Petalinux Tabanlı CORDIC HW/SW CoDesign

Recep GEMALMAZ, Berk TUNÇ, Ogün Berat GÜRSES, Damla Su KARADOĞAN
Fenerbahçe Üniversitesi
Bilgisayar Mühendisliği
İstanbul, Türkiye
e-mail: { recep.gemalmaz, berk.tunc, ogun.gurses, damla.karadogan }@stu.fbu.edu.tr,

***Özetçe*— Donanım hızlandırıcı olarak Xilinx'in CORDIC (COordinate Rotation DIgital Computer) IP'si kullanıldığı ve işlemcinin hesap yükünü donanım hızlandırıcıya aktardığı bir proje yapılacaktır. Bu donanım tasarımı PL tarafında hazırlandıktan sonra Petalinux ortamından veriler aktarılacaktır.**

Anahtar Kelimeler — FPGA, CPU, PYNQ, CORDIC

***Abstract*— The project where Xilinx, as the hardware developer, transferred the budget hardware accelerator project of the CORD (Coordinate Rotation DIgital Computer) IP training and hardware. Once this hardware design is prepared for PL, Petalinux will be handed over.**

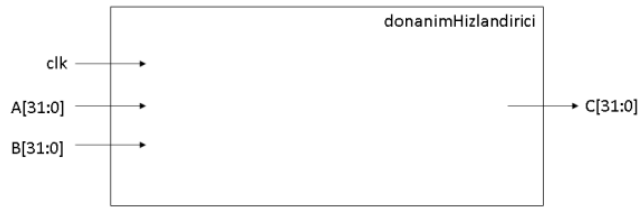
Keywords — FPGA, CPU, PYNQ, CORDIC

I. GİRİŞ

Geçtiğimiz dönem SOC Tasarımı dersinde CORDIC IP'si kullanılarak işlemcinin hesap yükünün donanım hızlandırıcıya aktarıldığı projenin üstüne donanımın PL tarafında hızlandırılması yapılmıştı. Ama bu PL tarafında yapılan donanım işletim sistemi olmadan kontrol eden bir tasarımdı. Bu dönemki projemizde PL tarafı aynı kalmasına karşın PS tarafında bir işletim sistemi varken gerekli işlemleri gerçekleştirmektir. Proje kapsamda dışarıdan verilecek sayıya göre donanım hızlandırıcı gidip, donanım hızlandırıcının sonucu hesaplayıp kullanıcıya geri vermesidir. Bu kapsamda C# kodu, UART'dan gelen veriyi kabul edecek, bunu C# kodu aracılığı ile okuyup HP portundan PL kısmına basıp modül çıktılarını okuyup sonucu UART arayüzü ile dışarı basacaktır.

II. SİSTEM MİMARİSİ

ZYNQ mimarisine sahip olan PYNQ geliştirme kartı üzerinde proje geliştirilmiştir. ZYNQ'in PS bölümü, tasarlanan özel bir modüle verileri besleyip, sonucunu alacak şekilde tasarlanmıştır. Özel modülün giriş ve çıkışları aşağıda verilmektedir.



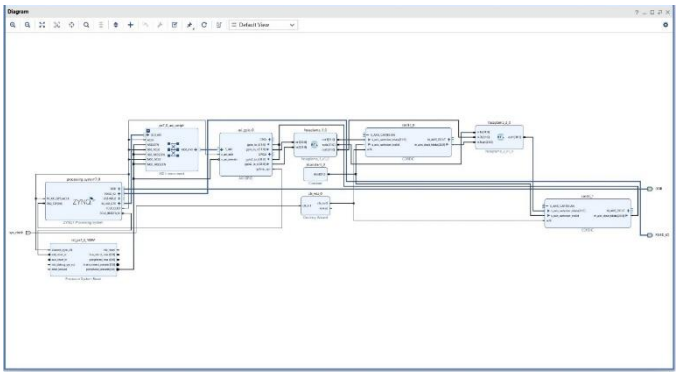
Bu modülde görselde de görüldüğü üzere clk, referans clock sinyali, A[31:0] ve B[31:0] giriş sinyalleri, C[31:0] çıkış sinyalleri bulunmaktadır.

