EDA Technology

Chapter 9

16/32-bit CPU Innovation Design

Architecture and Characteristics of KX9016

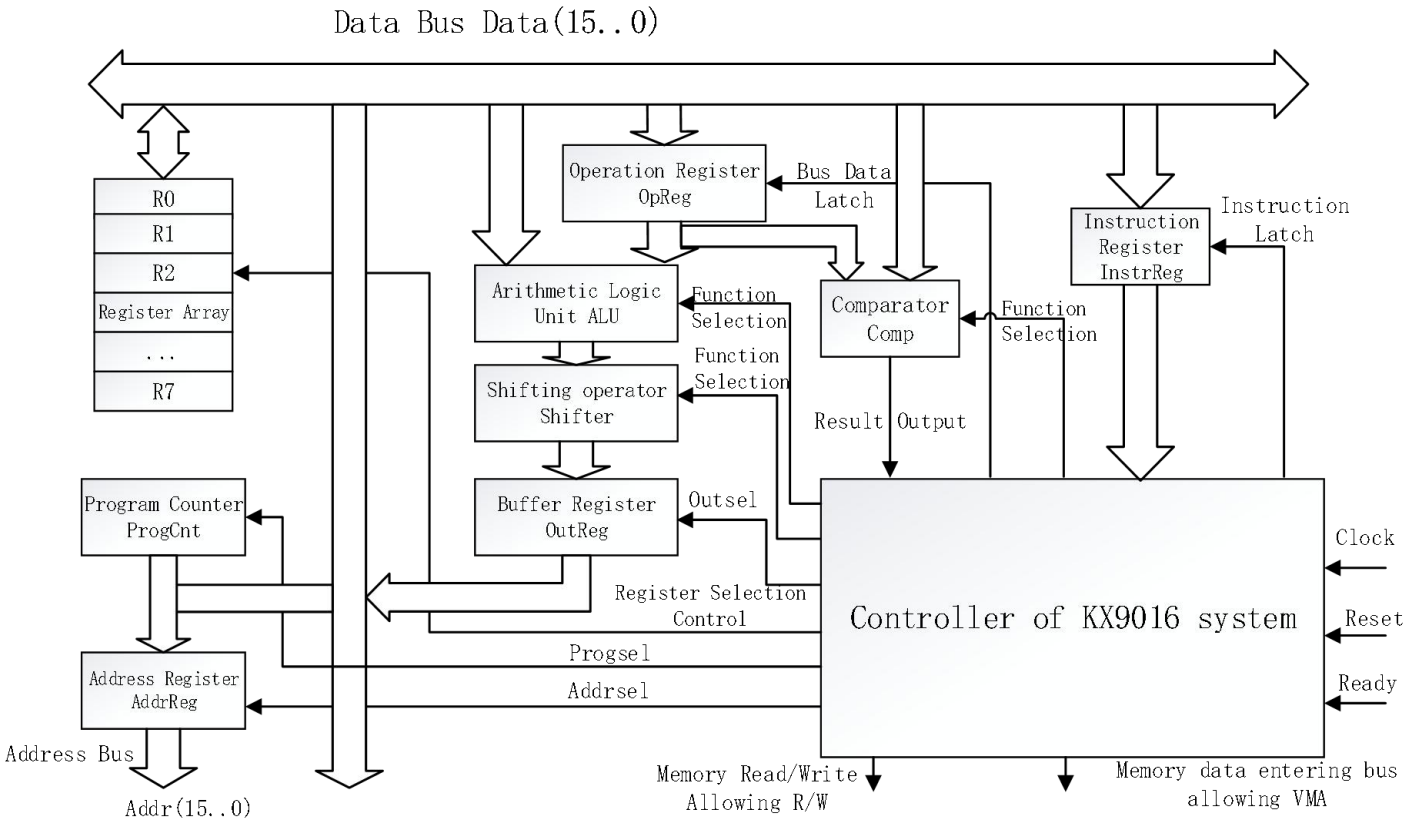

Figure: The top-level structural diagram of 16-bit KX9016v1 CPU

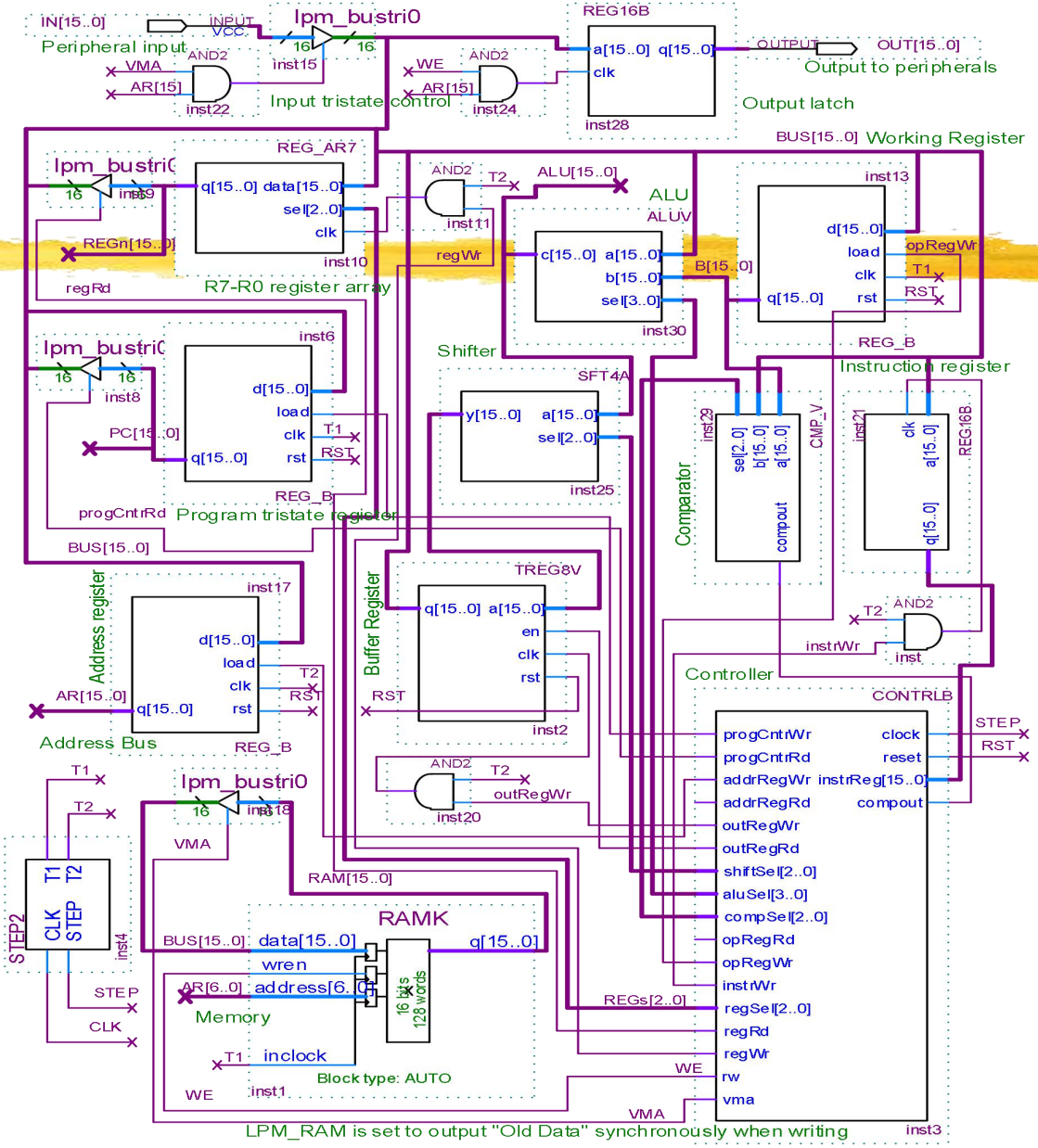

Figure: The structural diagram of the top-level circuit of KX9016v1

Design of KX9016 Basic Hardware System

Module of One-step Beat Generation

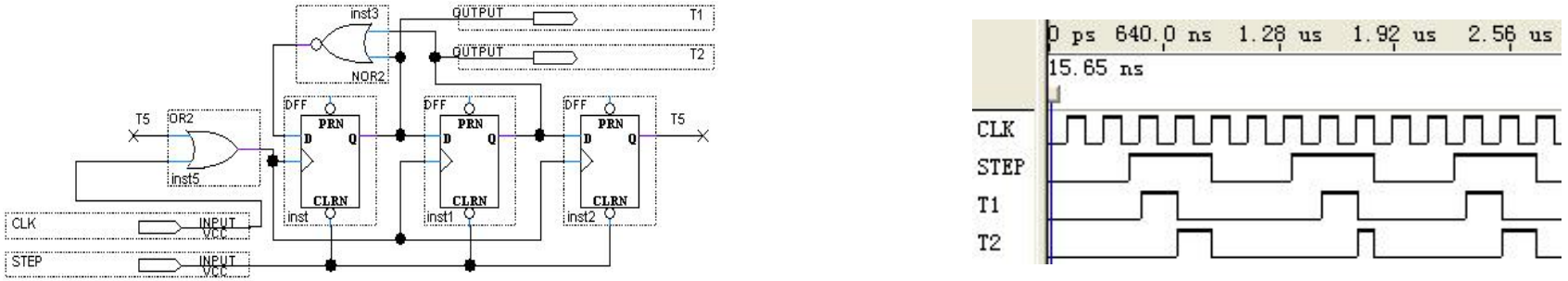

Figure: The circuit and its simulation waveform of beat pulse generator STEP2

ALU Module

```
module ALUV (input[15:0] a, b, input[3:0] sel, output reg [15:0] c);
  parameter  alupass=0, andOp=1, orOp=2, notOp=3, xorOp=4, plus=5,
 alusub=6,  inc=7, dec=8, zero=9;
  always @(a or b or sel)
    case (sel)
      alupass : c <= a ;          //Bus data is directly connected to ALU
       andOp : c <= a & b ;       //Logic AND operation
