Университет ИТМО
Факультет ПИиКТ
Системы на Кристалле

Лабораторная работа №3

Нестеров Дали Константинович

Лабушев Тимофей Михайлович

Группа P3402

Цель работы

Получить базовые навыки проектирования, отладки и тестирования гетерогенной СнК, включающей блоки с конфигурируемым и фиксированным алгоритмом функционирования.

Задание

1. Часть 1. Разработка проекта микропроцессорной СнК и реализация расчета значения математической функции.

    1.1. Спроектировать СнК для ПЛИС XC7A100T-1CSG324C отладочной платы Nexys A7 на базе процессорного ядра Microblaze.

    1.2. На базе созданной СнК реализовать алгоритм расчета математической функции согласно варианту задания. Исходные данные для расчетов должны быть представлены в виде массивов данных (задаются непосредственно в коде программы), а результат вычисления должен отображаться на дискретных портах вывода.

    1.3. Разработать тестовое окружение для созданной системы и провести её тестирование в симуляторе.

    1.4. Выполнить профилирование системы по занимаемым ресурсам и производительности при частоте тактового сигнала в 100МГц.

2. Часть 2. Разработка аппаратного ускорителя для вычисления значения математической функции.

    2.1. Разработать аппаратный ускоритель с использованием САПР высокоуровневого синтеза Vivado HLS согласно варианту задания. В процессе разработки необходимо стремиться создать оптимальную реализацию по критериям используемых ресурсов и производительности. Должны быть использованы различные варианты оптимизации циклов и способов хранения данных.

    2.2. Разработать тестовое окружение для созданного ускорителя и проверить его работу в режиме косимуляции.

    2.3. Провести профилирование разработанного ускорителя по занимаемым ресурсам и производительности при частоте тактового сигнала в 100 МГц в Vivado HLS.

    2.4. Создать IP ядро на базе проекта разработанного аппаратного ускорителя.

3. Часть 3. Разработка гетерогенной СнК с блоком аппаратного ускорителя математических операций.

3.1. Интегрировать в созданную в части 1 СнК аппаратный ускоритель, разработанный в части 2 лабораторной работы.

3.2. Выполнить профилирование системы по занимаемым ресурсам и производительности при частоте тактового сигнала в 100МГц.

3.3. Провести тестирование созданной системы в симуляторе с использованием тестового окружения из части 1 лабораторной работы.

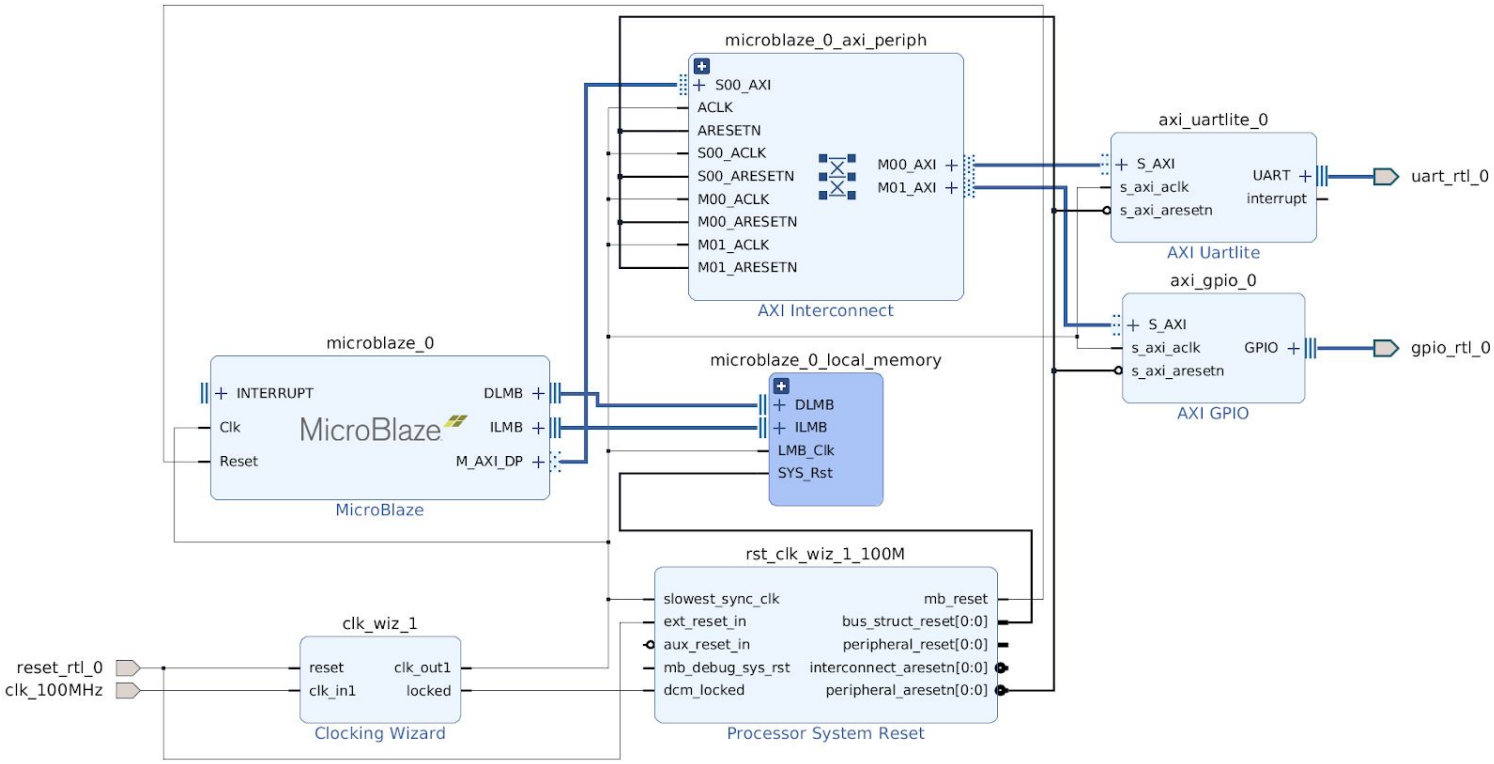
3.4. Выполнить профилирование системы по занимаемым ресурсам и производительности при частоте тактового сигнала в 100МГц.

4. Выполнить анализ полученных результатов и написать вывод по работе.

