Application Note
ZHCA703 – Apr 2017

在 VisionSDK 框架下开发 EVE 算法的方法

James Li Automotive FAE

摘要

EVE 的全称是 Embedded Vision/Vector Engine。是集成在 TI TDAxx 系列芯片中的视觉协处理器。TDA2x 芯片中集成了 4 个 EVE，主频可以达到 650MHz。TDA3x 芯片中集成了 1 个 EVE，主频可以达到 667MHz。在图像处理应用中，EVE 适合做逐像素的密集运算，例如滤波，梯度计算，sobel 算子等。处理这些算子时单核 EVE 的处理能力是同主频的单核 C66 DSP 的 2～3 倍。EVE 的运算能力很强，但是软件开发难度较大。本文结合作者的开发经验，探索了一套 EVE 软件开发流程。使得在 VisionSDK 框架下用户可以方便的集成调用自有的 EVE 算法模块。本文是对上述 EVE 软件开发流程的介绍。先简单介绍 EVE 的硬件架构，使读者对其硬件能力，编程特点有初步的认识。然后分三步介绍了作者提出的 EVE 算法开发步骤：算法内核的开发，算法主体的开发，和集成到 VisionSDK 框架的步骤。作者根据这套流程实现了一个简单的算法样例：图像梯度计算。最后是样例的实测结果和效率分析。

目录

Figures

Tables

1 EVE 简介

EVE 内部包括一个 RISC 处理器 ARP32 和一个向量处理器 VCOP，还集成了一个 DMA 控制器。ARP32 的运算能力很弱，主要执行控制功能，例如控制 DMA 把 block 数据输入到 VCOP 的 RAM 或者从 VCOP 的 RAM 输出结果到 DDR。VCOP 用来执行算法。VCOP 可以被看作一个协处理器。VCOP 的运行是受 ARP32 控制的。ARP32 首先初始化 VCOP 的循环参数，然后启动 VCOP。VCOP 运行中受循环控制电路的控制（循环参数是 ARP32 初始化的）。VCOP 运行中 ARP32 通常处于查询等待的状态。

