دانشگاه فردوسی

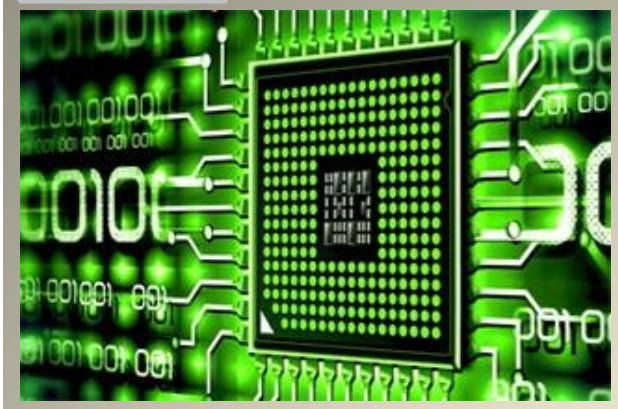
دانشکدهٔ مهندسی

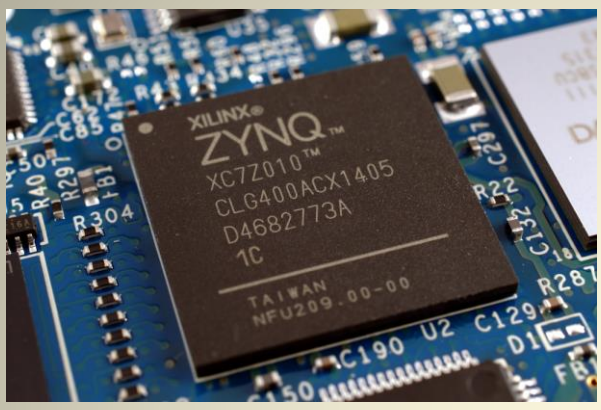
گروه برق

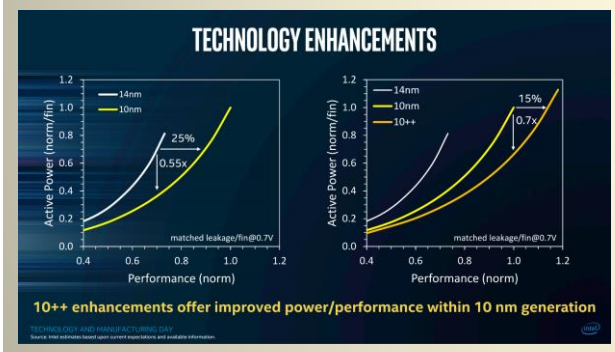
دستورکار آزمایشگاه اجزاء کامپیوتر

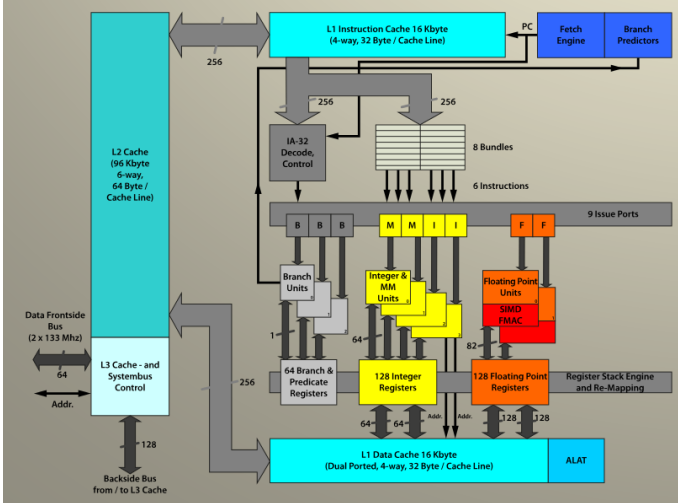
تدوین کننده : مجید شرکاء

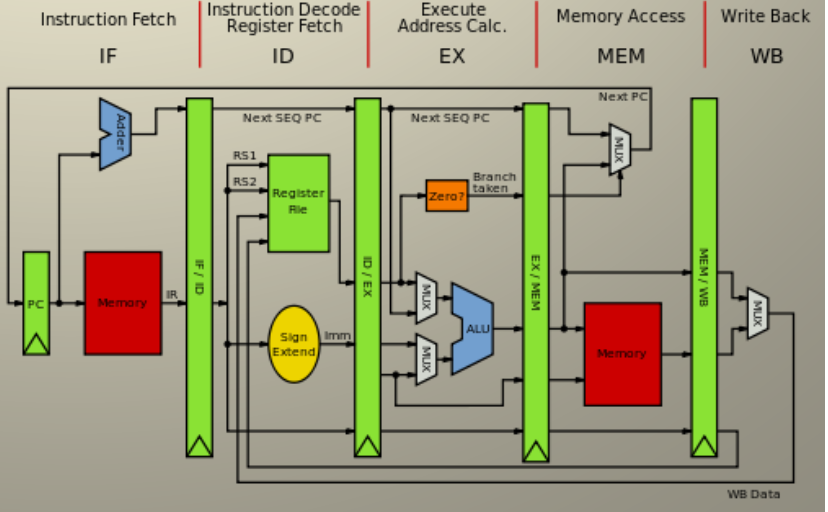
پاییز ۱۳۹۶











بنام خدا

کامپیوترها و سایلی هستند که می‌توانند داده‌ای (داده‌هایی) را دریافت نموده و بر اساس برنامه‌ای که به آن داده شده عملیاتی را بر روی داده اعمال نموده (پردازش داده : شامل عملیات ریاضی و منطقی) و نتایج را گزارش کنند. هدف درس اجزاء کامپیوتر : آشنایی با ساختارها، روشها و اجزاء مورد استفاده در کامپیوترها است.

در طول زمانی که از معرفی و سایل/ساختارهایی به نام کامپیوتر می‌گذرد، علم و تکنولوژی کامپیوترها بکلی و در چند دوره متحول شده‌اند. امروزه سخت‌افزارها با پیچیدگی و وسعت منابع بسیار بیشتر اما در ابعاد بسیار کوچکتر، بسیار ارزان‌تر، کم‌مصرف‌تر، سریع‌تر، ضمناً در دسترس‌تر هستند.

با فرض این که دانشجو در درس اجزاء کامپیوتر دانش لازم را فرا گرفته و برخی تمرینها را انجام داده است، در آزمایشگاه سعی بر این است که با افزودن بر تمرینها و تجربیات، با برخی مسائل نزدیک به واقعی نیز درگیر شود و در این مسیر، روش دستیابی به راه حل را نیز در عمل تجربه کند. همچنین در این فرآیند با ویژگیهای فیزیکی پیاده‌سازی و خطاهایی که ممکن است رخ دهند نیز آشنا خواهد شد.

