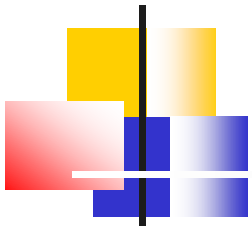
GRAPHICS & MEDIA LAB
VIDEO GROUP

Программирование обработки видео на процессорах NXP TriMedia

Марат Арсаев

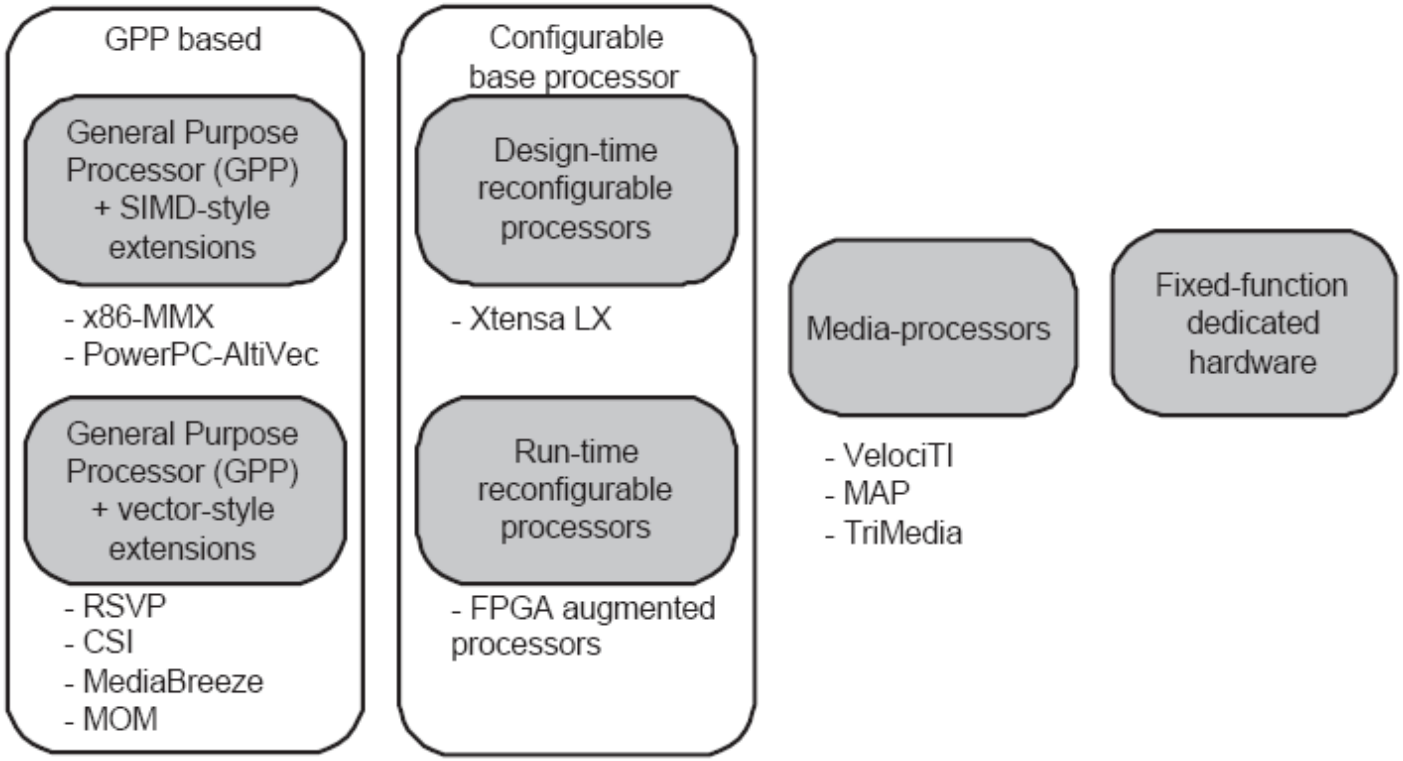
Video Group
CS MSU Graphics & Media Lab



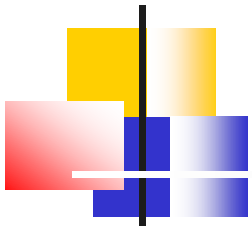
Содержание доклада

- **Введение**
- Архитектура систем PNX1x00
- Оптимизация обработки видео
- Пример фильтра
- Выводы

Введение

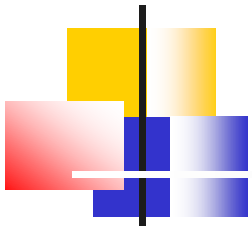


Процессоры для обработки видео и аудио



Введение

- TriMedia — чипы для встроенных систем
- Идеально подходят для low-power систем
- Гибок и легко программируем
- Расширенные встроенные средства
- Широко используется во всем мире

