre | 50 ms | 96 ms | 30 ms |
| Çalışma sayısı | 200 | 200 | 200 |
| Çalışma süresi | 20,02 sn | 20,02 sn | 20,03 sn |
| Ortalama çalışma süresi | 100,090 ms | 100,078 ms | 100,145 ms |
| En uzun çalışma süresi | 104,859 ms | 194,855 ms | 125,879 ms |
| En kısa çalışma süresi | 50,494 ms | 96,570 ms | 92,234 ms |
| IPC gecikmesine harcanan süre | ~0,02 sn | ~0,02 sn | ~0,03 sn |

Şekil 3.9'da aynı örneğin, 1 HWT alanının aktif olduğu sistemde çalıştırılmasıyla elde edilen ekran çıktısı verilmiştir.

```
pid:9236 uid:1000 mem:1216kb
  hwt:0    WAITING      test-1       20.38  %34
  hwt:1    RUNNING(0)   test-2       20.13  %28
  hwt:2    WAITING      test-3       20.18  %33
----------------------
 FPGA usage: %96 - CPU usage: %9
```

Şekil 3.9. 1 HWT alanında zamanlaması ideal olmayan 3 HWT çalıştırılması

Çalışma sırasında sistemdeki yük değişimi durumlarında performans anlık olarak düşebileceği için programlarda HWT'lerin zorla kesilme durumunu göz önünde bulundurması gerekmektedir. Zorla kesilme durumunda HWT Sunucu, programa gerekli hata bilgisini iletmekte ve HWT'in yarım kalan işlemini baştan başlatmasına olanak sağlamaktadır.

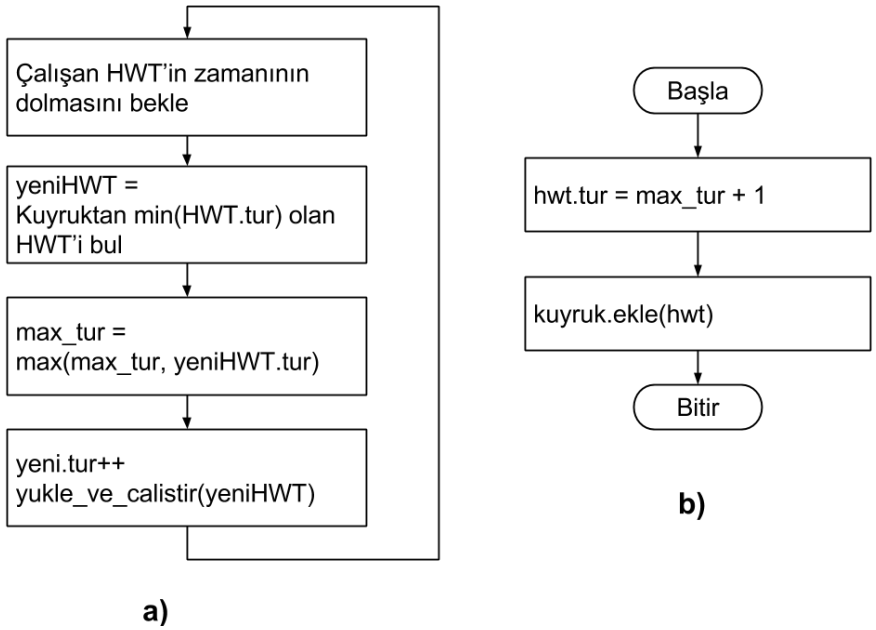
Açıklanan zaman paylaşımı yöntemi her HWT alanı için ayrı olarak çalıştırılmaktadır. Test platformunda 8 tane HWT alanı kullanılabildiği için 8 HWT'e kadar iş kesme yapılmadan eş zamanlı çalıştırma yapılabilmektedir. 8'den daha fazla HWT olduğu durumda HWT'ler eşit zaman alacak şekilde alanlar arasında dağıtılarak çalıştırılmaktadır. Şekil 3.10'da 4 HWT alanının aktif olduğu sistemde 4 HWT çalıştırılması durumu gösterilmiştir.

```
pid:2380 uid:1000 mem:3200kb
  hwt:0    RUNNING(1)   test-1       13.19  %96
  hwt:1    RUNNING(0)   test-1       13.87  %100
  hwt:2    RUNNING(3)   test-1       13.19  %96
  hwt:3    RUNNING(2)   test-2       13.08  %96
----------------------
 FPGA usage: %98 - CPU usage: %12
```

Şekil 3.10. 4 HWT alanında 4 HWT çalıştırılması

HWT alanı sayısından fazla HWT çalıştırıldığı durumda HWT seçimi için, Linux Completely Fair Scheduler (CFS) algoritmasında [31] CPU çekirdeği için kullanılan görev seçimi yöntemi uyarlanmıştır. HWT'lerde içerik değiştirme yazılıma göre maliyetli olduğu için CFS'de kullanılan çalışma süresi üzerinden seçim yerine çalışma sayısı üzerinden seçim kullanılmıştır. HWT'lerin eşit sayıda çalıştırılması sonucunda eşit süre kullanması Şekil 3.6.'da gösterilen yöntem ile sağlanmıştır.

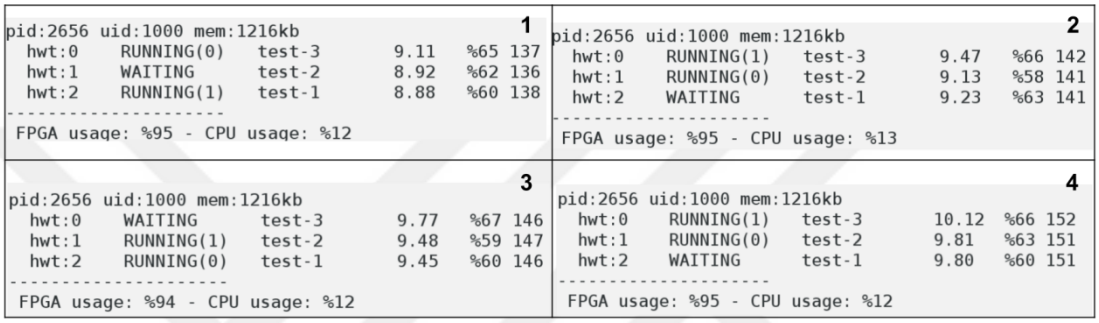
Şekil 3.11'de HWT seçim algoritması verilmiştir. Algoritma her seçimde en az çalıştırılan HWT'i seçerek HWT'lerin çalıştırılma sayılarını eşitlemeye çalışmaktadır. Her HWT için "tur" isimli değişkende kaç tur çalıştırıldığı bilgisi saklanmaktadır. Çalıştırılan en yüksek tur değeri "max_tur" isimli değişkende saklanmaktadır. Her alan içeriği değiştirmede en düşük tur değerine sahip HWT seçilerek tüm HWT'lerin eşit sayıda çalıştırılması sağlanmaktadır.