        orOp : c <= a | b ;       //Logic OR operation
       xorOp : c <= a ^ b ;       //Logic XOR operation
       notOp : c <= ~a ;          //Inverse operation
        plus : c <= a + b ;       //Arithmetic addition operation
      alusub : c <= a - b ;       //Arithmetic subtraction operation
         inc : c <= a + 1 ;       //Adding 1 operation
         dec : c <= a - 1 ;       //Subtracting 1 operation
        zero : c <= 0 ;           //Clearing the output
     default : c <= 0 ;
  endcase          endmodule
```

Comparator Module

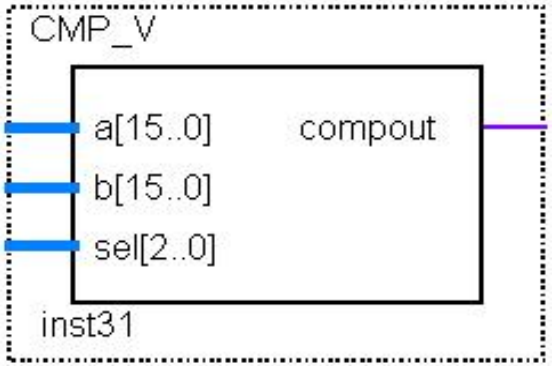

Figure: The module symbol of the comparator

```
module CMP_V (input[15:0] a, b,  input[2:0] sel,  output reg compout);
  parameter eq=0, neq=1, gt=2, gte=3, lt=4, lte =5;
  always @(a or b or sel)
    case (sel)
       eq : if (a==b)  compout<=1; else  compout<=0; // If a equals b, the output is 1
                                                     //otherwise the output is 0
       neq : if (a!=b)  compout<=1; else  compout<=0; // If a is not equal to b, the
                                                       //output is 1
        gt : if (a>b)   compout<=1; else  compout<=0; // If a is greater than b, the
                                                      //output is 1
       gte : if (a>=b)  compout<=1; else  compout<=0; //If a is not less than b, the
                                                       //output is 1
        lt : if (a<b)   compout<=1; else  compout<=0; //If a is less than b, the output
                                                      //is 1
       lte : if (a<=b)  compout<=1; else  compout<=0; //If a is not greater than b,
                                                       //the output is 1
      default :        compout<=0;
  endcase          endmodule
```

