گزارشِ تکلیفِ هفتمِ درسِ هم‌طرّاحی

روزبه صیّادی – امیرعلی منجر

۱ مقدّمه

هدف از این تکلیف پیاده‌سازیِ یک سیستم متشکّل از سخت‌افزار و نرم‌افزار با استفاده از محیطِ شبیه‌سازیِ GEZEL و با استفاده از میکروکنترلر ۸۰۵۱ است. الگوریتمِ پیاده‌سازی شده یک الگوریتمِ ساده برای محو[1] کردن یک عکس می‌باشد. در قسمتِ نرم‌افزار از زبانِ C استفاده شده است. کدهای مربوطه در فایل‌های blur.c و blur.fdl قابلِ دسترسی هستند.

۲ نحوه‌ی جداسازیِ سخت‌افزار از نرم‌افزار

برای پیاده‌سازیِ الگوریتم کارِ تولید کردنِ عکس را به نرم‌افزار سپردیم. نرم‌افزار پس از تولیدِ عکس، با استفاده از یک روشِ خاص (که در ادامه خواهیم دید) به سخت‌افزار پیام می‌دهد که کارِ خودش را شروع کند. سخت‌افزار ابتدا محتویاتِ عکسِ اصلی و- بعد از انجامِ محاسبات- محتویاتِ عکسِ محوشده را در خروجی چاپ می‌کند و به نرم‌افزار خبر می‌دهد که کارش تمام شده است. سپس نرم‌افزار دوباره یک عکس تولید کرده و مراحلِ قبلی را تکرار می‌کند. این الگوریتم روی ۸ عکسِ ۱۰ در ۱۰ متفاوت اعمال می‌شود و سپس برنامه به پایان می‌رسد.

دلیلِ این جداسازی این است که سخت‌افزار قابلیّتِ موازی‌سازی و همچنین سرعتِ بیشتری دارد و اگر محاسباتِ اصلیِ برنامه را به سخت‌افزار محوّل کنیم، به عملکردِ بهتری دست خواهیم یافت.

۳ توضیحِ کد

۳.۱ سخت‌افزار

۳.۱.۱ میکروکنترلر ۸۰۵۱ و RAM

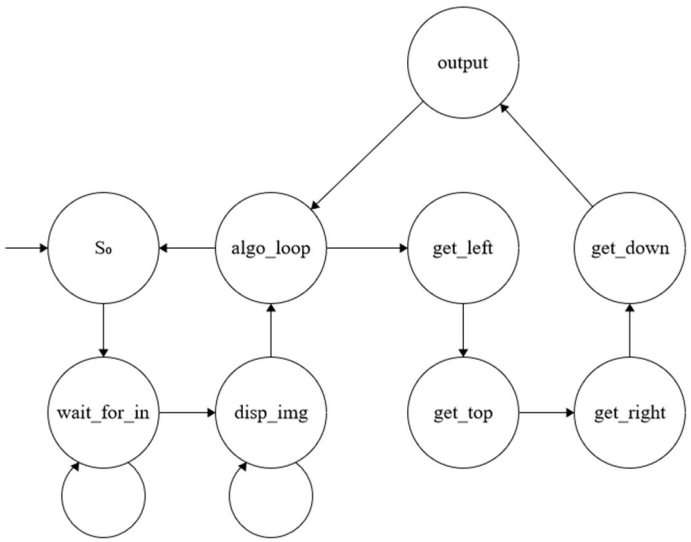
در قسمتِ سخت‌افزار یک میکروکنترلر ۸۰۵۱ وجود دارد. این میکروکنترلر با کمکِ ۲ پورت (P0 و P1) به dp ما متّصل است و با آن ارتباط برقرار می‌کند. پورتِ P0 وظیفه‌ی گرفتنِ instructionها از نرم‌افزار را دارد، و پورتِ P1 خروجیِ میکروکنترلر است و وظیفه‌ی اعلامِ وضعیّت به نرم‌افزار را دارد. همچنین یک RAM نیز در سیستم موجود است که بینِ سخت‌افزار و نرم‌افزار مشترک است و وظیفه‌ی نگه‌داریِ عکس‌ها را دارد.