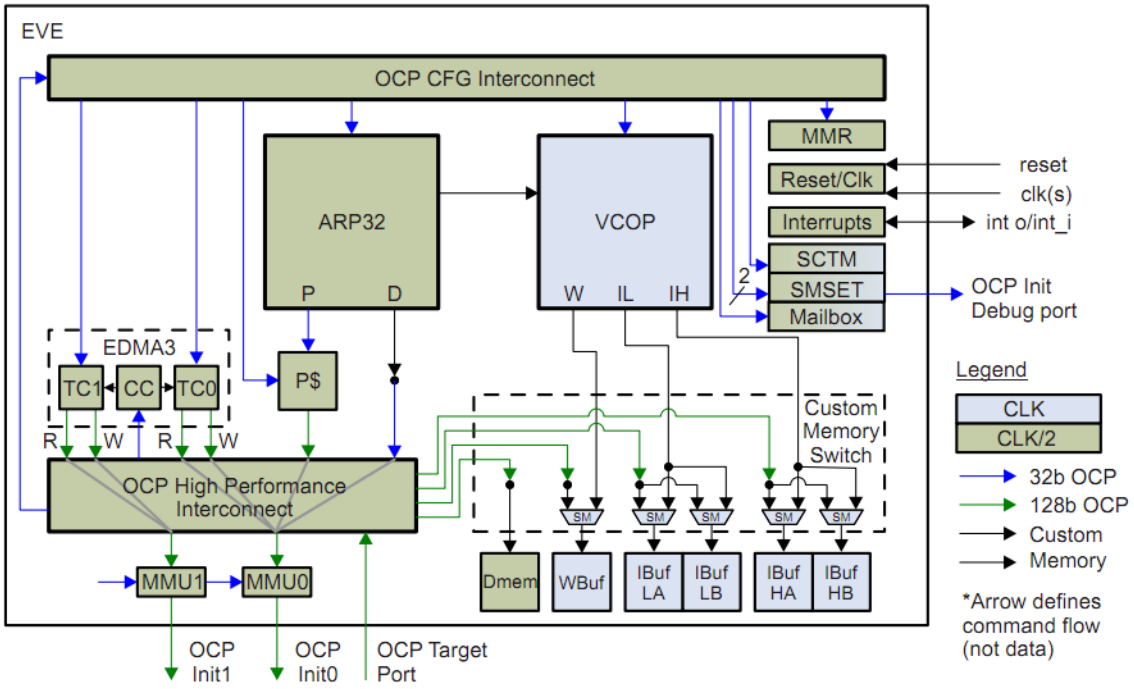

Figure 1. EVE 硬件框架

EVE 片内也集成少量 RAM，最主要的是 3 个 32K 的独立 buffer： WBUF， IBUFA 和 IBUFB。WBUF 是 VCOP 的 working buffer，主要用来存储 VCOP 运行时参数，常量系数和中间数据。IBUFA 和 IBUFB 通常用来存储输入输出数据。ARP32 可以访问 DDR 和 EVE 的片内 RAM，但 VCOP 只能访问 WBUF，IBUFA 和 IBUFB。因为 EVE 的片内 RAM 非常小，远装不下整幅图像，所以图像需要切块，用 DMA 搬移到 EVE 片内 RAM 给 VCOP 处理，处理的结果再从 EVE 片内 RAM 搬回 DDR，拼接成完整的输出图像。因为 ARP32 和 VCOP 都可以访问这 96K bytes 内部 RAM，为了防止冲突，EVE 内部有 buffer switch 硬件。当一个 buffer 切换给 SYSTEM 控制时，ARP32 和 DMA 才能访问它，在这期间 VCOP 访问这个 buffer 的硬件通路被切断。当一个 buffer 切换给 VCOP 控制， VCOP 才能向其中读写数据，在这期间 ARP32 和 DMA 访问这个 buffer 的硬件通路被切断。为了提高效率，有专用指令控制 Buffer switch 切换。WBUF, IBUFA, IBUFB 的 buffer switch 可以独立控制。IBUFA 和 IBUFB 还可以分成低 16K 和高 16K 分别控制 buffer switch。标记为 IBUFLA，IBUFHA 和 IBUFLB，IBUFHB。EVE 适合做像素级的处理，适合做不含 if/else 判断，不含 break, continue 等控制跳转的循环代码。EVE 支持常规的算术逻辑运算，例如：加、减、乘、乘加、乘减、求绝对值，逻辑与、或、异或，二选一求最大、求最小，移位，比较等等。还有专用的查表指令和计算直方图指令。单指令可以执行 8 个 SIMD 操作。

本文介绍的 EVE 算法开发流程包括三个步骤（如 Figure2）：第一步开发算法内核，也就是 VCOP 的 kernel 函数，代码优化是这步工作的重点。第二步实现算法的主体，算法的主体是一个控制 DMA 输入输出数据和并行调度 VCOP 执行 kernel 函数的框架。一个高效的框架要避免 VCOP 处于空闲状态并且尽可能简化 DMA 的控制。第三步是把算法集成到 VisionSDK 中，有四个修改点，缺一不可。

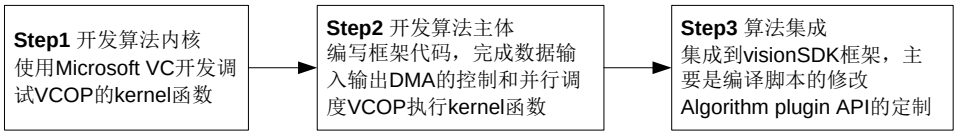

Figure 2.　EVE 算法开发的步骤

2　开发算法内核

EVE 算法的内核就是 VCOP 执行的 kernel 函数。前面已经提到 EVE 内部包括一个 RISC 处理器 ARP32 和一个向量处理器 VCOP。ARP32 用普通的 C 编程，VCOP 要使用 kernel C 编程。Kernel C 和 C 语言的语法基本相同，只是受限于 VCOP 硬件架构，kernel C 的循环编写有一些限制，例如：循环中输入输出 buffer 的地址偏移的步进有限制，除了加减等运算可以直接使用+、-符号外，很多 VCOP 特有的运算要调用 intrinsic 指令。具体请参见参考文献 1，2 和 4。EVE 硬件和 EVE simulator 都不支持 VCOP kernel 循环的单步调试，本文推荐的做法是用 Microsoft VC 做功能仿真。因为 EVE 的所有指令的功能仿真函数都定义在 visionSDK_path\ti_components\cg_tools\windows\arp32_1.0.7\include\vcop\目录下的 vcop.h 中。只要把 EVE 的 kernel C 文件后缀由.k 修改成.cpp，然后 include “vcop.h” 就可以把 kernel 程序编译成 VC 程序在 Microsoft Visual Studio 中运行。