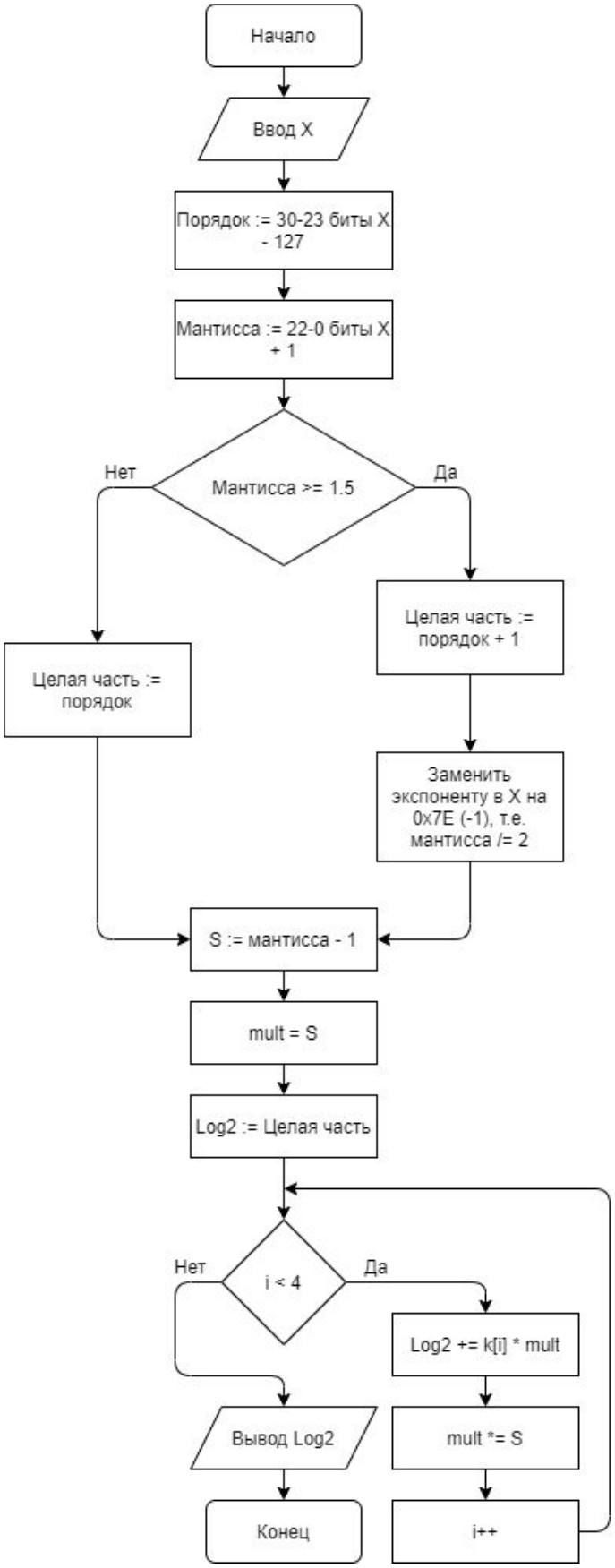
Вариант 1.
Реализовать вычисление логарифма по основанию 2.
Разрядность входных данных и результата: 32 бита.
В расчетах могут быть использованы как числа с плавающей точкой, так и с фиксированной.

Раздел 1. Разработка проекта микропроцессорной СнК и реализация расчета значения математической функции

Структурная схема разработанной СнК



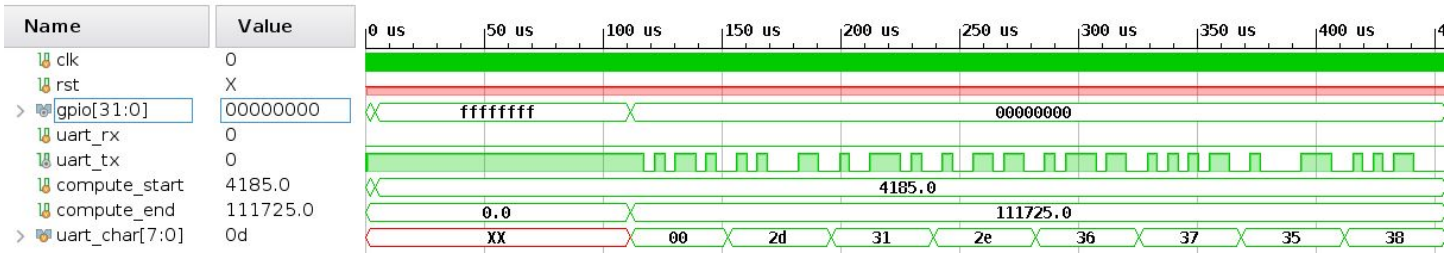
Блок-схема алгоритма



Исходный код программной реализации

```
/* A polynomial approximation of float log2(float).
 *
 * An IEEE 754 single-precision floating-point number x is represented as
 * sign (1 bit) exponent - 127 (8 bits) fraction (23 bits),
 * x = (-1)^sign * 2^(exp-127) * (1+frac)
 *
 * We ignore the sign bit (assume it to be 0) and compute log2(x) as:
 * log2(x) = log2(2^(exp-127) * (1+frac)) = log2(2^(exp-127)) + log2(1+frac) =
 *      = (exp-127) + log2(1+frac)
 *
 * The interval of [1,2) for (1+frac) is reduced to [0.75,1.5) for better approximation [1].
 *
 * Finally, a 4th degree polynomial is chosen to satisfy the assignment constraints
 * (delta < 0.001), with the coefficients a, b, c, d taken from [2].
 *
 * Sources:
 * [1] https://tech.ebayinc.com/engineering/fast-approximate-logarithms-part-i-the-basics/
 * [2] https://tech.ebayinc.com/engineering/fast-approximate-logarithms-part-iii-the-formulas/
 */
float log2_sw(float x)
{
	union { float f; unsigned int i; } fp32;

	fp32.f = x;
	int bexp = (fp32.i & 0x7F800000) >> 23;

	float signif, fexp;
	// Is the highest bit of frac set? Then significand >= 1.5
	if (fp32.i & 0x00400000)
	{
		// significand >= 1.5? Divide by 2 by setting exponent to -1
		fp32.i = (fp32.i & 0x007FFFFF) | 0x3f000000;
		signif = fp32.f;
		fexp = bexp - 126; // compensate for division by 2
	}
	else
	{
		// Set exponent to 0 to get the significand
		fp32.i = (fp32.i & 0x007FFFFF) | 0x3f800000;
		signif = fp32.f;
		fexp = bexp - 127;
	}
	signif -= 1.0;

	const float k[4] = {1.442547, -0.726980, 0.496404, -0.268344};
	float acc = fexp;
	float mult = signif;
	for (int i = 0; i < 4; ++i)
	{
		acc += k[i] * mult;
		mult *= signif;
	}
	return acc;
}
```

