باسمه تعالی

هم‌طراحی سخت‌افزار و نرم‌افزار

(سال تحصیلی ۱۴۰۰-۱۴۰۱، نیمسال اول)

تمرین ششم: Implementing a System Using Hardware/Software Codesign

هدف از انجام این تمرین شبیه‌سازی یک سیستم متشکل از سخت‌افزار و نرم‌افزار با استفاده از محیط شبیه‌سازی GEZEL، می باشد. به این منظور قصد داریم، محاسبه سینوس و کسینوس به وسیله الگوریتم CORDIC را به صورت هم‌طراحی سخت افزار و نرم افزار و با استفاده از پردازنده ARM، شبیه‌سازی نماییم.

الگوریتم CORDIC(Coordinate Rotation Digital Computer) یک روش کارآمد برای پیاده‌سازی سخت‌افزاری توابع بر مبنای محاسبه با استفاده از دوران‌های متعدد حول مبدأ مختصات می‌باشد. در زیر الگوریتم محاسبه سینوس و کسینوس با استفاده از این الگوریتم به زبان C نشان داده شده است.

```c
#include <stdio.h>
// Table of arctan's for use with CORDIC algorithm
// Store in decimal representation N = ((2^16)*angle_deg) / 180
#define ATAN_TAB_N 16
int atantable[ATAN_TAB_N] = {   0x4000,   //atan(2^0) = 45 degrees
                                0x25C8,   //atan(2^-1) = 26.5651
                                0x13F6,   //atan(2^-2) = 14.0362
                                0x0A22,   //7.12502
                                0x0516,   //3.57633
                                0x028B,   //1.78981
                                0x0145,   //0.895174
                                0x00A2,   //0.447614
                                0x0051,   //0.223808
                                0x0029,   //0.111904
                                0x0014,   //0.05595
                                0x000A,   //0.0279765
                                0x0005,   //0.0139882
                                0x0003,   //0.0069941
                                0x0002,   //0.0035013
                                0x0001    //0.0017485
};
```

```
// Inputs:
//  theta = any (integer) angle in degrees
//  iterations = number of iterations for CORDIC algorithm, up to 16
int main(){
  int s, x1, x2, y, i, quadAdj, shift;
  int *atanptr = atantable;
  int theta;
  char iterations;
  int sin_result,cos_result;

  theta = 30;
  iterations = 12;

  //Limit iterations to number of atan values in our table
  iterations = (iterations > ATAN_TAB_N) ? ATAN_TAB_N : iterations;
  //Shift angle to be in range -180 to 180
  while(theta < -180) theta += 360;
  while(theta > 180) theta -= 360;
  //Shift angle to be in range -90 to 90
  if (theta < -90){
    theta = theta + 180;
    quadAdj = -1;
  } else if (theta > 90){
    theta = theta - 180;
    quadAdj = -1;
  } else{
    quadAdj = 1;
  }

  //Shift angle to be in range -45 to 45
  if (theta < -45){
    theta = theta + 90;
    shift = -1;
  } else if (theta > 45){
    theta = theta - 90;
    shift = 1;
  } else{
    shift = 0;
  }

  //convert angle to decimal representation N = ((2^16)*angle_deg) / 180
  if(theta < 0){
    theta = -theta;
    theta = ((unsigned int)theta<<10)/45;   //Convert to decimal representation
    theta = (unsigned int)theta<<4;
    theta = -theta;
  } else{
    theta = ((unsigned int)theta<<10)/45;   //Convert to decimal representation
    theta = (unsigned int)theta<<4;
  }

  //Initial values
  x1 = 0x4DBA;    //this will be the cosine result,initially the number 0.60725
  y = 0;          //y will contain the sine result
  s = 0;          //s will contain the final angle
  for (i=0; i<iterations; i++){
    if(theta < s){
      x2 = x1 + (y >> i);
      y = y - (x1 >> i);
      x1 = x2;
      s -= atanptr[i];
    } else{
      x2 = x1 - (y >> i);
      y = y + (x1 >> i);
      x1 = x2;
      s += atanptr[i];
    }
  }
```

```
90    //Correct for possible overflow in cosine result
91    if(x1 < 0) x1 = -x1;
92    //Push final values to appropriate registers
93    if(shift > 0){
94      sin_result = x1;
95      cos_result = -y;
96    } else if (shift < 0){
97      sin_result = -x1;
98      cos_result = y;
99    } else {
100     sin_result = y;
101     cos_result = x1;
102   }
103
104   //Adjust for sign change if angle was in quadrant 3 or 4
105   sin_result = quadAdj * sin_result;
106   cos_result = quadAdj * cos_result;
107
108   // printf("sin_result:%d\tcos_result:%d\t",sin_result,cos_result);
109
110   return 0;
111 }
```

۱. در این پروژه باید الگوریتم فوق را به صورت هم‌طراحی سخت‌افزار و نرم‌افزار، در محیط شبیه‌سازی GEZEL با استفاده از پردازنده ARM، پیاده‌سازی نمایید. شبیه‌ساز GEZEL امکان استفاده از یک پردازنده ARM به صورت یک ipcore و تعریف ارتباطات مورد نیاز بین سخت‌افزار تولید شده و کد نرم‌افزاری را فراهم می‌نماید. مطابق مثال ارائه شده در بخش 13.3 از کتاب مرجع، سیستم پیاده‌سازی شده با استفاده از زبان GEZEL شامل قسمت‌های ARM Core، ARM Interfaces، Hardware Kernel و Hardware Interface می‌باشد. قسمت Hardware Kernel همان کد سخت‌افزار است و Hardware Interface ارتباط ورودی و خروجی‌های سخت‌افزار با ارتباطات تعریف شده در قسمت ARM Interfaces را مشخص می‌کند. علاوه بر این، کد نرم‌افزار یا Software Driver نیز بایستی به زبان C و با استفاده از روش Memory-Mapped نوشته شده و فایل کامپایل شده آن همانند مثال در قسمت ARM Core معرفی شود. سرانجام پس از آماده شدن سیستم، عملیات شبیه‌سازی با استفاده از دستور gplatform انجام می‌شود.

به این منظور باید قسمتی از کد فوق را به صورت سخت‌افزاری و قسمتی را به صورت نرم‌افزاری انجام دهید و در پایان نتیجه محاسبات را در هر دو قسمت نمایش دهید. برای انجام قسمتی از کد به صورت سخت‌افزاری و استفاده از پردازنده ARM به این شکل عمل شود که بخش محاسباتی که شامل حلقه موجود در خطوط 89-77 می‌باشد، در سخت‌افزار انجام شود و بقیه بخش‌ها شامل کنترل شروع و پایان حلقه و همچنین مشخص کردن مقادیر اولیه توسط نرم‌افزار صورت پذیرد. در واقع نرم‌افزار باید مقادیر x1 و iterations را برای سخت افزار ارسال و مقادیر سینوس و کسینوس را از سخت افزار دریافت کند. مابقی فرض‌ها از جمله تعداد و نوع پایه‌های ارتباطی می‌تواند به دلخواه انجام شود.

توجه: شبیه سازی باید برای ۳ مقدار مختلف از theta انجام شود.

۲. گزارش مختصری از فرآیند انجام پروژه به همراه تصاویر مناسب، به انضمام کدهای نوشته شده و فایل‌های خروجی ابزار را به صورت فشرده، همراه با نام و شماره دانشجویی در سامانه درس‌افزار بارگذاری نمایید (قالب گزارش از قسمت فایل‌ها قابل دسترسی است).

توجه: در گزارش نوشته شده باید تمامی فرض‌های گرفته شده توضیح داده شوند.

موفق باشید