Basic Register and Register Array

Basic register

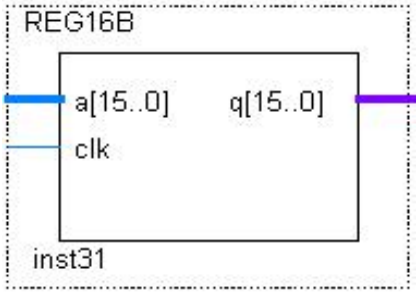

Figure: Basic register

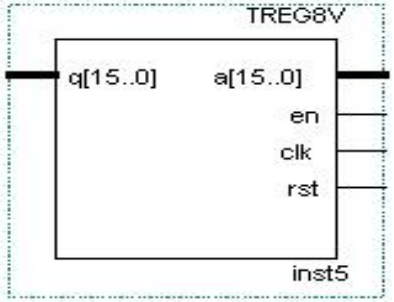

Figure: Register containing three-state gate

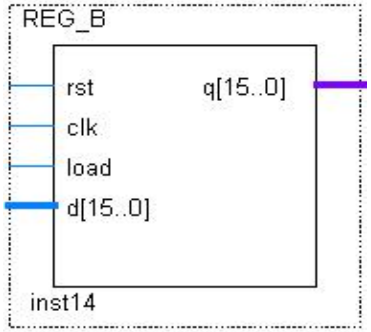

Figure: The register containing load enabling

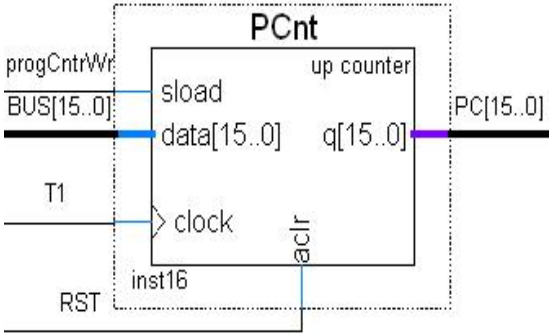

Figure: The alternative circuit of PC

```
module REG16B (input[15:0] a, input clk, output reg [15:0] q);
always @(posedge clk)      q <= a ;    endmodule
```

```
module TREG8V (a, en, clk,rst, q);
  input rst,en, clk; input[15:0] a;   output[15:0] q;  reg[15:0] q, val;
   always @(posedge clk or posedge rst)
     if (rst==1'b1)  val <= {16{1'b0}} ;  else  val <= a ;
   always @(en or val)
     if (en == 1'b1)  q <= val ;  else  q <= 16'bZZZZZZZZZZZZZZZZ ;
endmodule
```

REG_B.v

```
   module REG_B (input rst, clk, load, input[15:0]d, output reg [15:0] q);
       always @(posedge clk or posedge rst)
         if (rst==1'b1)  q <= {16{1'b0}} ;    else
       begin   if (load == 1'b1)    q <= d ;   end          endmodule
```

Register array

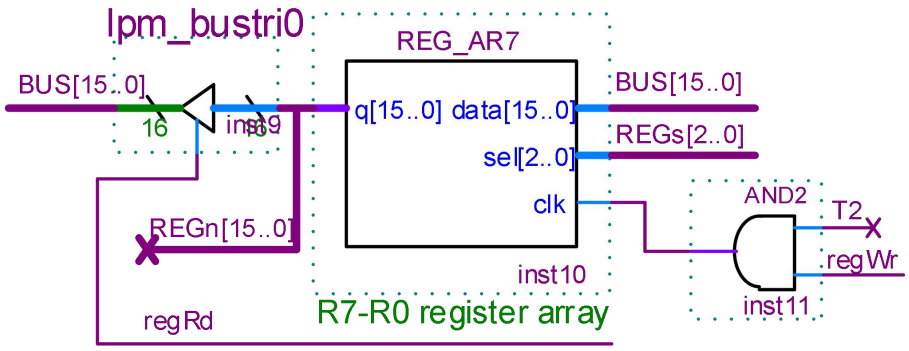

Figure: The register array element and the circuit of three-state control gate

```
module REG_AR7 (data, sel, clk, q);
   input[15:0] data;  input[2:0] sel; input clk;   output[15:0] q;
   reg[15:0] ramdata[0:7];
   always @(posedge clk)    ramdata[sel] <= data;
assign q = ramdata[sel] ;       endmodule
```