Временная диаграмма результатов симуляции с комментариями

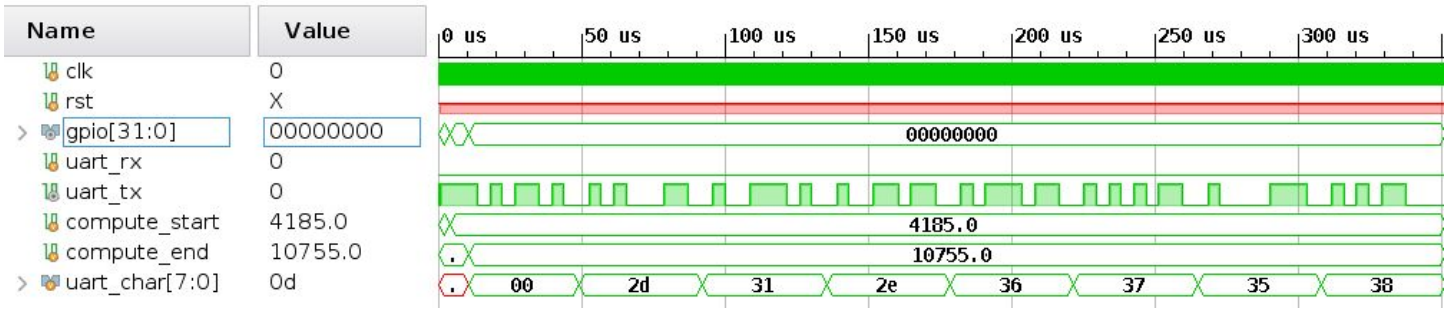


Перед вызовом функции log2 программа выставляет -1 на GPIO, а по окончании вычислений выставляет 0 и передает результат в ASCII представлении по UART.

В данном случае результат равен -1.6758 (“-” = 2d, “1” = 31, “.” = 2e, и т.д.), что соответствует ожидаемому для log2(0.313).

compute_start и compute_end являются переменными тестовой среды и устанавливаются в $realtime по положительному и отрицательному сигналу на GPIO соответственно. Таким образом, можно установить, что время вычисления составило 107540.00 нс.

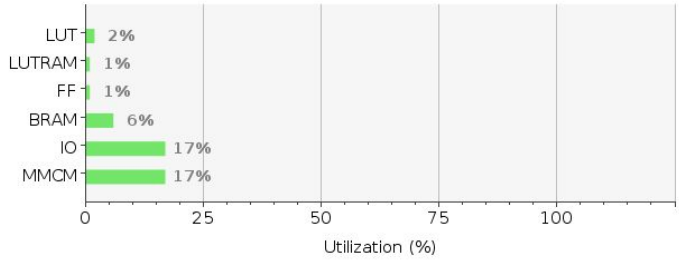
Результат можно существенно улучшить, добавив аппаратный модуль работы с числами с плавающей запятой (FPU) в MicroBlaze. Время вычисления сокращается до 6570.00 нс:



Отчет по занимаемым ресурсам и производительности

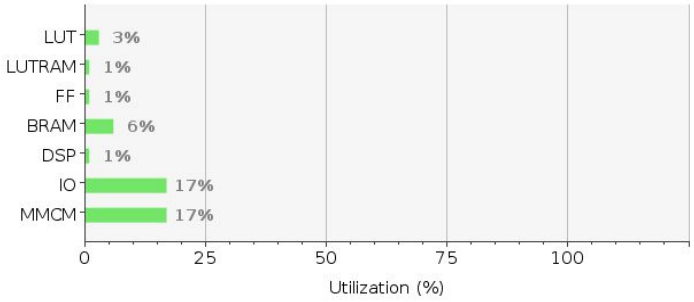
Для версии с программной реализацией FP операций:

Resource	Utilization	Available	Utilization %
LUT	1148	63400	1.81
LUTRAM	116	19000	0.61
FF	1023	126800	0.81
BRAM	8	135	5.93
IO	36	210	17.14
MMCM	1	6	16.67



После добавления аппаратного FPU:

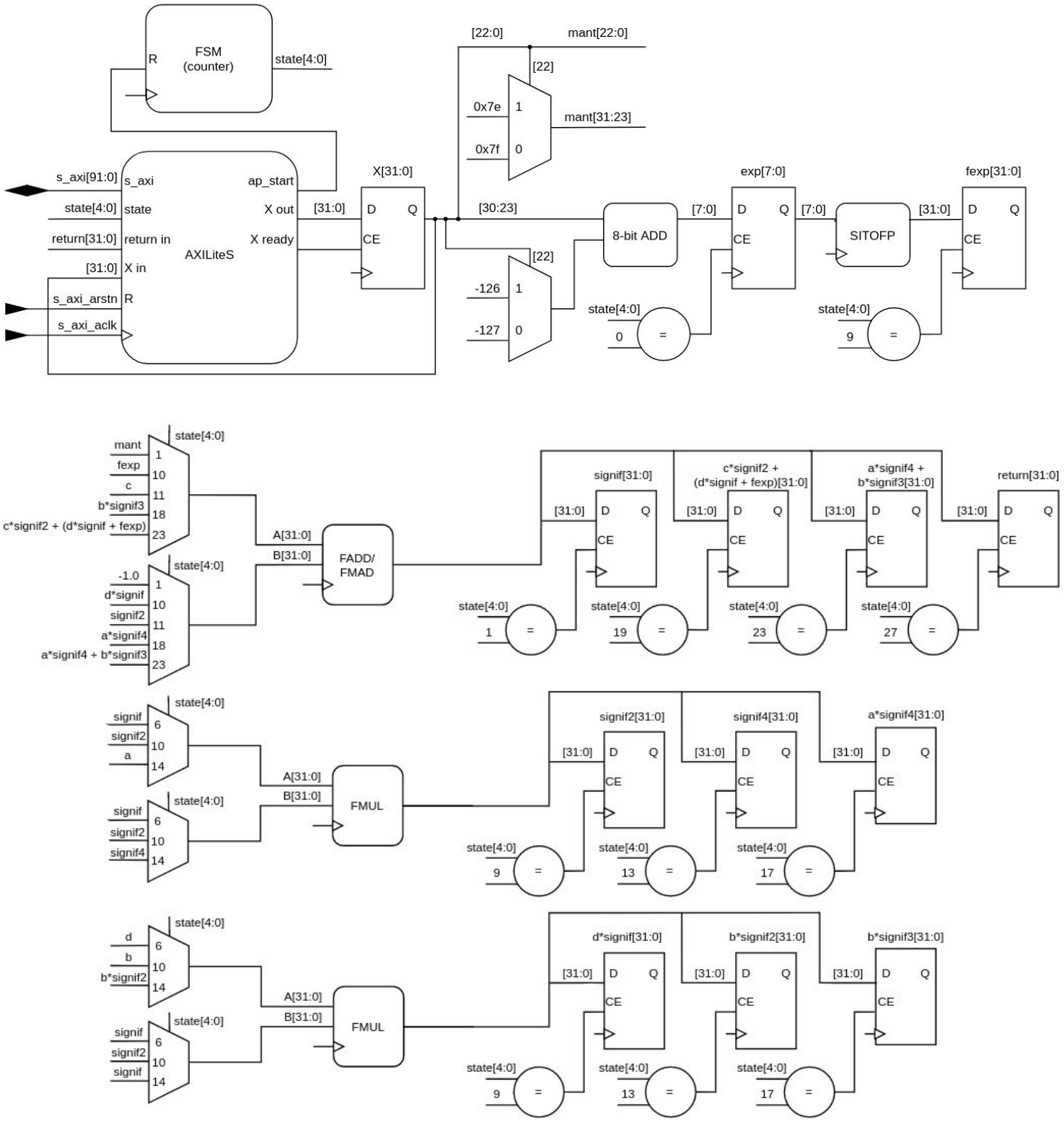
Resource	Utilization	Available	Utilization %
LUT	2177	63400	3.43
LUTRAM	123	19000	0.65
FF	1767	126800	1.39
BRAM	8	135	5.93
DSP	2	240	0.83
IO	36	210	17.14
MMCM	1	6	16.67