Bu modül içerisinde aşağıda verilen aritmetik işlemi yapan donanım bulunur.

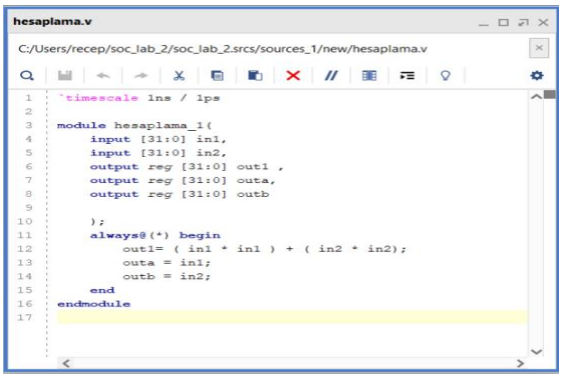
$$C = \sqrt{\left(\sqrt{A^2 + B^2}\right) + (A\ x\ B)}$$

SQRT işlemi için CORDIC IP'si kullanılmıştır. CORDIC IP'sinde bulunan SQRT fonksiyonu için unsigned integer seçeneği seçilmiştir.

AXI GPIO IP'si ile tasarlanan modülün giriş ve çıkışlarına bağlanmıştır. PS tarafında A ve B sayıları örnek olarak 10 ve 20 olarak ayarlanıp giriş verilip, sonuç doğru üretildiğinde geriye değer alınması gözlenmiştir.



Öncelikle ZYNQ IP'si eklenir daha sonra Gpio IP eklenir. Yapmaya çalıştığımız şey Gpio IP'sinin çıkışlarından iki adet sayı göndermek, gönderilen sayıları hesaplama modülü içerisinde istenen hesaplama adımlarından 1. kısmını yapmaktır. Oluşturulan sayıları Gpio IP'si ile hesaplama modülüne yolladık.


hesaplama.v

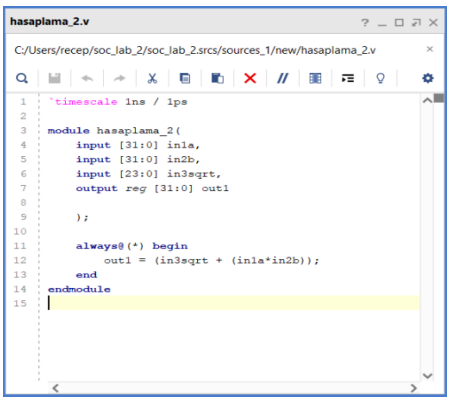
C:/Users/recep/soc_lab_2/soc_lab_2.srcs/sources_1/new/hesaplama.v

```
1  `timescale 1ns / 1ps
2
3  module hesaplama_1(
4      input [31:0] in1,
5      input [31:0] in2,
6      output reg [31:0] out1 ,
7      output reg [31:0] outa,
8      output reg [31:0] outb
9
10     );
11     always@(*) begin
12         out1= ( in1 * in1 ) + ( in2 * in2);
13         outa = in1;
14         outb = in2;
15     end
16 endmodule
17
```

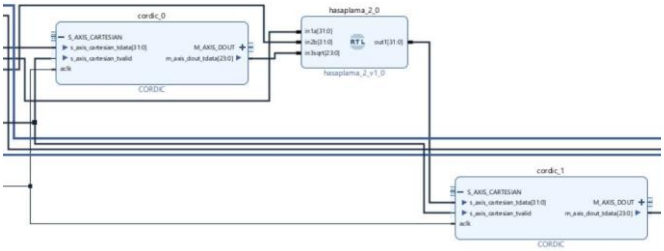
Hesaplama modülünde input ve outputlar görülmektedir. Bu modülde iki sayının da karelerini alıp toplanır ve out1

çıkışına beslenir. Başlangıçtaki input değerlerini de başka modülde kullanmak için 2 farklı çıkışa beslenmiştir.

