Харьковский национальный университет радиоэлектроники
(полное название высшего ученого заведения)

Кафедра «Электронных вычислительных машин»
(полное название кафедры)

КУРСОВОЙ ПРОЕКТ
(РАБОТА)

по курсу: Компьютерная схемотехника
(название дисциплины)

на тему: Проектирование процессорного модуля

Студента (ки) 3-го курса КИ-14-4 группы
направления подготовки Компьютерная инженерия
специальности КИ
Костюк С.А.
(фамилия и инициалы)

Руководитель Старший преподаватель
Дяченко В.А.
(должность, ученое звание, научная степень, фамилия и инициалы)

Национальная шкала
Количество балов: Оценка: ECTS

г. Харьков – 2016 год

РЕФЕРАТ

Пояснительная записка к курсовой работе содержит: 46 страницы, 15 рисунков, 4 раздела, 4 приложения, 6 источников.

Данная курсовая работа посвящена проектированию, синтезу и верификации цифрового арифметико-логического устройства, реализующего заданный набор операций над двоичными числами.

Целью данной работы является закрепление знаний и умений проектирования процессорных модулей на основе принципа микропрограммного управления, создание моделей цифровых устройств на языках описания аппаратуры, обобщение и углубление знаний по дисциплинам, связанным с проектированием средств вычислительной техники и реализации арифметических операций в ЭВМ.

Результатом выполнения курсового проектирования является верифицированная модель арифметико-логического устройства на языке описания аппаратуры VHDL, структурно-функциональная схема итогового устройства, а также другая дополнительная информация, необходимая для реализации АЛУ.

Операции, выполняемые АЛУ, соответствуют заданному варианту: деление целых двоичных знаковых чисел методом без восстановления остатка и умножение целых двоичных знаковых чисел.

АРИФМЕТИКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО, ПРОЦЕССОРНЫЙ МОДУЛЬ, УПРАВЛЯЮЩИЙ АВТОМАТ, ОПЕРАЦИОННЫЙ АВТОМАТ, I-АВТОМАТ, P-АВТОМАТ, ПРОГРАММИРУЕМЫЙ АВТОМАТ, МИКРОПРОГРАММНОЕ УПРАВЛЕНИЕ, АРИФМЕТИЧЕСКАЯ ОПЕРАЦИЯ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные электронно-вычислительные машины на данный момент представлены в самых разнообразных формах. Это и производительные серверы, позволяющие хранить и обрабатывать огромные массивы данных. И мобильные компьютеры, важнейшим фактором для которых является время автономной работы. И рабочие станции, позволяющие выполнять множество операций с мультимедиа и текстовыми данными, решать задачи разработки, исследований и симуляции различных систем и процессов. И встраиваемые системы, которые предназначены для решения специализированных задач с максимальной эффективностью в режиме 24/7.

Несмотря на все разнообразие вычислительных систем, все они имеют один важнейший компонент – центральный процессор, который выполняет заданные программы, управляет работой всего устройства, отвечает за большинство операций преобразования над поступающими данными.

Процессор, в свою очередь, включает в себя два основных модуля: устройство управления (УУ) и арифметико-логическое устройство (АЛУ). От скорости работы АЛУ зависит быстродействие компьютера и его способность к выполнению некоторого рода поставленных задач. При этом АЛУ различаются по производительности выполнения операций над целочисленными значениями, над числами с плавающей точкой и по скорости выполнения некоторых специализированных задач.

Зачастую задача разработки АЛУ ставится с целью повышения производительности ЭВМ в некоторых специфичных задачах, как задачи 3D-моделирования, криптографии, кодирования и декодирования мультимедиа или моделирования физических процессов. В таких случаях составляется спецификация, выбирается набор реализуемых операций, указывается желаемая скорость их выполнения и т.д.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

1.1 Общая характеристика арифметико-логических устройств, представление информации в цифровых ЭВМ

Процессор – центральная часть ЭВМ, которая отвечает за управление работой устройства и выполнение основной функции вычислителя – преобразования над данными. В свою очередь, процессор состоит из двух основных модулей: арифметико-логическое устройство (АЛУ) и устройство управления (УУ).

Арифметико-логическое устройство отвечает за выполнение арифметических и логических операций над данными, которые поступают в ЭВМ. Данные операции включают в себя умножение, деление, суммирование и вычитание, операции сравнения и сдвига, логические операции. Порядок выполнения данных операций, а также набор обрабатываемых данных определяется устройством управления и программой, которую он выполняет.

Важной характеристикой ЭВМ является формат данных, над которыми выполняются операции преобразования. В частности, разделяют аналоговые и цифровые ЭВМ. В аналоговых ЭВМ вычисления производятся над непрерывными величинами (например, напряжением на линии), а в цифровых – над дискретными значениями.

Цифровые ЭВМ в свою очередь разделяются по используемой системе счисления и ее основанию. В частности, существуют такие разновидности, как троичная (с основанием 3), десятичная и двоичная системы счисления.

Двоичная позиционная система счисления является наиболее распространённой на данный момент. Данные в ней представляются набором нулей и единиц (логических уровней), а значения зависят от их позиции в представлении числа.

Способы представления целых двоичных чисел также различаются. Беззнаковые числа представляются в прямом коде, для представления знаковых

чисел оптимальным является дополнительный код, в котором для представления знака используется старший бит числа.

С учетом указанных факторов, для выполнения курсового проектирования выбрано представление чисел в двоичном позиционном дополнительном коде.

1.2 Анализ используемых алгоритмов

В связи с особенностями представления чисел в цифровых ЭВМ, возникает необходимость в разработке оптимальных алгоритмов, выполняющих стандартный набор логических и арифметических операций над данными. В частности, созданы специализированные алгоритмы деления и умножения знаковых чисел.

Алгоритм деления целых знаковых двоичных чисел методом без восстановления остатка позволяет вычислить результат без использования дополнительного этапа восстановления остатка. Он состоит из нескольких стадий:

- проверка делителя на равенство нулю, генерация прерывания о недопустимой операции;
- пробное вычитание и проверка вместимости разрядной сетки, генерация прерывания о недостаточном ее размере;
- основной цикл деления;
- восстановление частичного остатка;
- коррекция результата деления.

Использование знаковых чисел приводит к некоторым отличиям по сравнению с базовым алгоритмом беззнакового деления. В частности, знак второго операнда в операции пробного вычитания выбирается в зависимости от знаков исходных аргументов. Аналогично, отличается алгоритм анализа результата пробного вычитания и генерации результата. Также вводятся

дополнительные этапы коррекции результата вычислений, изменяются условные переходы.

Граф-схема алгоритма деления представлена в Приложении А.

Алгоритм умножения двоичных знаковых чисел, начиная с младших разрядов множителя также отличается от аналогичного метода для беззнаковых чисел. В частности, вносятся изменения в алгоритм формирования старших разрядов произведения (используется флаг переполнения OF для выбора старшего бита при сдвиге вправо), добавляется дополнительный этап коррекции полученного произведения.

Граф-схема алгоритма умножения представлена в Приложении А.

Более подробная информация об используемых алгоритмах представлена в источнике [4].

1.3 Принцип микропрограммного управления

Функциональная и структурная организация операционных устройств, определяющая порядок функционирования и структуру устройств, базируется на принципе микропрограммного управления, который состоит в следующем:

1. Любая операция, выполняемая устройством, представляется как сложное действие, которое разделяется на множество элементарных действий – микроопераций.
2. Для управления порядком следования микроопераций используются логические условия, которые в зависимости от текущих значений и состояния устройства принимают одно из двух значений: «истина» или «ложь».
3. Процесс выполнения операций устройством описывается в виде алгоритма, называемого микропрограммой.
4. Микропрограмма используется как форма представления функции устройства, которая определяет структуру и порядок функционирования устройства во времени.

Описываемое устройство удобно разделять на две части: операционный автомат (ОУ) и управляющий автомат (УА).

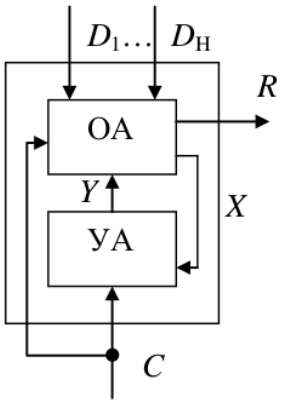

Рисунок 1.1 – Устройство как композиция автоматов

Операционный автомат непосредственно выполняет преобразования над данными в соответствии с заданными микрооперациями, формирует множество осведомительных сигналов X и результаты вычислений R. Управляющий автомат генерирует последовательность управляющих сигналов Y. Управляющая последовательность генерируется в соответствии с заданным алгоритмом (микропрограммой) и значениями осведомительных сигналов.

1.4 Постановка задачи

Задачей данного курсового проектирования является разработка арифметико-логического устройства, основанного на принципе микропрограммного управления, состоящего из операционного и управляющего автоматов и выполняющего операцию деления целых двоичных знаковых чисел методом без восстановления чисел, а также операцию умножения целых двоичных знаковых чисел, начиная с младших разрядов множителя. Числа представлены в двоичном позиционном дополнительном коде.

2 ПРОЕКТИРОВАНИЕ ОПЕРАЦИОННОГО АВТОМАТА

2.1 Общие сведения функционирования операционного автомата

Операционный автомат является исполнительной частью устройства. Он выполняет хранение, преобразование и пересылку данных, а также формирование множества осведомительных сигналов X. В соответствии с этим он, в общем случае, состоит их таких элементов, как элементы памяти и комбинационные схемы. Элементы памяти (регистры) выполняют промежуточное хранение аргументов и результатов вычислений, а также служебную информацию (как, например, флаги переполнения и переноса). Комбинационные схемы выполняют операции над данными (КС Ф) и формирование множества выходных осведомительных сигналов (КС ψ).

Общая структура ОА представлена на рисунке 2.1.

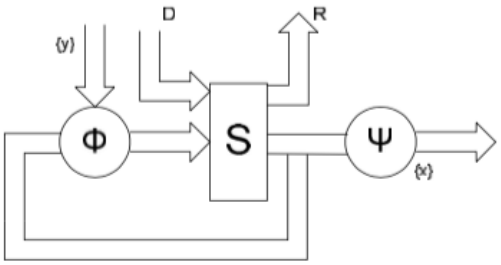

Рисунок 2.1 – Общая структура ОА

Функция операционного автомата определяется следующими множествами:

1. Множество входных слов D={d1, … , dH} (операндов).
2. Множеством выходных слов R={r1, … , rQ} (результатов вычислений).
3. Множеством внутренних слов S={s1, … , sN} (промежуточных результатов).
4. Множеством микроопераций Y.

5. Множеством логических условий X.

Основными характеристиками операционного автомата являются:

- производительность;
- быстродействие;
- аппаратные затраты.

Производительность ОА зависит от степени обобщения комбинационных схем, т.е. чем выше степень обобщения, тем ниже производительность. В зависимости от степени обобщения выделяют автоматы типов I, M, IM и канонического типа.

Наиболее производительной структурой ОА является каноническая структура, в которой отсутствует степень обобщения. Но она также имеет максимальные аппаратные затраты.