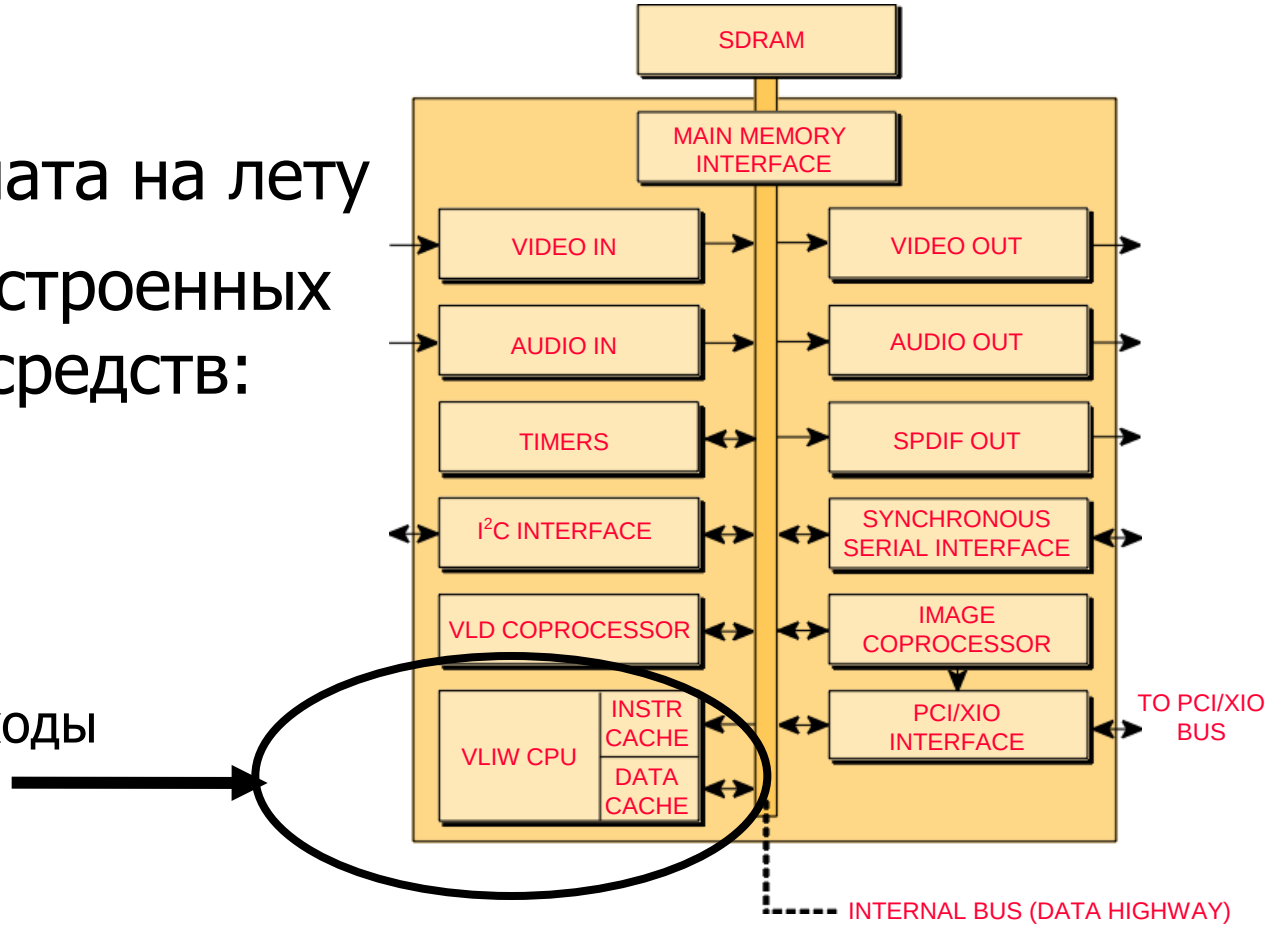


Содержание доклада

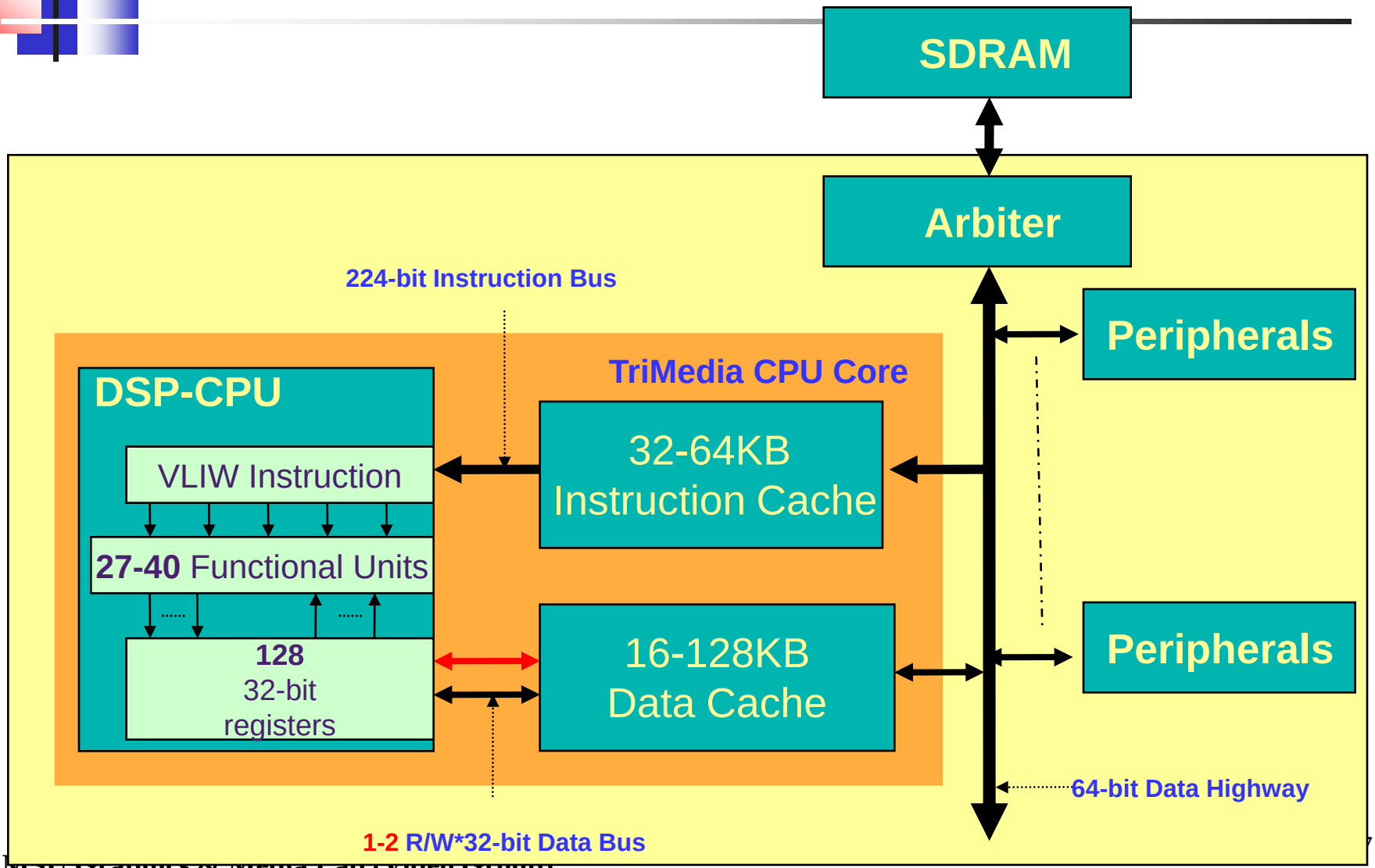
- Введение
- **Архитектура систем PNX1x00**
- Оптимизация обработки видео
- Пример фильтра
- Выводы

Архитектура систем PNX1x00

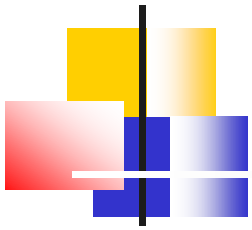
- Изменение формата на лету
- Использование встроенных вспомогательных средств:
  - скейлер \ FRC
  - сеть
  - видео декодеры
  - нестандартные выходы