این دستورکار بر اساس تجربیات تدوین کننده در تدریسِ درس و آزمایشگاه اجزاء کامپیوتر طی سالهای متمادی در گروه برق دانشگاه فردوسی تدوین شده است. فرض بر این است که دانشجو در درس اجزاء کامپیوتر تحلیل و طراحی مدارهای مورد استفاده در کامپیوترها را بطور کامل آموخته است و حداقل به صورت مقدماتی با یکی از زبانهای بیان مدارهای منطقی آشنا شده و بکمک یکی از نرم‌افزارهای شبیه‌سازی مدارهای منطقی تجربیاتی بدست آورده است.

مجموعهٔ فعالیتها در دستورکار آزمایشگاه را می‌توان را در سه بخش در نظر گرفت. در بخش اول پیاده‌سازی برخی مدارهای منطقی تجربه می‌شوند. هدف در این بخش، آشنایی دانشجو با ابزارهای موجود در آزمایشگاه و مرور پیاده سازی مدارها در FPGA است. در بخش دوم سعی می شود برخی ساختارهای پر کاربرد در سخت‌افزارها که بخشهایی مشابه با کامپیوترها داشته اما کامپیوتر نیستند تجربه شوند. در بخش سوم که عمدهٔ تلاش و زمان آزمایشگاه را در بر می‌گیرد، یک پروسسور با جزئیاتش پیاده‌سازی شده، عملکرد آن و میزان مؤثر بودن پیاده‌سازی مورد بررسی قرار می‌گیرند.

نحوهٔ ارزیابی نهایی : شامل ۱۰ نمره فعالیت کلاسی بر اساس حضور دانشجو، تمرینهای ابتدای جلسات، تلاش مؤثر در جلسات و نتایج بدست آمده در آزمایشها (وزن آزمایشها بترتیب ۱، ۲ و ۷) و ۱۰ نمره امتحان عملی پایان دوره است.

امید است که این دستور کار در تأمین هدفِ بالا بردن تواناییهای علمی و فنی تحلیل، طراحی و پیاده‌سازی مدارهای مبتنی بر ساختار پردازشی کامپیوترها مؤثر واقع شود.

نهایتاً، تدوین‌کننده از دانشجویان آزمایشگاه و سایر مطالعه کنندگان این دستورکار تقاضا دارد که خطاها و معایب دستورکار و همچنین نکاتی که ممکن است در بهبود کیفیت دستورکار مؤثر باشند را به صورت حضوری یا از طریق پست الکترونیک mshora@yahoo.com به اطلاع برسانند.

مجید شرکاء

پاییز ۱۳۹۶

فعالیت قبل از آزمایش :

- در ادامهٔ متن، توضیحات مرتبط با محاسبهٔ حاصل جمع دو عدد به روش ripple-carry را مطالعه نموده و کد پیاده‌سازی آن را بنویسید.
- در ادامهٔ متن، توضیحات مرتبط با آزمایش سرعت عمل محاسبهٔ حاصل جمع دو عدد را مطالعه نموده و کد پیاده‌سازی آن را بنویسید.
- در ادامهٔ متن، توضیحات مرتبط با محاسبهٔ حاصل ضرب دو عدد علامت‌دار را مطالعه نموده و کد پیاده‌سازی آن را بنویسید.

توجه : برنامه‌های نوشته شده در ابتدای جلسه بررسی می‌شود.

پیش‌زمینه : مقدمهٔ روشهای محاسبهٔ عملیات ریاضی

امروزه روشهای متنوعی برای محاسبهٔ عملیات ریاضی تعریف شده که بویژه از جهت بازده : سرعت عمل (تأخیر) در برابر سطح مصرفی بر روی قطعه (هزینه) با یکدیگر قابل مقایسه هستند. مثلاً در مورد مدارهای جمع‌کننده می‌توان بر اساس نحوهٔ اثر رقم نقلی به بیتهای بالاتر (carry) به روشهای ripple-carry ، look-ahead-carry و carry-select اشاره نمود. خوشبختانه در جمع (و تفریق) برای اعداد بدون علامت و اعداد علامت‌دار (نمایش مکمل دو : 2's complement) از ابزار (مدار) مشابهی استفاده می‌شود. اما برای مدارهای ضرب‌کننده (و تقسیم‌کننده) روشهای موجود برای بدون علامت قابل تعمیم به اعداد علامت‌دار نیستند و لذا مدارهای متفاوتی برای انجام عمل بر روی اعداد علامت‌دار یا بدون علامت تعریف می‌شوند.

در متن درس ۳ نوع مدار جمع‌کنندهٔ فوق را مطالعه نموده‌اید. در این آزمایش ضمن مروری بر پیاده‌سازی مدارهای فوق، سرعت عمل آنها نیز مورد آزمایش قرار می‌گیرد.

در بخش اول آزمایش، جمع‌کنندهٔ ۱۶بیتی به روش ripple-carry پیاده‌سازی می‌شود. در ابتدا در حاصل جمع موقت مقدار صفر قرار دارد. یک ورودی جمع‌کننده از کلیدهای dipswitch در نظر گرفته شود (۱۲ کلید وجود دارد. ۴ بیت بالا را صفر در نظر بگیرید). ورودی دیگر از خروجی جمع‌کننده باشد. به ازای هر بار فشار دادن کلید فشاری روی بورد یک جمع انجام شود. نتیجهٔ جمع‌کننده همچنین بر روی 7segهای روی بورد نمایش داده شود. برای اطلاعات مربوط به نحوهٔ ارتباط با کلیدها و 7segها و نمونهٔ کد به دستورکار آزمایشگاه مدارهای منطقی مراجعه شود. جدول (نقشهٔ) ارتباطها نیز در فایل nasr-pin-assignment.pdf قرار دارد.

سپس سرعت عمل جمع‌کننده آزمایش می‌شود. هنگام کامپایل شدن کد، حداکثر تأخیر مدار سنتز شده (ناشی از طولانی‌ترین مسیر عبور سیگنال) گزارش می‌شود و بر اساس آن حداکثر سرعت clock اعلام می‌گردد. با استفاده از پالس ساعت موجود بر روی بورد و تنظیم مدار برای انجام تعداد مشخصی جمع متوالی، صحت نتیجه را می‌توان بررسی نمود.

در نهایت کد ضرب‌کنندهٔ دو عدد علامت‌دار ۸بیتی مورد بررسی قرار می‌گیرد. مسئله به صورت یک ماشین حالت ۳مرحله‌ای پیاده‌سازی می‌شود. در مرحلهٔ اول مقدار حاصل‌ضرب قبلی بر روی 7segها نمایش داده شده و سیستم آمادهٔ دریافت عدد ۸بیتی اول می‌باشد. حضور در هر مرحله بر روی LEDها مشخص گردد. با زدن کلید فشاری به مرحلهٔ دوم می‌رود که در آن عدد اول را روی 7segها نمایش می‌دهد و آمادهٔ دریافت عدد دوم می‌شود. با زدن مجدد کلید فشاری عدد دوم وارد شده و بر روی 7segها نیزنمایش داده می‌شود. زدن مجدد کلید فشاری حاصل ضرب را بر روی 7segها نمایش داده و مجدداً آمادهٔ دریافت عدد اول در محاسبهٔ بعدی می‌شود.