2.2 Структурная организация I-автоматов

Определим структуру ОА, производительность которой не ниже канонической, а затраты оборудования меньше. Это может быть обеспечено в случае, если синтезируемая структура не будет носить ограничений на совместимость микроопераций. Это может быть выполнено тогда, когда каждая комбинационная схема, используемая для выполнения эквивалентных микроопераций, связанных с вычислением значений одного слова, следовательно, для минимизации аппаратных средств необходимо обобщать комбинационные схемы для выполнения нескольких микроопераций, которые принадлежат одному подмножеству микроопераций, вычисляющему одно слово. ОА, структура которых обеспечивает возможность одновременного выполнения всех функционально совместимых микроопераций при использовании минимально возможного числа комбинационных схем, называются I-автоматами.

Т.о. особенностью I-автоматов является то, что каждый регистр обслуживается своей комбинационной схемой.

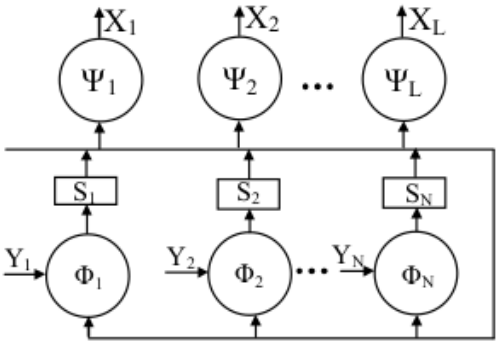

Рисунок 2.2 – Общая структура ОА типа I

2.3 Проектирование I-автомата

Проектирование I-автомата состоит из следующих этапов:

1. Описание входных и выходных слов.
2. Выбор множества регистров и определение их разрядности.
3. Составление списка всех микроопераций и логических условий.
4. Разбитие всех микроопераций на подмножества, соответствующие внутренним словам (регистрам).
5. Определение классов эквивалентных микроопераций на выделенных подмножествах, нахождение обобщенных операторов.

На первом этапе определяем множество входных и выходных слов в соответствии с алгоритмом, а также их разрядность. Получим, следующий список:

- D1 (2n-1 : 0), делимое или множимое;
- D2 (n-1 : 0), делитель или множитель;
- COP, код выполняемой операции (0 – деление, 1 – умножение);
- IRQ1, прерывание, деление на ноль;

– IRQ2, прерывание, недостаточный размер разрядной сетки;
– RL (n-1 : 0), результат деления либо младшая часть результата умножения;
– RH (n-1 : 0), старшая часть результата умножения.

На втором этапе определяем множество регистров:

– A (2n-1 : 0);
– B1 (n-1 : 0);
– B2 (n : 0);
– CnT(m-1 : 0) – счетчик;
– OF, CF – флаги переполнения и сброса соответственно;
– TgS – регистр для хранения знака делимого или младшего бита регистра B1.

На третьем этапе составляем список всех микроопераций и логических условий:

y1: A (2n-1 : 0) : = D1(2n-1 : 0)
y2: B1 (n-1 : 0) : = D2(n-1 : 0)
y3: C (n-1 : 0) : = 0
y4: CnT (m-1 : 0) : = n10
y5: C (n-1 : 0) : = C (n-1 : 0) + A(n-1 : 0)
y6: TgS := B1(0)
y7: B1(n-1 : 0) := R1(C(0) . B1(n-1 : 0)
y8: CnT : = CnT-1
y9: C (n-1 : 0) := R1(CF . C(n-1 : 0)
y10: C (n-1 : 0) := R1(C(n-1) . $\overline{C(n-1 : 0)}$
y11: C (n-1 : 0) : = C (n-1 : 0) + $\overline{A(n-1 : 0)}$ + 1
y12: RL (n-1 : 0) := B1 (n-1 : 0)
y13: RH (n-1 : 0) := C (n-1 : 0)
y14: B2 (n : 0) : = L1 (D2 (n-1 : 0) . 0)
y15: TgS := A(2n-1)
y16: B2 (n : 0) : = R1 (B2(n) . B2(n : 1))

x0: COP
x1: B1(0)
x2: OF
x3: CnT = 0
x4: TgS
x5: B2 = 0
x6: A(2n-1) $\oplus$ B2(n)
x7: A(2n-1) $\oplus$ TgS
x8: B2(n)
x9: B2(n) $\oplus$ TgS
x10: A = 0

y17: A (2n-1 : n-1) : = A (2n-1 : n-1) + $\overline{B2(n:0)}$ + 1
y18: A (2n-1 : n-1) := A (2n-1 : n-1) + B2(n : 0)
y19: C (n-1 : 0) := L1(C(n-2 : 0) . 1)
y20: C (n-1 : 0) := L1(C(n-2 : 0) . 0)
y21: A (2n-1 : 0) := L1(A(2n-2 : 0) . 0)
y22: C (n-1 : 0) : = C (n-1 : 0) + 1
y23: RL (n-1 : 0) := C(n-1 : 0)
y24: IRQ1 := 1
y25: IRQ2 := 1

На четвертом этапе группируем микрооперации по регистрам, в которые записывается результат:

$$Y_A = \{y1, y17, y18, y21\}$$
$$Y_{B1} = \{y2, y7\}$$
$$Y_{B2} = \{y14, y16\}$$
$$Y_C = \{y3, y5, y9, y10, y11, y19, y20, y22\}$$
$$Y_{CnT} = \{y3, y8\}$$
$$Y_{CF} = \{y5\}$$
$$Y_{OF} = \{y5\}$$
$$Y_{TgS} = \{y6, y15\}$$

На пятом этапе выполняем разбиение полученных множеств на классы эквивалентных микроопераций:

$$K_{A_1} = \{y1\} \qquad K_{C_1} = \{y3\}$$
$$K_{A_2} = \{y17, y18\} \qquad K_{C_2} = \{y5, y11, y22\}$$
$$K_{A_3} = \{y21\} \qquad K_{C_3} = \{y9, y10\}$$
$$K_{B1_1} = \{y2\} \qquad K_{C_4} = \{y19, y20\}$$
$$K_{B1_2} = \{y7\} \qquad K_{CnT_1} = \{y3\}$$
$$K_{B2_1} = \{y14\} \qquad K_{CnT_2} = \{y8\}$$
$$K_{B2_1} = \{y16\} \qquad K_{TgS} = \{y6, y15\}$$

Для каждого подмножества, содержащего не меньше двух микроопераций, находим обобщенный оператор, где отличающиеся аргументы заменяем на Arg1, Arg2 и т.д.:

$$K_{A_1}=\begin{cases} y17: A(2n\text{-}1:n\text{-}1) := A(2n\text{-}1:n\text{-}1) + \overline{B(n:0)} + 1 \\ y18: A(2n\text{-}1:n\text{-}1) := A(2n\text{-}1:n\text{-}1) + B(n:0) \end{cases} \Rightarrow$$

$$\Rightarrow A(2n\text{-}1:n\text{-}1) := A(2n\text{-}1:n\text{-}1) + Arg1 + Arg2$$

$$Arg1=\begin{cases} \overline{B},\ y17=1; \\ B,\ y18=1; \end{cases} \qquad Arg2=\begin{cases} 1, y17=1; \\ 0 \end{cases}$$

$$K_{C_2}=\begin{cases} y5: C(n\text{-}1:0) := C(n\text{-}1:0) + A(n\text{-}1:0) \\ y11: C(n\text{-}1:0) := C(n\text{-}1:0) + \overline{A(n\text{-}1:0)} + 1 \\ y22: C(n\text{-}1:0) := C(n\text{-}1:0) + 1 \end{cases} \Rightarrow C(n\text{-}1:0) := C(n\text{-}1:0) + Arg3 + Arg4$$

$$Arg3=\begin{cases} A(n\text{-}1:0),\ y5=1; \\ \overline{A(n\text{-}1:0)},\ y11=1; \\ 0,\ y22=1; \end{cases} \qquad Arg4=\begin{cases} 1, y11=1,\ y22=1; \\ 0 \end{cases}$$

$$K_{C_3}=\begin{cases} y9: C(n\text{-}1:0) := R1(CF\ .\ C(n\text{-}1:1) \\ y10: C(n\text{-}1:0) := R1(C(n\text{-}1\ .\ C(n\text{-}1:1) \end{cases} \Rightarrow C(n\text{-}1:0) := R1(Arg5\ .\ C(n\text{-}1:1)$$

$$Arg5=\begin{cases} CF,\ y9=1; \\ C(n\text{-}1),\ y10=1; \end{cases}$$

$$K_{C_4}=\begin{cases} y19: C(n\text{-}1:0) := L1(C(n\text{-}2:0)\ .\ 1) \\ y20: C(n\text{-}1:0) := L1(C(n\text{-}2:0)\ .\ 0) \end{cases} \Rightarrow C(n\text{-}1:0) := L1(C(n\text{-}2:0)\ .\ Arg6)$$

$$Arg6 = \begin{cases} 1, y19 = 1; \\ 0, y20 = 1; \end{cases}$$

$$K_{TgS}=\begin{cases} y6:\ TgS := B1(0) \\ y15:\ TgS := A(2n\text{-}1) \end{cases} \Rightarrow TgS := Arg7)$$

$$Arg7 = \begin{cases} B1(0), y6 = 1; \\ A(2n-1), y15 = 1; \end{cases}$$

По полученным данным составляются структурно-функциональная схема ОА (приложение Б) и его VHDL-модель (листинг Г.2). Для выбора аргументов используются мультиплексоры, у которых вход управления – это соответствующие коды микроопераций.

3 ПРОЕКТИРОВАНИЕ УПРАВЛЯЮЩЕГО АВТОМАТА

3.1 Описание управляющих автоматов с программируемой логикой

Функцией любого управляющего автомата является генерирование последовательности микрокоманд, определенной реализуемым алгоритмом. В автоматах с жесткой логикой формирование последовательности происходит в результате работы комбинационных схем выхода и переходов. Как результат – автомат может выполнять только тот алгоритм, который был заложен при его проектировании. Его структура не является универсальной и такой автомат нельзя перепрограммировать для выполнения каких-либо других алгоритмов.

В противоположность автоматам с жесткой логикой существуют автоматы с программируемой логикой (Р-автоматы). У таких автоматов вся информация, необходимая для реализации алгоритма, находится в памяти. Их структура слабо зависит от реализуемых алгоритмов, а для внесения изменений в работу автомата зачастую достаточно всего лишь изменить содержимое памяти.

Структурная схема Р-автомата в общем виде приведена на рисунке 3.1. Автомат включает в себя запоминающее устройство, содержащее набор микрокоманд, регистр микрокоманд и устройство формирования адреса микрокоманды.