قسمتِ اوّلِ کد مربوط به تعریفِ خودِ ۸۰۵۱ است. در این‌جا ما فایلِ HEX نرم‌افزارمان را وارد کرده‌ایم تا بتواند از آن استفاده کند. در قسمتِ دوم و سوم دو پورتی که قبلاً توضیح دادیم را تعریف کرده‌ایم. یکی از این پورت‌ها source و دیگری sink است. قسمتِ چهارم نیز مربوط به کامپوننتِ RAM است که آدرسِ ابتداییِ آن 0x4000 و حجمِ در نظر گرفته شده برای آن ۱۰۰ بایت (0x64) است.

۳.۱.۲ داده‌مسیر و ماشینِ حالتِ متناهی

مسیرِ کلّیِ ماشینِ حالت را می‌توانید در شکلِ (۱) ببینید. برای جلوگیری از شلوغی شرط‌ها و عملیاتی که در هر یال انجام می‌گیرد در شکل نوشته نشده است.

[1] blur



شکل ۱. ماشینِ حالتِ سخت‌افزار

توضیحِ دقیق‌تر از عملکردِ این ماشینِ حالت در جدولِ (۱) نوشته شده است. توضیحِ دقیق‌ترِ هر sfg در جدولِ (۲) نوشته شده است. اگر در کدِ اصلی دقّت کنید می‌بینید که در sfgها علاوه بر کدهای ذکر شده چند مقداردهیِ دیگر هم وجود دارند که بینِ این‌ها مشترک هستند. این مقداردهی‌ها فقط برای این نوشته شده‌اند که برخی از سیم‌هایی که تعریف کرده‌ایم همیشه باید مقدار داشته باشند. این مقداردهی‌ها تأثیری روی روندِ اجرای کد ندارند.

جدول ۱. عملکردِ ماشینِ حالت

نامِ حالت	شرط‌ها و sfgهای فراخوانی شده	مقصد	توضیح
S_0	init()	wait_for_in	متغیّرها را مقداردهیِ اوّلیّه کن و به حالتِ wait_for_in برو.
wait_for_in	data_in_ready ? announce_disp()	disp_img	اگر عکس آماده شده بود (روی RAM قرار داشت) روی صفحه بنویس که می‌خواهی عکس را نشان بدهی و سپس به حالتِ disp_img برو.
	!data_in_ready ? idle()	wait_for_in	اگر عکس هنوز آماده نبود، هیچ کاری انجام نده و روی همین حالت بمان.
disp_img	ramcnt == 100 ? reset_ramcnt(), announce_blur()	algo_loop	اگر متغیّرِ شمارنده‌ی RAM به ۱۰۰ رسیده بود، یعنی تمامِ عکس چاپ شده است. شمارنده را ریست کن، روی صفحه بنویس که می‌خواهی عکسِ تارشده را نشان بدهی و سپس به حالتِ algo_loop برو.
	ramcnt != 100 ? display_pixel()	display_img	اگر هنوز پیکسلی برای نشان دادن باقی مانده بود، آن را چاپ کن و در همین حالت بمان.
algo_loop	ramcnt == 100 ? announce_ready()	S_0	اگر حلقه‌ی الگوریتم به پایان رسیده بود، به نرم‌افزار اعلام کن که کار تمام شده است. سپس به حالتِ اوّلیّه برگرد.