کار آزمایشگاهی

۱- جمع‌کنندهٔ ۱۶بیتی به روش ripple-carry را پیاده‌سازی نمایید. صحت عملکرد برای ورودیهای مختلف بررسی گردد.

۲- سرعت عمل جمع‌کننده برای انجام مثلاً ۱۰۰۰۰۰ جمع متوالی آزمایش شود. با تغییر پریود جمع کردن در حد امکان، حد (مرز) سرعت عمل جمع‌کننده را برای تأمین نتیجهٔ صحیح بررسی نمایید. تغییر پریود بکمک پالس ساعت روی بورد فراهم می‌شود.

۳- کد ضرب‌کنندهٔ دو عدد علامت‌دار ۸بیتی را پیاده‌سازی نموده و صحت عملیات را آزمایش کنید.

فعالیت قبل از آزمایش :

- در ادامهٔ متن، توضیحات مرتبط با نمونهٔ مسئله مرتبط با طراحی مسیر داده (datapath design) را مطالعه نموده و کد پیاده‌سازی آن را طی مراحل خواسته شده بنویسید.

- **برنامه‌های نوشته شده در ابتدای جلسه بررسی می‌شود.**

پیش‌زمینه : تعاریف مورد نیاز در طراحی مسیر داده و مشخصات مسیر دادهٔ مورد نظر

از مهمترین بخشهای یک عملکرد محاسباتی، مسیر دادهٔ آن است. در کاربردهای مختلفی انجام محاسبات لازم می‌شود که چند مثال را ذیلاً مشاهده می‌نمایید :

- فیلتر دیجیتال
- محاسبهٔ مقدار RMS سیگنال
- محاسبهٔ توان، توان رآکتیو، ضریب توان (از مقادیر لحظه‌ای ولتاژ و جریان)
- محاسبهٔ میزان هارمونیکها (تبدیل فوریه)
- ترکیب سیگنالها : جمع یا تفریق همراه/بر اساس ویژگیهای انتخابی از سیگنالها (از جمله در سیگنالهای صوت یا تصویر)
- انجام عملیات بر روی سیگنالها : ایجاد effectها بر روی سیگنالها

برای پیاده‌سازی بلوک محاسباتی، پس از تعیین روش حل مسئله، مجموعهٔ رجیسترها و ALU و ارتباط بین آنها دقیقاً باید تعیین شوند. همچنین ترتیب عملیات از بلوکهایی که محاسبات در آنها نحوهٔ دسترسی به این منابع نیز ساختار مسیر داده را مشخص می‌نماید.

فیلتر دیجیتال

شکل سادهٔ یک فیلتر دیجیتال به صورت سیستمی است دارای یک ورودی و یک خروجی که رابطهٔ بین ورودی و خروجی از طریق عبارت مناسب بگونه‌ای تعریف می‌شود تا بتواند اهداف فیلتر مورد نظر را تأمین نماید. بر خلاف نوع آنالوگ، در این سیستم سیگنال به صورت پیوسته اعمال نشده بلکه از ورودی در لحظه‌های مشخصی نمونه‌برداری شده، محاسبات انجام گرفته و نتیجه به خروجی اعمال می‌گردد. معمولاً فاصلهٔ بین دو نمونه‌برداری ثابت است لذا عملیات با فرکانس مشخصی f_s (ω_s) انجام (تکرار) می‌شود.

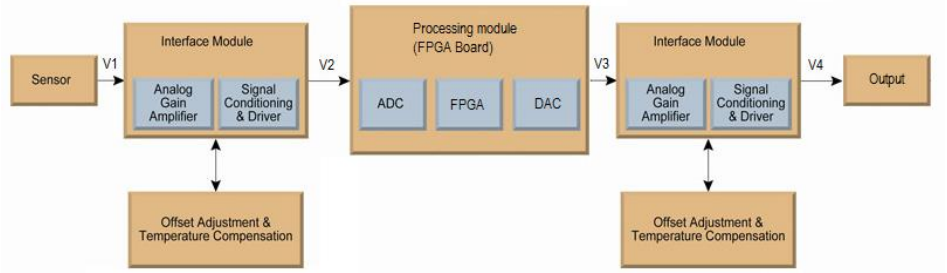
گسسته‌سازی در یک سیستم دیجیتال (در اینجا «یک ورودی-یک خروجی» : SISO)، باید در دو بخش در نظر گرفته شود. نخست اینکه سیگنال ورودی با نرخ مشخصی نمونه‌برداری می‌شود (f_s یا ω_s). به این ترتیب تابع انتقال زمان‌گسسته G(z) بدست می‌آید که در صفحهٔ z بیان می‌شود (در مقابل زمان‌پیوسته G(s) که در صفحهٔ مختلط s بیان می‌گردد). از طرفی برای نمایش و ذخیره‌سازی مقادیر در سیستم دیجیتال، نیاز به گسسته‌سازی در حوزهٔ مقادیر نیز وجود خواهد داشت.

جهت طراحی فیلتر مورد نظر، استفاده از ابزار fdatool از نرم‌افزار Matlab توصیه می‌شود. نخست اهداف فیلتر شامل نوع آن (پایین‌گذر، میان‌گذر، ...)، بهره و فرکانس(های) مرزی باند عبور، بهره و فرکانس(های) مرزی باند قطع، ساختار فیلتر (باتروورث، چبیشف، ...)، رتبهٔ فیلتر (۱، ۲، ...) و نوع فیلتر (IIR، FIR) را تعیین می‌کنیم. پس از وارد کردن مشخصات مورد نظر در صفحهٔ ابزار طراحی فیلتر، می‌توانیم فرمان طراحی را صادر کنیم که پس از انجام طراحی، نمودار پاسخ فرکانسی فیلتر طراحی شده در صفحهٔ ابزار مشاهده خواهد شد. سپس با انتخاب filter coefficients از نوار ابزار بالای صفحه، می‌توان ضرایب فیلتر را مشاهده نمود که شامل ضرایب صورت b_i، ضرایب مخرج a_i (ضریب a_0 همواره ۱ در نظر گرفته می‌شود) و بهره (که در صورت رابطه ضرب می‌شود و میتواند از ابتدا به ضرایب صورت اعمال گردد) می‌باشد $G(z) = \frac{Y(z)}{X(z)} = k\frac{\sum b_i z^i}{\sum a_i z^i}$. در فیلترهای با مرتبهٔ بالاتر از ۲،