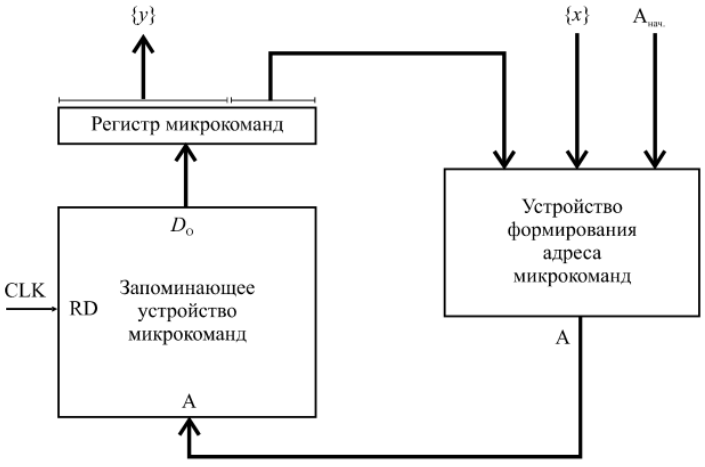

Рисунок 3.1 – Общая структура Р-автомата

В общем виде последовательность выполнения микропрограммы следующая:

1. При инициализации счетчик микрокоманд указывает на первую команду в памяти.
2. Из памяти выбирается микрокоманда, номер которой соответствует значению программного счетчика. Микрокоманда заносится в регистр микрокоманд.
3. Выполняется декодирование микроопераций, установка значений на выходах y.
4. Выполняется декодирование значений исполнительной части микрокоманды, декодируются условия перехода и определяется адрес следующей микрокоманды.
5. Адрес следующей микрокоманды заносится в счетчик микрокоманд.
6. Возвращение к пункту 2.

Р-автоматы различаются по типам адресации на такие категории:

– с принудительной адресацией и полным форматом микрокоманд;
– с принудительной адресацией и сокращенным форматом микрокоманд;
– с естественной адресацией.

В автоматах с принудительной адресацией каждая микрокоманда состоит из двух полей: поля микроопераций и поля адреса. Поле адреса содержит код условия перехода X и адрес (адреса) следующей команды A (рисунок 3.2).

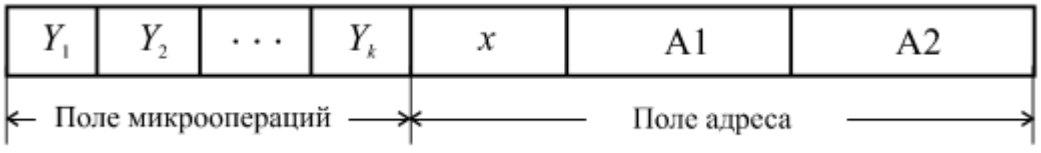

Рисунок 3.2 – Формат микрокоманды при принудительной адресации

В зависимости от количества адресных полей различают автоматы с полным и сокращенным форматом микрокоманды. Микрокоманды с полным форматом содержат два адресных поля: A1 и A2. Переход к номеру команды, указанной в поле A1, выполняется при удовлетворении условия X. Иначе выполняется переход к операции с кодом A2.

Микрокоманды с сокращенным форматом содержат только одно адресное поле A1. Переход к указанной команде выполняется при удовлетворении условия X либо при пустом поле условий. Иначе выполняется переход к команде с индексом, который на единицу больше указанного (A1 + 1).

Такой подход обеспечивает максимальное быстродействие автомата, но приводит к значительному увеличению используемого объема памяти. Особенно сильно эта особенность проявляется в случаях, когда множество микроопераций выполняется подряд. В таком случае поле условий перехода X пустует, а адреса следующих команд равны A(t) + 1.

Еще более заметное снижение эффективности проявляется при наличии в ГСА множества условных вершин, находящихся подряд. В таком случае пустует поле микроопераций и простаивает ОА. Как результат, получаем потери в скорости работы устройства и нерациональное использование памяти. В таких случаях в поле X имеет смысл указывать несколько условий одновременно.

Как следствие, в ряде случаев использование автоматов с принудительной адресацией не является рациональным, альтернативным решением является использование естественной адресации.

Следует также заметить, что по формированию сигналов Y Р-автоматы напоминают автоматы Мура: код микрооперации формируется на выходе в момент нахождения в некотором конкретном состоянии A, а не в момент перехода между ними (а не в момент перехода и выбора очередной микрокоманды).

3.2 Описание Р-автоматов с естественной адресацией

Р-автоматы с естественной адресацией имеют несколько важных отличий по сравнению с другими Р-автоматами. Данные отличия состоят в логике работы и формате микрокоманд автомата.

Зачастую все микрокоманды в данном типе устройств разделяются на два типа: операционные и управляющие. Операционные микрокоманды содержат тип команды и код выполняемых микроопераций Y. Управляющие команды в свою очередь содержат код условия перехода X и адрес следующей выполняемой команды A1.

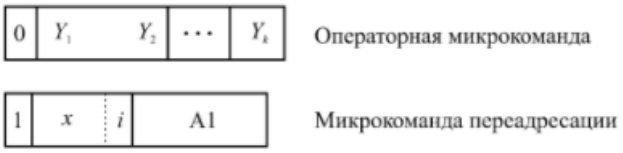

Рисунок 3.3 – Формат микрокоманд при естественной адресации

Определение типа микрокоманды определяется по старшему ее биту (T). Если команда операционная – выполняется декодирование указанных микроопераций, формирование выходных сигналов Y и переход к следующей по списку микрокоманде (A(t+1) = A(t) + 1). Если же команда управляющая – выполняется анализ условных сигналов X. Если указанное условие выполняется – автомат переходит к выполнению следующей по списку микрокоманде (A(t+1) = A(t) + 1). Иначе выполняется переход по адресу, указанному в поле A1 микрокоманды (A(t+1) = A1).

Рассмотрим структурную организацию автомата, который реализует данные принципы функционирования (рисунок 3.4). Как и любой другой Р-автомат, он содержит постоянную память (ROM, МПЗУ), хранящую таблицу микрокоманд (микропрограмму) автомата, устройство формирования выходных сигналов, устройство сравнения условных значений X и регистр, содержащий копию текущей команды РМК. В дополнение к этому схема

содержит счетчик микрокоманд РАМК (регистр адреса микрокоманды, программный счетчик), который содержит адрес текущей выполняемой микрокоманды.

Программный счетчик подключен на адресный вход ПЗУ. По указанному адресу выполняется выборка строки ПЗУ. Выбранная строка поступает на выход ПЗУ, к которому подключен регистр микрокоманд. Старший бит регистра микрокоманды подается на вход параллельной загрузки в РАМК и на разрешающий вход устройства формирования сигналов Y (обычно – регистр либо дешифратор, в зависимости от способа кодирования микрокоманд). Если бит типа команды равен нулю – параллельная загрузка в РАМК блокируется, а формирование сигналов Y разрешается. Иначе – на вход формирователя управляющих сигналов Y приходит неактивное значение, формирование Y-ков блокируется, активизируется работа мультиплексора MS. Мультиплексор MS выполняет выборку входных значений X по адресу, который содержится в адресном поле регистра текущей команды ОМК. Выбранное значение вместе с типом команды поступают на элемент И-НЕ. Выход элемента И-НЕ поступает на счетный вход РАМК через элемент И и на вход разрешения параллельной загрузки РАМК. Если приходит новый тактовый импульс – элемент И открывается и полученный сигнал подается на счетный вход РАМК, выполняется увеличение программного счетчика.

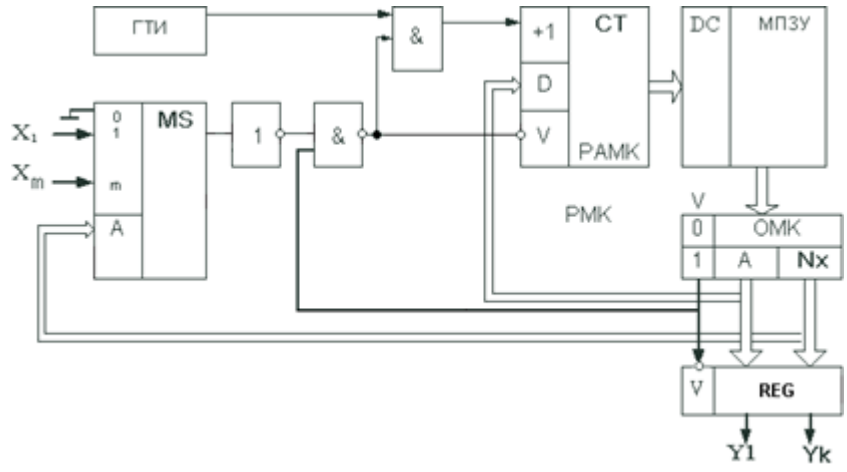

Рисунок 3.4 – Структура П-автомата с естественной адресацией

3.3 Проектирование Р-автомата с естественной адресацией

Разработка структуры УА с естественной адресацией подразумевает выполнение следующих этапов:

- выбор способа адресации и формата микрокоманды;
- определение формата операционной части микрокоманды;
- разметка ГСА в соответствии с выбранным способом адресации и формата микрокоманды;
- определение формата адресной части микрокоманды (формат управления микрокоманды);
- составление кодирования микропрограммы функционирования Р-автомата;
- карты программирования памяти;
- построение схемы;
- оценка характеристик.

3.3.1 Выбор способа адресации и формата микрокоманды

В соответствии с заданием на курсовое проектирование был выбран естественный способ адресации микрокоманд. Микрокоманды делятся на два типа: управляющие и операционные. Форматы микрокоманд подробно описаны в пункте 3.2 данной работы.

3.3.2 Определение формата операционной части микрокоманды

Различают три способа кодирования поля микроопераций:

- горизонтальный;
- вертикальный;
- смешанный.

При горизонтальном способе кодирования каждой микрооперации y i ∈ {y_1, ..., y_n} ставится в соответствие разряд поля микроопераций микрокомандного слова. В этом случае количество разрядов поля микроопераций N равно числу n различных микроопераций, вырабатываемых УА.

При вертикальном способе кодирования микроопераций каждая микрооперация кодируется двоичным позиционным кодом. При этом код, содержащий все нули, резервируется для обозначения отсутствия микроопераций на выходе управляющего автомата, а за один такт происходит выполнение только одной микрооперации.

При смешанном способе кодирования выполняется составление матрицы совместимых операций и распределение совместимых микроопераций по различным подмножествам. Микрооперации в каждом из подмножеств кодируются индивидуально по алгоритму вертикального кодирования.

Каждое подполе поля микроопераций кодирует микрооперации только одного подмножества $Y_i \subset Y$. Поскольку $\forall\ Y_i < Y$, разрядность k_i каждого из подполей может быть меньше k. Очевидно, при «удачном» распределении микроопераций по подмножествам можно будет реализовать любую операторную вершину ГСА микропрограммы с помощью одной микрокоманды (т.е. достигнуть быстродействия, характерного для горизонтального способа кодирования), при этом значительно уменьшить разрядность поля микроопераций даже по сравнению с каноническим способом смешанного кодирования.

Составляем матрицу совместимых микроопераций. На пресечении строк и столбцов, соответствующих микрооперациям с одинаковыми индексами находятся нули. На пересечении столбцов и строк с различными индексами ставится «1», если микрооперации являются совместимыми и «0» в другом случае. Результаты составления матрицы представлены в таблице 3.1.

Таблица 3.1 – Таблица совместимых микроопераций

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

После составления матрицы выполняем распределение микрокоманд по подмножествам. Максимальное количество совместимых микроопераций – 4, следовательно выделяются 4 подмножества микроопераций Y1, Y2, Y3, Y4. Распределение выполняем таким образом, чтобы получить минимальный размер итоговой микрокоманды. Для этого количество микроопераций в каждом из множеств должно стремиться к значению, равному степени двойки. При распределении следует отдавать предпочтение тем множествам, которые уже включают в себя большее количество микроопераций.