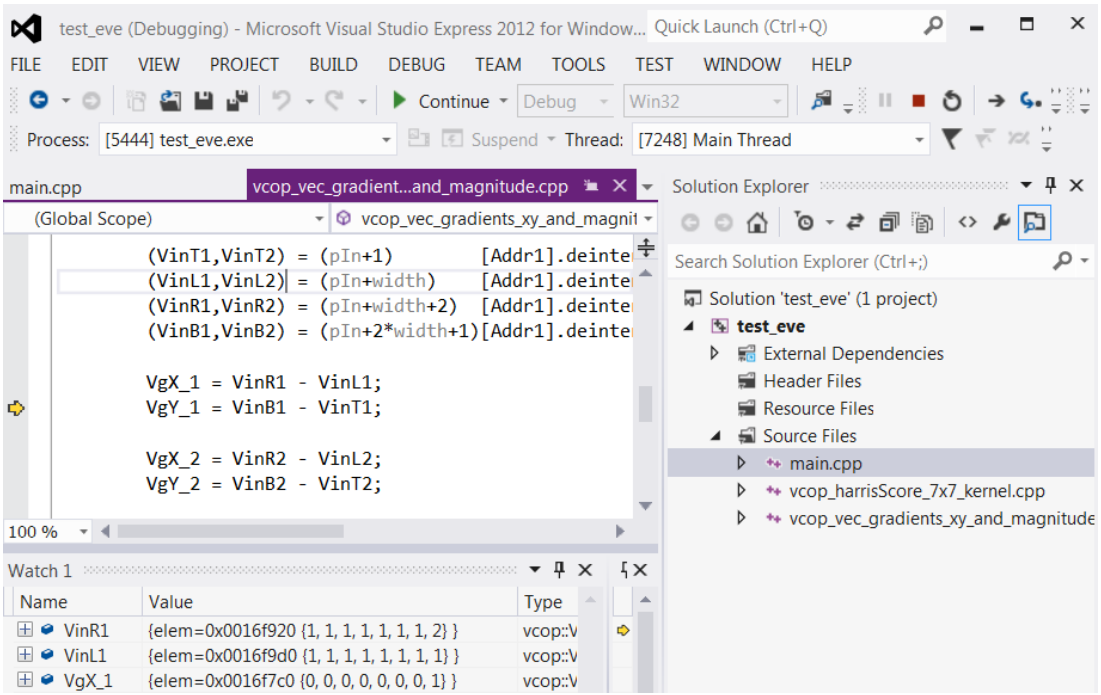

Figure 3.　VCOP kernel 函数的 VC 仿真

使用 VC 仿真的几个明显优点是：

- VC 程序可以单指令仿真。可以方便的查看向量寄存器的内容。
- VC 仿真可以通过文件输入输出的方式便捷的把中间结果存储成文件做 debug。
- VC 仿真不受 VCOP 内部 buffer size 的限制，可以直接处理完整幅图像。

VCOP kernel 函数的编写有很多优化技巧。TI 提供了很多开源的优化样例可供参考学习。例如 VisionSDK 安装包中包含了 VLIB，其最新的 release 中有 64 个 TI 专家优化过的视觉处理算法相关的功能函数。

3 开发算法主体

EVE 算法主体负责数据输入输出 DMA 的控制并调度 VCOP 执行 kernel 函数。如前所述，EVE 实际处理一帧图像的时候需要把图像切成小块用 DMA 搬移到内部 RAM。VCOP 的处理结果也只能存储在 VCOP 内存，还要用 DMA 从 VCOP 内存搬到 DDR，拼成一帧完整的输出图像。如果把输入输出缓冲区设计成 ping/pong buffer，就可以在当前 block 计算的同时把前一 block 的结果输出并输入下一 block 的数据。