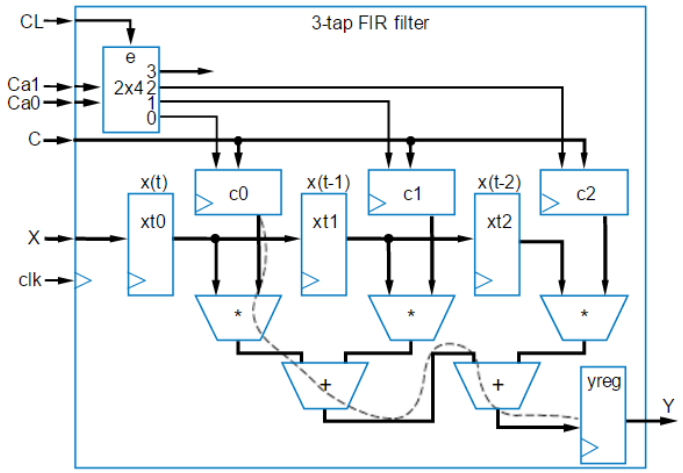
فیلتر به صورت ترکیب چند بلوک مرتبهٔ ۲ (بر اساس اتصال سری بلوکها : cascade) و یک بلوک احتمالی مرتبهٔ ۱ (در صورتی که رتبهٔ فیلتر فرد باشد) محاسبه و ارائه می‌گردد $G(z)=\frac{Y(z)}{X(z)}=(k\frac{\sum b_i z^i}{\sum a_i z^i})\dots(k\frac{\sum b_i z^i}{\sum a_i z^i})$.

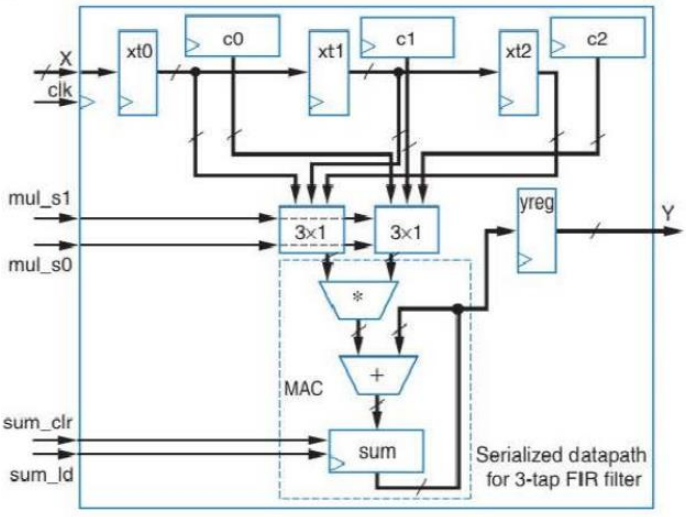
تابع انتقال فیلتر (یا هر سیستم دیجیتال siso) به صورت $G(z)=\frac{Y(z)}{X(z)}=\frac{\sum b_i z^i}{\sum a_i z^i}$ بیان می‌شود (با فرض درجهٔ m برای صورت و درجهٔ n برای مخرج) که همچنین می‌توان آن را به صورت $G(z)=z^{n-m}\frac{\sum_{i=0}^{m-1} b_i z^{i-m}}{\sum_{i=0}^{n-1} a_i z^{i-n}}$ نوشت (چند جمله‌ایهای صورت و مخرج بر حسب توانهای منفی z هستند). با توجه به اینکه z^{-1} معادل یک واحد تأخیر (بمدت یک پریود نمونه‌برداری) است، رابطه به صورت $\sum a_i y(n-i)=\sum x_i(n-i)$ در حوزهٔ زمان‌گسسته بدست می‌آید که همچنین می‌توان آن را به صورت $y(n)=\frac{1}{a_0}(-\sum_{i=1}^{n-1} a_i y(n-i)+\sum_{i=0}^{n-1} b_i x(n-i))$ نوشت. از این رابطه می‌توان مقدار جدید خروجی سیستم دیجیتال را بر حسب مقادیر فعلی و قبلی ورودی و مقادیر قبلی خروجی محاسبه نمود. طبعاً ممکن است برخی ضرایب صفر باشند. اگر ضرایب مخرج بجز a_0 صفر باشند، فیلتر از نوع FIR و در غیر این صورت از نوع IIR می‌باشد. در واقع فیلتر FIR فقط به مقادیر ورودی (فعلی و قبلی) بستگی دارد اما IIR به مقادیر قبلی خروجی نیز وابسته است.

در سیستم سخت‌افزاری-نرم‌افزاری نکاتی باید در نظر گرفته شود. با توجه به اینکه می‌دانیم سیگنال نمونه‌برداری خواهد شد، فرکانس نمونه‌برداری باید تعیین شود. معیارهای مختلفی برای کاربردهای مختلف در نظر گرفته می‌شوند که معمولاً منجر به تعیین پهنای باند شده و متعاقباً f_s (ω_s) بر اساس پهنای باند مشخص خواهد شد. مسیر سیگنال از ورود سیگنال پیوسته به مدار شروع می‌شود. در این بخش مبدل آنالوگ به دیجیتال ADC قرار دارد.



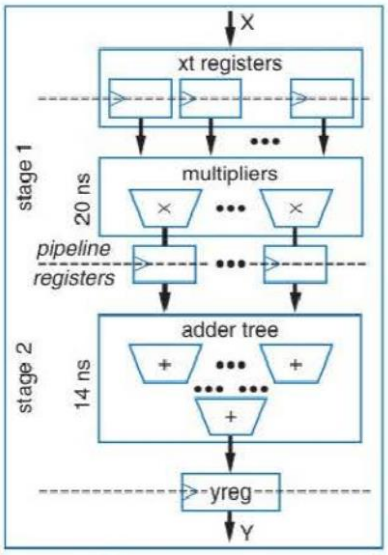
محدودهٔ سیگنال ورودی باید با محدودهٔ مؤثر مبدل منطبق باشد. لذا در حالت کلی مداری شامل بهره و شیفت DC باید در نظر گرفته شود (ولتاژ ورودی V1 که در محدودهٔ $[V1_{min},V1_{max}]$ قرار دارد را به محدودهٔ V2 : $[V2_{min},V2_{max}]$ که برای A/D تعریف شده نگاشت می‌نماید). مقدار دیجیتال که قاعدتاً به صورت باینری است معمولاً به صورت خام (بدون علامت و بدون در نظر گرفتن نقطه‌ممیز) در اختیار قرار می‌گیرد. لذا، معمولاً، مجدداً در سیستم دیجیتال نگاشت دیگری بر روی این مقدار اعمال می‌گردد تا آن را به صورت مناسب برای پردازش درآورند. نمونهٔ این فرآیند : اِعمال محاسباتی به جهت رسیدن به عددی دارای علامت و مقادیر کوچکتر از یک (ممیزشناور یا ممیز ثابت) است. مثلاً مقدار ورودی می‌تواند در محدودهٔ ۱± قرار گیرد (بازهٔ کل مقادیر ورودی). همچنین، با توجه به طرح سیستم دیجیتال، باید مشخصات مقادیری که متعاقباً در محاسبات مورد استفاده قرار خواهد گرفت (ضرایب، مقدار ورودی و مقادیر محاسباتی متعاقب) تعیین شود. این مشخصات شامل نوع نمایش مقادیر (ممیزثابت یا ممیزشناور) و ابعاد مقادیر (تعداد بیت و محل ممیز در عدد) هستند که با توجه به امکانات سخت‌افزاری و نیاز سیستم انتخاب می‌شود. در حالت کلی محاسبات شامل تعدادی ضرب وجمع است. البته همچنان ساختارهای