Первым делом выполняем распределение совместимых микроопераций:

Y1 : ∅, y1, y6, y12, y15

Y2 : ∅, y2, y13, y14, y16

Y3 : ∅, y3, y7

Y4 : ∅, y4, y8

Затем выполняем распределение оставшихся микроопераций:

Y1 : ∅, y1, y6, y12, y15, y5, y9, y10, y20, y21, y22, y23, y24, y25

Y2 : ∅, y2, y13, y14, y16, y11, y17, y18

Y3 : ∅, y3, y7, y19

Y4 : ∅, y4, y8

Следующий этап – кодирование микроопераций в каждом из множеств.

Для осведомительных сигналов используем алгоритм горизонтального кодирования. Результаты кодирования осведомительных и управляющих сигналов приведены в таблице 3.2.

Таблица 3.2 – Таблица совместимых микроопераций

Y1	K(Y1)	Y2	K(Y2)	Y3	K(Y3)	Y4	K(Y4)	X	K(X)
1	0001	2	001	3	01	4	01	0	0001
5	0010	11	010	7	10	8	10	1	0010
6	0011	13	011	19	11			2	0011
9	0100	14	100					3	0100
10	0101	16	101					4	0101
12	0110	17	110					5	0110
15	0111	18	111					6	0111
20	1000							7	1000
21	1001							8	1001
22	1010							9	1010
23	1011							10	1011
24	1100								
25	1101								

3.3.3 Разметка ГСА в соответствии с выбранным способом адресации и формата микрокоманды

Разметка ГСА ведется в соответствии с правилами разметки для естественной адресации. Алгоритм разметки следующий:

1. Выход первой вершины и вход последней вершины отмечаются меткой состояния a0.
2. После каждой операторной вершины ставим метки состояний, со значениями на единицу большими, чем у предыдущей метки.
3. Обходим граф-схему алгоритма «по единицам». Если на пути обхода встречается операторная вершина, выбирается ветка, соответствующая выполнению указанного условия. В процессе обхода расставляем по метки по порядку после каждой встреченной вершины.
4. После завершения обхода выполняем возврат к неотмеченным дугам. Ставим новую отметку на неотмеченном переходе, возвращаемся к пункту 3 и расставляем оставшиеся метки по ходу следования алгоритма.
5. Отметки, после которых не следуют операторные либо условные вершины, соответствуют управляющим микрокомандам, которые выполняют безусловный переход к следующей метке.

Результатом выполнения разметки является закодированная ГСА, приведенная в приложении В (рисунок В.2).

3.3.4 Составление микропрограммы функционирования Р-автомата

Составление микропрограммы функционирования Р-автомата выполняется по отмеченной граф-схеме алгоритма с использованием теоретических данных из пунктов 3.1 и 3.2 данной работы. Результатом составления микропрограммы является таблица программирования автомата.

№	T	ОР и Х				А					
		Y1				Y2		Y3		Y4	
a0	1	0	0	0	1	0	0	1	1	1	0	0
a1	0	0	0	0	1	0	0	1	0	1	0	1
a2	1	0	0	1	0	0	0	0	1	0	0	0
a3	0	0	0	1	0	0	0	0	0	0	0	0
a4	0	0	0	1	1	0	0	0	1	0	1	0
a5	1	0	0	1	1	0	0	1	1	0	0	0
a6	0	0	1	0	0	0	0	0	0	0	0	0
a7	1	0	1	0	1	0	0	1	0	1	0	0
a8	1	0	1	0	1	0	0	1	0	1	0	0
a9	0	0	0	0	0	0	1	0	0	0	0	0
a10	0	0	1	0	1	0	0	1	1	0	0	0
a11	1	0	0	0	0	0	0	0	0	0	0	0
a12	0	0	1	0	1	0	0	0	0	0	0	0
a13	1	0	0	0	0	0	0	0	1	1	1	0
a14	0	0	0	0	1	1	0	0	0	0	0	0
a15	1	0	1	1	0	0	1	0	0	1	0	0
a16	0	1	1	0	0	0	0	0	0	0	0	0
a17	1	0	0	0	0	0	0	0	0	0	0	0
a18	0	0	1	1	1	1	0	1	0	0	0	0
a19	1	0	1	1	1	0	1	1	0	0	0	0
a20	0	0	0	0	0	1	1	1	0	0	0	0
a21	1	0	1	1	1	0	1	1	0	1	0	0
a22	0	1	1	0	1	0	0	0	0	0	0	0
a23	1	0	0	0	0	0	0	0	0	0	0	0
a24	0	0	0	0	0	1	1	0	0	0	0	0
a25	1	0	1	1	1	0	1	0	1	1	0	0
a26	0	0	0	0	0	0	0	0	0	1	0	1

№	T	ОР и Х				А					
		Y1				Y2		Y3		Y4	
a27	1	0	1	1	1	1	0	0	1	0	1	0
a28	0	1	0	0	0	0	0	0	0	0	0	0
a29	0	0	0	0	0	0	0	0	0	0	1	0
a30	1	0	1	0	0	1	0	0	1	1	1	0
a31	1	1	0	0	0	1	0	1	1	1	0	0
a32	1	0	1	0	1	1	1	0	1	0	1	0
a33	1	1	0	1	1	1	1	0	1	0	1	0
a34	1	1	0	1	1	1	1	0	1	0	1	0
a35	0	1	0	1	1	0	0	0	0	0	0	0
a36	1	0	0	0	0	0	0	0	0	0	0	0
a37	0	0	0	0	0	0	0	0	1	1	0	0
a38	1	0	0	0	0	0	1	1	1	0	1	0
a39	1	0	1	1	1	1	0	1	0	1	1	0
a40	0	1	0	0	1	0	0	0	0	0	0	0
a41	0	0	0	0	0	1	1	1	0	1	1	0
a42	1	0	0	0	0	0	1	1	0	1	1	0
a43	0	1	0	0	1	0	0	0	0	0	0	0
a44	0	0	0	0	0	1	1	0	0	0	0	0
a45	1	1	0	0	1	1	1	0	0	0	1	0
a46	1	1	0	0	1	1	1	0	0	0	1	0
a47	0	0	0	0	0	1	1	0	0	0	0	0
a48	1	0	0	0	0	1	0	0	0	0	0	0
a49	0	0	0	0	0	1	1	1	0	0	0	0
a50	1	0	0	0	0	1	0	0	0	0	0	0
a51	1	0	1	0	1	1	0	0	0	1	1	0
a52	1	1	0	1	1	1	0	0	0	1	1	0
a53	0	1	0	1	0	0	0	0	0	0	0	0
a54	1	0	0	0	0	1	0	0	0	1	1	0

Таблица 3.3 – Кодирование микропрограммы функционирования

3.3.5 Построение схемы

Построение схемы автомата выполняется в соответствии с пунктами 3.1 и 3.2 данной работы. Для формирования выходных сигналов Y используются дешифраторы, адресные входы которых подключены на соответствующие выходы регистра команд. Для проверки условий X используется мультиплексор, выполняющий выборку входных значений X по адресу,

указанному в регистре микрокоманд. Результат построения схемы приведен в приложении В (рисунок В.3).

3.3.6 Оценка характеристик

Характеристики полученного автомата обусловлены характеристиками его составляющих частей: операционного и управляющего автоматов. Операционный автомат типа I обеспечивает выполнение совместимых микроопераций за один такт при умеренных аппаратных затратах. Управляющий автомат типа P обеспечивает высокую гибкость полученного устройства, но дополнительные микрокоманды безусловного перехода приводят к уменьшению производительности итогового устройства.

Как следствие, полученный вычислительный модуль требует умеренных аппаратных затрат на свою реализацию, обладает не самой высокой производительностью и может перепрограммироваться для выполнения измененных алгоритмов.

4 МОДЕЛИРОВАНИЕ И АНАЛИЗ РЕЗУЛЬТАТОВ СИНТЕЗА

Результатом автоматизированного синтеза VHDL-модели является RTL-схема процессорного модуля, приведенная на рисунке 4.1. Сгенерированная структура процессорного модуля а также его составляющих соответствует общим структурам ОА типа I и УА типа P, что свидетельствует об успешном составлении VHDL-модели устройства.

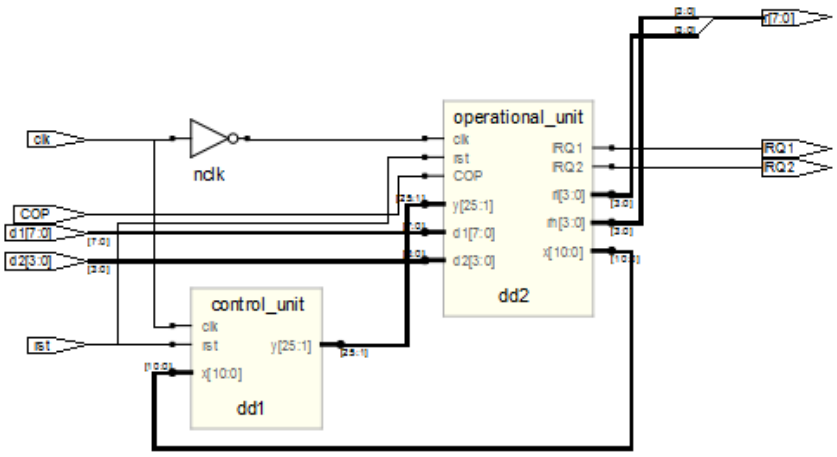

Рисунок 4.1 – Результат автоматизированного синтеза АЛУ

Верификация VHDL-модели разработанного устройства выполнена средствами компилятивной системы моделирования Aldec Active-HDL. Верификация показала соответствие созданного устройства заданным требованиям и корректное выполнение указанных арифметических операций деления и умножения.

Результат выполнения умножения был успешно получен на выходах устройства RL и RH, где RL – младшая часть результата, а RH – старшая часть результата. Результат выполнения умножения был успешно получен на выходе RL. До момента получения результатов соответствующие выходы устанавливаются в состояния высокого импеданса.

Для тестирования выбраны комбинации аргументов с различными знаками и разрядностями. Выполнена проверка корректности результатов для

«крайних» случаев, таких как деление и умножение на ноль, умножение и деление на отрицательные числа, деление большого числа на малое с последующим переполнением разрядной сетки результата.

Примеры результатов тестирования отображены на временных диаграммах ниже (рисунки 4.2 и 4.3). Листинги ОА, УА, а также процессорного модуля и их компонентов представлены в приложении Г.