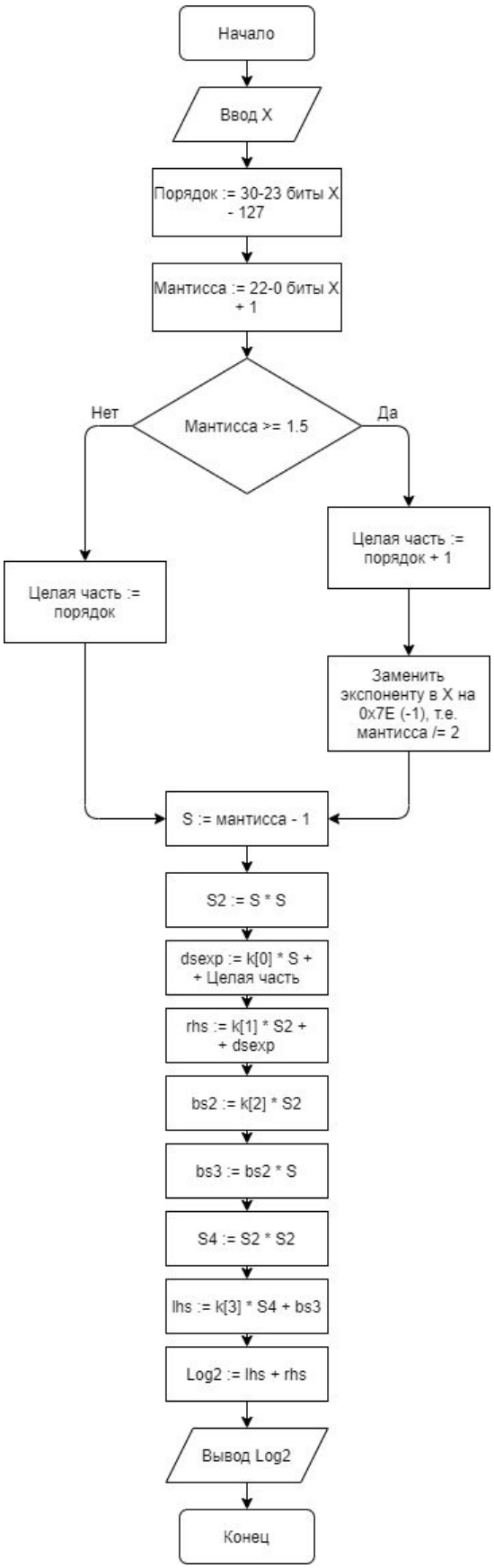
Раздел 2. Разработка аппаратного ускорителя для вычисления значения математической функции

Структурная схема аппаратного ускорителя на уровне RTL

Вычислениями управляет конечный автомат — фактически, счетчик тактов начиная с поступления сигнала ap_start. Текущее состояние указывается на схеме как fsm. Запись в регистры осуществляется по сигналу Clock Enable (CE).


FSM
(counter)
R
state[4:0]
s_axi[91:0]
s_axi
state[4:0]
state
return[31:0]
return in
[31:0]
X in
s_axi_arstn
R
s_axi_aclk
AXILiteS
ap_start
X out
X ready
[31:0]
X[31:0]
D
Q
CE
[22:0]
mant[22:0]
[22]
0x7e
1
mant[31:23]
0x7f
0
[30:23]
-126
-127
8-bit ADD
[7:0]
exp[7:0]
SITOFP
fexp[31:0]
state[4:0]
0
9
=
mant
fexp
c
b*signif3
c*signif2 + (d*signif + fexp)
23
-1.0
d*signif
signif2
a*signif4
a*signif4 + b*signif3
A[31:0]
B[31:0]
FADD/
FMAD
signif[31:0]
c*signif2 +
(d*signif + fexp)[31:0]
a*signif4 +
b*signif3[31:0]
return[31:0]
1
19
23
27
signif
a
FMUL
signif4
signif2[31:0]
signif4[31:0]
a*signif4[31:0]
13
17
d
b
b*signif2
d*signif[31:0]
b*signif2[31:0]
b*signif3[31:0]