Şekil 3.11. a) HWT seçimi yönteminin akış şeması b) Kuyruğa HWT ekleme yönteminin akış şeması

Yeni HWT oluşturulduğunda veya bir HWT uykudan uyandırıldığında çalışma kuyruğuna eklenirken tur değeri max_tur olarak güncellenmektedir. Bu sayede kuyrukta olan HWT'lerin tamamı çalıştırıldıktan sonra yeni eklenen HWT çalıştırılmaktadır.

Şekil 3.12.'de 2 HWT alanında 3 HWT çalıştırılması örneğinin yaklaşık 1 saniye aralıkla alınmış 4 ekran çıktısı verilmiştir. Ekran çıktılarında 6. sütunda HWT'in o andaki tur değeri gösterilmektedir.

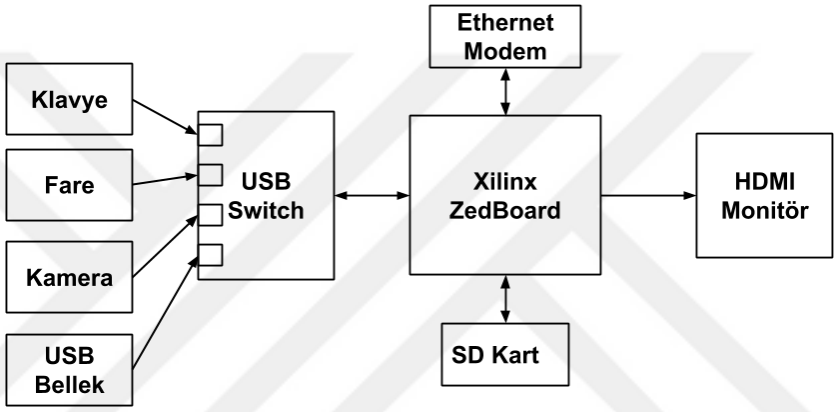



Şekil 3.12. 2 HWT alanında 3 HWT çalıştırıldığı durumdan 1 saniye aralıkla alınmış 4 ekran görüntüsü

# 4. TEST PLATFORMU VE KULLANICI PROGRAMLARI

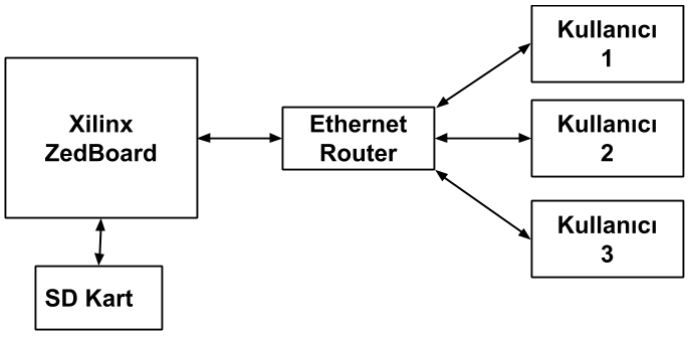
## 4.1. Test Platformu Ayarları

Kişisel bilgisayar modelinin testi için klavye, fare, monitör, USB kamera gibi G/Ç aygıtlarını kullanılmıştır. Şekil 4.1'de kişisel bilgisayar modeli için kullanılan test ortamının şeması gösterilmiştir.




Şekil 4.1. Kişisel bilgisayar modeli için kullanılan test ortamı

Sunucu modelinin testi, kullanıcıların SSH protokolü ile ethernet üzerinden sisteme bağlanması ve işlemlerini gerçekleştirmesi şeklinde yapılmıştır. Şekil 4.2'de sunucu modelinin testinde için kullanılan test ortamının şeması gösterilmiştir.




Şekil 4.2. Sunucu modeli için kullanılan test ortamı

USB desteği için Linux çekirdeğinin [39] web adresindeki modülleri aktifleştirilmiştir. ZedBoard geliştirme kitinin kullandığı USB sürücüsü bazı Linux

versiyonlarında çalışmamaktadır. Testte Linux çekirdeğinin [40] web adresindeki 3.14 versiyonu kullanılmıştır.

ZedBoard geliştirme kitinin HDMI portu sadece FPGA tarafından kontrol edilebildiğinden dolayı RAM'e doğrudan erişerek çalışan bir HDMI sürücü geliştirilmiştir. HDMI sürücüde bellek erişimi için "2. Yüksek Performanslı Bellek Arayüzü" başlığında anlatılan yöntemin basitleştirilmiş hali kullanılmıştır. Geliştirilen HDMI sürücü Linux çekirdeğine "simple-framebuffer" [41] aygıtı olarak tanıtılarak, aygıtın işletim sistemi tarafından ekran kartı olarak kullanılması sağlanmıştır.

Sisteminin testinde YYH yöntemini kullanan komut satırı üzerinden çalışan basit test uygulamaları ve video oynatma yazılımı kullanılmıştır. Geliştirilen HWT Sunucu, test programları ve HDMI sürücü Linaro/Ubuntu 14.04 dağıtımı [42] kullanılarak test edilmiştir.

### 4.2. Kişisel Bilgisayar Modeli Testi: Video Oynatıcı

Kişisel bilgisayarlarda YYH'nın kullanımının sağlayabileceği faydayı göstermek için günlük yaşamda sıklıkla kullanılan video oynatma problemi üzerinden karşılaştırma yapılmıştır. Testler internet ortamında standart olarak kullanılan h264 [43] ve vp8 [44] codecleri ile sıkıştırılmış 1280x720 çözünürlükteki saniyede 25 kare hızındaki videoların oynatılması şeklinde gerçekleştirilmiştir. Bu özellikteki videoların izlenebilmesi için saniyede en az 25 kare hızında kod çözme yapılması gerekmektedir.