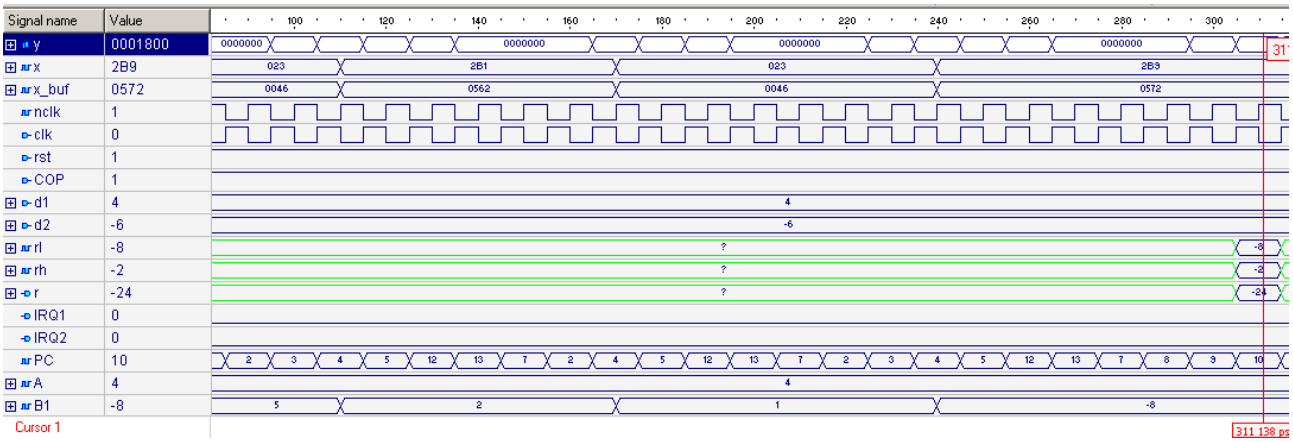

Рисунок 4.2 – Результат выполнения операции умножения целых двоичных знаковых чисел методом старшими разрядами вперед

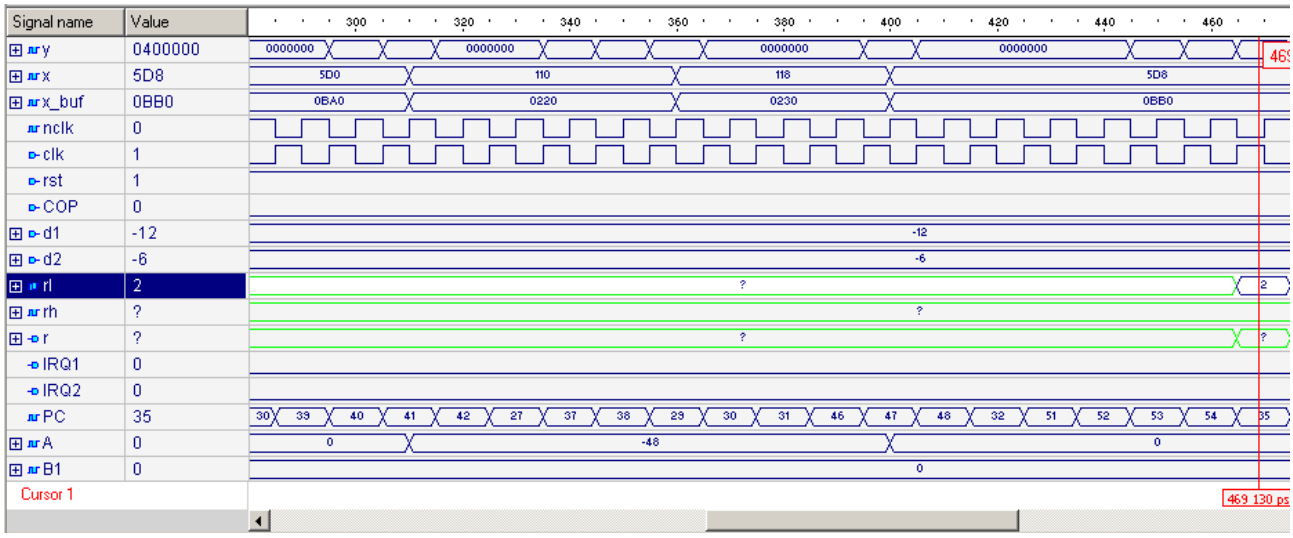

Рисунок 4.3 – Результат выполнения операции деления целых двоичных знаковых чисел методом без восстановления остатка

ВЫВОДЫ

В ходе выполнения курсовой работы был разработан процессорный модуль, основанный на композиции управляющего и операционного автоматов и выполняющий операции умножения и деления знаковых двоичных целых чисел.

Были исследованы алгоритмы выполнения заданных операций преобразований над двоичными числами, составлена закодированная граф-схема алгоритма, выбраны разрядности портов и регистров операционного автомата, составлен список выполняемых микроопераций Y и осведомительных сигналов X. Были закодированы осведомительные и управляющие сигналы, составлена микропрограмма и таблица программирования P-автомата с естественной адресацией. Для управляющего и операционного автоматов составлены структурные схемы и модели устройств на языке описания аппаратуры VHDL. Созданные модели были верифицированы в среде Active-HDL, составлены временные диаграммы, демонстрирующие работу устройства.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Жмакин А. П. – Архитектура ЭВМ: 2-е изд., перераб. и доп.: учеб. пособие. – СПб.: БХВ- Петербург, 2010
2. Кулак Э. Н. – Конспект лекций по дисциплине «Прикладная теория цифровых автоматов» – Харьков, 2009 г.
3. Кораблев Н. М. – Конспект лекций по дисциплине «Компьютерная схемотехника» – Харьков, 2010 г.
4. Кораблев Н. М., Саранча С. Н., Саранча О. Н. – Методические указания к лабораторным работам по дисциплине «Компьютерная схемотехника»: Часть 2 «Проектирование сложных систем» – Харьков: ХНУРЭ, 2006 г.
5. Конспект лекций по дисциплине «Основы компьютерных вычислений»
6. Савельев А. Я. Прикладная теория цифровых автоматов – М. «Высшая школа». 1987.