Блок-схема алгоритма

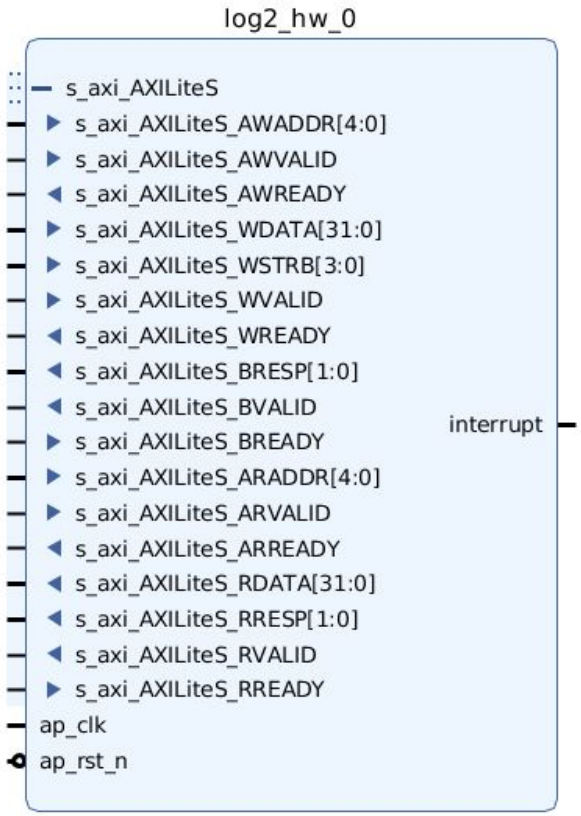


Исходный код аппаратной реализации

```
#include <hls_math.h>

float log2_hw(float x)
{
	#pragma HLS INTERFACE s_axilite port=x
	#pragma HLS INTERFACE s_axilite port=return

	union { float f; unsigned int i; } fp32;

	fp32.f = x;
	// bexp = exp-127 (bits 23-30)
	int bexp = (fp32.i & 0x7F800000) >> 23;

	float signif, fexp;
	if (fp32.i & 0x00400000)
	{
		fp32.i = (fp32.i & 0x007FFFFF) | 0x3f000000;
		signif = fp32.f;
		fexp = bexp - 126;
	}
	else
	{
		fp32.i = (fp32.i & 0x007FFFFF) | 0x3f800000;
		signif = fp32.f;
		fexp = bexp - 127;
	}
	signif -= 1.0;

	const float k[4] = {1.442547, -0.726980, 0.496404, -0.268344};

	float signif2 = signif*signif;

	float dsigniffexp = hls::fmaf(k[0], signif, fexp);
	float rhs = hls::fmaf(k[1], signif2, dsigniffexp);

	float bsignif2 = k[2]*signif2;
	float bsignif3 = bsignif2*signif;
	float signif4 = signif2*signif2;
	float lhs = hls::fmaf(k[3], signif4, bsignif3);

	return lhs + rhs;

	/*
	float acc = fexp;
	float mult = signif;
	for (int i = 0; i < 4; ++i)
	{
		#pragma HLS unroll
		acc += k[i] * mult;
		mult *= signif;
	}
	return acc;
	*/
}
```

Схема разработанного ускорителя

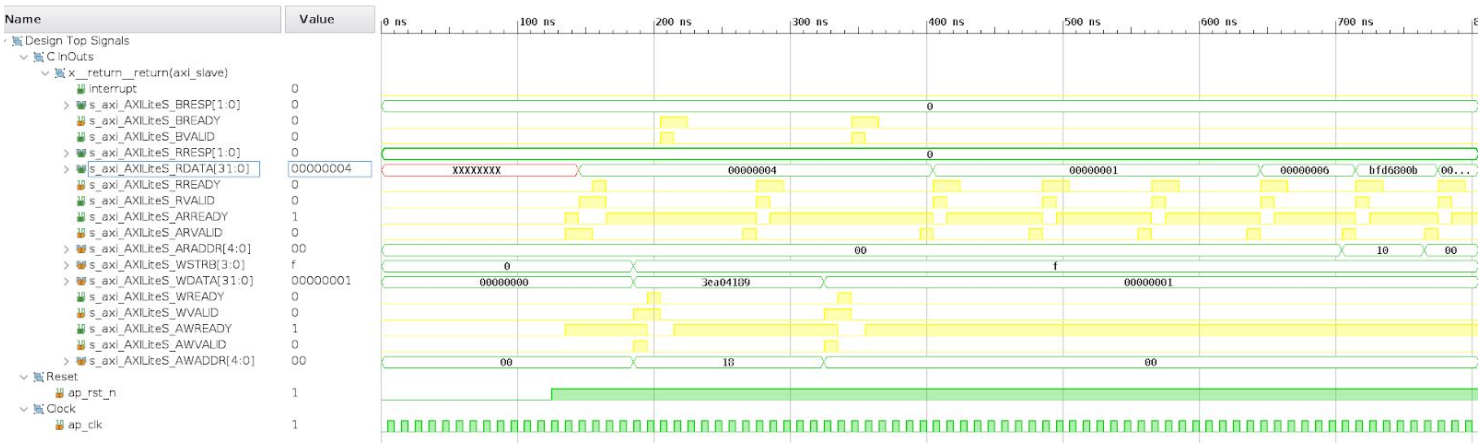
Для взаимодействия с остальными частями СнК используется шина AXI-Lite.



Предназначения отдельных сигналов рассмотрены ниже.

Временная диаграмма результатов косимуляции

В ходе косимуляции вычисляется log2(0.313) = -1.6757825613, то есть log2(`0x3ea04189`) = `0xbfd6800b`:



Взаимодействие с устройством осуществляется по следующим адресам:

- `0x0`: Статус
    - Бит 0 (RW): `ap_start`

    - Бит 1 (R): `ap_done`
    - Бит 2 (R): `ap_idle`
    - Бит 3 (R): `ap_ready`
- `0x10`: 32-х разрядное возвращаемое значение (R)
- `0x18`: 32-х разрядное входное значение (RW)

Порядок взаимодействия:

1. Проверка состояния устройства:
   1.1. ведущий подает сигнал на считывание статуса: `ARADDR = 0x0, ARVALID = 1`;
   1.2. ведомый устанавливает `ARREADY = 0` (состояние по умолчанию — 1) и возвращает значение: `RDATA = 0x4, RVALID = 1`;
   1.3. ведущий устанавливает `RREADY = 1`.

   Считанное значение `0x4` указывает на состояние `ap_idle`.

2. Передача входного значения:
   2.1. ведущий подает X: `AWADDR = 0x18, WDATA = 0x3ea04189, AWVALID = 1, WSTRB = 0xf`; (`WSTRB[n]` устанавливается в 1 для каждого валидного байта в `WDATA`)
   2.2. ведомый устанавливает `WREADY = 1, BRESP = 0` (статус записи `OKAY`), `BVALID = 1`.

3. Ведущий записывает в статус `ap_start` (нулевой бит = 1).

4. Ведущий считывает статус до тех пор, пока не встретит `ap_done` (первый бит = 1).

5. Ведущий считывает выходное значение по адресу `0x18`. Ведомый возвращает `0xbfd6800b`.