محاسباتی متفاوتی برای پیاده‌سازی قابل استفاده هستند (همچون فرمهای موازی، مستقیم ۱ و ۲ ، ترانهاده و ...) که هر یک واجد خصلتهایی از جمله : تعداد عملیات، خطای محاسبات، حافظهٔ مورد نیاز، نوع عملیات مورد نیاز، حساسیت به گرد شدن ضرایب و ... هستند. نمونهٔ نمودار جریان داده برای فیلتر FIR با ۳ ضریب در دو شکل آمده است. شکل اول پیاده‌سازی به صورت موازی و شکل دوم برای پیاده‌سازی به صورت سری هستند. در شکل سوم سامان‌دهی ساختار پیاده‌سازی سری جهت ایجاد فرصت pipeline نمودن عملیات است.

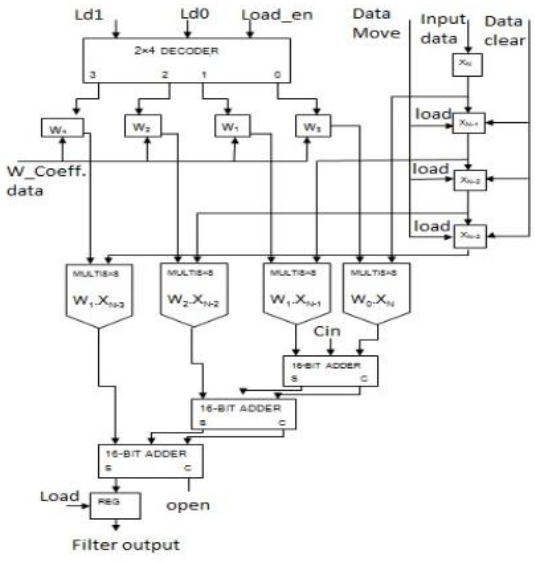
پس از انجام کل عملیات، مقدار خروجی بدست آمده را باید جهت اعمال به مبدل دیجیتال به آنالوگ DAC آماده نمود که شامل ضرب در مقدار مناسب، شیفت DC (عددی) و برش (انتخاب d بیت از n بیت) خواهد بود. مقدار خروجی DAC نیز مجدداً احتمالاً به مداری شامل بهره و شیفت DC نیاز خواهد داشت تا سیگنال آنالوگ متناسب با نیاز طبقهٔ بعد بدست آید.



از جملهٔ کاربردهای مشابه فیلتر، تولید سیگنال سینوسی است. می‌دانیم که مولد سینوسی در یک سیستم خطی معادل سیستمی در آستانهٔ ناپایداری است (قطب بر روی محور jω در حوزهٔ پیوسته یا بر روی دایرهٔ واحد در حوزهٔ گسسته). کاربردهای مختلفی را برای مولدهای سیگنال سینوسی می‌توان نام برد که ازجمله در مولدهای قدرت، سیستمهای مخابراتی و صوتی است. روشهای متنوعی برای این منظور ارائه شده و بکار می‌روند که از جمله روش Direct Digital Synthesis : DDS می‌باشد.

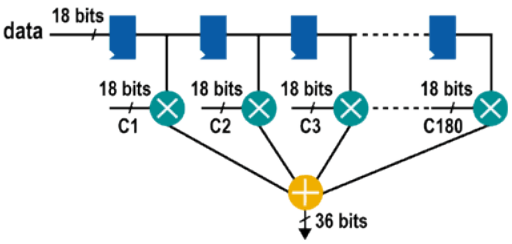
نحوهٔ پیاده‌سازی فیلتر موجود

با توجه به محدودهٔ فرکانسی و میزان پردازش مورد نیاز که در حوزهٔ فرکانس صوتی است، از حداکثر فرکانس کار ممکن استفاده می‌شود (۱۶MHz). واضح است که در این شرایط بازهٔ فرکانسی به ۸MHz محدود میشود. همچنین معمولاً به جهت پرهیز از رخداد پدیدهٔ alias، فیلتر پایین‌گذر آنالوگ مناسبی (سخت‌افزاری : مدار) در ورودی سیستم دیجیتال (قبل از A/D) قرار داده می‌شود. طبیعی است در صورت استفاده از فرکانس نمونه‌برداری پایین‌تر، ویژگیهای مطلوب به همان نسبت محدود می‌شوند.



از جمله نکات خاصی که در محاسبات می‌تواند در نظر گرفته شود موضوع سرریز (overflow) و برش (truncation) است. می‌دانیم در محاسبات امکان رخداد سرریز وجود دارد. در یک فیلتر که شامل چندین عمل ضرب و جمع می‌باشد این احتمال افزایش پیدا می‌کند. همچنین توجه داریم که در صورت سرریز، مقدار بدست آمدهٔ جدید، در محدودهٔ مقادیر با اندازهٔ بزرگ و با علامتِ مخالفِ مقدارِ درست خواهد بود (معادل تولید مقادیر بسیار پرت نسبت به مقدار واقعی). یک راه جبران این مشکل، ایجاد پدیدهٔ اشباع است که بطور طبیعی در سیستمهای آنالوگ نیز رخ می‌دهد. لذا هنگام انجام محاسبات، وقوع سرریز چک شده و در صورت رخداد، برای نتیجهٔ محاسبات انتهایی‌ترین مقدار در نظر گرفته می‌شود (بزرگترین مقدار از نظر اندازه در سمت مثبت یا منفی). از طرفی میدانیم نتیجهٔ عمل ضرب بر روی دو مقدار nبیتی، مقداری ۲nبیتی خواهد بود. اگر قرارباشد در مسیر عملیات ابعاد مقادیر محاسباتی میانی تغییر داده نشود، ابعاد مقادی حاصل به سرعت افزایش خواهد یافت. لذا پس از هر عمل ضرب(و جمع)، ابعاد حاصل محاسبات موقت تنظیم میشود که در شکل سادهاش دور ریختن مقادیر کمارزش نتیجهٔ محاسبات است (برش : truncation یا گرد کردن round-off). میدانیم که در اعداد صحیح حذف بیتهای وزن پایین منجر به از دست رفتن ارزش عدد میشود. به همین دلیل معمولاً اعداد را حتیالامکان به صورت مقدار کوچکتر از یک در نظر میگیرند : حاصلضرب دو عدد کوچکتر از یک، کوچکتر از

یک خواهد بود و با حذف مقادیر وزن پایین (truncation, round-off) تنها خطایی معادل ارزش بیتهای حذف شده ایجاد شده اما ارزش کلی عدد از دست نمیرود. .