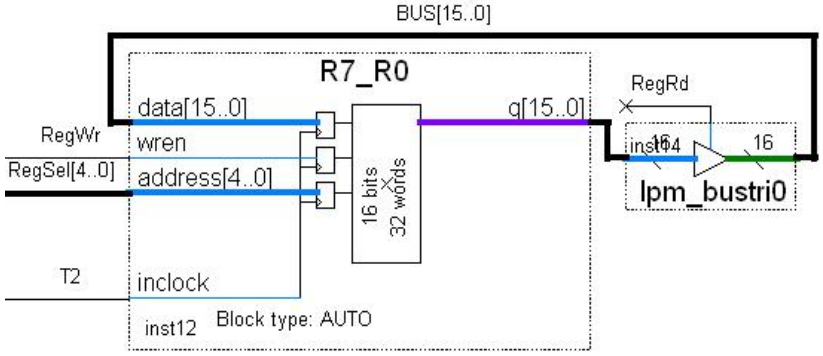

Figure: A circuit that uses LPM_RAM to replace a register array

Shifting Register Module

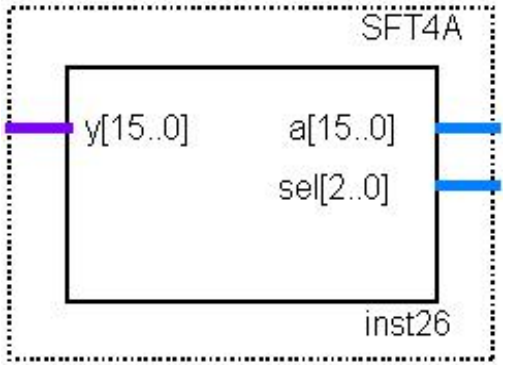

Figure: The shifter symbol

```
module SFT4A (input[15:0] a, input[2:0] sel, output reg[15:0] y);
   parameter shftpass=0, sftl=1, sftr=2, rotl=3, rotr=4;
  always @(a or sel)
   case (sel)
  shftpass :  y<=a ;                  //Data direct passing
     sftl :  y<={a[14:0], 1'b0};      //left shift
     sftr :  y<={1'b0, a[15:1]};      //right shift
     rotl :  y<={a[14:0], a[15]};     // Circulatory left shift
     rotr :  y<={a[0], a[15:1]};      // Circulatory right shift
   default :  y<=0 ;
endcase          endmodule
```

Design of KX9016v1 Instruction System

Instruction Format

Table **Single word instruction format**

Operation code							Source operation code 2			Source operation code 1			Destination operand		
Opcode							SRC			SRC			DST		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table Double word instruction format

Operation code													Destination operand		
Opcode													DST		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit operand															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table Double word instruction

Operation code						Destination operand		
0	0	1	0	0		0	0	1

0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1
0				0				1				5			

Instruction Operation Code

Table KX9016 preset instructions and their function table

Operation code	Instruction	Function	Operation code	Instruction	Function
00000	NOP	Null operation	01111	IN	Peripheral data output instruction
00001	LD	Loading data to registers	10000	JMPLTI	If less than, transferring to an immediate number address
00010	STA	Storing the numbers of registers into memory	10001	JMPGT	If greater than, transferring
00011	MOV	Transferring operands between registers	10010	OUT	Data output instruction
00100	LDR	Loading the immediate number into the register	10011	MTAD	16-bit multiplicative accumulation
00101	JMPI	Transferring to the address specified by an immediate number	10100	MULT	16-bit multiplication
00110	JMPGTI	If greater than, transferring to the immediate number address	10101	JMP	Unconditional transferring
00111	INC	After adding 1, putting back to the register	10110	JMPEQ	If equal, transferring
01000	DEC	After subtracting 1, putting back to the register	10111	JMPEQI	If equal, transferring to an immediate number address
01001	AND	AND operation between two registers	11000	DIV	32-bit Division
01010	OR	OR operation between two registers	11001	JMPLTE	If not greater than, transferring
01011	XOR	XOR operation between two registers	11010	SHL	Left logical shift
01100	NOT	NOT operation for the register	11011	SHR	Right logical shift
01101	ADD	Addition operation for two registers	11100	ROTR	Circular right shift
01110	SUB	Subtraction operation for two registers	11101	ROTL	Circular left shift