ZedBoard'da NEON FPU destekli 667Mhz hızında çalışan çift çekirdek ARM Cortex-A9 işlemci bulunmaktadır. Yaptığımız testlerde FFmpeg [45] kütüphanesi kullanılarak geliştirilmiş, sadece işlemciyi kullanılarak çalışan bir program ile test videoları ortalama saniyede 10 kare gibi hızlarda gösterilebilmiştir. Videoların 25 kare hızında gösterilebilmesi için video çözme aşamalarının bir kısmının FPGA kaynakları ile gerçekleştirilmesi gerekmektedir.

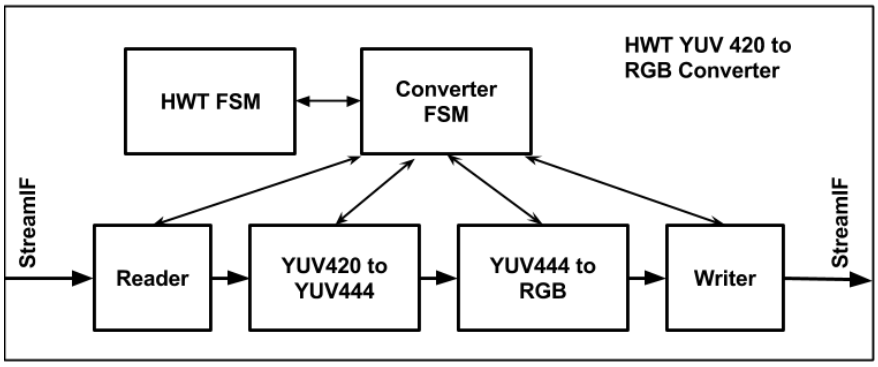
Yaptığımız testlerde FFmpeg kütüphanesinin h264 ve vp8 codeclerini saniyede 25 kareden daha yüksek hızda çözebildiği fakat video sıkıştırmada kullanılan yuv420

piksel formatının monitörde göstermek için kullanılan RGB piksel formatına dönüşümünü saniyede 14 kareden daha yavaş yaptığı gözlemlenmiştir. Ardından piksel formatı dönüştürme işlemi, FFMpeg kütüphanesi yerine, C dilinde 32 bit tam sayı kullanarak yaklaşık değerler hesaplanarak yazılmış ve derleyici optimizasyonları aktif olarak derlenmiş denenmiş kod ile saniyede 10 kare hızında işlem yapılabilmiştir.

Yapılan testler sonucunda, video oynatma programının piksel formatı dönüşümünün HWT ile yapılması gerektiğine karar verilmiştir. Yazılımın codec çözme hızı yeterli olduğu için bu işlem SWT şeklinde FFmpeg Kütüphanesi ile yapılmıştır.

### 4.2.1. HWT ile YUV420 - RGB renk formatı dönüşümü

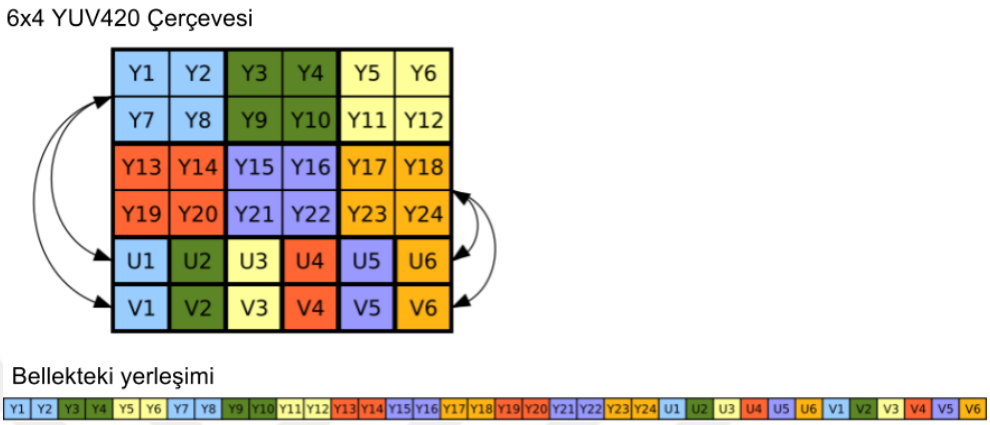
YUV420 – RGB renk formatı dönüşümü için geliştirilen modülün içyapısı Şekil 4.3'te gösterilmiştir. "HWT FSM", modülün genelinin yönetildiği ve ReconOS sistem çağrıları ile IPC işlemlerinin yapıldığı modüldür. "Converter FSM" renk formatı dönüşümünü takip eden, okuma ve yazma işlemlerini yönetmektedir. "StreamIF Reader" ve "StreamIF Writer" modülleri, Başlık 2'de anlatılan yüksek performanslı bellek arayüzü ile bağlantıyı sağlamaktadır. "YUV420 to YUV444" ve "YUV444 to RGB" modülleri sırasıyla YUV444 ve RGB'ye dönüşümü sağlamaktadır.




Şekil 4.3. YUV420 – RGB dönüştürücü HWT'in iç yapısı

YUV420 formatında [46], HDMI sürücüde kullanılan RGB formatından farklı olarak pikselin bileşenleri bellekte farklı alanlarda saklanmaktadır. Şekil 4.4'te YUV420 formatında 6 x 4 boyutundaki resmin matris ve dizi biçimindeki yerleşimi gösterilmektedir. Her piksel için bir parlaklık bileşeni (Y) ve her 4 piksel için ortak

olan renk bileşenleri (Cb ve Cr) kullanılmaktadır. Örneğe göre "0,0" pikselinin Y değeri Y1, Cb değeri U1, Cr değeri V1'dir. Matriste "0,0", "0,1", "1,0", "1,1"de bulunan pikseller için renk bileşenleri olarak U1 ve V1 ortak kullanılmaktadır. 8 bitlik YUV formatında Y, Cb ve Cr bileşenleri bellekte 1'er byte yer kaplamaktadır.