Архитектура чипов TM



Архитектура CPU



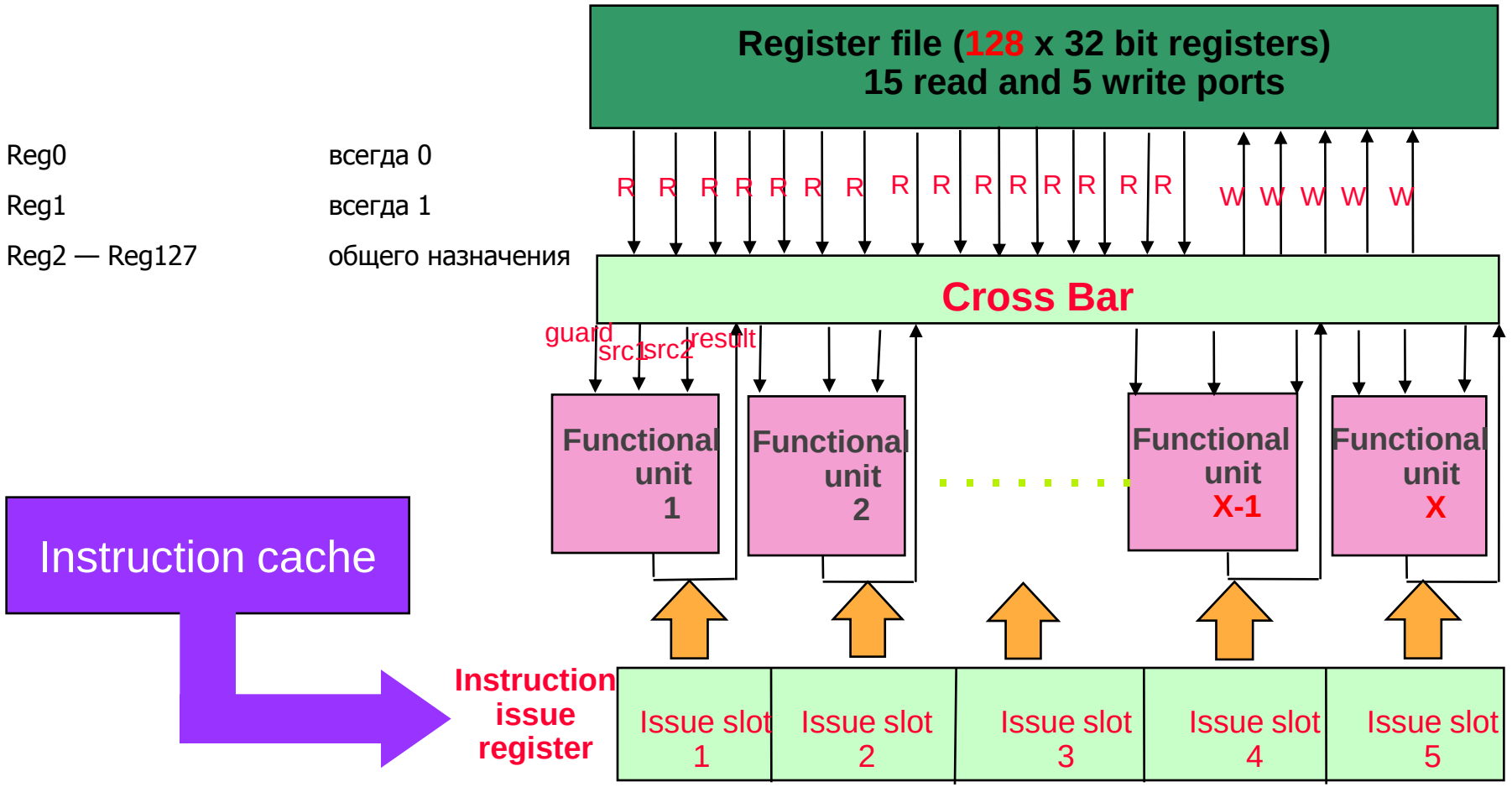
Архитектура VLIW процессра:

- 5 32-битных слотов операций
- каждая операция обращается к необходимым ей ФЭ
- ФЭ обращаются к массиву регистров

Архитектура CPU

GRAPHICS & MEDIA LAB
VIDEO GROUP

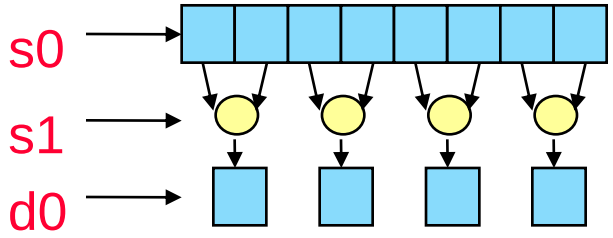
Reg0	всегда 0
Reg1	всегда 1
Reg2 — Reg127	общего назначения

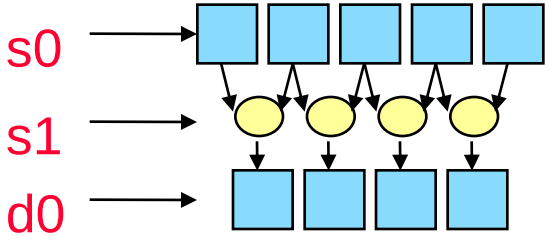


Архитектура CPU

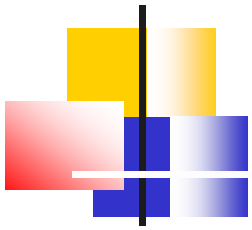
От версии к версии меняются:

- набор, количество и параметры ФЭ
- набор встроенных препроцессоров:
  - скейлер
  - интерполятор
- поддерживаемые декодеры
  - H264
  - MPEG2, MPEG4
  - аудио кодеки





Поддерживаемые операции



GRAPHICS & MEDIA LAB
VIDEO GROUP

- Инструкция – константа (не нужно обращаться в кэш)
- Супероперации (super_ld32, super_quadumedian, и т.д.)
- Целочисленная арифметика :
- +\- , логические операции, сравнения, abs, и т.д.
- Pack, merge, и т.д.
- Сдвиги:
- <<, >>, rol, и т.д.
- Специфические операции сборки векторов (abcd % efgh > cdef)