	ramcnt != 100 ? idle()	get_left	اگر الگوریتم به پایان نرسیده بود، کاری انجام نده و به حالتِ get_left برو.
get_left	calculate_left()	get_top	مقدارِ پیکسلِ سمتِ چپ را ذخیره کن و سپس به حالتِ get_top برو.
get_top	calculate_top()	get_right	مقدارِ پیکسلِ بالا را ذخیره کن و سپس به حالتِ get_right برو.
get_right	calculate_right()	get_down	مقدارِ پیکسلِ سمتِ راست را ذخیره کن و سپس به حالتِ get_down برو.
get_down	calculate_down()	output	مقدارِ پیکسلِ پایین را ذخیره کن و سپس به حالتِ output برو.
output	calculate_and_display_mean()	next_pixel	با کمکِ ۴ مقداری که از قبل ذخیره کرده‌ایم مقدارِ پیکسلِ فعلی را محاسبه کن و نتیجه را نشان بده. سپس به حالتِ next_pixel برو.
next_pixel	inc_ramcnt()	algo_loop	مقدارِ شمارنده‌ی RAM را زیاد کن و به ابتدای حلقه‌ی الگوریتم برگرد.

جدول ۲. عملکردِ هر sfg

نامِ sfg	مقداردهی‌ها و دستورهای اصلی	توضیح
always	data_in_ready = upins == INS_START ? 1 : 0;	اگر instructionای که وارد شده بود INS_START بود، یعنی عکس آماده است. بنابراین مقدارِ data_in_ready را ۱ می‌کنیم.
init	ramcnt = 0 image_counter = image_counter + 1	شمارنده‌ی RAM (که در واقع شمارنده‌ی پیکسل‌ها است) را برابر با صفر قرار می‌دهیم و شمارنده‌ی عکس (که مشخّص می‌کند الآن در حالِ پردازشِ عکسِ چندم هستیم) را یکی زیاد می‌کنیم.
calculate_left	ramadr = ramcnt % 10 == 0 ? ramcnt : ramcnt – 1 temp_left = ramcnt % 10 == 0 ? 0 : ramodata;	در این‌جا می‌خواهیم آدرسِ پیکسلِ سمتِ چپ را به دست بیاوریم تا بتوانیم مقدارِ پیکسل را ذخیره کنیم. اگر پیکسلِ فعلی سمتِ چپِ تصویر بود مقدارِ ramadr برای ما بی‌اهمّیّت می‌شود (چون مقدارِ پیکسل ۰ است). در غیرِ این صورت مقدارِ ramcnt-1 را در آدرسِ RAM می‌نویسیم. سپس در خطّ بعد یا ۰ یا مقداری که RAM به ما برمی‌گرداند را در temp_left می‌نویسیم.
calculate_top	ramadr = ramcnt < 10 ? ramcnt : ramcnt – 10; temp_up = ramcnt < 10 ? 0 : ramodata;	مشابهِ حالتِ قبل، ولی برای پیکسلِ بالا.
calculate_right	ramadr = ramcnt % 10 == 9 ? ramcnt : ramcnt + 1; temp_right = ramcnt % 10 == 9 ? 0 : ramodata;	مشابهِ حالتِ قبل، ولی برای پیکسلِ راست.
calculate_down	ramadr = ramcnt >= 90 ? ramcnt : ramcnt + 10; temp_down = ramcnt >= 90 ? 0 : ramodata;	مشابهِ حالتِ قبل، ولی برای پیکسلِ پایین.

calculate_and_display_mean	temp_sum = (temp_left + temp_up + temp_right + temp_down) >> 2;
	ramcnt_temp = ramcnt + 1;
	$display("pixel ", $dec, ramcnt_temp, ": ", $dec, temp_sum);
		میانگینِ چهار مقداری که در sfgهای قبل به دست آوردیم را محاسبه می‌کنیم و در temp_sum می‌ریزیم. سپس مقدارِ ramcnt را با ۱ جمع می‌کنیم و در ramcnt_temp می‌ریزیم (که شمارشمان در خروجی از ۱ شروع شود) و در نهایت مقادیرِ به دست آمده را چاپ می‌کنیم.