Diyagramda da görüldüğü üzere yapılan işlem out1 çıkışından kök alma işlemini yapan Cordic IP'sine gitmektedir. Outa ile Outb çıktıları ise tekrar hesaplama yapmak için hesaplama2 modülüne gider.

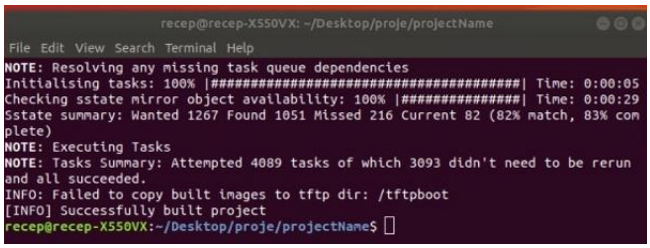


Hesaplama_2 modülü input olarak Cordic'den çıkan kök sonucunu ve önceki modülden gelen Outa ve Outb'yi alınır. Bu kısımda yapılan işlem; önceki modülden gelen iki sayıyı çarpıp onları kökü alınmış değer ile toplamaktır.

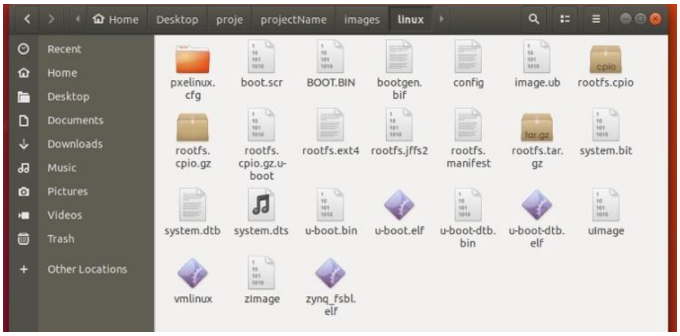


Hesaplama_2 modülünde işlem tamamlandıktan sonra elde edilen çıktı yeni oluşturulan yine kök alma için kullanılacak olan Cordic IP'sinin içine gider ve son elde edilen sayının kökü alınır. 2. oluşturduğumuz Cordic IP'sinin çıktısı ise Gpio IP'sinin girişlerine besleniyor. Gpio IP'sinin girişleri değiştiği andan itibaren interrupt üretilir ve bu interrupt ZYNQ Ip'sine gelir. Bu dallanma oluşmuş olur.

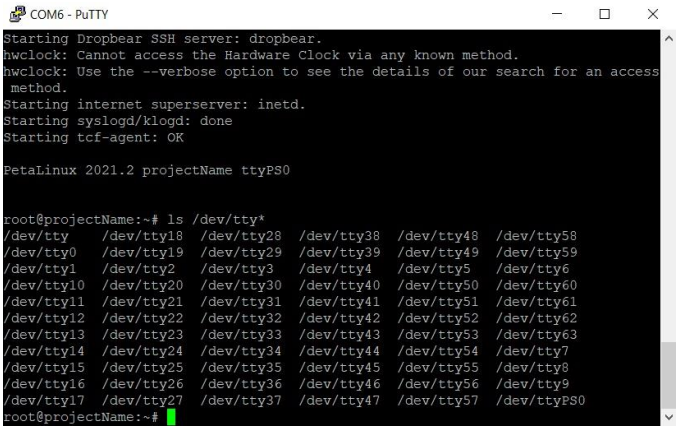
Oluşturulan blok tasarımın ardından derlendiğinde ortaya ortaya çıkan .xsa uzantılı dosya linux ortamında yapıştırılır. Linux ortamında terminal ekranı açılarak Petalinux Tool'ları çalıştırılır.