Поддерживаемые операции

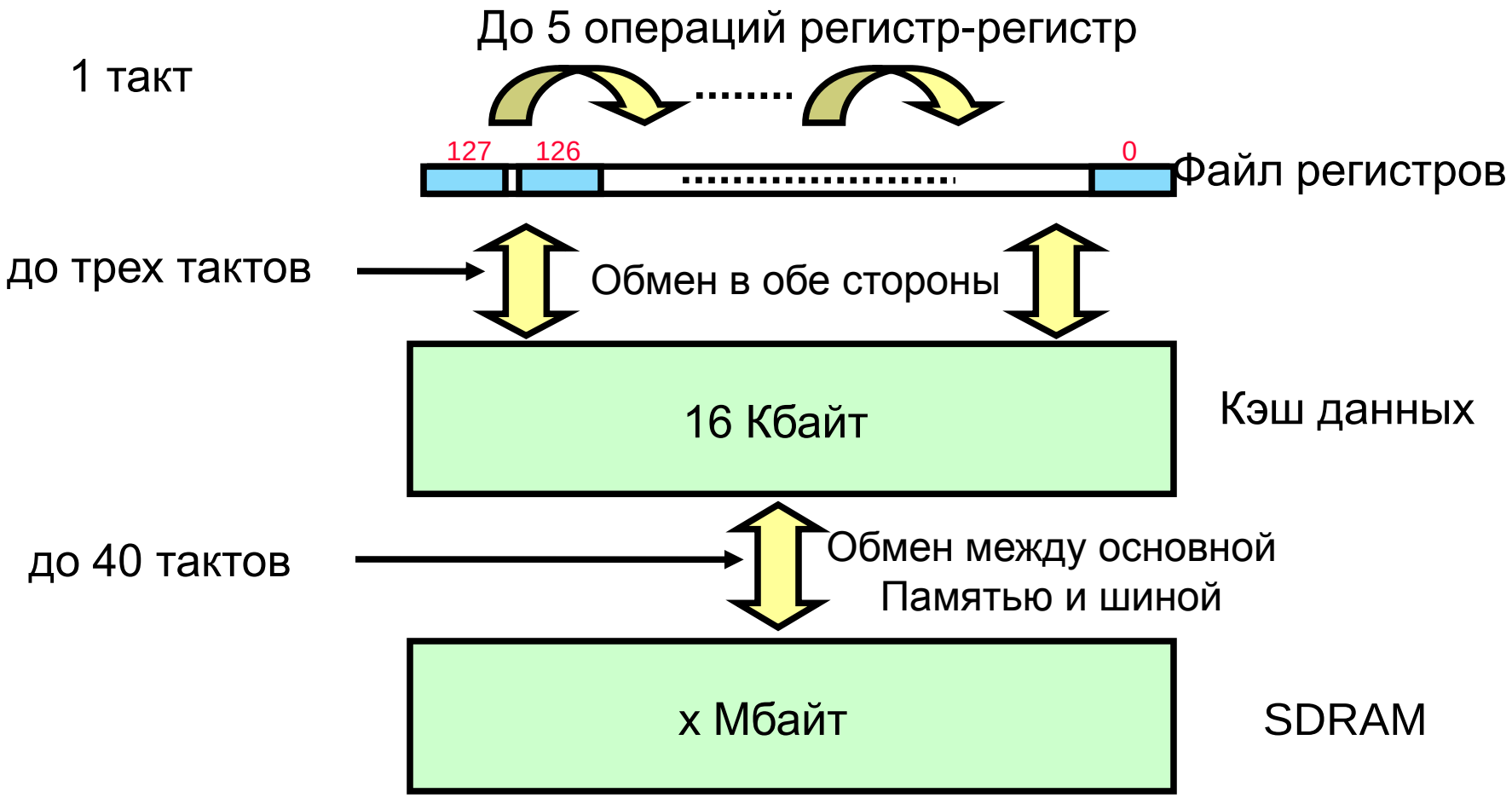
- DSP ALU:
  - Clipping operations
    - SIMD операции
      - Quadadd( , ) =>

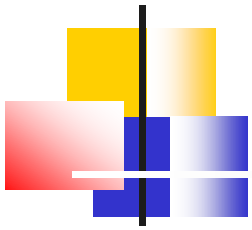
    - Motion Estimation операции
  - DSP MUL:
    - SIMD умножения над 8- и 16-битными операндами
      - FIR инструкции над 8- и 16-битными данными
        - Ifir8ii( ,) =>
          a*e+b*f+c*g+d*h

Таблица задержек ФЭ TM3260

Functional Unit	Quantity	Latency / Delay	Recovery	Issue Slots				
constant	5	1	1	1	2	3	4	5
integer ALU	5	1	1	1	2	3	4	5
load/store	2	3	1				4	5
DSP ALU	3	2	1	1		3		5
DSP MUL	2	3	1		2	3		
shifter	5	1	1	1	2	3	4	5
branch	3	3	1		2	3	4	
int/float mul	2	3	1		2	3		
float ALU	2	3	1	1			4	
float compare	1	1	1			3		
float sqrt/div	1	17	16		2			

Кэш данных TM3260



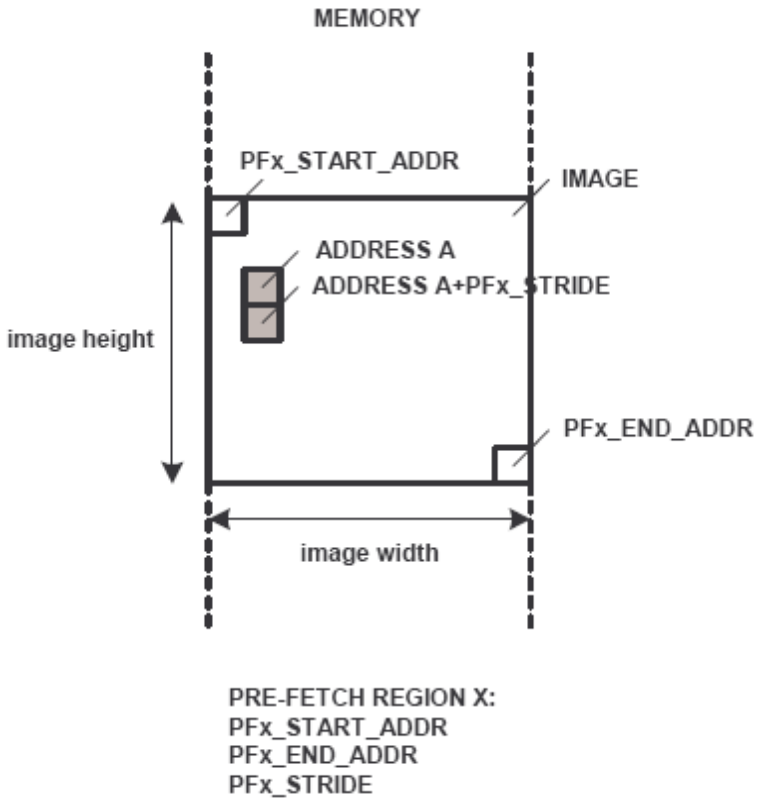


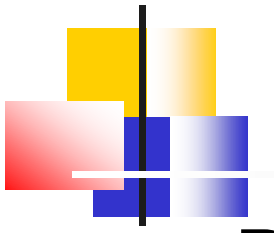
Управление кэшем

- Кэш — одна из самых узких и в то же время гибких мест процессоров TriMedia
- Начиная с процессора 3270 кэш явно программируем
- При неэффективном программировании процессор может простаивать до 90% времени

Управление кэшем

Встроенная реализация пре-кэширования: необходимо верно выставить начало, конец региона, текущий указатель и размер выборки кэша

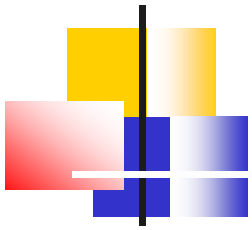




Содержание доклада

- Введение
- Архитектура систем PNX1x00
- **Оптимизация обработки видео**
- Пример фильтра
- Выводы

Стандартные оптимизации



GRAPHICS & MEDIA LAB
VIDEO GROUP

- Разворачивание циклов, использование look-up tables
- Использование встроенных и SIMD операций
- Эффективные выборки из кэша