Şekil 4.4. YUV420 formatında 6 x 4 boyutundaki resmin piksellerinin matris ve dizi modelindeki yerleşimleri [47]

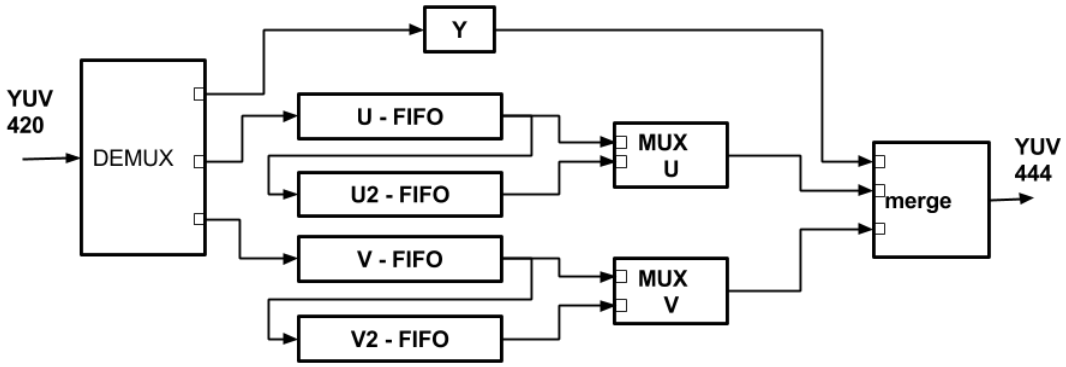
YUV420 formatı, piksele karşılık gelen Y, Cb ve Cr bileşenleri bir araya getirilerek YUV444 formatına dönüştürülmektedir. Ardından Denklem (4.1), (4.2), (4.3) kullanılarak RGB formatına dönüştürülmektedir [46];

$$R' = 1{,}164(Y - 16) + 1{,}596(Cr - 128) \quad (4.1)$$

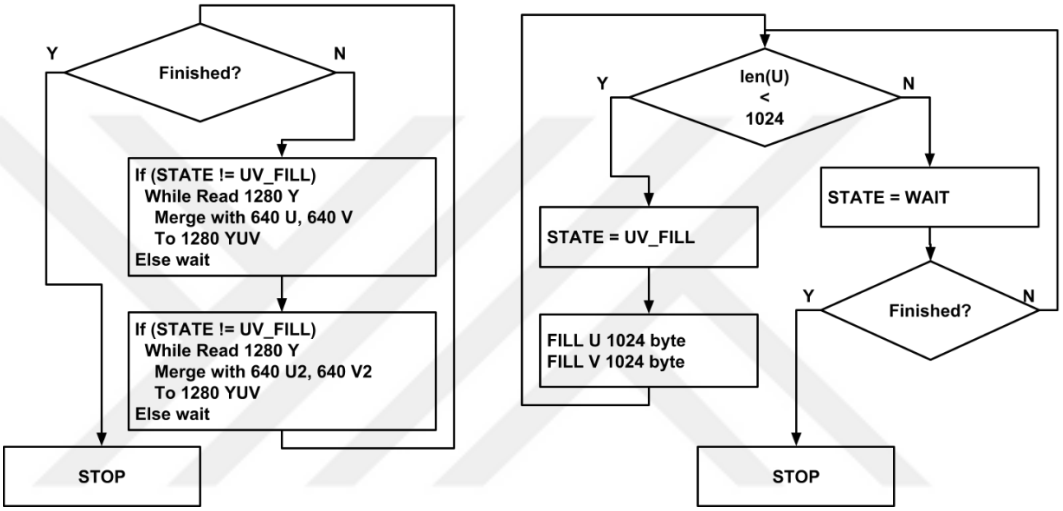
$$G' = 1{,}164(Y - 16) - 0{,}813(Cr - 128) - 0{,}391(Cb - 128) \quad (4.2)$$

$$B' = 1{,}164(Y - 16) + 2{,}018(Cb - 128) \quad (4.3)$$

YUV420 formatından YUV444 formatını dönüşüm, iki Y bileşenine karşılık bir U ve V bileşeninin bellekten okunması gerekmektedir. Ayrıca Y'nin ikinci satırında, U ve V bileşeninin birinci satırdaki değerlerinin okunması gerekmektedir. Bu problemin çözümü için U, U2, V, V2 olarak 4 FIFO'nun kullanılan bir tasarım geliştirilmiştir. Şekil 4.5'te YUV444'e dönüşümü sağlayan modülün içyapısı gösterilmiştir. Modülün akış şeması Şekil 4.6'te verilmiştir.




Şekil 4.5. YUV420 – YUV444 dönüştürücünün içyapısı




Şekil 4.6. YUV420 – YUV444 dönüştürücünün akış şeması

YUV444'den RGB'ye dönüşümde, işlemlerde 16 bit ondalık, 16 bit tam kısımdan oluşan sabit noktalı sayı kullanılmıştır. İşlemlerde kullanılan katsayılar formüldeki ondalık katsayıların 65536 ile çarpılması ile elde edilmiştir. İşlem 4 saat darbesinde gerçekleşecek şekilde bölünmüştür. Birinci saat darbesinde yapılan işlemler Denklem (4.4), (4.5), (4.6)'da verilmiştir;

$$y = Y - 16 \tag{4.4}$$

$$u = Cb - 128 \tag{4.5}$$

$$v = Cr - 128 \tag{4.6}$$

Önceki aşamada elde edilen değerler kullanılarak ikinci saat darbesinde yapılan sabit ile çarpma işlemleri Denklem (4.7), (4.8), (4.9), (4.10), (4.11)'de verilmiştir;

$$a = 76284\ y \tag{4.7}$$

$$b = 104596\ v \tag{4.8}$$

$$c = 53281\ v \tag{4.9}$$

$$d = 25625\ u \tag{4.10}$$

$$e = 132252\ u \tag{4.11}$$

İkinci saat darbesinde hesaplanan değerler kullanılarak üçüncü saat darbesinde yapılan işlemler Denklem (4.12), (4.13), (4.14)'te verilmiştir;

$$R = (a + b) / 65536 \tag{4.12}$$

$$G = (a - c - d) / 65536 \tag{4.13}$$

$$B = (a + e) / 65536 \tag{4.14}$$

4. saat darbesinde, elde edilen R, G, B değerleri 0'dan küçük olması durumunda 0, 255'den büyük olması durumunda 255 olacak şekilde çıkışa verilmektedir.