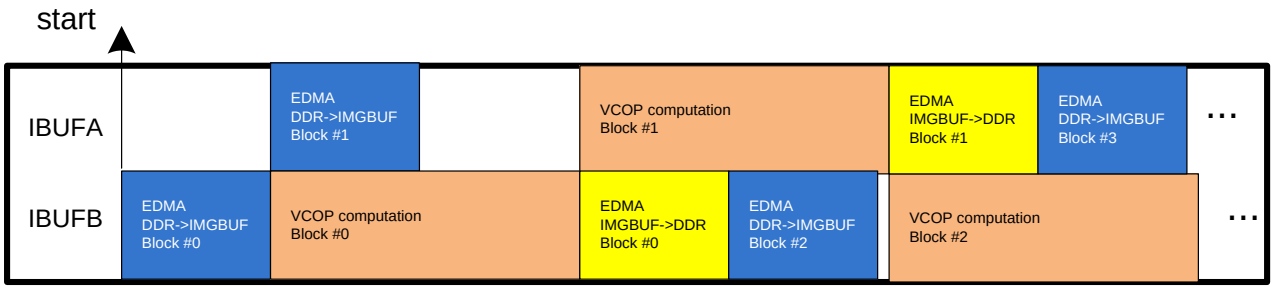

Figure 4. 数据分块 DMA 和 VCOP 并行的示意图

在介绍 DMA 配置前，有必要阐述清楚 EVE 内部 RAM 的两种地址映射模式。为了更高效的支持数据输入输出的 ping/pong， EVE 内部 RAM 支持两种地址映射模式：Full memory map 模式和 Aliased memory map 模式。我们已经知道 IBUFA 的基地址是 0x40050000，IBUFB 的基地址是 0x40054000。在 Full memory map 模式下，VCOP 和 DMA 都以地址 0x40050000 访问 IBUFA，以地址 0x40054000 访问 IBUFB。在 Aliased memory map 模式下，IBUFA 和 IBUFB 的地址映射是相同的，VCOP 和 DMA 物理上访问的是哪一个 buffer 取决于 buffer switch 的设置。当 buffer switch 切换成 VCOP 控制 IBUFA， DMA 控制 IBUFB 的时候，VCOP 以地址 0x40050000 访问 IBUFA，DMA 以地址 0x40050000 访问 IBUFB。当 buffer switch 切换成 VCOP 控制 IBUFB， DMA 控制 IBUFA 的时候，VCOP 以地址 0x40050000 访问 IBUFB，DMA 以地址 0x40050000 访问 IBUFA。使用 Aliased memory map 模式的好处是：

- VCOP 参数 buffer 中的数据指针不需要随 IBUFA/IBUFB 的 ping/pong 切换而更新。
- 输入 DMA 的目的地址和输出 DMA 的源地址不需要随 IBUFA/IBUFB 的切换而更新。

算法主体的框架要避免 VCOP 处于空闲状态并且尽可能使 DMA 的控制简单。下面以输入 DMA 为例来分析 DMA 的控制。输入 DMA 每次输入一个 block(如 Figure5)，除源地址外 DMA 参数不需要更新。