inc_ramcnt	ramcnt = ramcnt + 1;	مقدارِ شمارنده‌ی RAM را یکی زیاد می‌کنیم.
announce_disp	image_counter_temp = image_counter; $display("---------- Image ", $dec, image_counter_temp, " ----------"); $display(" Original image: ");	ابتدا مقدارِ شمارنده‌ی عکس را در یک sig می‌ریزیم (برای این که نمایشش زیباتر بشود). سپس شماره‌ی عکس را در صفحه چاپ می‌کنیم و همچین چاپ می‌کنیم که قرار است عکسِ اصلی را در ادامه چاپ کنیم.
announce_blur	$display(" Blurred image: ");	در صفحه پیامی می‌نویسیم که مشخّص شود می‌خواهیم عکسِ تارشده را چاپ کنیم.

display_pixel	ramadr = ramcnt;
	ramcnt_temp = ramcnt + 1;
	wr = 0;
	$display("pixel ", $dec, ramcnt_temp, ": ", ramodata);
	ramcnt = ramcnt + 1;
		با توجّه به مقدارِ شمارنده‌ی RAM مقدارِ پیکسلِ فعلی را به دست می‌آوریم و آن را در خروجی چاپ می‌کنیم. در آخر نیز مقدارِ شمارنده را یکی اضافه می‌کنیم.

announce_ready	upstatus = 1;	مقدارِ پورتِ مربوط به اعلامِ وضعیّت را یک قرار می‌دهیم که نرم‌افزار متوجّهِ اتمامِ کارِ سخت‌افزار شود و به کارِ خود ادامه دهد.
reset_ramcnt	ramcnt = 0;	مقدارِ شمارنده‌ی RAM را صفر می‌کنیم.
idle	-	هیچ کاری انجام نمی‌دهیم.

در نهایت نیز سیستمی که طرّاحی کرده‌ایم را با بلوکِ system که در انتهای کد قرار دارد اجرا می‌کنیم.

۳.۲ نرم‌افزار

در قسمتِ نرم‌افزار ابتدا instructionها، تابعِ terminate و تعریفِ حافظه‌ی مشترک را را طبقِ متنِ کتابِ مرجع[۲] می‌نویسیم. در ادامه یک حلقه‌ی ۸ تایی داریم که در هر بار اجرای حلقه یک عکس مقداردهی می‌شود و دستورالعملِ INS_START با استفاده از پورتِ P0 به سخت‌افزار ارسال می‌شود. سپس یک حلقه‌ی بدونِ بدنه داریم که وظیفه‌ی منتظرِ سخت‌افزار ماندن را دارد. شرطِ این حلقه مقدارِ موجود در P1 است که در واقع همان پورتِ مربوط به وضعیّت در سخت‌افزار است.

بعد از اتمامِ حلقه نیز terminate را صدا می‌زنیم که شبیه‌سازی به اتمام برسد.

۴ اجرا و تستِ برنامه

برای اجرای برنامه ابتدا برنامه‌ی C را با برنامه‌ی sdcc کامپایل می‌کنیم. سپس با دستورِ cpp -P ماکروهای برنامه‌ی جزل را پیش‌پردازش می‌کنیم. در انتها نیز کدِ جزلِ حاصل را با دستورِ gplatform اجرا می‌کنیم. نمونه‌ای از خروجیِ برنامه در فایلِ result.txt قابلِ دسترسی است.

۵ نتیجه‌گیری

ما در این تکلیف با نحوه‌ی کار با میکروکنترلرِ ۸۰۵۱ و تعریفِ instructionهای جدید برای آن آشنا شدیم. همچنین مشاهده کردیم که چه‌گونه می‌توانیم یک حافظه‌ی مشترک بینِ سخت‌افزار و نرم‌افزار داشته باشیم و با آن کار کنیم.

[2] Patrick R. Schaumont – A Practical Introduction to Hardware/Software Codesign – Second Edition