### 4.2.2. Performans testleri

Video oynatıcı testlerinde YUV420 – YUV444 dönüşümü için 4 yöntem kullanılmıştır:

- FFmpeg YUV çevirici: FFmpeg kütüphanesi kullanılarak tüm işlemlerin CPU'da gerçekleştirilmesi
- C YUV çevirici: Denklem (4.1) ile Denklem (4.14) arasındaki işlemlerin C dilinde tam sayı kullanılarak yazılan program ile CPU'da gerçekleştirilmesi
- HWT: Girişin CPU ile FPGA bellek alanına kopyalanması ve hesaplama işlemlerinin FPGA'de gerçekleştirilmesi
- HWT + DMA sürücü [48]: Girişin FPGA bellek alanı üzerinde oluşturulması, işlemlerin FPGA'de gerçekleştirilmesi

YUV420 – YUV444 dönüşüm yöntemlerin 1 kare için ortalama CPU ve FPGA kullanım süreleri Tablo 4.1'de verilmiştir. FFmpeg, kayan noktalı sayı işlemlerinde CPU'nun FPU birimini kullandığı için 32 bit tam sayı ile hesaplamaya göre daha iyi

sonuç elde etmiştir. FFmpeg kütüphanesi platform özel işlemci optimizasyonlarını kullandığından dolayı sadece yazılım kullanılarak daha iyi çözüm üretilmesi pek mümkün değildir.

Tablo 4.1. YUV-RGB dönüşümü yöntemlerin 1 kare için CPU ve FPGA kullanım süreleri ve hız karşılaştırması

| Yöntem | CPU Kullanımı | FPGA Kullanımı | Hız |
|---|---|---|---|
| FFmpeg YUV çevirici | 76 ms | 0 ms | ~13 kare/sn |
| C YUV çevirici | 104 ms | 0 ms | ~10 kare/sn |
| 1 tane HWT | 16 ms | 15 ms | ~63 kare/sn |
| 1 tane HWT + DMA sürücü | 0 ms | 15 ms | ~67 kare/sn |

HWT yönteminin işlem süresi seri olarak hesaplandığında saniyede 32 kare hızında işlem yapabilmektedir. Fakat CPU ve FPGA eş zamanlı olarak işlem yapabildiğinden dolayı işlem CPU hesaplama süresinde tamamlanabilmektedir. Bu yüzden hız saniyede 63 kare olarak hesaplanmıştır. Bu değer FFmpeg kütüphanesinin yaklaşık 4,8 katı hızındadır.

HWT + DMA sürücü yönteminde FFmpeg yönteminin 5,2 katı hıza ulaşılmıştır. Bu yöntemde CPU kaynakları senkronizasyon haricinde kullanılmamaktadır. Hesaplama yapılmadığından dolayı CPU süresi nanosaniye seviyesinde olduğu için 0 milisaniye olarak hesaplanmıştır.

Gerçek video ile yapılan testlerde vp8 ve h264 codecleri için yüksek ve düşük bit hızına olmak üzere 4 video kullanılmıştır:

- VP8 Test 1: VP8 Codec ile düşük bit hızı
- VP8 Test 2: VP8 Codec ile yüksek bit hızı
- H264 Test 1: H264 Codec ile düşük bit hızı
- H264 Test 2: H264 Codec ile yüksek bit hızı

Testler videonun ilk 100 karesinin beklemeden oynatılma süresinin CPU zamanı ölçülmesi şeklinde yapılmıştır. Tablo 4.2'de VP8 Codec, Tablo 4.3'de H264 Codec için performans sonuçları verilmiştir. "Gereken CPU %" sütunu, videonun saniyede 25 kare hızında oynatılabilmesi için gereken CPU kullanım yüzdesini ifade etmektedir. Videonun izlenebilmesi için ilk 100 karesinin 4 saniyede gösterilmesi

gerekmektedir. İşlemci 2 çekirdekli olduğu için 4 saniye video, %100 CPU kullanımı ile oynatıldığında 8 saniye CPU zamanına denk düşmektedir. "Gereken CPU %" sütunu, kullanılan CPU zamanı ile %100 kullanımdaki CPU zamanı oranlanarak hesaplanmıştır. %100'den büyük değerler videonun 25 kare hızında oynatılamayacağı anlamına gelmektedir.

Tablo 4.2. VP8 Codec video oynatma performans testi sonuçları

| Yöntem | Test 1 | | Test 2 | |
|---|---|---|---|---|
| | CPU Zamanı | Gereken CPU % | CPU Zamanı | Gereken CPU % |
| Sadece FFmpeg decoding | 1,910s | %23,9 | 4,960s | %62,0 |
| FFmpeg decoding & FFmpeg YUV çev. | 9,510s | %118,9 | 12,590s | %157,4 |
| FFmpeg decoding & C YUV çev. | 12,130s | %151,6 | 15,340s | %191,8 |
| FFmpeg decoding & HWT | 3,550s | %44,4 | 6,550s | %81,9 |
| FFmpeg decoding & HWT DMA opt. | 1,910s | %23,9 | 4,960s | %62,0 |

VP8 Codec testlerinde, sadece yazılım kullanılarak oynatılması mümkün olmayan videolar HWT ile renk dönüşümü yapıldığında oynatılabilmektedir. HWT kullanıldığında düşük bit hızına sahip videoda FFmpeg kütüphanesine göre yaklaşık 5 kat, yüksek bit hızına sahip videoda yaklaşık 2,5 kat performans elde edilmektedir.

Tablo 4.3. H264 Codec video oynatma performans testi sonuçları

| Yöntem | Test 1 | | Test 2 | |
|---|---|---|---|---|
| | CPU Zamanı | Gereken CPU % | CPU Zamanı | Gereken CPU % |
| Sadece FFmpeg decoding | 4,120s | %51,5 | 8,270s | %103,4 |
| FFmpeg decoding & FFmpeg YUV çev. | 11,710s | %146,4 | 15,770s | %197,1 |
| FFmpeg decoding & C YUV çev. | 14,430s | %180,4 | 18,510s | %231,4 |
| FFmpeg decoding & HWT | 5,780s | %72,3 | 9,830s | %122,9 |
| FFmpeg decoding & HWT DMA opt. | 4,120s | %51,5 | 8,270s | %103,4 |

H264 Codec, VP8'e göre çözmede daha fazla CPU kullanmaktadır. Yüksek bit hızındaki videoda HWT kullanıldığında yaklaşık 2 katı performansa ulaşılmasına rağmen yeterli hıza ulaşılamamıştır. H264 Codec testinde düşük bit hızındaki video oynatılabilmiştir.