در FPGAها معمولاً بلوکهای ویژهٔ پردازشی از پیش در نظر گرفته شده‌اند که از جمله بلوکهای ضرب‌کننده هستند (مثلاً ۴ یا ۱۰ بلوک ضرب‌کنندهٔ ۱۸بیتی). لذا پیاده‌سازی فیلتر به صورت موازی تا حدی که تعداد ضرب‌کنندهها از تعداد تعبیه شده در FPGA نگذرد به سهولت میسر است. با استفاده از ساختارهای سریال (ساده و pipeline) می‌توان در استفاده از منابع درون FPGA صرفه‌جویی کرد.

مجموعه منابع و فرآیندهای مورد نیاز می‌توند چنین باشد :

- ضرب‌کننده(ها) و جمع‌کنندهٔ(های) nبیتی، با در نظر گرفتن امکان سرریز و اشباع
- مشخص شدن فرکانس نمونه‌برداری و ایجاد آن
- بلوک ورودی : خواندن از A/D شامل نگاشت به مقدار در بازهٔ ۱±
- ایجاد ساختار محاسبات فیلتر (موازی، سری، pipeline با تعداد طبقات مناسب)، اعمال محاسبات بر مقادیر تا تولید خروجی
- نگاشت خروجی فیلتر مطابق با نیازهای خروجی فیزیکی (D/A) و قرار دادن در D/A
- در صورت استفاده از ساختارهای سری یا pipeline تکرار عملیات به تعداد طبقات مناسب

در برنامه‌ای که همراه متن می‌آید نمونهٔ بلوکهای مورد استفاده در یک فیلتر (به عنوان راهنما، ایجاد ایده) مشاهده می‌شود.

کار آزمایشگاهی

۱- بر روی بورد FPGA ، با فرکانس نمونه‌برداری ثابت، از ورودی آنالوگ مقدار را خوانده و در خروجی آنالوگ بنویسید. ورودی می‌تواند در ابتدا مقدار ثابت باشد (در محدودهٔ مجاز ورودی) و در نهایت سیگنالی AC (مجدداً در محدودهٔ مجاز ورودی) با فرکانس مناسب در نظر گرفته شود. از صحت عملکرد مطمئن شوید.

۲- کد واحد ریاضی–منطقی را بر اساس نیاز در دستورالعملهای مختلف نوشته و صحت عملکرد آن را آزمایش نمایید.

۳- از ترکیب دو بخش قبلی ساختار کامل مسیر داده را در ساختار موازی برای فیلتر FIR مرتبهٔ ۲ ساخته و با اعمال ورودیهای مناسب از صحت عملکرد آن مطمئن شوید.

۴- ساختار سری را برای فیلتر ایجاد و مورد آزمایش قرار دهید : نخست برای فیلتر مرتبهٔ ۲ و سپس برای فیلتر تا مرتبهٔ ۲۰.

۵- ساختار pipeline را برای فیلتر ایجاد نموده و مورد آزمایش قرار دهید.

فعالیت قبل از آزمایش :

- در ادامهٔ متن، توضیحات مرتبط با طراحی پروسسور را مطالعه نموده و کد پیاده‌سازی آن را طی مراحل خواسته شده بنویسید.

- **برنامه‌های نوشته شده در ابتدای جلسه بررسی می‌شود.**

پیش‌زمینه : تعاریف مورد نیاز در طراحی پروسسور : رجیسترها، مسیر داده و کنترل عملیات لازم در پروسسور

از مهمترین بخشهای یک پردازنده مسیر دادهٔ آن است. منابع اصلی مورد نیاز در دستورالعملهای پردازنده در این بخش تعریف می‌شوند. نحوهٔ دسترسی به این منابع نیز ساختار مسیر داده را مشخص می‌نماید. مجموعهٔ رجیسترها (شامل رجیسترهای مشهود برای کاربر و رجیسترهای مخفی) و ALU و ارتباط بین آنها دقیقاً باید تعیین شوند.

پردازندهٔ مورد نظر در این آزمایش، یک پردازندهٔ سادهٔ ۸بیتی با نام VX2 است : رجیسترها، گذرگاه دادهٔ داخلی و خارجی، حافظهٔ داده و برنامه و آدرس همگی ۸بیتی هستند. لیست دستورالعملها و ساختار آنها در جدول آورده شده است. مشخصات کلی آن چنین است :

- رجیسترهای PC و IR ۸بیتی هستند.
- مجموعهٔ پرچمها در رجیستر ۸بیتی F (C ، Z ، H ، V/P ، S، I) قرار دارند.
- دارای ۸ رجیستر است (R0 تا R7).
- پشته به صورت سخت‌افزاری با عمق ۳۱ مرحله است (۳۱ بایت : SP0 تا SP30).
- دارای ساختار هاروارد است (حافظهٔ برنامه جدا از حافظهٔ داده) : هر کدام ۲۵۶ بایت
- فضای آدرس حافظه از I/O جداست. دسترسی به فضای I/O از طریق دستورهای IN و OUT است. سیگنال MR دسترسی به حافظهٔ داده و IOR دسترسی به I/O را مشخص می‌کنند.
- برخی دستورالعملها نیاز به ۲ بار دسترسی به حافظهٔ برنامه دارند (نخست Opcode Fetch و سپس خواندن عملوند n).
- دارای یک ورودی اینتراپت و پرچم اینتراپت I است. اجرای اینتراپت موکول به فعال (1 بودن) پرچم اینتراپت I است. در این صورت و در صورت فعال بودن ورودی اینتراپت، بر روی لبهٔ clk (و در انتهای اجرای یک دستورالعمل)، یک Call به آدرس FE(hex) اجرا می‌گردد (معمولاً در این آدرس JP به آدرس روتین اینتراپت قرار داده می‌شود). در پرچم I نیز صفر قرار داده می‌شود.

کار آزمایشگاهی

۱- کد مجموعهٔ رجیسترهای تعریف شده را بر اساس نحوهٔ دسترسی به آنها در دستورالعملهای مختلف نوشته و از عملکرد صحیح آن مطمئن شوید.

۲- کد واحد ریاضی–منطقی را بر اساس نیاز در دستورالعملهای مختلف نوشته و صحت عملکرد آن را آزمایش نمایید.