ПРИЛОЖЕНИЕ А

Объединенная ГСА процессорного модуля

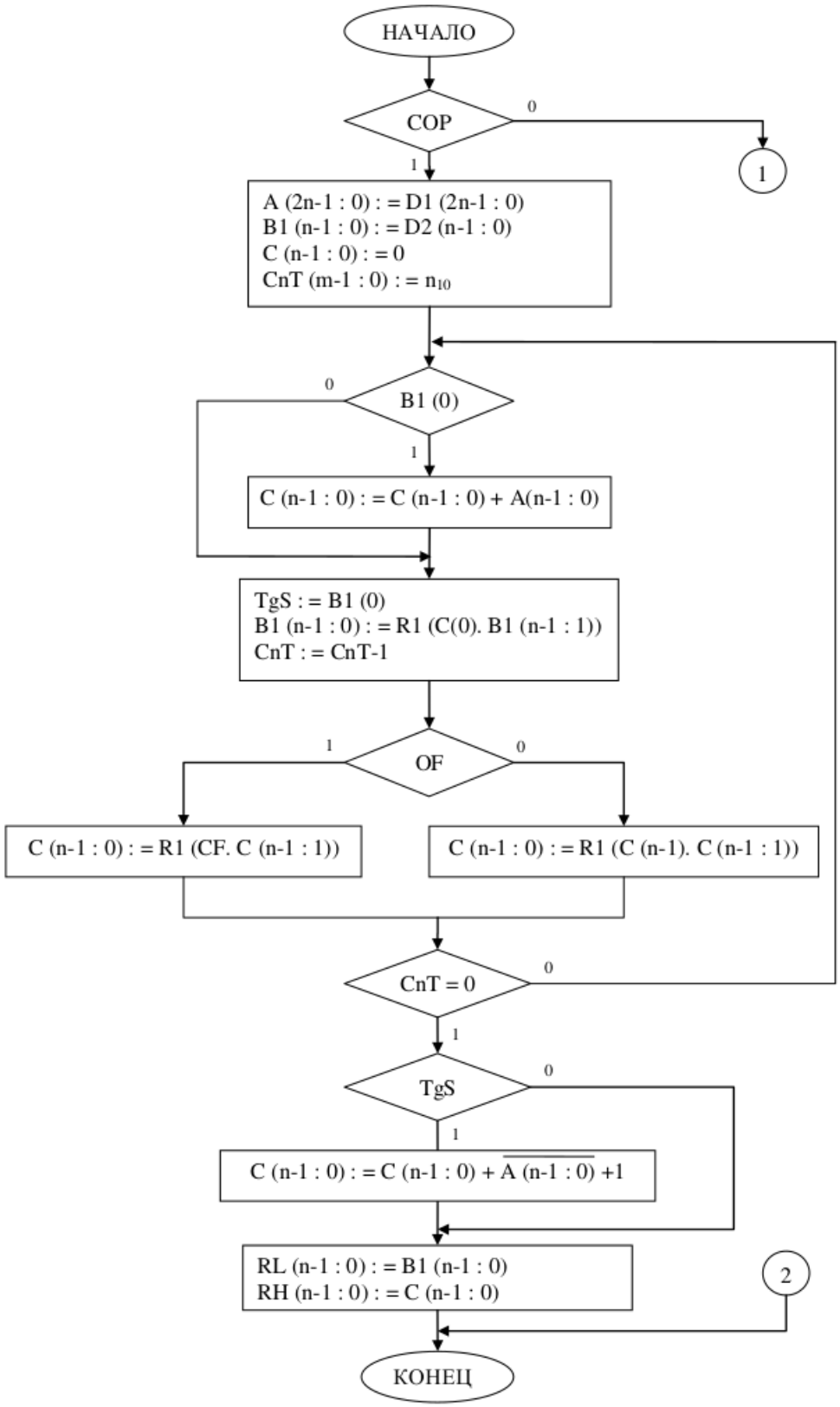

Рисунок А.1 – Объединенная ГСА, часть 1

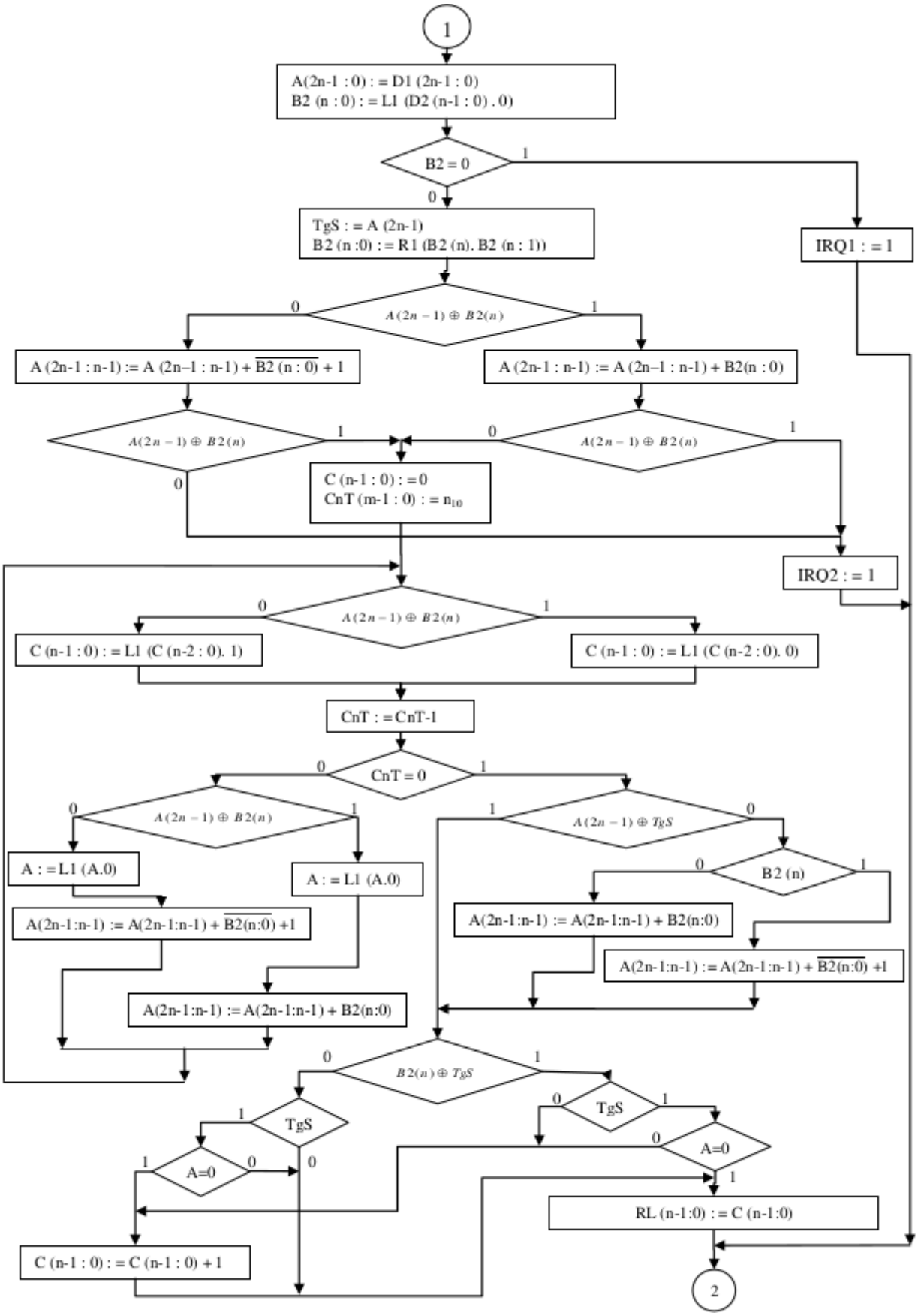

Рисунок А.2 – Объединенная ГСА, часть 2

ПРИЛОЖЕНИЕ Б

Проектирование структурной модели ОА типа I.

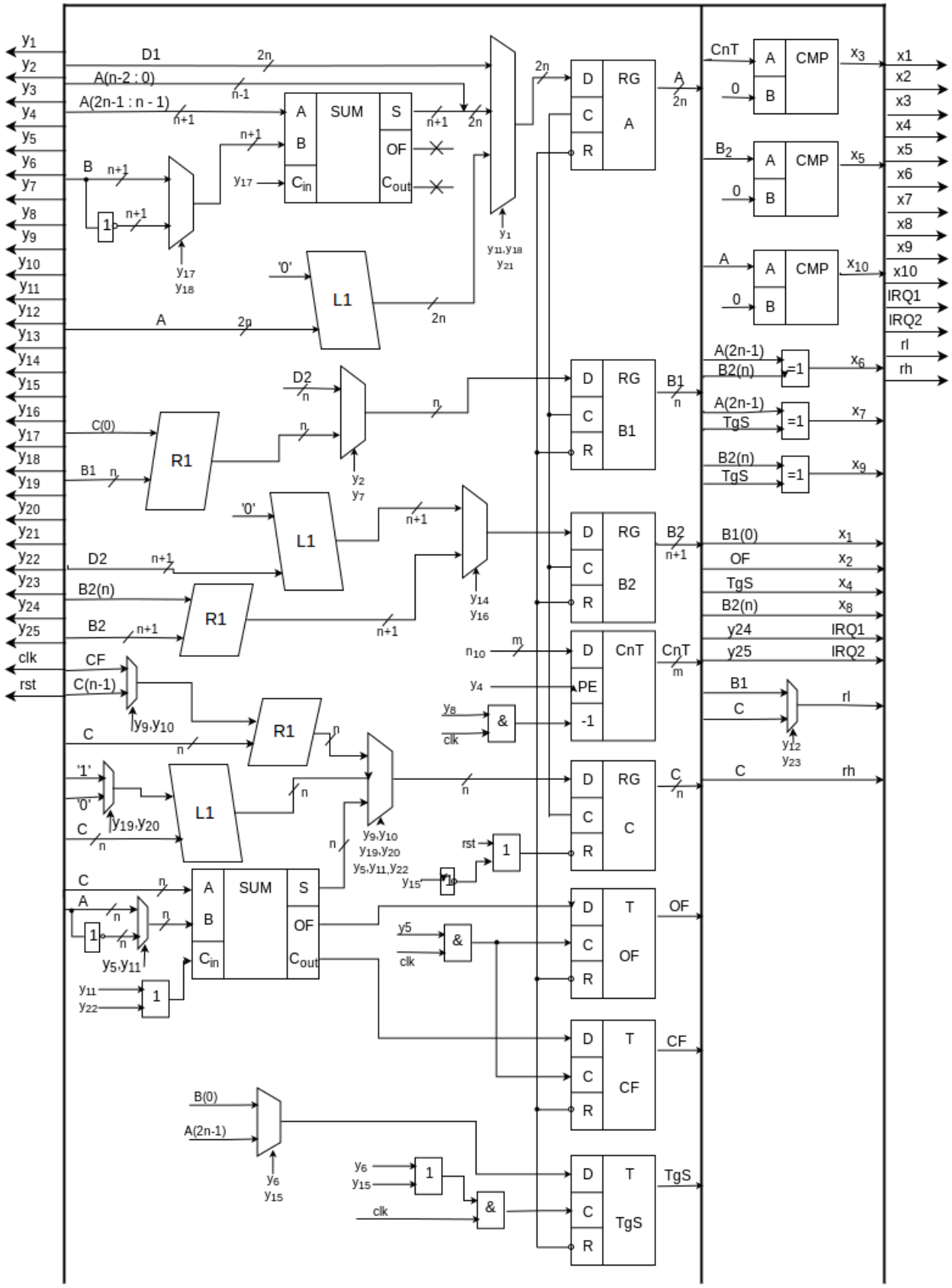

Рисунок Б.1 – Структурно-функциональная схема I-автомата

ПРИЛОЖЕНИЕ В

Результаты проектирования УА с программируемой логикой

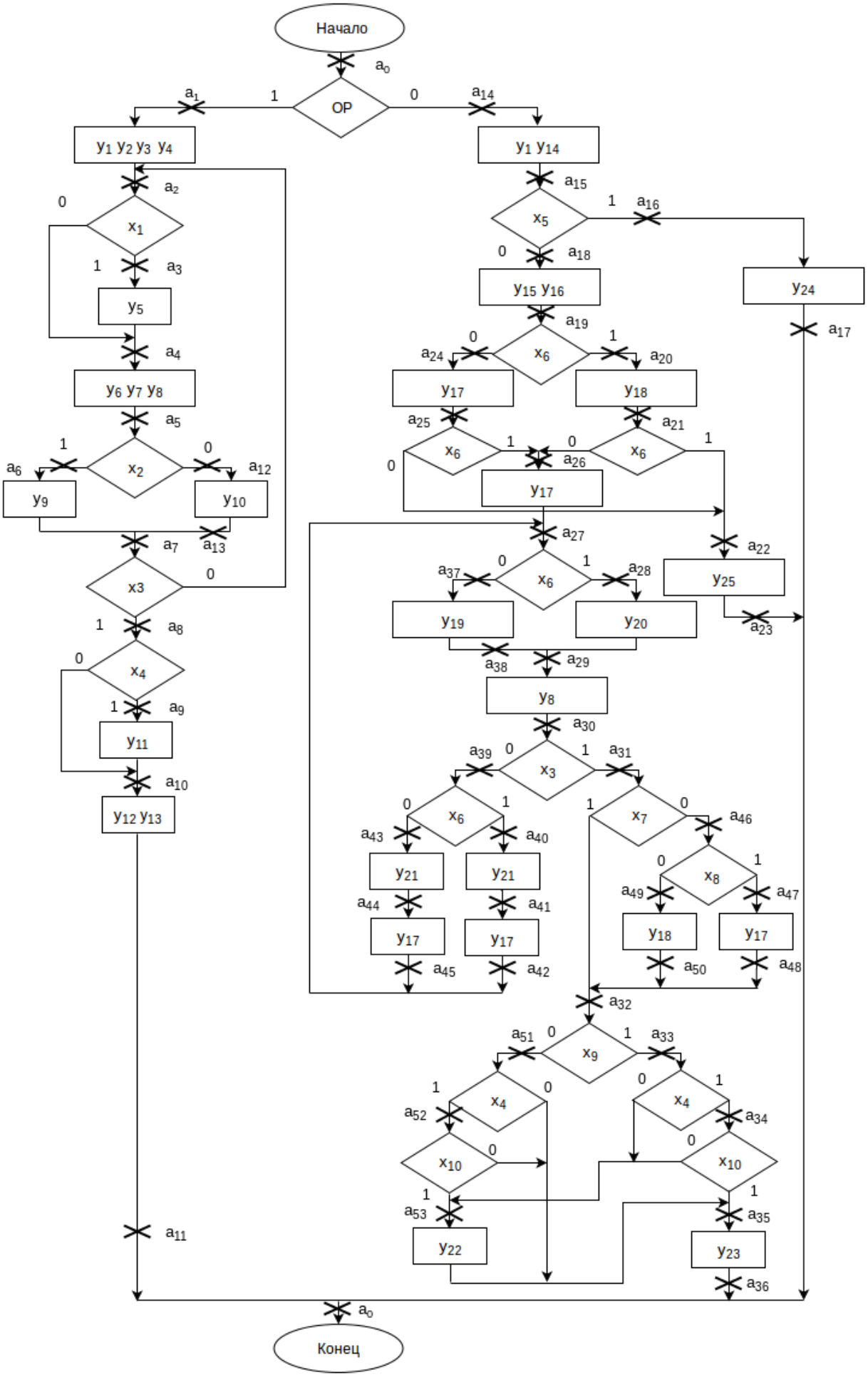
Начало
a0
OP
a1
a14
y1 y2 y3 y4
y1 y14
a2
a15
x1
x5
a3
a16
a18
y5
y15 y16
y24
a4
a19
a17
y6 y7 y8
x6
a5
a24
a20
y17
y18
a25
a21
x2
x6
x6
a6
a12
a26
y9
y10
y17
a7
a13
a27
a22
x3
x6
a37
a28
y25
y19
y20
a23
a8
a38
a29
x4
y8
a9
a30
y11
a39
a31
x3
a10
x6
x7
a46
y12 y13
a43
a40
x8
y21
y21
a49
a47
a44
a41
y18
y17
y17
y17
a50
a48
a45
a42
a32
a11
a51
a33
x9
x4
x4
a52
a34
x10
x10
a53
a35
y22
y23
a36
a0
Конец