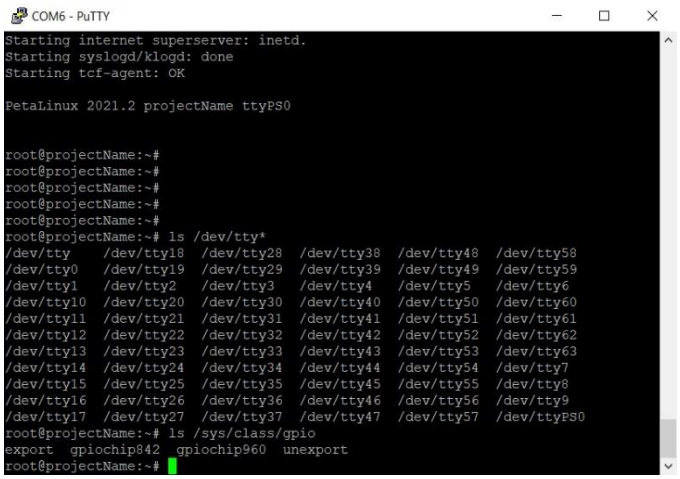
Linux işletim sisteminde bir proje oluşturulur ve xsa dosyası kullanılarak proje ayarlatılır. Petalinux'u oluştururken ssd karttan ayağa kalkacak şekilde build edilir. Ve image dosyaları elde edilir.



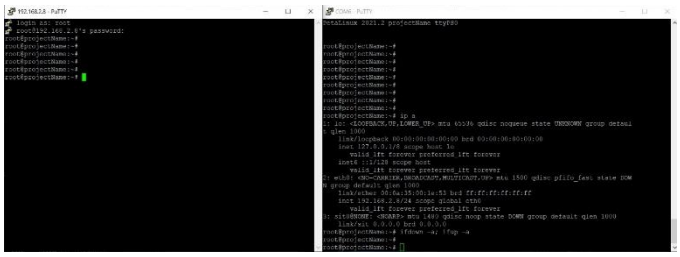
1. partiton'a koymak için boot.bin ve image.ub dosyasını ekledik ve 2. Patiton'da bir linux dağıtımı olan Yocto'nın kaynak dosyalarını ekledik (rootfs.tar.gz). Ayrıca gcc derleyicisi de koda eklendi.



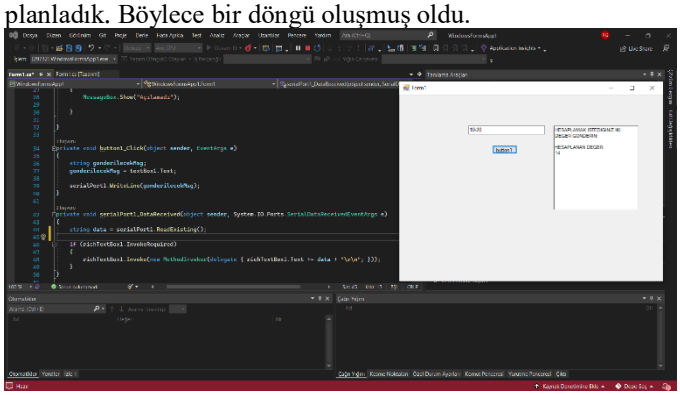
Ardından PS tarafındaki UART tty/PS0'ın var olup olmadığı kontrolü yapılır. Çünkü PS0 olmadığı takdirde PS tarafına konulan UART'lar sürülemez.



Daha sonra GPİO IP'ye ulaşılıp ulaşılamadığı kontrol edilir. Komut yazıldıktan sonra çıkan mesajda Gpio'a ulaşılabildiği gözlemlemiştir.



Proje kapsamında bilgisayar ve çip arasındaki haberleşmenin oluşturduğu engel durumunu kaldırmak için Ethernet ara yüzü kullanılmıştır. Bu yüzden çipe "ip.a" kodu ile statik ip adresi atadık. Çipin statik ip adresi (192.168.2.8) ve bilgisayarın Ethernet (192.168.2.9) çıktısına ip adresi atması yapıldı. Bu kapsamda proje planımıza göre; çip verilerini göndereceği dataları bilgisayarın Ethernet arayüzü ile gönderecek ve biz de yazdığımız C# kodu aracılığı ile bilgisayardan karta verileri COM-6 ile göndermesini planladık. Böylece bir döngü oluşmuş oldu.