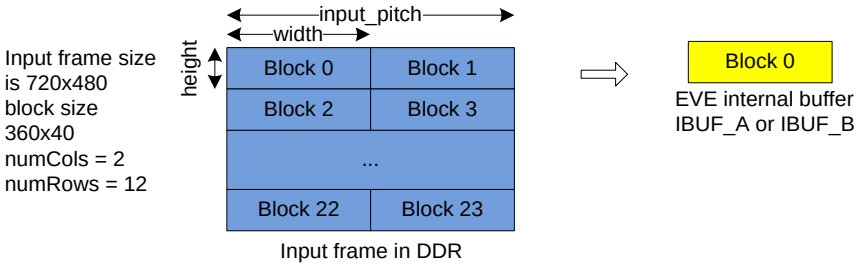

Figure 5. 数据分块 DMA 示意图

本文按照 Figure6 的参数配置 DMA，关于 DMA 相关参数的详细定义见参考文献 3。

- PaRAM->Opt->syndim = 1， DMA 配置成 AB SYNC 模式。每次传输 acnt*bcnt 个数据。
- PaRAM->Opt->static = 0，每个 block 传输完后需要 DMA 硬件自动更新 PaRAM 参数。
- PaRAM->Opt->ITCINTEN = 1 并且 PaRAM->Opt->TCINTEN = 1，使得每个 block 传输完成后都可以置位中断标志。软件要通过查询中断标志来等待传输完成然后调度 VCOP 处理数据。
- 设置 acnt 等于 block 的宽度加 padding size， bcnt 等于 block 的高度加 padding size。Ccnt 等于一行的 block 个数（例如 Figure5 中一行有两个 block）。因为 block 的尺寸相同，所以 acnt，bcnt 和 ccnt 不需要更新。考虑 padding size 是必需的，因为分块计算的时候数据块的边界需要做 padding，padding 的尺寸跟算法相关。
- 设置 sbidx 等于输入图像帧的 input pitch（如图）， dbidx 等于 block 的宽度。
- 设置 scidx 等于 block 的宽度， dcidx 等于 0。

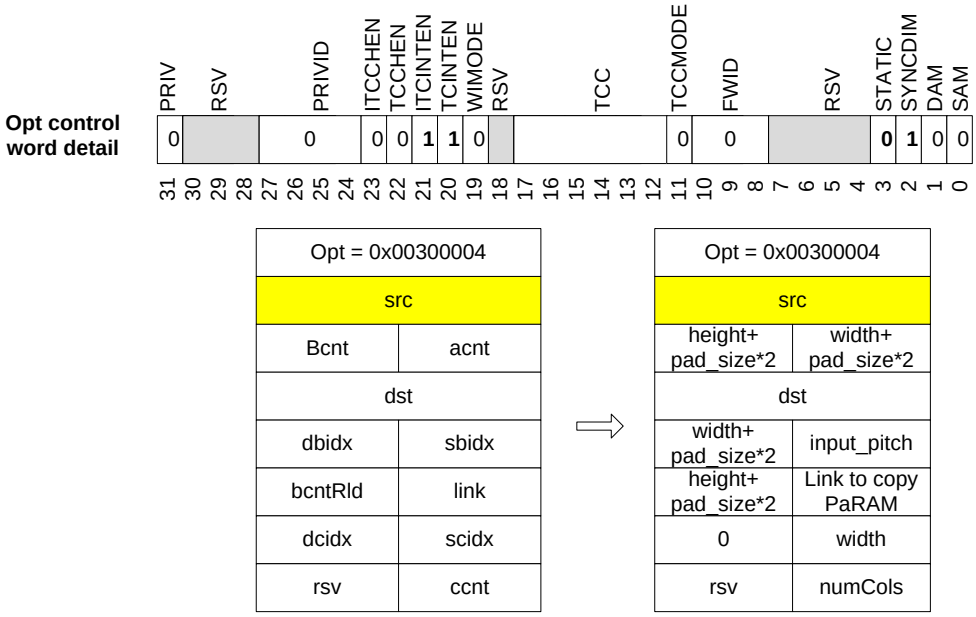

Figure 6. DMA PaRAM 参数配置

– 设置 bcntRld = bcnt，使得一个 block 传输完后 PaRAM 能重新加载 bcnt

– 关于 src 的设置，因为使用了 aliased memory map 模式，所以目的地址不需要更新。因为配置了 scidx，所以同一行 block 的源地址会自动更新。每行第一个 block(例如图 4 中的 block0,2,4,6…)的源地址需要软件更新。

– 关于 link 域的设置，在申请 DMA 资源的时候，软件为 input DMA channel 申请了两个 PaRAM 如 Figure6。我们分别称之为 running PaRAM 和 copy PaRAM。初始化的时候，这两个 PaRAM 被初始化成相同的内容，然后把 running PaRAM link 到 copyPaRAMA，把 copy PaRAMA link 到自己。因为 PaRAM->Opt->static=0，所以每个 block 传输完后 DMA 硬件都会自动更新 running PaRAM，当一行 block 传输完以后（ccnt 递减到 0 的时候）running PaRAM 会被清零。因为 running PaRAM link 到了一个新的 PaRAM，这时 DMA 硬件会把所 link 的 PaRAM 的内容拷贝到当前 PaRAM。就不需要软件重配 running PaRAM 了。