Table Sample program

Instruction	Machine code	Word length	Operation code	Idle code	Source operand	Destination operand	Functional description
LDR R1, 0025H	2001H 0025H	2	00100 0000 0000 0010 0101	xxxxx	xxx	001	The immediate number 0025H is sent to R1
LDR R2, 0047H	2002H 0047H	2	00100 0000 0000 0100 0111	xxxxx	xxx	010	The immediate number 0047H is sent to R2
LDR R6, 0036H	2006H 0036H	2	00100 0000 0000 0011 0110	xxxxx	xxx	110	The immediate number 0036H is sent to R6
LD R3, [R1]	080BH	1	00001	xxxxx	001	011	Taking the number from the RAM storage unit specified by R1 to R3
STA [R2], R3	101AH	1	00010	xxxxx	011	010	Saving the content of R3 into the RAM unit specified by R2
JMPGTI 0000H	300EH 0000H	2	00110 0000000000000000	xxxxx	001	110	If R1>R6,transferring to address [0000H]
INC R1	3801H	1	00111	xxxxx	xxx	010	R1+1 ? R1
INC R2	3802H	1	00111	xxxxx	xxx	010	R2+1 ? R2
JMPI 0006H	2800H 0006H	2	00101	xxxxx	xxx	xxx	Absolute address transferring instruction: transferring to address 0006H

Example of Software Program Design

Table The example of the assembler program of 7 instructions

Address	Operation code	Instruction	Functional description
0000H 0001H	2001H 0032H	LDR R1, 0032H	Sending the immediate number of 0032H to the register R1
0002H 0003H	2002H 0011H	LDR R2, 0011H	Sending the immediate number of 0011H to the register R2
0004H	680AH	ADD R1, R2, R3	Sending the addition of the contents of the registers R1 and R2 to R3
0005H	1819H	MOV R1, R3	Sending the contents of the register R3 into the R1
0006H	3802H	INC R2	R2+1→R2
0007H	101AH	STA [R2], R3	Saving the content of the R3 into the RAM unit of the R2 specified address
0008H	080BH	LD R3, [R1]	Sending the data of the RAM unit of the R1 specified address to R3
0009H	0000H	NOP	Null operation

Table The contents of the memory initialization file RAM_16.mif

WIDTH = 16;	03 :0011;	0B :0000;	13 :0000;
DEPTH = 256;	04 :680A;	0C :0000;	...
ADDRESS_RADIX = HEX;	05 :1819;	0D :0000;	41 :0000;
DATA_RADIX = HEX;	06 :3802;	0E :0000;	42 :0000;
CONTENT BEGIN	07 :101A;	0F :0000;	43 :A6C7;
00 :2001;	08 :080B;	10 :0000;	…
01 :0032;	09 :0000;	11 :0000;	4F :0000;
02 :2002;	0A :0000;	12 :1524;	END;

Design of KX9016 Controller

```
module CONTRLB (clock, reset, instrReg, compout, progCntrWr, progCntrRd,
  addrRegWr, addrRegRd, outRegWr, outRegRd, shiftSel, aluSel, compSel,
  opRegRd, opRegWr, instrWr, regSel, regRd, regWr, rw, vma);
    input clock;    input reset;                  //Clock and reset signal
     input[15:0] instrReg; input compout;// The input of instruction register operation
                                          //code and input of comparator result
    output progCntrWr;             // Program register synchronous loading allowing,
                                    //but the rising edge of T1 is required to be
                                    //effective.
    output progCntrRd;             // Program register data is output to bus three-state
                                    //switch allowing control
     output addrRegWr;            // The address register allows bus data to lock in, but
                                  //it needs T2 to be effective
    output addrRegRd;             // Address register read bus allowing
     output outRegWr;             // Output registers allow bus data to be written, but
                                  //it needs T2 to be effective
     output outRegRd;             // The allowing of the output register data to enter
   the                            //bus, that is, to open the three-state gate
     output[2:0] shiftSel; output[3:0] aluSel; // Function selection of shifter and
                                               //ALU
     output[2:0] compSel;  output opRegRd; // Function selection of comparator and
                                           //readout allowing of working register
     output opRegWr;              // Bus data is allowed to be locked into the work
                                  //registers,but T1 is required to be effective
     output instrWr;              //Bus data is allowed to be locked into instruction
                                  //register, but T2 is required to be effective
    output[2:0] regSel;           //Register array selection
     output regRd;               // Register array data is output to bus three-state switch
                                 //allowing control
     output regWr;                // Data on bus is allowed to write to register array,
                                  //but it needs T2 to be effective.
    output rw;                   //rw=1, RAM write allowing; rw=0, RAM read allowing
     output vma;                 // Memory RAM data is output to bus three-state switch
                                 //allowing control
     reg progCntrWr, progCntrRd, addrRegWr, addrRegRd, outRegWr, outRegRd,
         opRegRd, opRegWr, instrWr, regRd, vma, regWr, rw;
     reg[2:0] shiftSel, regSel;  wire[2:0] compSel;   reg[3:0] aluSel;
     parameter shftpass=0,  alupass=0, zero=9, inc=7, plus=5;//Seeing example 8-1
                                                          //for reference
     parameter reset1=0, reset2=1, reset3=2, execute=3, nop=4, load=5, store=6,
    load2=7, load3=8, load4=9, store2=10, store3=11, store4=12,incPc=13,
    incPc2=14,incPc3=15,loadI2=19,loadI3=20,loadI4=21,loadI5=22,loadI6=23, inc2=24
    inc3=25, inc4=26,move1=27,move2=28,add2=29,add3=30,add4=31;
          //Three additional state variable elements for microoperation, namely add2,
          //add3, add4, are added to the state machine
     reg[4:0] current_state, next_state;      // Definition of current and next
                                              //state variables
always @(current_state or instrReg or compout)   begin :COM//Combinational process
   progCntrWr<=0; progCntrRd<=0; addrRegWr<=0; addrRegRd<=0; outRegWr<=0;
   outRegRd<=0; shiftSel<=shftpass; aluSel<=alupass;  opRegRd<=0; opRegWr<=0;
   instrWr<=0; regSel<=0;  regRd<=0;  regWr<=0; rw<=0; vma<=0;
      case (current_state)
       reset1 :  begin  aluSel<=zero; shiftSel<=shftpass;
                 outRegWr<=1'b1; next_state<=reset2;    end
       reset2 :  begin  outRegRd<=1'b1; progCntrWr<=1'b1;
                  addrRegWr<=1'b1; next_state<=reset3;   end
       reset3 :  begin  vma<=1'b1; rw<=1'b0; instrWr<=1'b1;
                  next_state<=execute;                   end
```