Отчет по занимаемым ресурсам и производительности

Синтезирование программной реализации дает следующие результаты:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	171	-
FIFO	-	-	-	-	-
Instance	0	8	936	1026	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	142	-
Register	-	-	215	-	-
Total	0	8	1151	1339	0
Available	270	240	126800	63400	0
Utilization (%)	0	3	~0	2	0

Latency		Interval		Type
min	max	min	max	
43	43	43	43	none

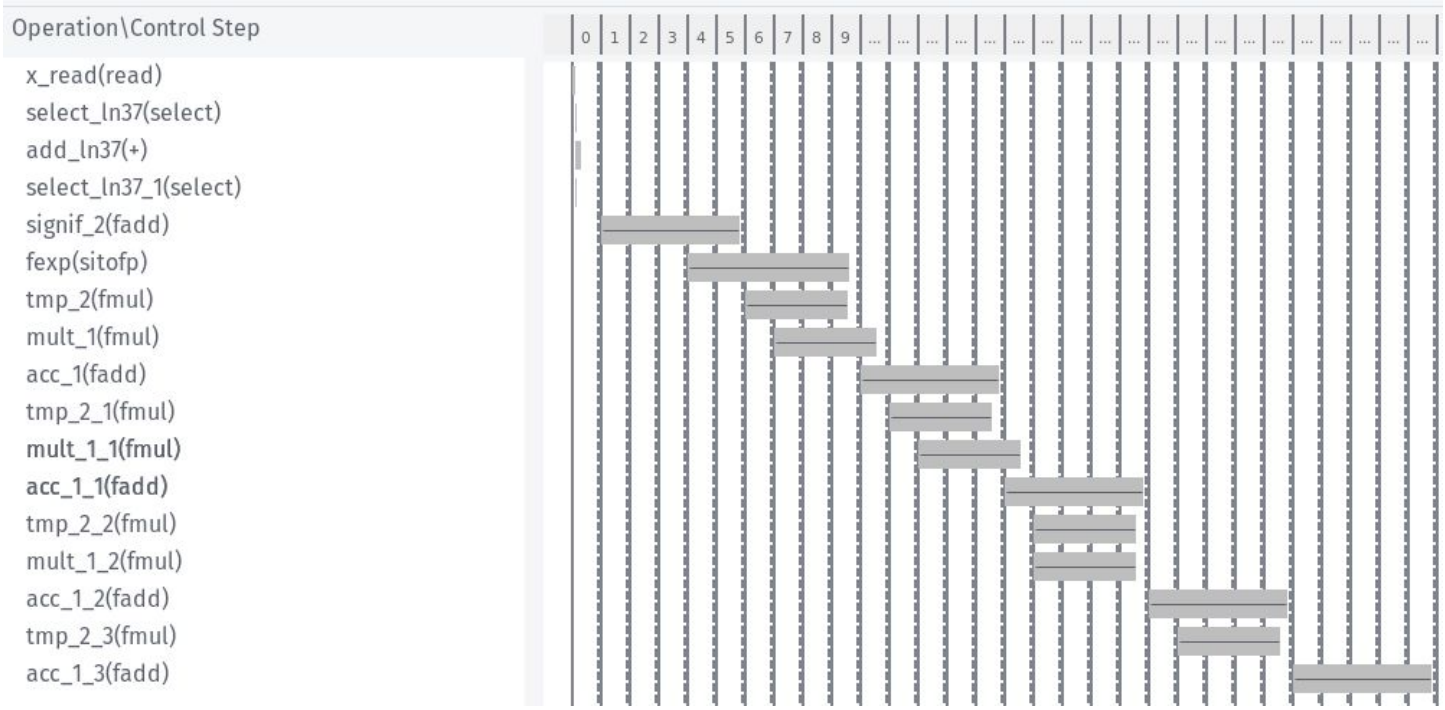
Оптимизируем исполнение цикла, выполнив полную развертку с помощью директивы `#pragma HLS unroll`. Задержка сокращается на 14 тактов (32%), объем используемых ресурсов практически не изменяется:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	27	-
FIFO	-	-	-	-	-
Instance	0	8	936	1026	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	237	-
Register	-	-	255	-	-
Total	0	8	1191	1290	0
Available	270	240	126800	63400	0
Utilization (%)	0	3	~0	2	0

Latency		Interval		Type
min	max	min	max	
29	29	29	29	none

Конвейеризация неприменима, так как вычисление каждой последующей итерации цикла зависит от предыдущей и задействует все функциональные блоки.

Возможна ли дальнейшая оптимизация? Обратимся к инструменту Schedule Viewer. Цикл состоит из последовательностей fmul-fadd, которые могут быть объединены в fused multiply-add:



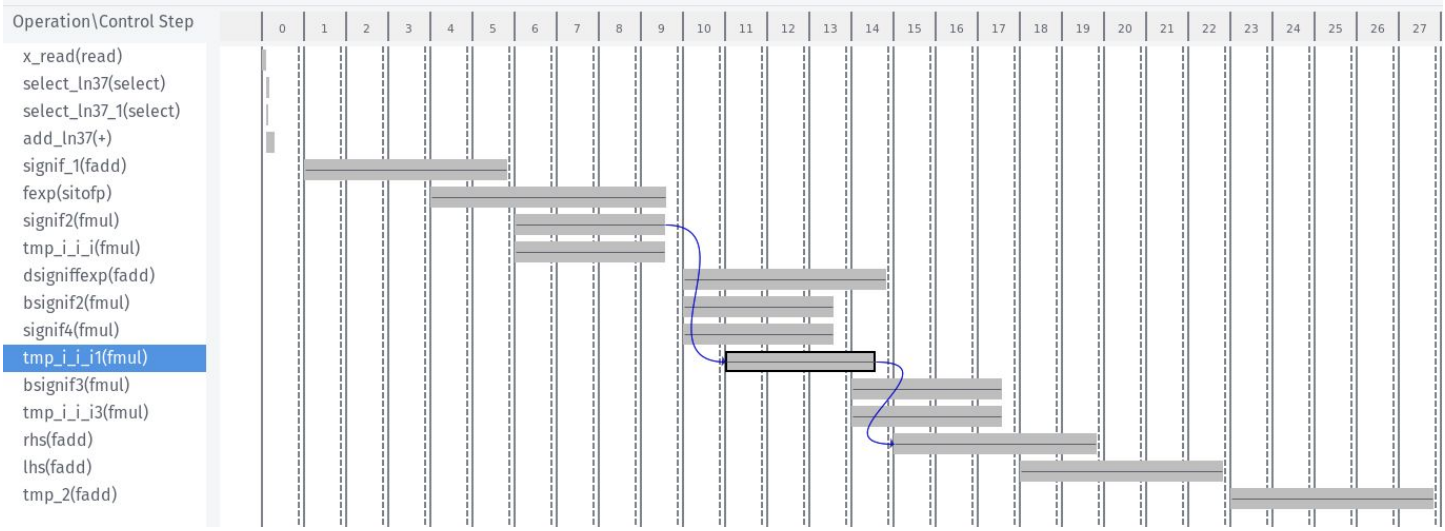
Простое добавление примитива `hls::fmaf` не дает никакого прироста. Необходимо также разделить вычисления на две независимые части, которые складываются для получения результата. Получаем сокращение задержки на 2 такта:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	27	-
FIFO	-	-	-	-	-
Instance	0	8	936	1026	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	277	-
Register	-	-	285	-	-
Total	0	8	1221	1330	0
Available	270	240	126800	63400	0
Utilization (%)	0	3	~0	2	0