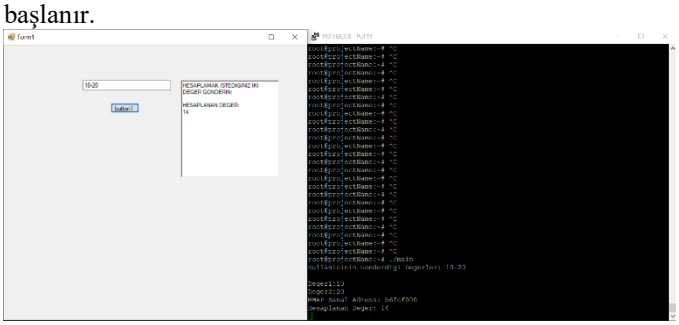
PYNQ kartı ssd'den ayağı kalkmıştı. İlk ayağa kalktığında Putty'den açtığımızda normal olarak COM-6'dan kalkar. Bunu değiştirebilmemiz için bir tane daha Putty ekranı açıp 192.168.2.8 adresi yazılır. Proje bu adresin yazılmasının ardından diğer putty penceresinde akmaya başlar. Comport arayüzünü yani diğer putty ekranını kapattığımız zaman artık çip verilerini bilgisayara Ethernet üzerinden göndermeye başlar. Comport arayüzü boşa çıkar.

III. KULLANILAN YAZILIM

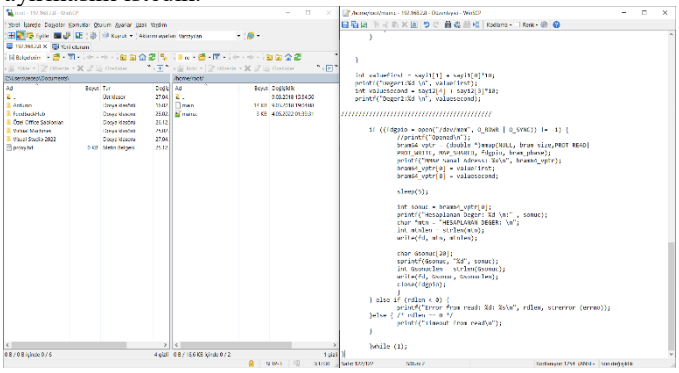
Gerekli ayarlamaların yapılmasının ardından gerekli spesifikasyonlara göre C# kodu yazımına başlanır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            serialPort1.BaudRate = 115200;
            serialPort1.PortName = "COM6";

            serialPort1.Open();
            if (serialPort1.IsOpen == true)
            {
                MessageBox.Show("Açıldı");
            }
            else
            {
                MessageBox.Show("Açılamadı");

            }

        }

        private void button1_Click(object sender, EventArgs e)
        {
            string gonderilecekMsg;
            gonderilecekMsg = textBox1.Text;

            serialPort1.WriteLine(gonderilecekMsg);
        }

        private void serialPort1_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            string data = serialPort1.ReadExisting();
            if (richTextBox1.InvokeRequired)
            {
                richTextBox1.Invoke(new MethodInvoker(delegate { richTextBox1.Text += data + "\r\n"; }));
            }
        }
    }
}
```

Kod içerisinde com-6'ya bağlanmaya çalışmaktadır. Haberleşmenin 115200 bound-rate ile yapılacağını söylemektedir. Com-6'nın açılıp açılamadığının kontrolü yapılır. Eğer açılamadıysa port boştadır. Açıldı uyarısını aldıktan sonra fonksiyon içerisinde com-6'dan veri transferine başlanır.



Oluşturduğumuz protokolde veriler transfer edilirken kullanıcı iki değer gireceği için girdiği sayıları "-" ile ayırmasını istedik.



Daha sonra bizim çipten bilgisayara veri gönderimini sağlamak için GPIO IP'sine ulaşmamıza ihtiyacımız vardır. Bunun için içeride gcc derleyicisini kullanarak C kodu çalıştırmamız gerekmektedir. Gcc derleyicisine ulaşırken Winscp uygulaması kullanılır. Winscp'ye ethernetten 192.168.2.8 yazılarak ulaşılır. İçerisinde PS bölümüne kodlarımızı çalışan petalinux'ün içerisine atmamızı sağlayan bir araçtır. Bu aracı kullanarak mainin içerisine yazdığımız kodu yapıştırarak kodu kartın içerisine gönderiyoruz.

Kodun içerisinde kullanacağımı Gpio ip'sinin fiziksel adresini öğrenmemiz gerekmektedir.