۳- از ترکیب دو بخش قبلی ساختار کامل مسیر داده را ساخته و با اعمال ورودیهای مناسب از صحت عملکرد آن مطمئن شوید.

۴- مدل حافظه‌های برنامه، داده و پشته را پیاده‌سازی و آزمایش نمایید.

۵- عملکرد اینتراپت را پیاده‌سازی و آزمایش کنید.

۶- مجموعهٔ کل پروسسور را ترکیب نموده و صحت عملکرد آن را برای دستورالعملهای مختلف آزمایش کنید.

۷- بکمک یک برنامهٔ آزمون، حداکثر سرعت پروسسور را بدست آورید.

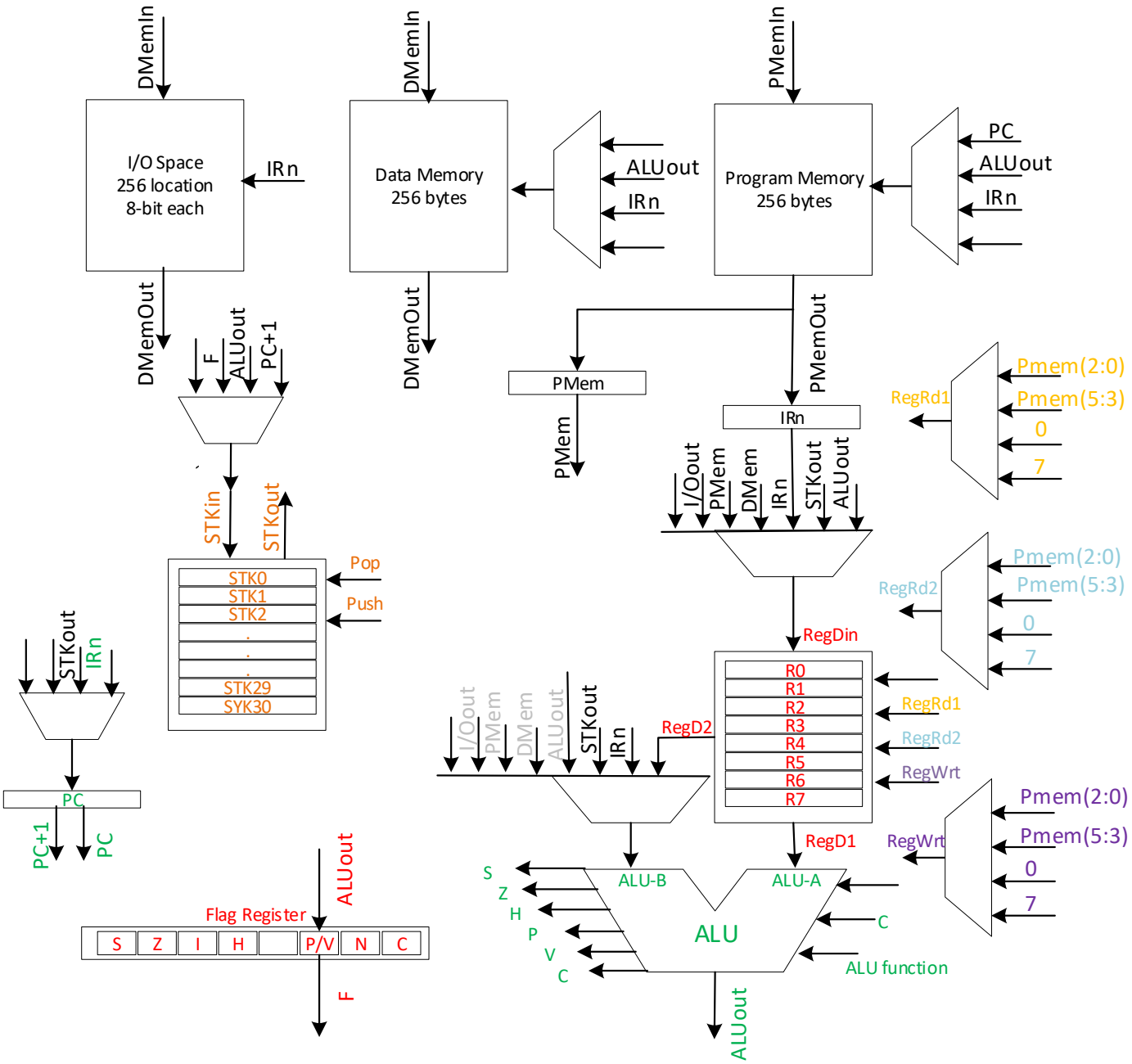
VX2 Instruction Set

	Mnemonic		Opcode Structure								Affected Flags					Operations in the Instruction										
			7	6	5	4	3	2	1	0	S	V/P	H	Z	C											
1	MOV	Rd , Rs	1	0	Rs			Rd			-	-	-	-	-	Rd	←	Rs								Register to Register Transfer
2	LD	Rd , n	0	1	0	0	1	Rd			-	-	-	-	-	Rd	←	n								Immediate to Register Transfer
			n																							
3	LD	R0 , (n)	0	1	0	1	0	0	0	0	-	-	-	-	-	Rd	←	(n)								Memory to Register Transfer
			n																							
4	LD	Rd , (R7+n)	0	1	0	1	1	Rd			-	-	-	-	-	Rd	←	(R7+n)								Memory to Register Transfer
			n																							(indirect Register Addressing)
5	LDP	R0 , (n)	0	1	0	1	0	0	0	1	-	-	-	-	-	R0	←	(n)								Prog. Memory to R0 Transfer
			n																							
6	LDP	R0 , (R7)	0	1	0	1	0	0	1	0	-	-	-	-	-	R0	←	(R7)								Prog. Memory to R0 Transfer
7	ST	(n) , R0	0	1	0	1	0	0	1	1	-	-	-	-	-	(n)	←	R0								R0 to Prog. Memory Transfer
			n																							
8	ST	(R7+n) , Rs	0	1	1	0	1	Rs			-	-	-	-	-	(R7+n)	←	Rs								Register to Memory Transfer
			n																							(indirect Register Addressing)
9	STP	(n) , R0	0	1	0	1	0	1	0	0	-	-	-	-	-	(n)	←	R0								R0 to Prog. Memory Transfer
			n																							
10	STP	(R7) , R0	0	1	0	1	0	1	0	1	-	-	-	-	-	(R7)	←	R0								R0 to Prog. Memory Transfer
11	PUSH	Rs	0	1	1	1	0	Rs			-	-	-	-	-	STK(i+1)	←	STK(i)	,	STK(0)	←	Rs				Register to Stack Transfer
12	POP	Rd	0	1	1	1	1	Rd			-	-	-	-	-	Rd	←	STK(0)	,	STK(i)	←	STK(i+1)				Stack to Register Transfer
13	PUSHF		0	1	1	0	0	1	0	0	-	-	-	-	-	STK(i+1)	←	STK(i)	,	STK(0)	←	F				Flag Register to Stack Transfer
14	POPF		0	1	1	0	0	1	0	1	↕	↕	↕	↕	↕	F	←	STK(0)	,	STK(i)	←	STK(i+1)				Stack to Flag Register Transfer
15	OUT	(n) , R0	0	1	1	0	0	1	1	0	-	-	-	-	-	I/O(n)	←	R0								R0 to I/O Transfer
			n																							
16	IN	R0 , (n)	0	1	1	0	0	1	1	1	-	-	-	-	-	R0	←	I/O(n)								I/O to R0 Transfer
			n																							
17	SWAP	R0	0	1	1	0	0	0	0	0	-	-	-	-	-	R0(3:0)	←	R0(7:4)	,	R0(7:4)	←	R0(3:0)				Swap Register Nibbles
18	ADC	R0 , Rs	0	0	0	0	0	Rs			↕	↕	↕	↕	↕	R0	←	Rs+R0+C									Add Register to R0 + Carry Flag
19	SBC	R0 , Rs	0	0	0	0	1	Rs			↕	↕	↕	↕	↕	R0	←	Rs-R0-C									Subtract Register from R0 - Carry Flag
20	AND	R0 , Rs	0	0	0	1	0	Rs			↕	↕	-	↕	0	R0	←	R0&Rs									And Register to R0
21	ANDI	Rd , n	0	0	0	1	1	Rd			↕	↕	-	↕	0	Rd	←	Rd&n									And Immediate to R0