Рисунок В.1 – Закодированная ГСА

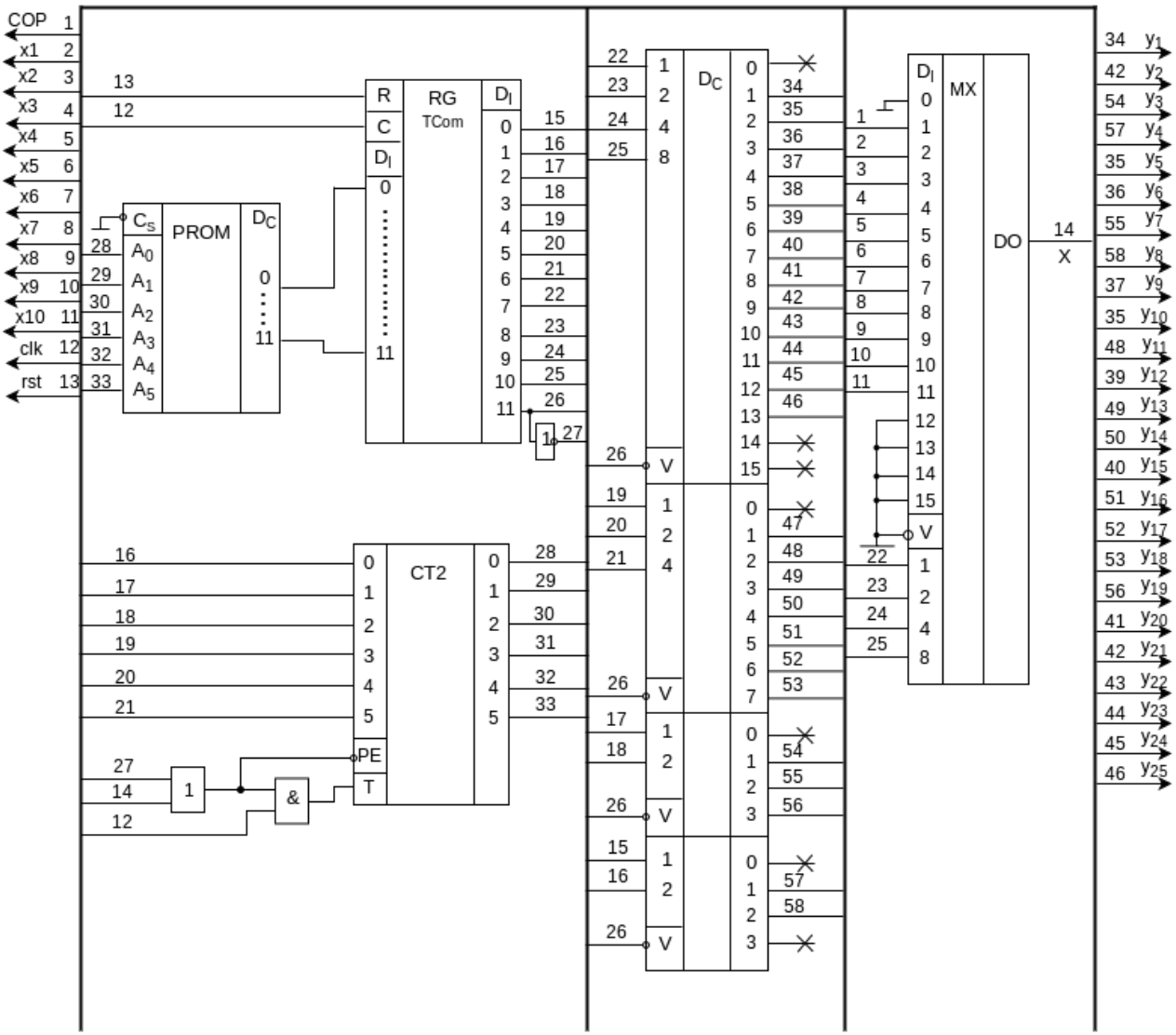

Рисунок В.2 – Структурно-функциональная схема УА

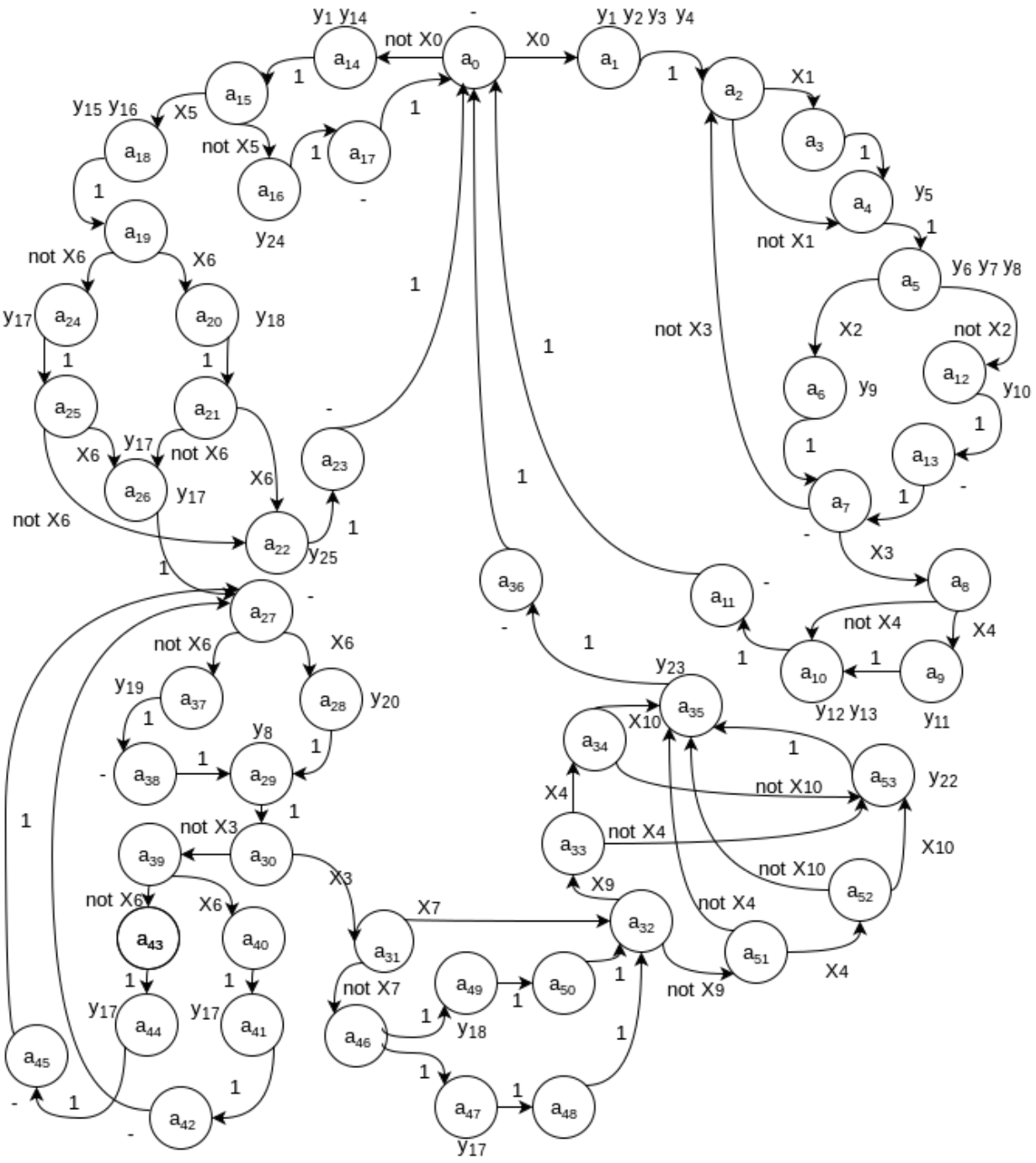

Рисунок В.3 – Граф переходов для объединенной ГСА

ПРИЛОЖЕНИЕ Г

Листинги элементов процессорного модуля

Листинг Г.1 – VHDL модель микропрограммного УА с естественной адресацией

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity control_unit is
	port(
		clk, rst : in STD_LOGIC;
		x : in STD_LOGIC_VECTOR(10 downto 1);
		y : out STD_LOGIC_VECTOR(25 downto 1);
		COP: in std_logic
		);
end control_unit;
architecture control_unit of control_unit is
	use ieee.std_logic_unsigned.all;
	subtype TCommand is std_logic_vector(11 downto 0);
	type TROM is array(0 to 54) of TCommand;
	constant ROM:TROM := (
	-- p, y1, y2, y3, y4
	-- p, x, a
	"1" & "0001" & "001" & "11" & "00", -- 0
	"0" & "0001" & "001" & "01" & "01", -- 1
	"1" & "0010" & "000" & "10" & "00", -- 2
	"0" & "0010" & "000" & "00" & "00", -- 3
	"0" & "0011" & "000" & "10" & "10", -- 4
	"1" & "0011" & "001" & "10" & "00", -- 5
	"0" & "0100" & "000" & "00" & "00", -- 6
	"1" & "0100" & "000" & "01" & "00", -- 7
	"1" & "0101" & "001" & "01" & "00", -- 8
	"0" & "0000" & "010" & "00" & "00", -- 9
	"0" & "0110" & "011" & "00" & "00", --10
	"1" & "0000" & "000" & "00" & "00", --11
	"0" & "0101" & "000" & "00" & "00", --12
	"1" & "0000" & "000" & "11" & "10", --13
	"0" & "0001" & "100" & "00" & "00", --14
	"1" & "0110" & "010" & "01" & "00", --15
	"0" & "1100" & "000" & "00" & "00", --16
	"1" & "0000" & "000" & "00" & "00", --17
	"0" & "0111" & "101" & "00" & "00", --18
	"1" & "0111" & "011" & "00" & "00", --19
	"0" & "0000" & "111" & "00" & "11", --20
	"1" & "0111" & "011" & "01" & "00", --21
	"0" & "1101" & "000" & "00" & "00", --22
	"1" & "0000" & "000" & "00" & "00", --23
	"0" & "0000" & "110" & "00" & "00", --24
	"1" & "0111" & "010" & "11" & "00", --25
	"0" & "0000" & "000" & "01" & "01", --26
	"1" & "0111" & "100" & "10" & "10", --27
	"0" & "1000" & "000" & "00" & "00", --28
	"0" & "0000" & "000" & "00" & "10", --29
	"1" & "0100" & "100" & "11" & "10", --30
	"1" & "1000" & "101" & "11" & "00", --31
	"1" & "1010" & "110" & "01" & "10", --32
	"1" & "0101" & "110" & "10" & "10", --33
	"1" & "1011" & "110" & "10" & "10", --34
```