Latency		Interval		Type
min	max	min	max	
27	27	27	27	none

В Schedule Viewer можно увидеть, что на одной из итераций производится сразу три операции умножения, хотя инстанций fmul используется две:

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
log2_hw_AXILiteS_s_axi_U	log2_hw_AXILiteS_s_axi	0	0	106	168	0
log2_hw_fadd_32ns_32ns_32_5_full_dsp_1_U1	log2_hw_fadd_32ns_32ns_32_5_full_dsp_1	0	2	205	205	0
log2_hw_fmul_32ns_32ns_32_4_max_dsp_1_U2	log2_hw_fmul_32ns_32ns_32_4_max_dsp_1	0	3	143	140	0
log2_hw_fmul_32ns_32ns_32_4_max_dsp_1_U3	log2_hw_fmul_32ns_32ns_32_4_max_dsp_1	0	3	143	140	0
log2_hw_sitofp_32s_32_6_1_U4	log2_hw_sitofp_32s_32_6_1	0	0	339	373	0
Total	5	0	8	936	1026	0

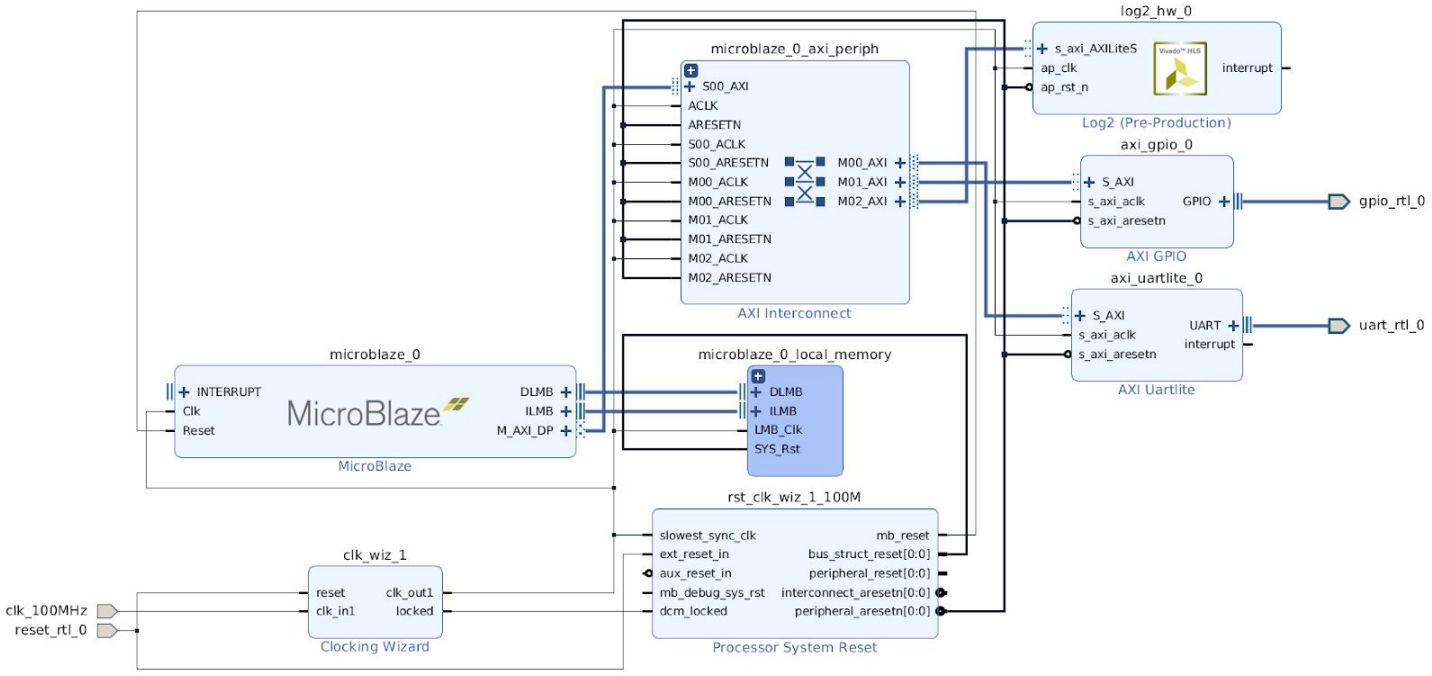


Сравнение реализаций:

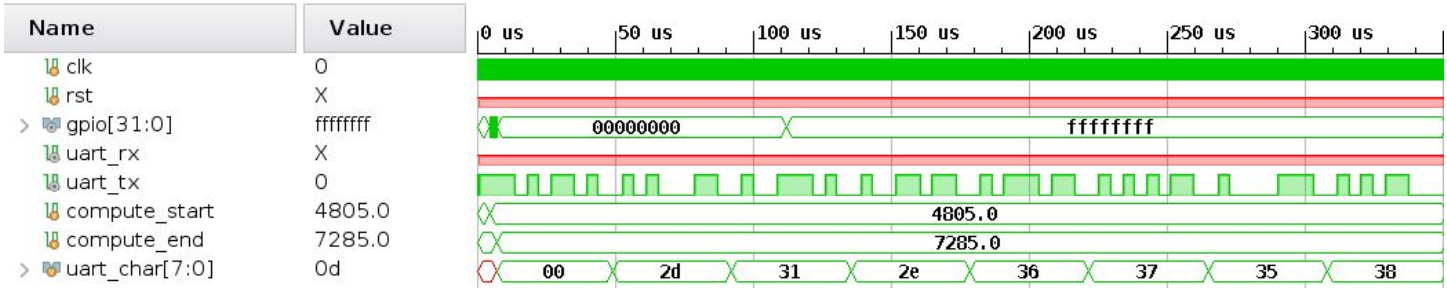
Реализация	Максимальная задержка, тактов	Использование FF	Использование LUT	Использование DSP блоков
Исходная	43	1151	1339	8
Развертка цикла	29	1191	1290	8
Ручная развертка, FMA, разделение на две независимые части	27	1221	1330	8