Timing Simulation and Hardware Testing of KX9016

Timing Simulation and Waveform Analysis of Instruction Execution

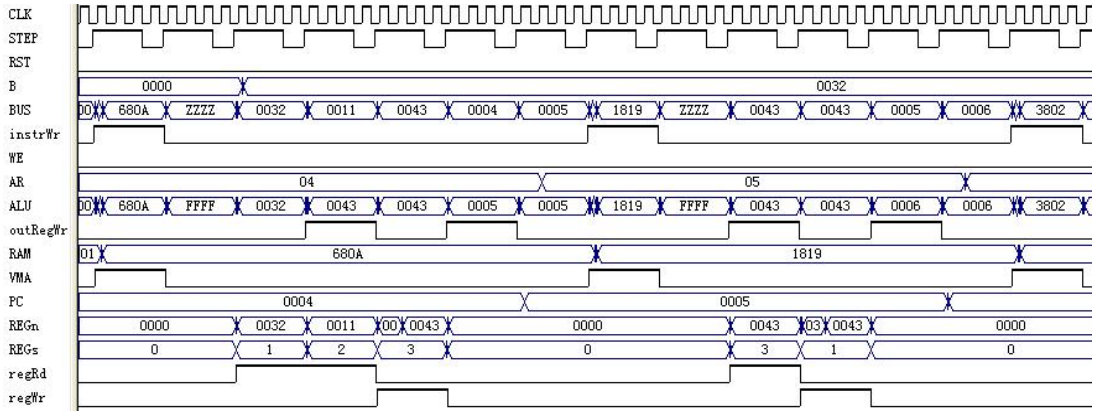

Figure: KX9016 simulation waveform, which includes the timing of ADD command and MOV command

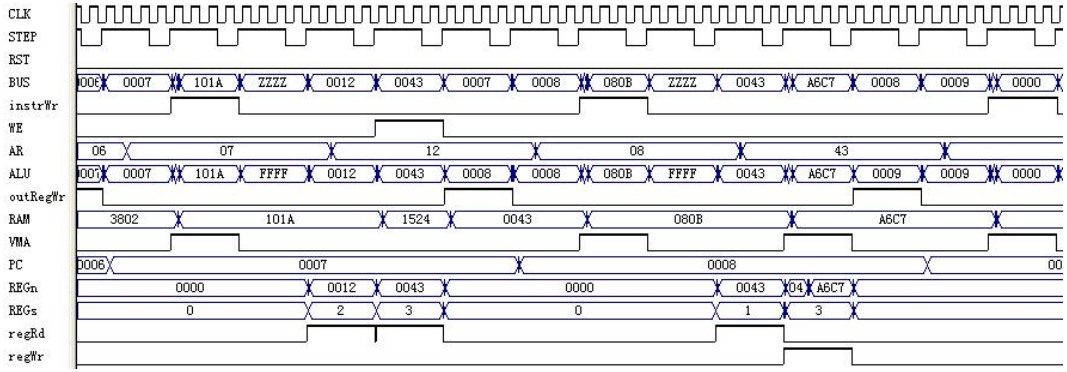

Figure: KX9016 simulation waveform, which includes the timing of STA command and LD command