Test sonuçları video oynatıcıda renk dönüşümü için HWT kullanılması durumunda, sadece CPU ile oynatılması mümkün olmayan videoların oynatılabildiğini göstermiştir. VP8 ve H264 Codec'lerinin FFmpeg kullanılarak çözülmesi yerine HWT yardımı ile çözülmesi durumunda daha az CPU kaynağı kullanılması, daha yüksek çözünürlüklü veya daha yüksek bit hızına sahip videoların oynatılması mümkün olacaktır.

### 4.3. Çok Kullanıcılı Sunucu Modeli Testi

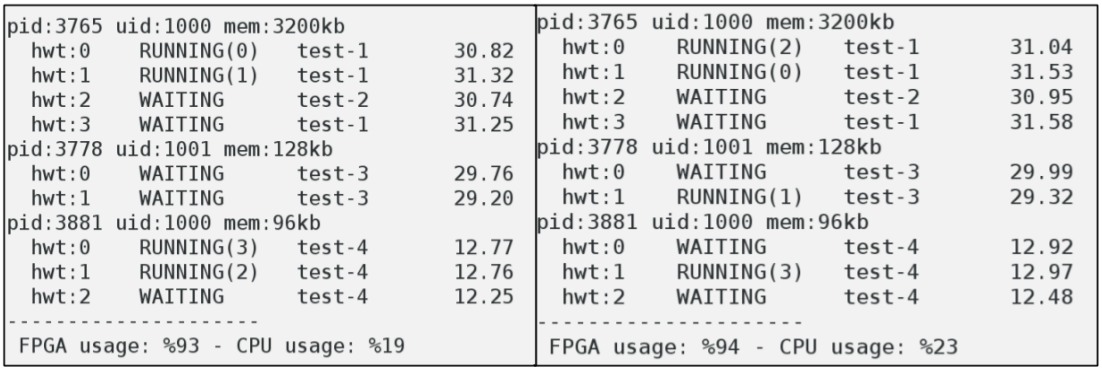
Sunucu modelinin testi 4 HWT alanı üzerinde farklı görev ve kullanıcılara ait HWT'lerin çalıştırılması şeklinde gerçekleştirilmiştir. Şekil 4.7'de farklı kullanıcılara ait iki görev çalıştırılması durumundaki ekran çıktısı verilmiştir. İlk görev (pid:3541), 1000 numaralı kullanıcıya, ikinci görev (pid:3583) 1001 numaralı kullanıcı tarafından çalıştırılmıştır.

```
pid:3541 uid:1000 mem:3200kb
  hwt:0    RUNNING(0)   test-1       27.77  %58
  hwt:1    RUNNING(3)   test-1       27.12  %64
  hwt:2    WAITING      test-1       27.14  %68
  hwt:3    RUNNING(2)   test-2       26.89  %64
pid:3583 uid:1001 mem:128kb
  hwt:0    RUNNING(1)   test-3       19.34  %60
  hwt:1    WAITING      test-3       19.59  %67
----------------------
 FPGA usage: %96 - CPU usage: %19
```

Şekil 4.7. Farklı kullanıcılara ait iki görevin eş zamanlı çalıştırılması

Örnekte görevler aynı anda başlatılmadığı için HWT'lerin çalışma süreleri eşit değildir. Tüm HWT'lerin son 4 saniyedeki çalışma oranları %58-%68 olduğundan olayı anlık olarak FPGA'den yararlanma süreleri birbirine yakındır.

Şekil 4.8'de üç görev çalıştırılan bir örneğin yaklaşık 2 saniye arayla alınan iki ekran çıktısı verilmiştir. Örnekte pid:3765 ve pid:3881 görevleri 1000 numaralı kullanıcı, pid:3778 görevi 1001 numaralı kullanıcı tarafından çalıştırılmaktadır. Tüm HWT'lerin son 4 saniyedeki çalışma oranları %36-%48 arasındadır.




Şekil 4.8. İki kullanıcıya ait 3 görev ve 9 HWT çalıştırılması durumunda 2 saniye arayla alınmış ekran çıktısı

Elde edilen test sonuçları ile farklı kullanıcılar ve görevlerin FPGA kaynaklarını eş zamanlı olarak kullanabileceği gösterilmiştir. Kullanılan zaman paylaşımı yöntemi kullanıcı ve görev numarasından bağımsız olarak çalıştığı için örneklerde tüm HWT'ler eşit oranda çalıştırılmıştır. Gelecekteki çalışmalarda Linux CFS'de kullanılan "Group Scheduling" özelliğine benzer bir yöntem geliştirilerek kullanıcı ve görev tabanlı zaman paylaşımı yapılması mümkündür.

# 5. SONUÇLAR VE ÖNERİLER

Yapılan çalışmada öncelikle temel alınan ReconOS işletim sistemindeki bellek erişimi performansının yetersiz olduğu tespit edilmiş ve bu açığın giderilmesi için yeni bir bellek arayüzü geliştirilmiştir. Geliştirilen bellek arayüzünün ReconOS'da kullanılan mevcut yöntem ve işletim sistemi kullanılmayan bir uygulama ile performans karşılaştırılması yapılmıştır. Performans verileri kullanılan platformun teorik limitleri ve donanım üreticisinin belirttiği verim değerleri ile karşılaştırılmıştır. Kaynak kullanımında %20'lik artışa karşın mevcut sisteme göre 12 kat performans artışı elde edilmiştir.

Yeniden yapılandırılabilir hesaplamanın çok kullanıcılı bilgisayar modelinde kullanılabilmesi için görevler ve HWT'ler arasında bellek soyutlaması çözümleri üzerinde çalışılmıştır. HWT'lerin bellek erişimini kısıtlamak için HWT'e özel sayfa tablosu yapısı tasarlanmıştır.

Birden fazla görevin ReconOS altyapısını kullanabilmesi için HWT Sunucu programı geliştirilmiştir. Görevlerin HWT Sunucusu'nu IPC üzerinden kullanması sağlanarak yetkili kullanıcı fonksiyonlarına erişmeden çalışabilmesi sağlanmıştır. IPC üzerinden kullanım ile görevler ve kullanıcılar arasında soyutlama ile hata izolasyonu sağlanmıştır.