```
        "0" & "1011" & "000" & "00" & "00", --35
        "1" & "0000" & "000" & "00" & "00", --36
        "0" & "0000" & "000" & "11" & "00", --37
        "1" & "0000" & "011" & "10" & "10", --38
        "1" & "0111" & "101" & "01" & "10", --39
        "0" & "1001" & "000" & "00" & "00", --40
        "0" & "0000" & "111" & "00" & "00", --41
        "1" & "0000" & "011" & "01" & "10", --42
        "0" & "1001" & "000" & "00" & "00", --43
        "0" & "0000" & "110" & "00" & "00", --44
        "1" & "0000" & "011" & "01" & "10", --45
        "1" & "1001" & "110" & "00" & "10", --46
        "0" & "0000" & "110" & "00" & "00", --47
        "1" & "0000" & "100" & "00" & "00", --48
        "0" & "0000" & "111" & "00" & "00", --49
        "1" & "0000" & "100" & "00" & "00", --50
        "1" & "0101" & "100" & "01" & "10", --51
        "1" & "1011" & "100" & "01" & "10", --52
        "0" & "1010" & "000" & "00" & "00", --53
        "1" & "0000" & "100" & "01" & "10"  --54
        );
        signal RegCom:TCommand;
        signal PC:integer;
        signal x_decoded: std_logic;
        signal y1_decoded: std_logic_vector(15 downto 0);
        signal y2_decoded: std_logic_vector(7 downto 0);
        signal y3_decoded: std_logic_vector(3 downto 0);
        signal y4_decoded: std_logic_vector(3 downto 0);
        signal not_p: std_logic;
        signal x_buf: std_logic_vector(15 downto 0);

        component decoder is
             generic(
                   N: integer := 4
                   );
             port(D: in std_logic_vector (N-1 downto 0);
                   En: in std_logic;
                   Q: out std_logic_vector (2**N -1 downto 0)
                   );
        end component;

        component mx is
             generic(
                   N: integer := 4
                   );
             port(
                   A: in std_logic_vector(N-1 downto 0);
                   D: in std_logic_vector(2**N-1 downto 0);
                   En: in std_logic;
                   Q: out std_logic
                   );
        end component;
begin
        not_p <= not RegCom(11);
        x_buf <= "0000" & x & COP & "0";

        x_mx: mx
        generic map(N => 4)
        port map(A => RegCom(10 downto 7), D=>x_buf, En=>'0', Q => x_decoded);

        y1_dc: decoder
             generic map(N => 4)
```

```
        port map(D => RegCom(10 downto 7), En=>RegCom(11), Q=>y1_decoded);

    y2_dc: decoder
        generic map(N => 3)
        port map(D => RegCom(6 downto 4), En=>RegCom(11), Q=> y2_decoded);

    y3_dc: decoder
        generic map(N => 2)
        port map(D => RegCom(3 downto 2), En=>RegCom(11), Q=> y3_decoded);

    y4_dc: decoder
        generic map(N => 2)
        port map(D => RegCom(1 downto 0), En=>RegCom(11), Q=> y4_decoded);


    process(rst,clk) is
        variable PCv:integer range 0 to 54;
    begin
        if rst='0' then PCv:=0;
        elsif rising_edge(clk) then
            if RegCom(11)='1' and x_decoded = '0' then
                PCv:=conv_integer(RegCom(6 downto 1));
            else
                PCv:=PCv+1;
            end if;
        end if;
        RegCom<=ROM(PCv);
        PC<=PCv;
    end process;

    y(1)  <= y1_decoded(1);
    y(5)  <= y1_decoded(2);
    y(6)  <= y1_decoded(3);
    y(9)  <= y1_decoded(4);
    y(10) <= y1_decoded(5);
    y(12) <= y1_decoded(6);
    y(15) <= y1_decoded(7);
    y(20) <= y1_decoded(8);
    y(21) <= y1_decoded(9);
    y(22) <= y1_decoded(10);
    y(23) <= y1_decoded(11);
    y(24) <= y1_decoded(12);
    y(25) <= y1_decoded(13);

    y(2)  <= y2_decoded(1);
    y(11) <= y2_decoded(2);
    y(13) <= y2_decoded(3);
    y(14) <= y2_decoded(4);
    y(16) <= y2_decoded(5);
    y(17) <= y2_decoded(6);
    y(18) <= y2_decoded(7);

    y(3)  <= y3_decoded(1);
    y(7)  <= y3_decoded(2);
    y(19) <= y3_decoded(3);

    y(4)  <= y4_decoded(1);
    y(8)  <= y4_decoded(2);
end architecture control_unit;
```

Листинг Г.2 – VHDL модель ОА типа I

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_signed.all;
use IEEE.STD_logic_arith.all;

entity operational_unit is
	generic(
		N: integer := 4;
		M: integer := 8
		);
	port(
		clk,rst : in STD_LOGIC;
		y : in STD_LOGIC_VECTOR(25 downto 1);
		d1 : in STD_LOGIC_VECTOR(2*N-1 downto 0);
		d2 : in STD_LOGIC_VECTOR(N-1 downto 0);
		rl, rh : out STD_LOGIC_VECTOR(N-1 downto 0);
		x : out STD_LOGIC_vector(10 downto 1);
		IRQ1, IRQ2 : out std_logic
		);
end operational_unit;

architecture operational_unit of operational_unit is
	signal A,Ain: STD_LOGIC_VECTOR(2*N-1 downto 0) ;
	signal B1,B1in: STD_LOGIC_VECTOR(N-1 downto 0) ;
	signal B2,B2in: STD_LOGIC_VECTOR(N downto 0) ;
	signal CnT, CnTin: std_logic_vector(M-1 downto 0);
	signal C, Cin:  STD_LOGIC_VECTOR(N-1 downto 0) ;
	signal overflow, carry: std_logic;
	signal of_in, cf_in: std_logic;
	signal of_sum, cf_sum: std_logic;
	signal TgS, TgSin: std_logic;
	signal sum_result: std_logic_vector(N-1 downto 0);

	component adder is
		generic(
			N: integer := 4
			);
		port(A, B: in std_logic_vector(N-1 downto 0);
			Cin: in std_logic;
			S: out std_logic_vector(N-1 downto 0);
			Cout: out std_logic;
			overflow: out std_logic);
	end component;

begin
	process(clk,rst)is
	begin
		if rst='0' then
			A <= (others=>'0');
			B1 <= (others=>'0');
			B2 <= (others=>'0');
			TgS <= '0';
			Overflow <= '0';
			Carry <= '0';
			CnT <= (others=>'0');
		elsif rising_edge(clk)then
			A <= Ain;
			B1 <= B1in;
			B2 <= B2in;
			TgS <= TgSin;
```

```
                CnT <= CnTin;
                C <= Cin;
                Overflow <= of_in;
                Carry <= cf_in;
            end if;
        end process;

        -- Подключение сумматора
        SUM : adder port map(A => C, B => A(N-1 downto 0), Cin => '0', Cout =>
cf_sum, overflow => of_sum, S => sum_result);

        -- Реализация микроопераций
        Ain <= D1 when y(1)='1'
        else (A(2*N-1 downto N-1) + not B2 + 1)& A(N-2 downto 0) when y(17)='1'
        else (A(2*N-1 downto N-1) + B2) & A(N-2 downto 0) when y(18) = '1'
        else A(2*N-2 downto 0) & '0' when y(21) = '1'
        else A;

        B1in <= D2 when y(2) = '1'
        else C(0) & B1(N-1 downto 1) when y(7) = '1'
        else B1;

        Cin <= (others => '0') when y(3) = '1'
        else sum_result when y(5) = '1'
        else carry & C(N-1 downto 1) when y(9) = '1'
        else C(N-1)& C(N-1 downto 1) when y(10) = '1'
        else C + not A(N-1 downto 0) + 1 when y(11) = '1'
        else C(N-2 downto 0) & '1' when y(19) = '1'
        else C(N-2 downto 0) & '0' when y(20) = '1'
        else C + 1 when y(22) = '1'
        else C;

        cf_in <= cf_sum when y(5) = '1'
        else carry;

        of_in <= of_sum when y(5) = '1'
        else overflow;

        CnTin <= conv_std_logic_vector(N, M) when y(4) = '1'
        else CnT - 1 when y(8) = '1'
        else CnT;

        TgSin <= B1(0) when y(6) = '1'
        else A(2*N-1) when y(15) = '1'
        else TgS;

        RL <= B1 when y(12) = '1'
        else C when y(23) = '1'
        else (others => 'Z');

        RH <= C when y(13) = '1'
        else (others => 'Z');

        B2in <= D2 & '0' when y(14) = '1'
        else B2(N) & B2(N downto 1) when y(16) = '1'
        else B2;

        IRQ1 <= '1' when y(24) = '1'
        else '0';

        IRQ2 <= '1' when y(25) = '1'
        else '0';
```

```
        -- Осведомительные сигналы
        x(1)  <= B1(0);
        x(2)  <= overflow;
        x(3)  <= '1' when CnT = 0 else '0';
        x(4)  <= TgS;
        x(5)  <= '1' when B2 = 0 else '0';
        x(6)  <= A(2*N-1) xor B2(N);
        x(7)  <= A(2*N-1) xor TgS;
        x(8)  <= B2(N);
        x(9)  <= B2(N) xor TgS;
        x(10) <= '1' when A = 0 else '0';

end operational_unit;
```

Листинг Г.3 – VHDL модель процессорного устройства

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

entity top is
	generic(
		N: integer := 4;
		M: integer := 8
		);
	port(
		clk : in STD_LOGIC;
		rst : in STD_LOGIC;
		COP : in std_logic;
		d1 : in STD_LOGIC_VECTOR(2*N-1 downto 0);
		d2 : in STD_LOGIC_VECTOR(N-1 downto 0);
		r : out STD_LOGIC_VECTOR(2*N-1 downto 0);
		IRQ1, IRQ2 : out std_logic
		);
end top;

architecture top of top is
	component control_unit
		port (
			clk,rst : in STD_LOGIC;
			X: in STD_LOGIC_vector(10 downto 1);
			Y: out STD_LOGIC_vector(25 downto 1);
			COP : in std_logic);
	end component;

	component operational_unit port(
			clk,rst : in STD_LOGIC;
			y : in STD_LOGIC_VECTOR(25 downto 1);

			d1 : in STD_LOGIC_VECTOR(2*N-1 downto 0);
			d2 : in STD_LOGIC_VECTOR(N-1 downto 0);
			rl, rh : out STD_LOGIC_VECTOR(N-1 downto 0);
			x : out STD_LOGIC_vector(10 downto 1);
			IRQ1, IRQ2 : out std_logic
			);
	end component;

	signal y : std_logic_vector(25 downto 1);
	signal x : std_logic_vector(10 downto 1);
	signal nclk : std_logic;
	signal rl, rh : std_logic_vector(N-1 downto 0);

begin
	nclk <= not clk;

	dd1 : control_unit port map (clk=>clk, rst=>rst, x=>x, y=>y, COP=>COP);

	dd2 : operational_unit port map (
		clk => nclk, rst => rst, d1 => d1, d2 => d2,
		y => y, x => x, rl => rl, rh => rh,
		IRQ1 => IRQ1, IRQ2 => IRQ2
		);

	r <= rh & rl;

end top;
```

Листинг Г.4 – VHDL модель дешифратора управляющих сигналов y

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity decoder is
	generic(
		N: integer := 4
		);
	port(D: in std_logic_vector (N-1 downto 0);
		En: in std_logic;
		Q: out std_logic_vector (2**N -1 downto 0)
		);
end entity;

architecture decoder of decoder is
begin
	process(D, En) is
		variable vQ: std_logic_vector(2**N - 1 downto 0);
	begin
		vQ := (others => '0');

		if (En = '0') then -- Èíâåðñíûé En
			vQ(conv_integer(D)) := '1';
		end if;

		Q <= vQ;
	end process;
end architecture;
```

Листинг Г.5 – VHDL модель мультиплексора сигналов X

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity mx is
	generic(
		N: integer := 4
		);
	port(
		A: in std_logic_vector(N-1 downto 0);
		D: in std_logic_vector(2**N-1 downto 0);
		En: in std_logic;
		Q: out std_logic
		);
end entity;

architecture mx of mx is
signal index: integer := 0;
begin
	index <= conv_integer(A);
	Q <= d(index) when En = '0'
	else '0';
end architecture;
```