Раздел 3. Разработка гетерогенной СнК с блоком аппаратного ускорителя математических операций

Структурная схема разработанной гетерогенной СнК

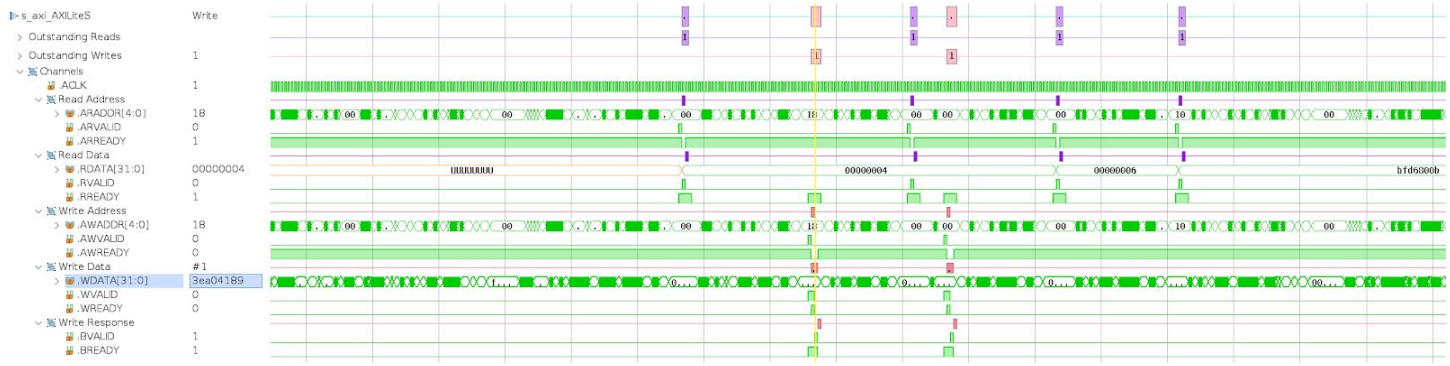


Временная диаграмма результатов симуляции



От вызова функции вычисления до получения результата прошло 2480.00 нс. Сигналы compute_start, compute_end, uart_char описаны в первом разделе работы.

Отдельно стоит рассмотреть временную диаграмму взаимодействия по шине AXI-Lite:



MicroBlaze осуществляет работу с устройством посредством функции `log2_hw`, представленной ниже:

```
float log2_hw(XLog2_hw* instance, float x)
{
        union { float f; unsigned int i; } fp32;
        fp32.f = x;

        while (!XLog2_hw_IsReady(instance));
        XLog2_hw_Set_x(instance, fp32.i);
        XLog2_hw_Start(instance);
        while (!XLog2_hw_IsDone(instance));

        fp32.i = XLog2_hw_Get_return(instance);
        return fp32.f;
}
```

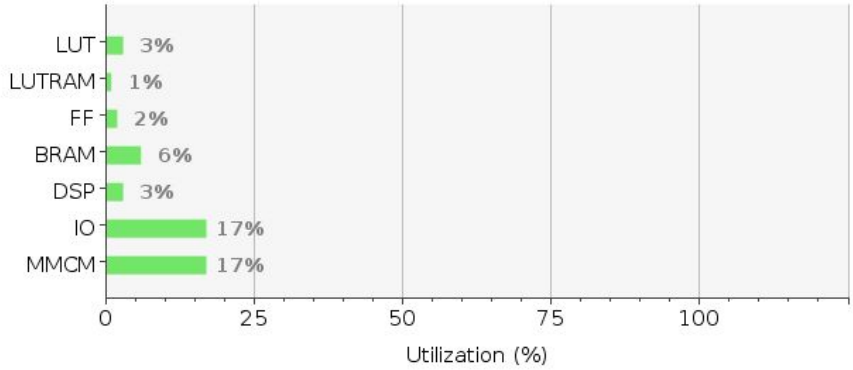
Функции, начинающиеся с `XLog2_hw`, входят в драйвер устройства и создаются автоматически при синтезе в Vivado HLS.

Порядок взаимодействия по шине AXI:

1. `XLog2_hw_IsReady` считывает статус готовности по адресу `0x0`.
2. `XLog2_hw_Set_x` записывает данные по адресу `0x18`. В нашем случае было записано число 0.313 (`0x3ea04189`).
3. `XLog2_hw_Start` выставляет нулевой бит по адресу `0x0` в единицу, что означает запуск функции.
4. `XLog2_hw_IsDone` считывает статус по адресу `0x0`. Значение `0x6` означает готовность, т.к. первый бит выставлен в единицу.
5. `XLog2_hw_Get_return` считывает выходное значение по адресу 0x10. В нашем случае был считан ответ -1.6757825613 (`0xbfd6800b`).

Отчет по занимаемым ресурсам и производительности

Resource	Utilization	Available	Utilization %
LUT	1946	63400	3.07
LUTRAM	117	19000	0.62
FF	1913	126800	1.51
BRAM	8	135	5.93
DSP	8	240	3.33
IO	36	210	17.14
MMCM	1	6	16.67



По сравнению с программной реализацией, использующей аппаратный FPU, время вычисления сократилось с 6570.00 нс до 2480.00 нс, то есть почти в три раза. Используемые ресурсы практически не изменились: объем LUT уменьшился с с 3.43% до 3.07%, объем DSP блоков увеличился с 0.83% до 3.33%.

Сравнение реализаций:

Реализация	Время вычисления, нс	Использование FF	Использование LUT	Использование DSP блоков
Программная, программный FPU	107540	1023	1148	0
Программная, аппаратный FPU	6570	1767	2177	2
Аппаратная	2480	1221	1330	8

Вывод

В результате выполнения работы была сначала реализована СнК, вычисляющая логарифм по основанию 2, затем был построен аппаратный ускоритель, реализующий логарифм, и в итоге этот ускоритель был внедрен в изначальную СнК.

При разработке аппаратного ускорителя была выполнена полная развертка цикла, а операции умножения и сложения были объединены в одну операцию `hls::fmaf`. В результате такой оптимизации время вычисления функции было уменьшено в 2.65 раза по сравнению с программной реализацией с аппаратной поддержкой чисел с плавающей точкой.