```
root@projectName:~# ls/sys/class/gpio
-sh: ls/sys/class/gpio: No such file or directory
root@projectName:~# ls /sys/class/gpio
export  gpiochip842  gpiochip960  unexport
root@projectName:~# more /sys/class/gpio/gpiochip842/label
zynq_gpio
root@projectName:~# more /sys/class/gpio/gpiochip960/label
41200000.gpio
root@projectName:~#
```

Bunun için kod çalıştırılır. Bu sayede Gpio ip'nin başlangıç adresi öğrenilmiş olunur.

```
char karakter[255];
int sayi1[255];
int sayi2[255];
int i;
int j;

int main()
{
    unsigned int gpio_size = 0x8000;
    off_t gpio_pbase = 0x41200000;
    long long *gpio64_vptr;
    int fdgpio;

    char *portname = TERMINAL;
    int fd;
    int wlen;
    char *xstr = "HESAPLAMAK ISTEDIGINIZ IKI DEGER GONDERIN: \n";
    int xlen = strlen(xstr);
fd = open (portname, O_RDWR | O_NOCTTY| O_SYNC);
if (fd < 0) {
    printf("Error opening %s: %s\n", portname, strerror(errno));
    return 1;
}
    wlen = write(fd, xstr, xlen); //PYNQ'den PC'ye bilgilendirme mesajı bastık.
    if (wlen != xlen){
        printf("Error from write: %d, %d\n", wlen, errno);
    }
```

Gpio'nun fiziksel adresini öğrendikten ve bunu koda ekledikten sonra 32 bitlik sayı okumak istediğimizi belirtiriz. Ethernet araryüzünü kullanarak Pynq'ten bilgisayara bilgilendirme mesajı bastırılır. Yapılan kodlama oluşturulan pointer ile mmap sanal adresine (0 ve 8. Adresler) değer atanır.

```
do{
    unsigned char buf[80];
    int rdlen;

    rdlen = read (fd, buf, sizeof(buf) -1); //Kullanıcının PC'den gönderdiği değerleri Array'e yazdık.
    if (rdlen > 0){

    buf[rdlen]=0;
    printf("Kullanicinin Gonderdigi Degerler: %s \n", buf);

    j = strlen(buf);
    int kontrol = 0;

    for (i = 0; i < j; i++) //Buf arayında aralarında tire "-" olacak şekilde verileri aldık.
    {
        if(buf[i]== '-'){ //Bu char "-" ile gönderilen iki haklı sayıya array'de ulaştık.
            kontrol = 1;
        } else{
            if(kontrol == 0){
                sayi1[i] = buf[i] - '0';
            }
            else {
                sayi2[i] = buf[i] - '0';
            }
        }
    }

    int valuefirst = sayi1[1] + sayi1[0]*10; //Char olarak gelen bu değerleri Integer haline dönüştürdük.
    printf("Deger1:%d \n", valuefirst);
    int valuesecond = sayi2[1] + sayi2[0]*10;
    printf("Deger2:%d \n", valuesecond);
```

```
if ((fdgpio = open("/dev/mem", O_RDWR | O_SYNC)) != -1) {

    gpio64_vptr = (long long *)mmap(NULL, gpio_size,PROT_READ|
    PROT_WRITE, MAP_SHARED, fdgpio, gpio_pbase); //MMAP Komutu ile GPIO'nun sanal adresine ulaştık.
    printf("PSGP Sanal Adresi: %x\n", gpio64_vptr); //Bu sanal adres sayesinde GPIO giriş ve çıkışlarına veri yazdık.
    *(gpio64_vptr)  = valuefirst; //gpio 0.adresine kullanıcının gönderdiği 1. değeri gönderdik.
    *(gpio64_vptr + 1) = valuesecond; //gpio 8.adresine kullanıcının gönderdiği 2. değeri gönderdik.

    long sonuc = *(gpio64_vptr + 1); //gpio 8.adresinden PL tarafında hesaplanan değeri aldık.
    printf("Hesaplanan Deger: %d \n:" , sonuc);
    char *mtn = "HESAPLANAN DEGER: \n";
    int mtnlen = strlen(mtn);
    write(fd, mtn, mtnlen);

    char Gsonuc[20];
    sprintf(Gsonuc, "%d", sonuc);
    int Gsonuclen = strlen(Gsonuc);
    write(fd, Gsonuc, Gsonuclen); // Uart arayüzü ile PYNQ'ten PC'ye hesaplanan değeri gönderdik.
    close(fdgpio);
    }
```

Daha sonra kullanıcıdan bilgisayar üzerinden (C#)' gönderdiği değerler buffer array'e yazılır. Buffer arrayde sayıların arasında tire olacak şekilde veriler alınır. Char olarak gelen veriler daha sonrasında Gpio IP integer ile çalıştığı için integer değere dönüştürülür.

Valuefirst ve valuesecond gpio ip'sinin girişlerine gönderilir. Petalinux fiziksel adreslerle değil sanal adreslerle çalışan bir işletim sistemi olduğu için mmap komutuna yukarıda belirlediğimiz Gpio Ip'sinin fiziksel adresini veririz ve sanal adresi elde ederiz. Bu sayede gpio'nun sanal giriş ve çıkışlarına veri yazarız. Daha sonra gpio'nun 0. adresine kullanıcının gönderdiği birinci değeri, 8. Adresine de ikinci değeri gönderdik. İkinci adrese ulaşmak için "+1" yazdık. Hesaplamaları yaptıktan sonra Gpio'nun 8. adresinden PL tarafından hesaplanan değeri geri aldık. Yani 10 ve 20 gönderdiğimiz değerlerin adresini okuduğumuzda sonuçta 14'ü elde ederiz.

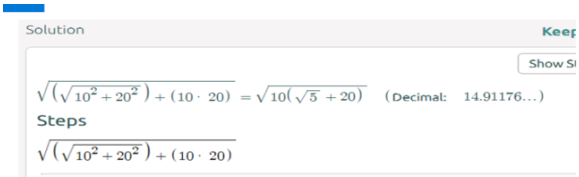
IV. SONUÇLAR

Projemizde C# kodu ile verileri bilgisayardan PYNQ kartına gönderilmesi istenmiştir. Fakat PYNQ kartı, COM-6'yı kart dinlediği için aynı zamanda yazdığımız C# kodu COM-6'yı dinleyememektedir. Bu sebepten ötürü biz çipten bilgisayara verileri Ethernet arayüzü ile gönderip, bilgisayardan çipe verileri seriport arayüzü ile göndersin olarak kurguladık.

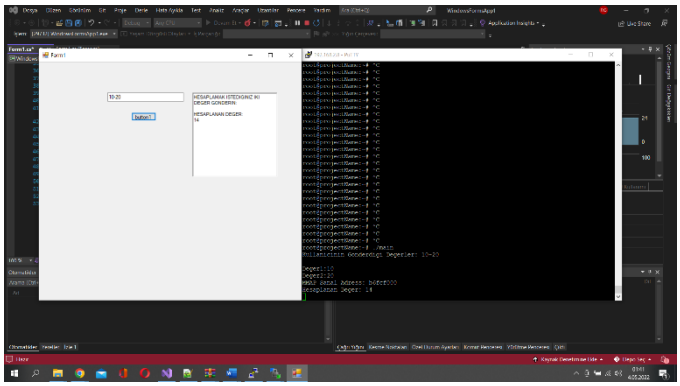
Bu kapsamda kodlamamızın sonuçlarını incelersek;

```
root@projectName:~#
root@projectName:~# devmem 0x41200000 32 0xA
root@projectName:~# devmem 0x41200008 32 20
root@projectName:~# devmem 0x41200008 32
0x0000000E
root@projectName:~#
```

C#'da kodu çalıştırmadan da linux'da komut satırı arayüzünde verileri komutlarla transfer edebiliriz. Yani fiziksel adreste de çalışabiliriz. Devmem komutu fiziksel adrese ulaşabilmektedir. Gpio ip'sinin fiziksel adresinin 41200000 olduğu belirttikten sonra hexadecimal olarak A=10 değeri, 41200008 adresine ise 20 değeri gönderilir. Adres sonrasında tekrar okunduğunda E=14 sonucu fiziksel adresten okunmuş olur. İşlemin doğruluğunu kontrol edersek de 14 çıktığını gözlemlemiş oluruz.



./main'i çalıştırarak C kodunu derlediğimizde pynq'ten bilgisayara "hesaplamak istediğiniz 2 değeri gönderin" mesajını C#'dan gönderir. C# com-6'yı okur ve kullanıcı karta com-6 aracılığı ile 10 ve 20 sayılarını gönderir. Alttaki görselde sağ ekranda kart, sol ekranda bilgisayar görünmektedir.



Kart kullanıcının gönderdiği değerleri okur ve değerleri integera çevirir. Gpio'nun sanal adresini öğrendikten sonra kodlamanın devamında hesaplama yapıp hesaplanan değerin 14 olduğunu öğrendik ve değeri putty'den (karttan bilgisayara) geri gönderdik.

PROJE EKİBİ

Berk TUNÇ: 04.02.2000 yılında Yalova ili Merkez ilçesinde doğdu. 2018 yılında Şehit Osman Altınkuyu Anadolu Lisesi'nden mezun oldu. Şu anda Fenerbahçe Üniversitesi'nde Bilgisayar Mühendisliği bölümünde bölümünde eğitim almaktadır. mySQL, HTML, C, C++, Verilog, Sys-Verilog ve Python ile ilgilenmektedir.

RECEP GEMALMAZ: 16.10.2000 tarihinde Kadıköy'de dünyaya geldi. 2018 yılında Alparslan Anadolu Lisesi'nden Mezun oldu. Şu anda Fenerbahçe Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimi almakta. Java, C, C++ ve Pyhton dillerinde bilgili. Android Programlama, Verilog ve Sys-Verilog ile ilgilenmektedir.

Ogün Berat GÜRSES: 10.11.2000 Sakarya ili Adapazarı ilçesinde doğdu. 2018 yılında Sakarya Anadolu Lisesi'nden mezun oldu. Şu anda Fenerbahçe Üniversitesi'nde Bilgisayar Mühendisliği bölümünde eğitim almaktadır. C, C++, mySQL, Verilog, Sys-Verilog ve Python ile ilgilenmektedir.

Damla Su KARADOĞAN, 11.02.2001 yılında doğdu. 2019 yılında Özel Envar Anadolu Lisesinden mezun oldu. Şu anda Fenerbahçe Üniversitesinde Endüstri Mühendisliği bölümünde lisans eğitimini almakta ve Bilgisayar Mühendisliğinde ÇAP eğitimi alıyor. Öğrenci numarası, 190302016.

REFERANS DOSYALAR

(26) FENERBAHÇE ÜNİVERSİTESİ-PETALINUX TABANLI CORDIC HW/SW CODESIGN - YOUTUBE

brktnc/Petalinux-Tabanl-CORDIC-HW-SW-CoDesign (github.com)

KAYNAKLAR

[1]http://www.levent.tc/courses/embedded-systems