Пример оптимизаций

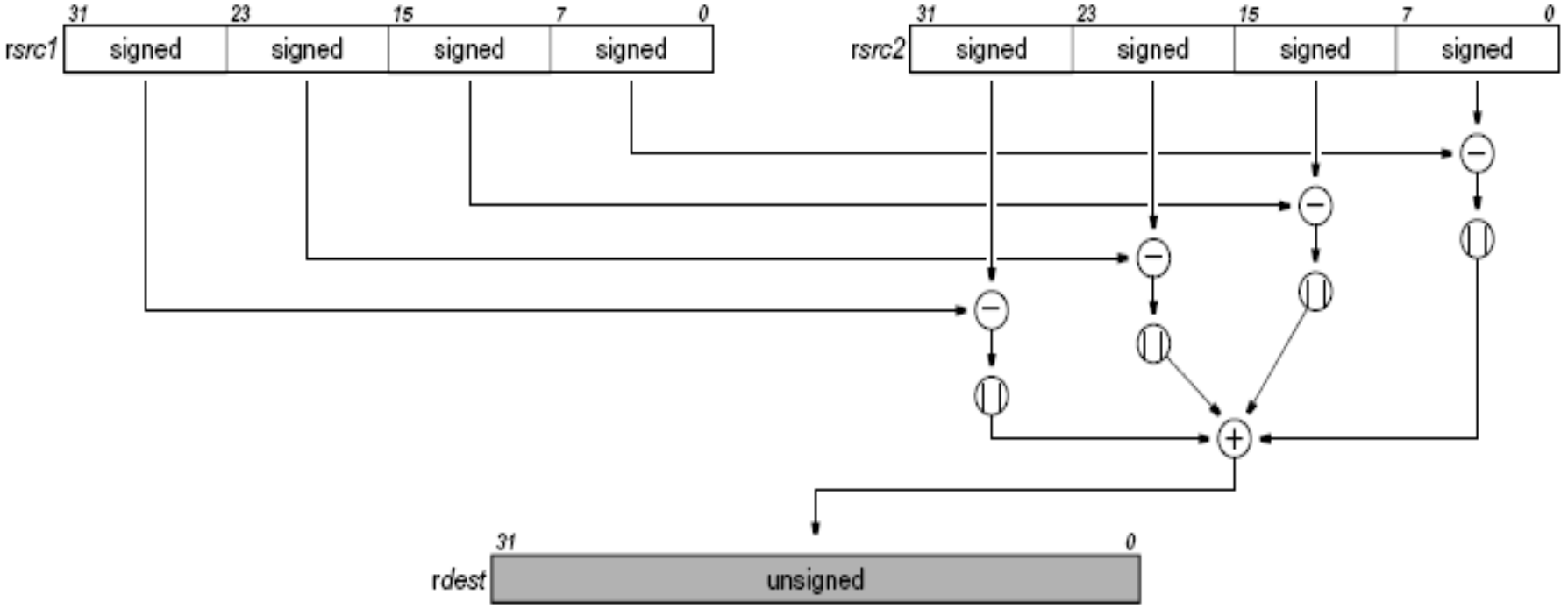
Развертка циклов позволяет компилятору автоматически распараллелить код

```
unsigned char A[16][16];
unsigned char B[16][16];
        .
        .
        .
for (row = 0; row < 16; row += 1)
{
    for (col = 0; col < 16; col += 1)
        cost += abs(A[row][col] - B[row][col]);
}
```

```
for (row = 0; row < 16; row += 1)
{
    for (col = 0; col < 16; col += 4)
    {
        cost0 = abs(A[row][col+0] - B[row][col+0]);
        cost1 = abs(A[row][col+1] - B[row][col+1]);
        cost2 = abs(A[row][col+2] - B[row][col+2]);
        cost3 = abs(A[row][col+3] - B[row][col+3]);

        cost += cost0 + cost1 + cost2 + cost3;
    }
}
```

Пример оптимизаций

Используем встроенные операции: ume8uu

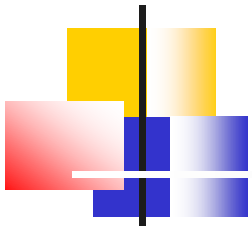


Пример оптимизаций

Новый код:

```
unsigned int *IA = (unsigned int *) A;
unsigned int *IB = (unsigned int *) B;

for (row = 0; row < 16; row += 1)
{
    int rowoffset = row * 4;

    for (col4 = 0; col4 < 4; col4 += 1)
        cost += UME8UU(IA[rowoffset + col4], IB[rowoffset + col4]);
}
```

Пример оптимизаций



GRAPHICS & MEDIA LAB
VIDEO GROUP

Разворчиваем цикл дальше:

```
unsigned int *IA = (unsigned int *) A;
unsigned int *IB = (unsigned int *) B;

for (i = 0; i < 64; i += 8)
{
    cost0 = UME8UU(IA[i+0], IB[i+0]);
    cost1 = UME8UU(IA[i+1], IB[i+1]);
    cost2 = UME8UU(IA[i+2], IB[i+2]);
    cost3 = UME8UU(IA[i+3], IB[i+3]);
    cost4 = UME8UU(IA[i+4], IB[i+4]);
    cost5 = UME8UU(IA[i+5], IB[i+5]);
    cost6 = UME8UU(IA[i+6], IB[i+6]);
    cost7 = UME8UU(IA[i+7], IB[i+7]);

    cost += cost0 + cost1 + cost2 +
            cost3 + cost4 + cost5 +
            cost6 + cost7;
}
```

Пример оптимизаций

Итого операций:

- Было:

16*16*(разность + abs + суммирование) = 768 операций

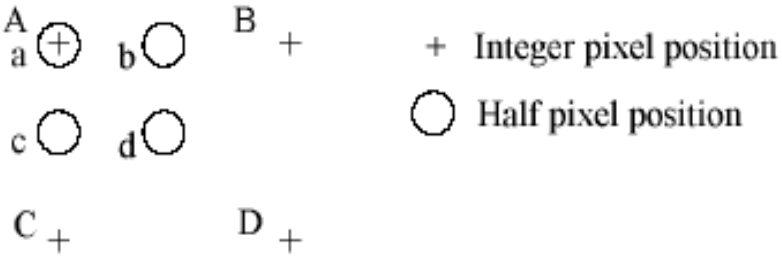
- Стало:

- 8*(8 ME операций + 8 суммирований) = 128 операций

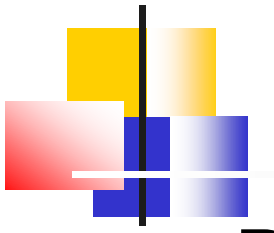
Учитывая развертку циклов, для последней реализации будет сгенерирован весьма эффективный код

Пример оптимизаций

Возможно уточнение при помощи полупиксельной интерполяции, используя funshift, mergeodd, bilinear — благодоря этому операции ведутся над регистрами, без обращения в кэш



a = A,
b = (A + B + 1 - rounding_control) / 2
c = (A + C + 1 - rounding_control) / 2,
d = (A + B + C + D + 2 - rounding_control) / 4

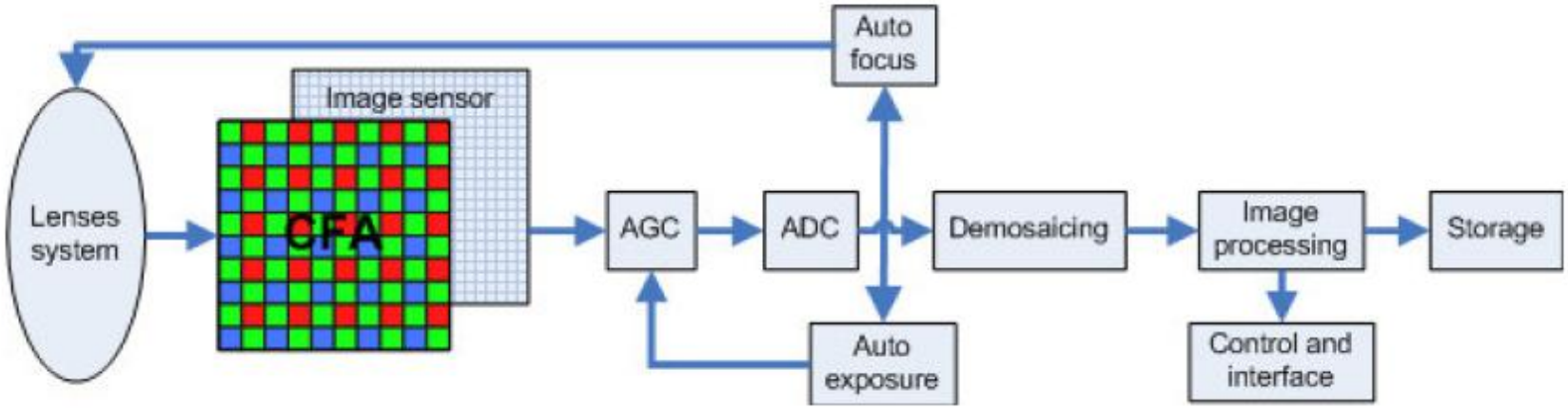


Содержание доклада

- Введение
- Архитектура систем PNX1x00
- Оптимизация обработки видео
- **Пример фильтра**
- Выводы

Bilateral denoising on TM3260

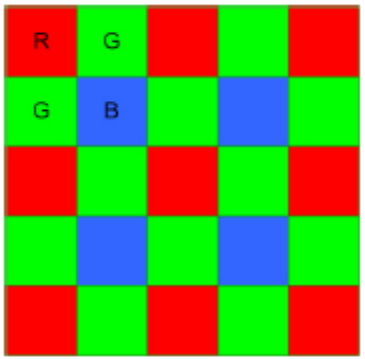
Рассмотрим цифровую камеру и попробуем запрограммировать Bilateral denoising на встроенном чипе TM3260



Bayer Bilateral Denoising on TriMedia3270
H.Phelippeau, M.Akil, Universite Paris-Est, Labinfo, Noisy-le-Grand Cedex France, 2007

Bilateral denoising on TM3260

Будем работать с Байесовской матрицей цветов

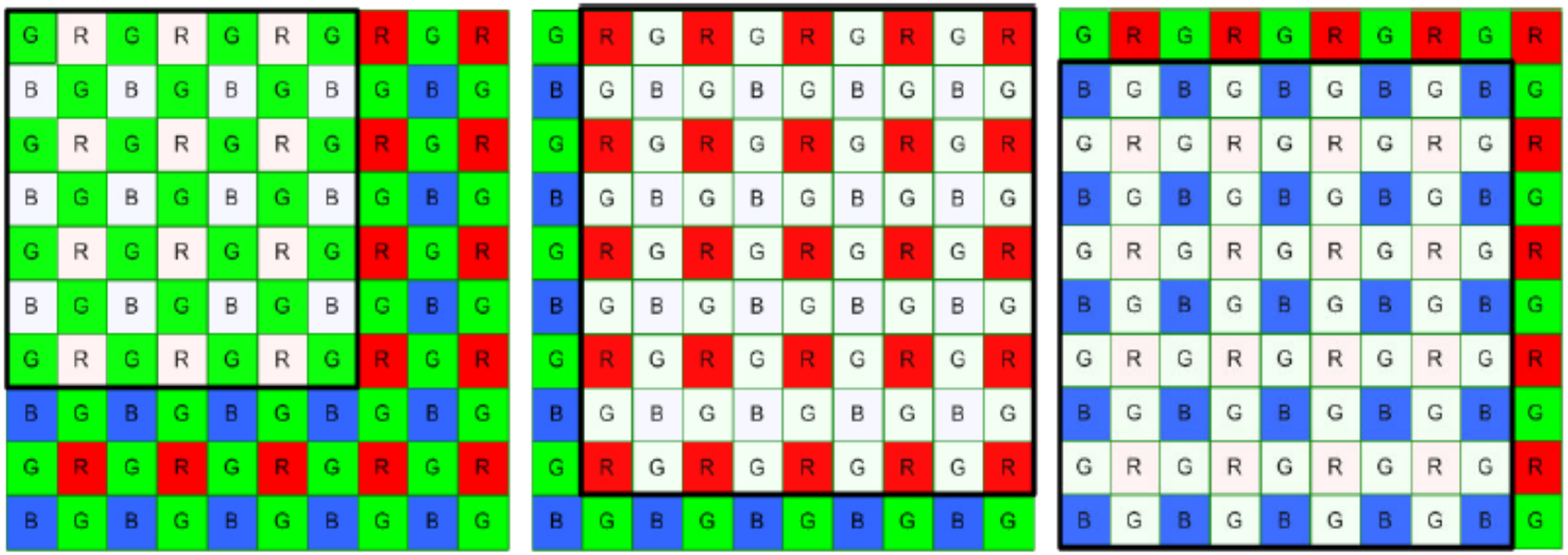


$$v(x) = \frac{1}{C(x)} \sum_{\beta} \exp \frac{-|x-y|}{\rho^2} \exp \frac{-|u(x)-u(y)|}{h^2} u(y)$$

x — координата текущего пикселя
y — координата обрабатываемого пикселя
u(x) — соответствующее значение пикселя
C(x) — нормализующая константа
β — окно фильтра
h, ρ — Гауссовское распределение

Bilateral denoising on TM3260

Окно фильтра для зеленого 7x7, для красного и синего - 9x9



Прямая реализация

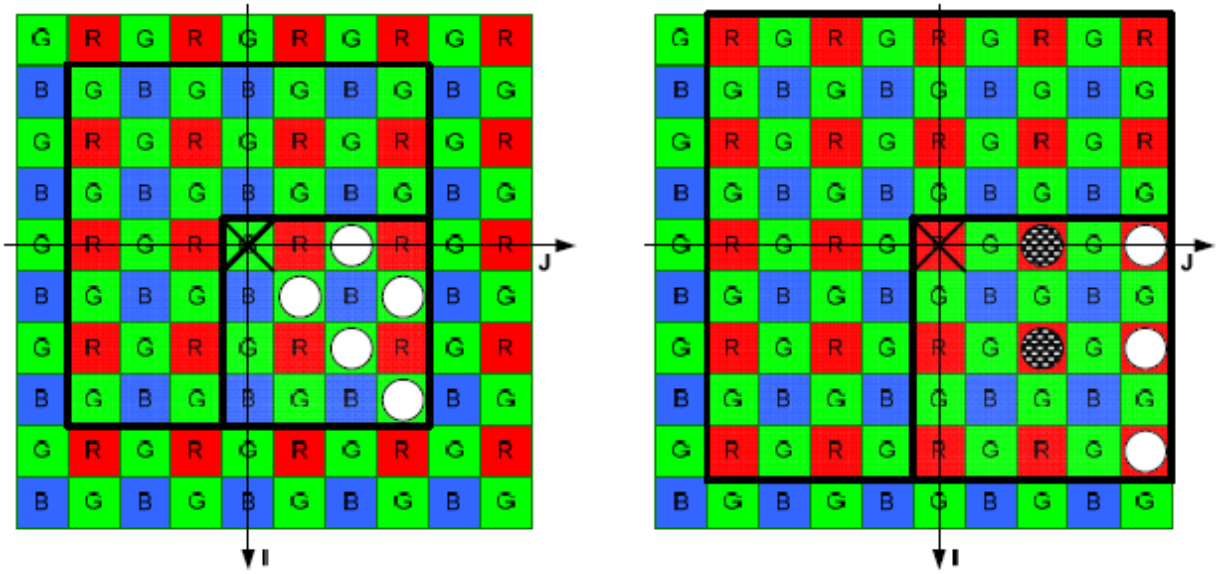
```
count_width=1
count_heigh=1

while count_width<image_width
        while count_heigh<image_heigh
                if pixel_color==red
                     new_pixel_value=red_bilateral_filtering
                else if pixel_color==green
                   new_pixel=green_ bilateral_filtering
                 else
                    new_pixel=blue_ bilateral_filtering
                 end if
          end while
end while
```

Operations	Exponentials	Floating point operations			Absolute values	Square roots
		Multiplications	Additions	Divisions		
location weights	25	75	25	25	0	25
intensity weights	25	25	25	25	25	0
mean and normalization	0	50	24	1	0	0
total	**50**	**150**	**74**	**51**	**25**	**25**

Так не программируют даже на PC!

Look-up tables

Используя симметричность заводим таблицу на 8*255 float значений



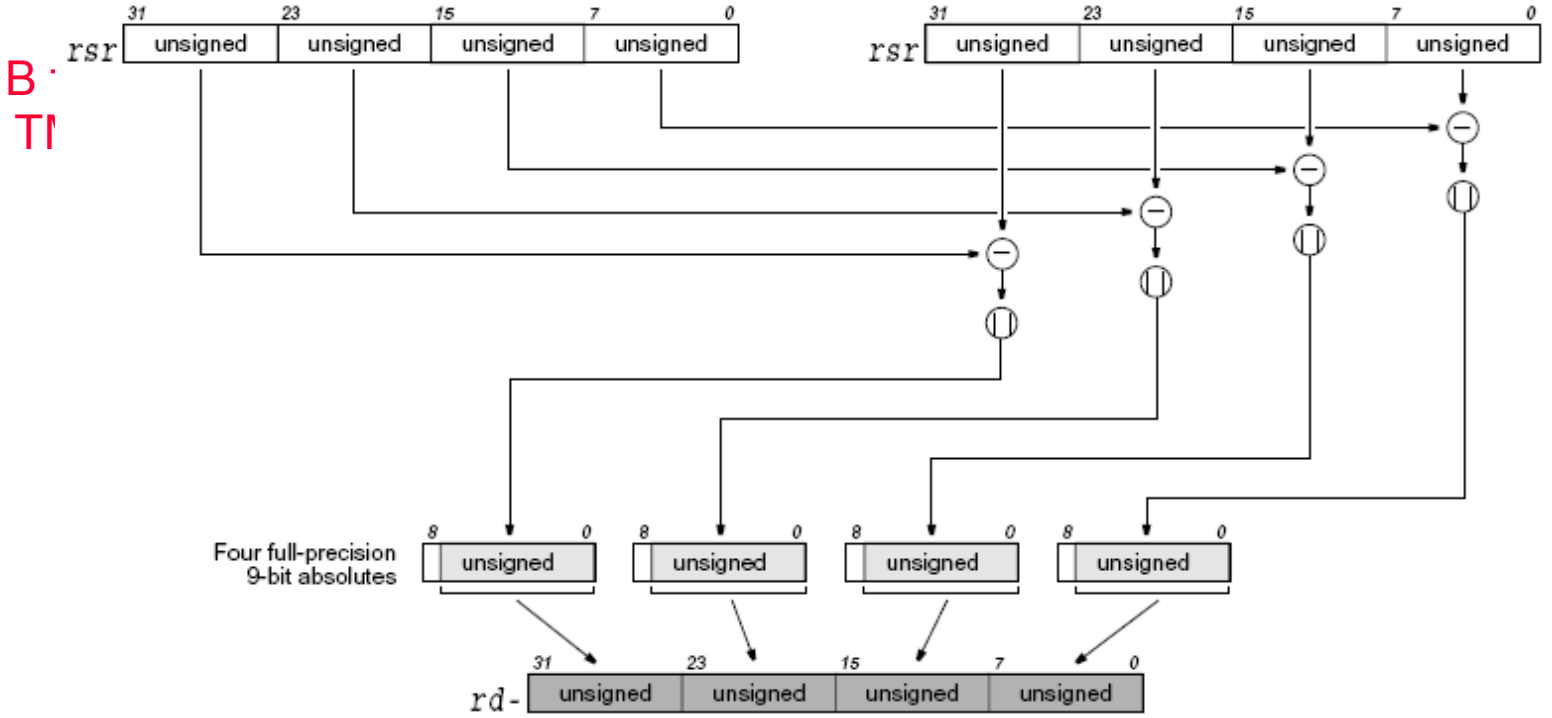
Operations	Exponentials	Floating point operations			Absolute values	Fetches
		Multiplications	Additions	Divisions		
location weights	0	0	0	0	0	0
intensity weights	0	0	0	0	25	25
mean and normalization	0	25	24	1	0	0
total	**0**	**25**	**24**	**1**	**25**	**25**
total-naïve version	*50*	*150*	*74*	*51*	*25*	*25*

Loop unrolling

Развернем внутренний цикл в 4 раза — такая простая операция на VLIW процессорах высоко повышает эффективность

```
for (j=0; j<height-1; j++)
        {
        Switch(Pixel_Line_Start):
          Case(Red):
          for (i=0; i<width-1; i+=4)
                    {
                    Img [j*width+i]   = Red_pixel ();
                    Img [j*width+i+1] = GreenR_pixel ();
                    Img [j*width+i+2] = Red_pixel ();
                    Img [j*width+i+3] = GreenR_pixel ();
                    }
          }
          Case(GreenB):
          for (i=0; i<width-1; i+=4)
                    {
                    Img [j*width+i]   = GreenB_pixel ();
                    Img [j*width+i+1] = Blue_pixel ();
                    Img [j*width+i+2] = GreenB_pixel ();
                    Img [j*width+i+3] = Blue_pixel ();
                    }
        }Update(Pixel_Line_Start);
        }
```

Custom operations



Custom operations

Результат – уменьшение загрузки более чем в два раза!

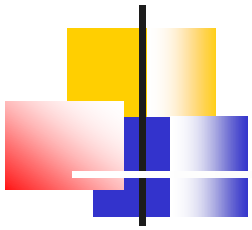
			Function latency	Latency per 4-pixel computation	Total Cycles
Memory data load	Classical load operation		5	20	20
	TriMedia *super_ld32()* function		**5**	**10**	**10**
LUT index computation	Classical C functions	Substraction	1	4	12
		Absolute value	2	8	
	TriMedia *dspquadabssub()* function			**3**	**3**

Сравнение

Простейшая оптимизация дает прирост почти в 100 раз

	Cycles per pixel	Improvement/ naive(%)	Improvement/ Look up table version (%)	Frame/second (640x480) @350MHz
Naïve version	6026.1			0.19
Look up table	445.7	92.6%		2.56
Loop unrolling	219.2	96.36%	50.82%	5.2
Integer	155.5	97.42%	65.12%	7.33
Trimedia custom functions	64.1	98.94%	85.62%	17.78

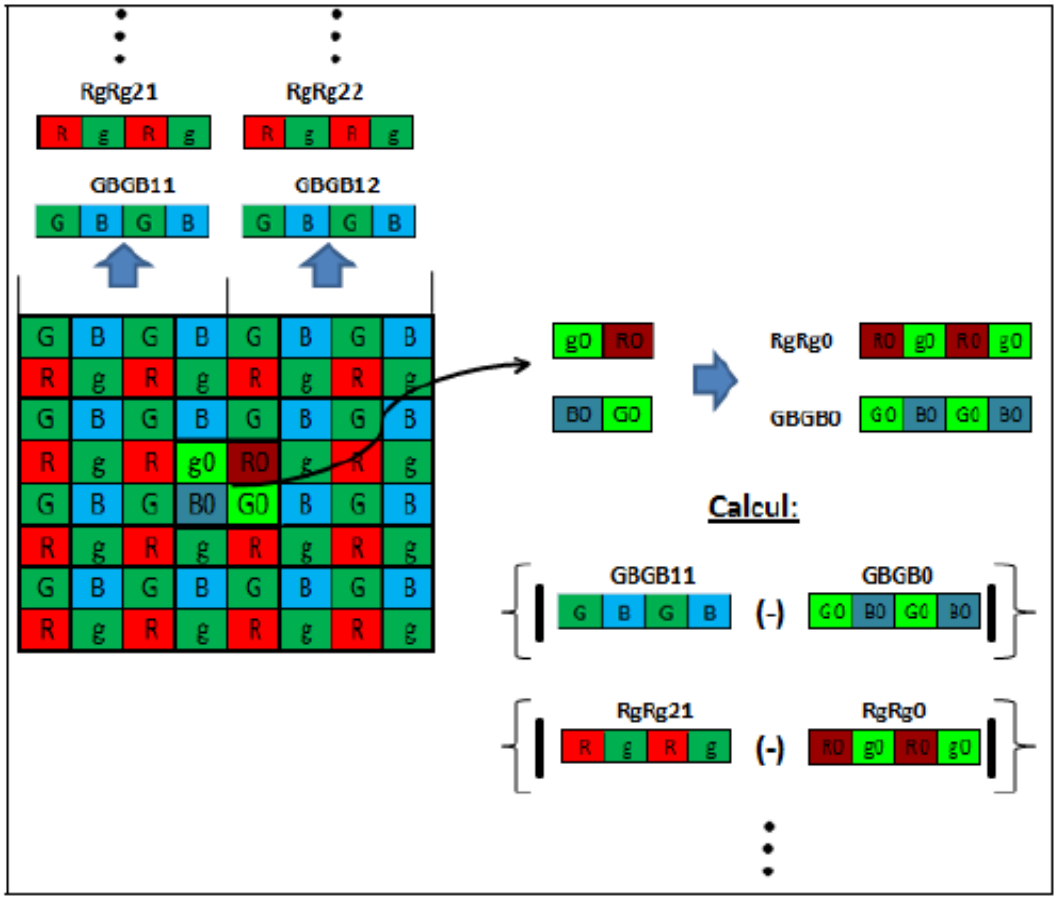
Проблемы дальнейшей оптимизации



- Загрузка подряд идущих 9 байтов требует двух операций super_ld32. Попробуем использовать окно 8x8
- Половина загруженных значений не используются — изменим реализацию для обработки всех цветов за раз

Оптимизация загрузок

Выход — считать весь центральный блок за итерацию в окне 8x8

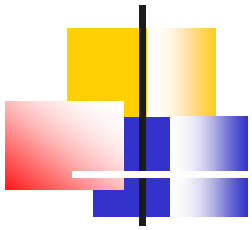


Результаты

	Cycles per pixel	Improvement/ naive(%)	Improvement/ Look up table version (%)	Improvement/ Naïve + TM custom fonctions version (%)	Frame/second (640x480) @350MHz
Naïve version	*6026.1*				*0.19*
Look up table	*445.7*	*92.6%*			*2.56*
Loop unrolling	*219.2*	*96.36%*	*50.82%*		*5.2*
Integer	*155.5*	*97.42%*	*65.12%*		*7.33*
Trimedia custom functions	*64.1*	*98.94%*	*85.62%*		*17.78*
New formulation-8x8 windows	32.3	99.46%	92.75%	49.61%	35.27
New formulation-6x6 windows	24.3	99.6%	94.55%	61.93%	46.9

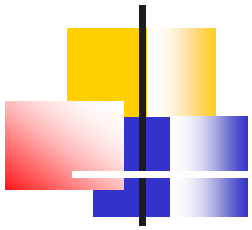
Уменьшение окна до 6x6 позволило авторам выйграть в скорости еще в 1.3 раза.

Даже малейшие оптимизации сильно влияют на скорость работы алгоритма благодаря компилятору



Выводы

- Процессоры TriMedia — мощное и удобное средство программирования обработки видео
- При хорошем уровне разроботки можно добиться высокой производительности
- NXP предлагает встроенный программный пакет ME/MC, кодеров\декодеров, фильтров с которыми можно соревноваться



Вопросы

GRAPHICS & MEDIA LAB
VIDEO GROUP

?

Список литературы

- "**TriMedia CPU Optimization**" Torsten Fink, Manoj Koul and Chuck Peplinski,TM Center of Excellence, August 2006
- "**TriMedia Optimizations by Hand**" Torsten Fink, Manoj Koul and Chuck Peplinski, TM Center of Excellence, August 2006
- "**Bayer Bilateral Denoising on TriMedia3270"** H.Phelippeau, M.Akil, Universite Paris-Est, Labinfo, Noisy-le-Grand Cedex France, 2007
- "**TM3260 Architecture Databook**" Koninklijke Philips Electronics N.V. 2003.
- "**PNX15xx Series Data Book**" Koninklijke Philips Electronics N.V. 2006.
- "**Motion Estimation Performance of the TM3270 Processor**" Jan-Willem van de Waerdt, Jean-Paul van Itegem, Gerrit A.Slavenburg, Philips Semiconductors, San Jose, CA, USA, NVIDIA Corporation Santa Clara, CA, USA, 2005