Donanımda zaman paylaşımı problemini çözmek için zaman aralıklarının esnek olduğu bir preemptive çoklu-görev yöntemi geliştirilmiştir. Yöntem, zaman paylaşımı simülasyonu ile farklı durumlar kullanılarak test edilmiştir. Geliştirilen yöntem ile görevlerin HWT'leri yönetirken çalışma sürelerini bildirmesi durumunda verimli şekilde çalışabilmektedir. HWT'lerin yanlış kullanımı sonucunda verim düşerek zaman paylaşımı sürdürülebilmektedir.

Geliştirilen işletim sistemi modelinin testi için YYH yöntemi ile son kullanıcıya yönelik test uygulamaları geliştirilmiştir. YYH destekli video oynatıcı uygulaması ile sadece yazılıma göre 2-5 kat arasında performans artışı elde edilmiştir. Elde edilen

sonuçlar ile yeniden yapılandırılabilir hesaplamanın kişisel bilgisayar modelinde kullanılabileceği gösterilmiştir.

Gelecekteki çalışmalarda, bellek ara yüzüne HWT'ler arasında doğrudan iletişim özelliği eklenilmesiyle ana bellek bant genişliği kullanımında iyileştirme yapılması mümkündür. Ayrıca HWT Sunucu'nun çekirdek içerisine eklenmesi ile IPC kaynaklı gecikmeler ve CPU kullanımında iyileştirme yapılabilir.

## KAYNAKLAR


[1] Compton K., Hauck S., Reconfigurable computing: a survey of systems and software, *ACM Computing Surveys (CSUR)*, 2002, **34**(2), 171-210.

[2] Najjar W. A., Ienne P., Reconfigurable computing, *IEEE Micro*, 2014, **34**(1), 4-6.

[3] Crockett L. H., Elliot R. A., Enderwitz M. A., Stewart R. W., *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc*, 1st ed. Strathclyde Academic Media, Glasgow, Scotland, 2014.

[4] Andrews D., Operating systems research for reconfigurable computing, *IEEE Micro*, 2014, **34**(1), 54-58.

[5] Kalte H., Porrmann M., Context saving and restoring for multitasking in reconfigurable systems, *International Conference on Field Programmable Logic and Applications*, Tampere, Finland, 24-26 August 2005.

[6] Jovanovic S., Tanougast C., Weber S., A hardware preemptive multitasking mechanism based on scan-path register structure for FPGA-based reconfigurable systems, *Second NASA/ESA Conference on Adaptive Hardware and Systems*, Edinburgh, UK, 5-8 August 2007.

[7] Jozwik K., Tomiyama H., Honda S., Takada H., A novel mechanism for effective hardware task preemption in dynamically reconfigurable systems, *International Conference on Field Programmable Logic and Applications (FPL)*, Milan, Italy, 31 August-2 September 2010.

[8] Happe M., Traber A., Keller A., Preemptive hardware multitasking in ReconOS, *Lecture Notes in Computer Science*, 2015, **9040**, 79-90.

[9] Jozwik K., Tomiyama H., Edahiro M., Honda S., Takada H., Rainbow: an OS extension for hardware multitasking on dynamically partially reconfigurable FPGAs, *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 30 November-2 December 2011.

[10] Morales-Villanueva A., Gordon-Ross A., HTR: on-chip hardware task relocation for partially reconfigurable FPGAs, *Lecture Notes in Computer Science*, 2013, **7806**, 185-196.

[11] Andrews D., Peck W., Agron J., Preston K., Komp E., Finley M., Sass R., Hthreads: A hardware/software co-designed multithreaded RTOS kernel, *10th IEEE Conference on Emerging Technologies and Factory Automation*, Catania, Italy, 19-22 September 2005.

[12] Peck W., Anderson E., Agron J., Stevens J., Baijot F., Andrews D., Hthreads: A computational model for reconfigurable devices, *International Conference on Field Programmable Logic and Applications*, Madrid, Spain, 28-30 August 2006.

[13] Andrews D., Sass R., Anderson E., Agron J., Peck W., Stevens J., Baijot F., Komp E., Achieving programming model abstractions for reconfigurable computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2008, **16**(1), 34-44.

[14] So H. K. H., Brodersen R. W., Improving usability of FPGA-based reconfigurable computers through operating system support, *International Conference on Field Programmable Logic and Applications*, Madrid, Spain, 28-30 August 2006.

[15] So H. K. H., Brodersen R., A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH, *ACM Transactions on Embedded Computing Systems (TECS)*, 2008, **7**(2), 1401-1428.

[16] So H. K. H., Brodersen R., Runtime filesystem support for reconfigurable FPGA hardware processes in BORPH, *16th International Symposium on Field-Programmable Custom Computing Machines*, Stanford, USA, 14-15 April 2008.

[17] So H. K. H., BORPH: An operating system for FPGA-based reconfigurable computers, PhD Thesis, University of California, Berkeley, 2007.

[18] Iturbe X., Benkrid K., Hong C., Ebrahim A., Torrego R., Martinez I., Arslan T., Perez J., R3TOS: a novel reliable reconfigurable real-time operating system for highly adaptive, efficient, and dependable computing on FPGAs, *IEEE Transactions on Computers*, 2013, **62**(8), 1542-1556.

[19] Lubbers E., Platzner M., ReconOS: An RTOS supporting hard-and software threads, *International Conference on Field Programmable Logic and Applications*, Amsterdam, Netherlands, 27-29 August 2007.

[20] Lübbers E., Platzner M., Communication and Synchronization in Multithreaded Reconfigurable Computing Systems, *8th International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, USA, 14-17 July 2008.

[21] Lubbers E., Platzner M., A portable abstraction layer for hardware threads. *International Conference on Field Programmable Logic and Applications*, Heidelberg, Germany, 8-10 September 2008.

[22] Lubbers E., Platzner M., Cooperative multithreading in dynamically reconfigurable systems, *International Conference on Field Programmable Logic and Applications*, Prague, Czech Republic, 31 August-2 September 2009.

[23] Lübbers E., Platzner M., ReconOS: Multithreaded programming for reconfigurable computers, *ACM Transactions on Embedded Computing Systems (TECS)*, 2009, **9**(1), 801-833.