			n																							
22	OR	R0 , Rs	0	0	1	0	0	Rs			↕	↕	-	↕	1	R0	←	R0|Rs									Or Register to R0
23	ORI	Rd , n	0	0	1	0	1	Rd			↕	↕	-	↕	1	Rd	←	Rd|n									Or immediate to R0
			n																							
24	XOR	R0 , Rs	0	0	1	1	0	Rs			↕	↕	-	↕	0	Rd	←	R0^Rs									Xor Register to R0
25	NOT	Rd	0	0	1	1	1	Rd			↕	↕	-	↕	-	Rd	←	~Rd									Complement Register
26	CPL	Rd	1	1	0	0	0	Rd			↕	↕	-	↕	-	Rd	←	-Rd									Negate Register (2's Complement)
27	RL	Rd	1	1	0	0	1	Rd			↕	↕	↕	↕	↕	C	←	Rd(7)	,	Rd(i+1)	←	Rd(i)	,	Rd(0)	←	C	Rotate Left
28	RR	Rd	1	1	0	1	0	Rd			↕	↕	↕	↕	↕	C	←	Rd(0)	,	Rd(i-1)	←	Rd(i)	,	Rd(7)	←	C	Rotate Right
29	SL	Rd	1	1	0	1	1	Rd			↕	↕	↕	↕	↕	C	←	Rd(7)	,	Rd(i+1)	←	Rd(i)	,	Rd(0)	←	0	Shift Left
30	SRA	Rd	1	1	1	0	0	Rd			↕	↕	↕	↕	↕	C	←	Rd(0)	,	Rd(i-1)	←	Rd(i)	,	Rd(7)	←	Rd(7)	Shift Arith. Right
31	SRL	Rd	1	1	1	0	1	Rd			↕	↕	↕	↕	↕	C	←	Rd(0)	,	Rd(i-1)	←	Rd(i)	,	Rd(7)	←	0	Shift Left
32	INC	Rd	1	1	1	1	0	Rd			↕	↕	-	↕	-	Rd	←	Rd+1									Increment Register
33	DEC	Rd	1	1	1	1	1	Rd			↕	↕	-	↕	-	Rd	←	Rd-1									Decrement Register

VX2 Instruction Set

	Mnemonic		Opcode Structure								Affected Flags					Operations in the Instruction
			7	6	5	4	3	2	1	0	S	V/P	H	Z	C	
34	JP	n	0	1	0	1	0	1	1	0	-	-	-	-	-	PC ← n
			n													
45	JPZ	n	0	1	0	0	0	0	0	0	-	-	-	-	-	if Z=1 then PC ← n
			n													
36	JPNZ	n	0	1	0	0	0	0	0	1	-	-	-	-	-	if Z=0 then PC ← n
			n													
37	JPC	n	0	1	0	0	0	0	1	0	-	-	-	-	-	if C=1 then PC ← n
			n													
38	JPNC	n	0	1	0	0	0	0	1	1	-	-	-	-	-	if C=0 then PC ← n
			n													
39	JPP	n	0	1	0	0	0	1	0	0	-	-	-	-	-	if S=0 then PC ← n
			n													
40	JPM	n	0	1	0	0	0	1	0	1	-	-	-	-	-	if S=1 then PC ← n
			n													
41	JPV	n	0	1	0	0	0	1	1	0	-	-	-	-	-	if V=1 then PC ← n
			n													
42	JPNV	n	0	1	0	0	0	1	1	1	-	-	-	-	-	if V=0 then PC ← n
			n													
43	CALL	n	0	1	1	0	0	0	0	1	-	-	-	-	-	STK(i+1) ← STK(i) , STK(0) ← PC , PC ← n
			n													
44	RET		0	1	1	0	0	0	1	0	-	-	-	-	-	PC ← STK(0) , STK(i) ← STK(i+1)
45	RETI		0	1	1	0	0	0	1	1	-	-	-	-	-	PC ← STK(0) , STK(i) ← STK(i+1) , I ← 1

نمونهٔ بخشهای سازندهٔ پروسسور و ارتباطها در شکل زیر آورده شده است :


DMemIn
I/O Space
256 location
8-bit each
IRn
DMemOut
DMemIn
Data Memory
256 bytes
ALUout
IRn
DMemOut
PMemIn
Program Memory
256 bytes
PC
ALUout
IRn
PMemOut
PMem
PMem
IRn
I/Oout
PMem
DMem
IRn
STKout
ALUout
RegRd1
Pmem(2:0)
Pmem(5:3)
0
7
F
ALUout
PC+1
STKin
STKout
STK0
STK1
STK2
STK29
SYK30
Pop
Push
RegDin
RegRd2
Pmem(2:0)
Pmem(5:3)
0
7
R0
R1
R2
R3
R4
R5
R6
R7
RegRd1
RegRd2
RegWrt
RegD1
RegWrt
Pmem(2:0)
Pmem(5:3)
0
7
STKout
IRn
PC
PC+1
PC
I/Oout
PMem
DMem
ALUout
STKout
IRn
RegD2
S
Z
H
P
V
C
ALU-B
ALU-A
ALU
C
ALU function
ALUout
Flag Register
S
Z
I
H
P/V
N
C
ALUout
F