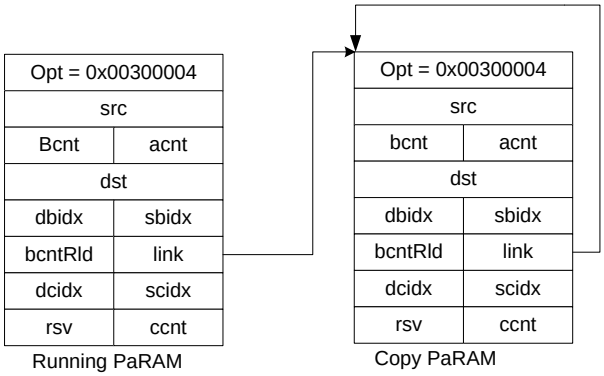

Figure 7. DMA PaRAM link

上述的 DMA 配置使得运行中软件只需要更新很少的 DMA 参数。下面的叙述中，本文把算法主体按功能分成三个函数：Input_one_block, process_one_block 和 output_one_block。

```
void input_one_blk(rowIn, colIn) // 输入一个 block
{
    if( colIn == 0 ) // 每一行的第一个 block，软件更新源地址
        update input DMA PaRAM source address
    triggert input DMA
}

void output_one_blk(rowOut, colOut) // 输出一个 block
{
    if( colOut == 0 ) // 每一行的第一个 block，软件更新目的地址
        update output DMA PaRAM destination address
    triggert output DMA
}

void process_one_blk() // 处理一个 block
{
    call VCOP
    wait for VCOP to finish
}
```

下面的伪代码调用上述三个处理步骤完成一帧图像的处理。参照了软件流水编写前缀，循环内核和后缀的方式，我们把这个调度过程分成前缀，内核和后缀三部分。前缀部分并行度逐渐增加，流水线开始充满，内核部分 VCOP 的运行和输入输出 DMA 的传输并行，后缀部分并行度又逐渐降低，流水线开始排空。

```
Allocate DMA resources
Initial DMA PaRAMs
Intial VCOP parameter buffers

//============== prelog =================
input_one_blk();// input A
DMA_WAIT(input_channel);
switchFlag = VCOP_BUF_SWITCH_TOGGLE (switchFlag);// A for VCOP, B for SYST
input_one_blk(rowIn, colIn);// input B
process_one_blk( paramBuf);  // process A

//============== kernel =================
for( i = 0; i < totalNumBlks-2; i++ )
{
    switchFlag = VCOP_BUF_SWITCH_TOGGLE (switchFlag);// B for VCOP, A for SYST
    output_one_blk(rowOut, colOut);// output A
    DMA_WAIT(input_channel);
    input_one_blk(rowIn, colIn);// input A
    process_one_blk( paramBuf); // process B
    DMA_WAIT(output_channel);
}
//============== epilog =================
switchFlag = VCOP_BUF_SWITCH_TOGGLE (switchFlag); // We assume now is: B for VCOP, A
for SYST
output_one_blk(rowOut, colOut); // output A
DMA_WAIT(input_channel);
process_one_blk( paramBuf);// process B
switchFlag = VCOP_BUF_SWITCH_TOGGLE (switchFlag); // A for VCOP, B for SYST
output_one_blk(rowOut, colOut); // output B
DMA_WAIT(output_channel);
```

至此，完成了算法主体的开发。可以进行单元测试。单元测试通过后就可以集成到 VisionSDK 中。

4 集成算法到 VisionSDK 框架

由于 TDAxx 系列芯片的相关软件都是基于 VisionSDK 框架开发的，所以用户的 EVE 算法需要集成到 VisionSDK 软件框架才能方便的调用。VisionSDK 软件框架是开放的，用户集成自己算法需要的操作见 Table1。