[24] Agne A., Happe M., Keller A., Lubbers E., Plattner B., Platzner M., Plessl C., ReconOS: An operating system approach for reconfigurable computing, *IEEE Micro*, 2014, **34**(1), 60-71.

[25] Agne A., Platzner M., Plessl C., Happe M., Lübbers E., ReconOS, Editörler: Koch D., Hannig F., Ziener D., *FPGAs for Software Programmers*, 1st ed., Springer International Publishing, 227-244, 2016.

[26] ReconOS - operating system approach for reconfigurable computing, http://reconos.de, (Ziyaret Tarihi: 22 Mayıs 2017).

[27] ReconOS - GitHub, http://github.com/ReconOS (Ziyaret Tarihi: 22 Mayıs 2017).

[28] Altuncu M. A., Guven T., Becerikli Y., Sahin S., Real-time system implementation for image processing with hardware/software co-design on the Xilinx Zynq platform, *International Journal of Information and Electronics Engineering*, 2015, **5**(6), 473-477.

[29] IEEE SA 1003.1-2008, Portable Operating System Interface (POSIX(R)), *IEEE Standard for Information Technology*, New York, 2008.

[30] Zynq-7000 All Programmable SoC Technical Reference Manual, https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, (Ziyaret Tarihi: 9 Kasım 2016).

[31] Jones M., Inside the Linux 2.6 Completely Fair Scheduler, https://www.ibm.com/developerworks/library/l-completely-fair-scheduler/, (Ziyaret Tarihi: 22 Mayıs 2017).

[32] AMBA AXI and ACE Protocol Specification, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html, (Ziyaret Tarihi: 15 Mart 2016).

[33] AMBA AXI4-Stream Protocol Specification, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0051a/index.html, (Ziyaret Tarihi: 15 Mart 2016).

[34] Vivado Design Suite User Guide - High-Level Synthesis, https://www.xilinx.com/support/documentation/application_notes/xapp1209-designing-protocol-processing-systems-hls.pdf, (Ziyaret Tarihi: 22 Mayıs 2017).

[35] AXI Interconnect (v1.06.a), https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v1_06_a/ds768_axi_interconnect.pdf, (Ziyaret Tarihi: 28 Eylül 2016).

[36] Lucero J., Arbel Y., Designing High-Performance Video Systems with the Zynq-7000 All Programmable SoC, https://www.xilinx.com/support/documentation/application_notes/xapp1205-high-performance-video-zynq.pdf, (Ziyaret Tarihi: 3 Kasım 2016).

[37] Linux Programmer's Manual - UNIX(7), http://man7.org/linux/man-pages/man7/unix.7.html, (Ziyaret Tarihi: 22 Mayıs 2017).

[38] System V IPC, http://www.tldp.org/LDP/lpg/node21.html, (Ziyaret Tarihi: 22 Mayıs 2017).

[39] Zynq Linux USB Device Driver, http://www.wiki.xilinx.com/Zynq+Linux+USB+Device+Driver, (Ziyaret Tarihi: 22 Mayıs 2017).

[40] Xilinx/linux-xlnx, https://github.com/Xilinx/linux-xlnx/tree/xilinx-v2014.2, (Ziyaret Tarihi: 22 Mayıs 2017).

[41] Simple Framebuffer, https://www.kernel.org/doc/Documentation/devicetree/bindings/display/simple-framebuffer.txt, (Ziyaret Tarihi: 22 Mayıs 2017).

[42] Linaro Releases, https://releases.linaro.org/archive/14.04/ubuntu/saucy-images/nano/, (Ziyaret Tarihi: 22 Mayıs 2017).

[43] FFmpeg - A complete, cross-platform solution to record, convert and stream audio and video, https://ffmpeg.org/, (Ziyaret Tarihi: 13 Şubat 2017).

[44] ISO/IEC 14496-10:2010, Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding, *International Organization for Standardization*, Geneva, 2010.

[45] rfc638 - VP8 Data Format and Decoding Guide, https://tools.ietf.org/html/rfc6386, (Ziyaret Tarihi: 22 Mayıs 2017).

[46] Jack K., *Video Demystified: A Handbook for the Digital Engineer*, 4rd ed., Elsevier Inc., Massachusetts, 2005.

[47] URL-1, https://en.wikipedia.org/wiki/File:Yuv420.svg, (Ziyaret Tarihi: 16 Mayıs 2017).

[48] User space mappable dma buffer device driver for Linux, https://github.com/ikwzm/udmabuf, (Ziyaret Tarihi: 22 Mayıs 2017).

## KİŞİSEL YAYINLAR VE ESERLER

[1] Küçükmanisa A., **Güven T.**, Urhan O., Gerçek Zamanlı Gömülü Şeritten Ayrılma Tespit Sistemi, *GömSis 2014 - Gömülü Sistemler ve Uygulamaları Sempozyumu*, İstanbul, Türkiye, 4-5 Aralık 2014.

[2] Altuncu M. A., **Guven T.**, Becerikli Y., Sahin S., Real-time system implementation for image processing with hardware/software co-design on the Xilinx Zynq platform. *International Journal of Information and Electronics Engineering*, 2015, **5**(6), 473-477.

[3] Özcan H., **Güven T.**, Altuncu M. A., Savaş B. K., Duman O., Şahin S., A Smart Tracking Systems for Museums, *Global Journal on Technology*, Antalya, Türkiye, 12 - 14 Mayıs 2016.

[4] Özcan H., **Güven T.**, Altuncu M. A., Savaş B. K., Şahin, S., Duman O., Design and Implementation of a Content Delivery Architecture For Museums, *International Conference on Research in Education and Science (ICRES)*, Kuşadası, Türkiye, 18-21 Mayıs 2017.

[5] **Guven T.**, Sahin S., StreamIF - High Performance Memory Interface for Reconfigurable Operating Systems on Xilinx ZedBoard Platform, *International Journal of Computer Applications*, 2017, **168**(6), 1-5.

## ÖZGEÇMİŞ

Taner Güven 1989'da Almanya'da doğdu. İlköğretim ve Lise öğrenimi Tekirdağ'da tamamladı. 2008 yılında girdiği Kocaeli Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2013 yılında mezun oldu. Aynı yıl içinde Kocaeli Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladı.