Университет ИТМО
Факультет ПИиКТ
Системы на Кристалле

Лабораторная работа №2

Нестеров Дали Константинович

Лабушев Тимофей Михайлович

Группа P3402

Цель работы

Получить базовые навыки использования средств высокоуровневого синтеза в процессе проектирования СнК

Задание

1. Спроектировать и описать функциональность аппаратного ускорителя для алгоритма из лабораторной работы №1 на языках C/C++, пригодную для синтеза в аппаратный блок.

2. Провести синтез аппаратного ускорителя.

3. Разработать тестовое окружение для проверки функциональности синтезированного аппаратного ускорителя.

4. Оценить следующие характеристики:

    4.1. Время выполнения алгоритма при частоте тактового сигнала в 100 МГц.

    4.2. Число занимаемых ресурсов ПЛИС (XC7A100T-1CSG324C).

    4.3. Время и занимаемые ресурсы ПЛИС с использованием следующих оптимизаций: раскрутка циклов, конвейеризация циклов.

Выполнение задания

Исходный код алгоритма

```
void fir_filter(short x[FIR_MAX_IN], char p, short y[FIR_MAX_OUT])
{
    if (p < 3 || p > 10)
        return;

    char y_count = 10 - p + 1;

    LoopOverOutputs:
    for (char i = 0; i < y_count; ++i)
    {
        int acc = 0; // Use a 32-bit accumulator to prevent overflow

        SumAccLoop:
        for (char j = 0; j < p; ++j)
        {
               acc += x[i + j];
        }

        y[i] = acc / p;
    }
}
```

Определение характеристик

Время выполнения алгоритма при частоте тактового сигнала в 100 МГц:

P	Время выполнения, нс
3	2490
4	2320
5	2110
6	1860
7	1570
8	1240
9	870
10	460

Число занимаемых ресурсов:

331 FF, 439 LUT (<1%)

Оптимизация алгоритма

Полная развертка циклов с использованием директивы `#pragma HLS unroll` невозможна, так как число итераций переменно и неделимо на какой-либо целочисленный фактор. При частичной развертке вставляются дополнительные проверки (аналоги `if (x > ...) break;` в высокоуровневом языке), которые замедляют время выполнения и используют дополнительные ресурсы.

Развертка внешнего цикла лишь увеличила максимальную задержку, при развертке внутреннего цикла был получен небольшой прирост.

Конвейеризацию внутреннего цикла можно считать оптимальной по используемым ресурсам реализацией: при задержке в 80% от исходной объем используемых ресурсов практически не изменился.

Для конвейеризации внешнего цикла пришлось развернуть внутренний цикл вручную, при этом система синтеза сообщает о том, что ограниченное число входных портов x не позволяет полностью сократить *Initiation Interval* (число тактов между итерациями). Оно составляет 5 (achieved) при возможном 1 (target).

В нашем случае число входных элементов ограничено, поэтому возможна их параллельная передача на вход (аналог шины шириной размерность массива * разрядность данных). Это достигается директивой `#pragma HLS array_partition variable=x complete.`

При рассмотрении получившегося плана вычислений видно, что большая часть задержки связана с операцией целочисленного деления. Поскольку делители известны заранее, модуль делителя можно заменить операциями битового сдвига и умножения на заранее вычисленные константы. Таким образом, мы получаем оптимальную по задержке реализацию: 4% от исходной при использовании 313% ресурсов.

Реализация	Максимальная задержка		Использование FF		Использование LUT	
	Значение, тактов	Относительно исходного, %	Значение	Относительно исходного, %	Значение	Относительно исходного, %
Исходная	361	100	331	100	439	100
Развертка внутреннего цикла	345	96	430	130	571	130
Конвейеризация внутреннего цикла	**289**	**80**	**330**	**100**	**458**	**104**
Ручная развертка внутреннего цикла, конвейеризация внешнего	66	18	1309	395	1470	335
Ручная развертка внутреннего цикла, конвейеризация внешнего, одновременная передача входных элементов	36	10	1624	491	1898	432
Ручная развертка внутреннего цикла, конвейеризация внешнего, одновременная передача входных элементов, деление через умножение и сдвиги	**14**	**4**	**841**	**254**	**1570**	**358**

Исходный код оптимальной реализации

```
void fir_filter(short x[FIR_MAX_IN], char p, short y[FIR_MAX_OUT])
{
	#pragma HLS array_partition variable=x complete

	if (p < 3 || p > 10)
		return;

	char y_count = 10 - p + 1;

	LoopOverOutputs:
	for (char i = 0; i < y_count; ++i)
	{
	#pragma HLS pipeline

	int acc = 0; // Use a 32-bit accumulator to prevent overflow

	switch (p)
	{
		case 10: acc += x[i + 9]; // @suppress("No break at end of case")
		case 9: acc += x[i + 8]; // @suppress("No break at end of case")
		case 8: acc += x[i + 7]; // @suppress("No break at end of case")
		case 7: acc += x[i + 6]; // @suppress("No break at end of case")
		case 6: acc += x[i + 5]; // @suppress("No break at end of case")
		case 5: acc += x[i + 4]; // @suppress("No break at end of case")
		case 4: acc += x[i + 3]; // @suppress("No break at end of case")
		case 3:
			acc += x[i + 2];
			acc += x[i + 1];
			acc += x[i + 0];
	}

	int tmp;
	switch (p)
	{
		case 3: y[i] = ((((long long)acc * 0x55555556) >> 32) - (acc >> 31)); break;
		case 4: y[i] = acc >> 2; break;
		case 5: y[i] = ((((long long)acc * 0x66666667) >> 33) - (acc >> 31)); break;
		case 6: y[i] = ((((long long)acc * 0x2AAAAAAB) >> 32) - (acc >> 31)); break;
		case 7:
			tmp = (((long long)acc * (int)0x92492493) >> 32) + acc;
			y[i] = (tmp >> 2) + (tmp >> 31);
			break;
		case 8: y[i] = acc >> 3; break;
		case 9: y[i] = ((((long long)acc * 0x38E38E39) >> 33) - (acc >> 31)); break;
		case 10: y[i] = ((((long long)acc * 0x66666667) >> 34) - (acc >> 31)); break;
	}
	}
}
```

Вывод

В результате выполнения работы была создана аппаратная реализация блока вычисления КИХ-фильтра, созданного в ходе первой лабораторной работы, с использованием средств высокоуровневого синтеза.

В процессе оптимизации было установлено, что использование правильных директив позволяет уменьшить время выполнения без увеличения используемых ресурсов. Тем не менее, достигнуть уменьшения задержки в несколько порядков возможно лишь путем анализа плана выполнения и применения ручных оптимизаций, которые будут отличаться для каждого конкретного алгоритма.

При этом были замечены и ограничения высокоуровневого анализа. При использовании типов `int8_t, int16_t` вместо `char, short` отчет по синтезу содержал некорректную аппроксимацию числа итераций цикла. Более того, в нашем алгоритме число итераций внутреннего цикла напрямую зависит от числа итераций внешнего, но при подсчете максимальной задержки это не учтено. В этом можно убедиться, сравнив пункт Определение характеристик со значениями для исходной реализации в пункте Оптимизация алгоритма: рассчитанная максимальная задержка не достигается ни при каких значениях P.