Table 1. 集成 EVE 算法到 VisionSDK 框架的步骤

修改内容	修改位置	说明
创建算法源码	In folder vision_sdk\examples\tda2xx\src\alg_plugins\	VSDK 要求每个算法包含 5 个功能函数，分别是： create, process, stop, control 和 delete。还要实现一个新算法的注册函数，注册函数把新算法的五个功能函数挂接到 VSDK 框架中。
在 VSDK 编译脚本中包含这个新算法	vision_sdk\examples\tda2xx\src\alg_plugins\makefile	例如：include $(MODULE_SRC_BASE_PATH)/eveAlgExam1/SRC_FILES.MK
在 VSDK 头文件中定义新算法的 ID	vision_sdk\include\link_api\algorithmLink.h	
在 VSDK 函数 AlgorithmLink_initAlgPlugins 中调用新算法的注册函数	vision_sdk\src\links_common\algorithm\algorithmLink_cfg.c	这个注册函数会在 VSDK 框架启动的时候被调用

新算法集成成功后，应用可以通过算法 ID 调用它。在 VisionSDK 的目录 vision_sdk\examples\tda2xx\src\usecases 下有很多调用算法的例子。

5 效率分析

本文以梯度计算为例在硬件上测试了上述调度框架的运行效率。输入输出图像的尺寸都是 720x480。代码中有记录时间戳的功能。在程序执行过程中把处理每个 block 的每个步骤（input_one_block，process_one_block，output_one_block，wait input DMA 和 wait output DMA）的起止时间记录在 DDR 中，然后导出分析，最终画出下面的执行图（Figure8）。图中用不同的符号表示不同的处理步骤，每个符号的时间刻度是 100 个 ARP32 clock（不足 100 clock 按 100 clock 计算）。

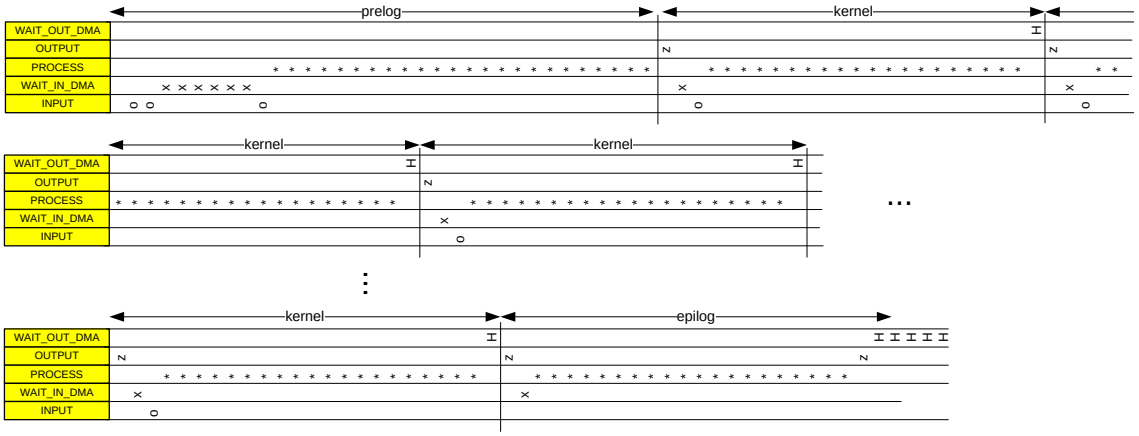

Figure 8. 梯度计算样例的执行时序图

各个步骤运行的时间和占比也统计在 Table2 中，可以看到 VCOP 的运行时间占到了总时间的 93%以上，说明本文所述的调度框架效率很高。VCOP 的空闲时间很短。

Table 2. 调度框架各步骤的时间占比

	INPUT	WAIT_IN	PROCESS	OUTPUT	WAIT_OUT
最短时间	27	25	3719	47	24
最长时间	269	1044	3883	174	42
平均时间	77	76	3725	85	23
总时间	1869	1824	89420	2047	573
占比	1.95%	1.91%	93.41%	2.14%	0.60%

本文还统计了帧图像切块尺寸对总运行时间的影响见 Table3。可以看出帧数据切块的大小对 EVE 的运算效率有很大影响。数据块小则数目多，DMA 搬移的次数多，附带开销会增加。而且 VCOP 每处理一个数据块，硬件 pipeline 都要重新 ramp up，大约需要 40 个 VCOP clock。因此要在 EVE 片内存储空间允许的情况下使用尽可能大的数据切块。

Table 3. 不同数据切块尺寸对应的处理时间

number of blocks	block width	block height	total cycle (arp32 clocks)
24	360	40	103632
48	180	40	113617
96	90	40	127139
192	90	20	157009
384	90	10	216291

6 总结

EVE 是 TI TDAxx 系列芯片中重要的视觉计算引擎。本文介绍了一种开发 EVE 算法并集成到 VisionSDK 框架的方法。实测表明使用这种方法可以简单高效的调用 EVE。所以本文对使用 TDAxx 系列芯片的用户具有很好的参考意义。

参考文献

1. *VCOP Kernel C Reference Guide SPRUHB9*
2. *VCOP CPU and Instruction Set Reference Guide SPRUHC0A*
3. *Technical Reference Manual (SPRUHK5) section 16 DMA Controllers*
4. *Embedded Vector Engine (EVE) Programmer's Guide SPRUHC1B*

有关 TI 设计信息和资源的重要通知

德州仪器 (TI) 公司提供的技术、应用或其他设计建议、服务或信息，包括但不限于与评估模块有关的参考设计和材料（总称“TI 资源”），旨在帮助设计人员开发整合了 TI 产品的 应用； 如果您（个人，或如果是代表贵公司，则为贵公司）以任何方式下载、访问或使用了任何特定的 TI 资源，即表示贵方同意仅为该等目标，按照本通知的条款进行使用。

TI 所提供的 TI 资源，并未扩大或以其他方式修改 TI 对 TI 产品的公开适用的质保及质保免责声明；也未导致 TI 承担任何额外的义务或责任。TI 有权对其 TI 资源进行纠正、增强、改进和其他修改。

您理解并同意，在设计应用时应自行实施独立的分析、评价和 判断， 且应全权负责并确保 应用的安全性， 以及您的 应用 （包括应用中使用的所有 TI 产品）） 应符合所有适用的法律法规及其他相关要求。你就您的 应用声明，您具备制订和实施下列保障措施所需的一切必要专业知识，能够 (1) 预见故障的危险后果，(2) 监视故障及其后果，以及 (3) 降低可能导致危险的故障几率并采取适当措施。您同意，在使用或分发包含 TI 产品的任何 应用前， 您将彻底测试该等 应用 和该等应用所用 TI 产品的 功能。除特定 TI 资源的公开文档中明确列出的测试外，TI 未进行任何其他测试。

您只有在为开发包含该等 TI 资源所列 TI 产品的 应用时， 才被授权使用、复制和修改任何相关单项 TI 资源。但并未依据禁止反言原则或其他法理授予您任何TI知识产权的任何其他明示或默示的许可，也未授予您 TI 或第三方的任何技术或知识产权的许可，该等产权包括但不限于任何专利权、版权、屏蔽作品权或与使用TI产品或服务的任何整合、机器制作、流程相关的其他知识产权。涉及或参考了第三方产品或服务的信息不构成使用此类产品或服务的许可或与其相关的保证或认可。使用 TI 资源可能需要您向第三方获得对该等第三方专利或其他知识产权的许可。

TI 资源系“按原样”提供。TI 兹免除对 TI 资源及其使用作出所有其他明确或默认的保证或陈述，包括但不限于对准确性或完整性、产权保证、无屡发故障保证，以及适销性、适合特定用途和不侵犯任何第三方知识产权的任何默认保证。

TI 不负责任何申索，包括但不限于因组合产品所致或与之有关的申索，也不为您辩护或赔偿，即使该等产品组合已列于 TI 资源或其他地方。对因 TI 资源或其使用引起或与之有关的任何实际的、直接的、特殊的、附带的、间接的、惩罚性的、偶发的、从属或惩戒性损害赔偿，不管 TI 是否获悉可能会产生上述损害赔偿，TI 概不负责。

您同意向 TI 及其代表全额赔偿因您不遵守本通知条款和条件而引起的任何损害、费用、损失和/或责任。

本通知适用于 TI 资源。另有其他条款适用于某些类型的材料、TI 产品和服务的使用和采购。这些条款包括但不限于适用于 TI 的半导体产品 (http://www.ti.com/sc/docs/stdterms.htm)、评估模块和样品 (http://www.ti.com/sc/docs/sampterms.htm) 的标准条款。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122
Copyright © 2017 德州仪器半导体技术（上海）有限公司