Hardware Testing of CPU Operation Condition

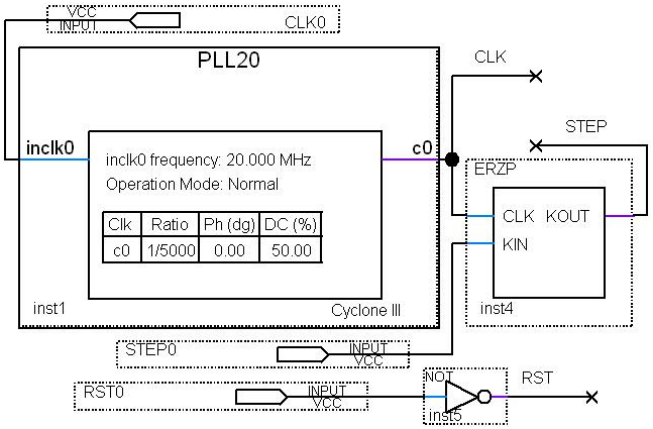

Figure: The circuit scheme for adding phase-locked loop

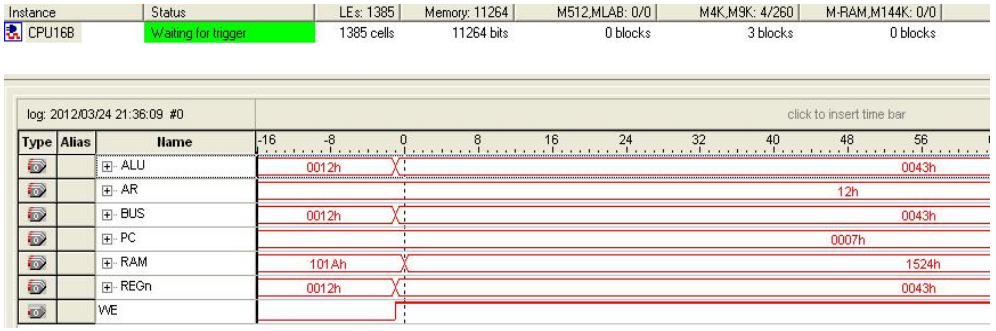

Figure: SignalTap II executes real-time test waveform for KX9016 to write data instructions from RAM

0 RAM8:

000000	20	01	00	32	20	02	00	11	68	0A	18	19	38	02	10	1A	08	0B	00	00	00	00	00	00	00	00	00	00
000010	00	00	00	00	00	43	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	B3	B4	B5	B6	B7	B8	D1	D2	D3	D4
000020	D5	D6	D7	D8	D9	DA	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	20	0D	20	0E	20	0F	20	10	20	11
000030	D5	D6	D7	D8	D1	23	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	00	00	00	00	00	00	00	00	00	00
000040	00	00	00	00	00	00	A6	C7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure: The testing of the change of RAM data in KX9016 by In-System Memory Content Editor

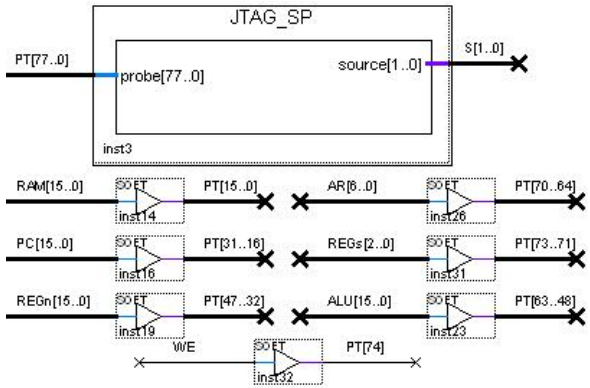

Figure: The connection of the S/P module to the KX9016 port

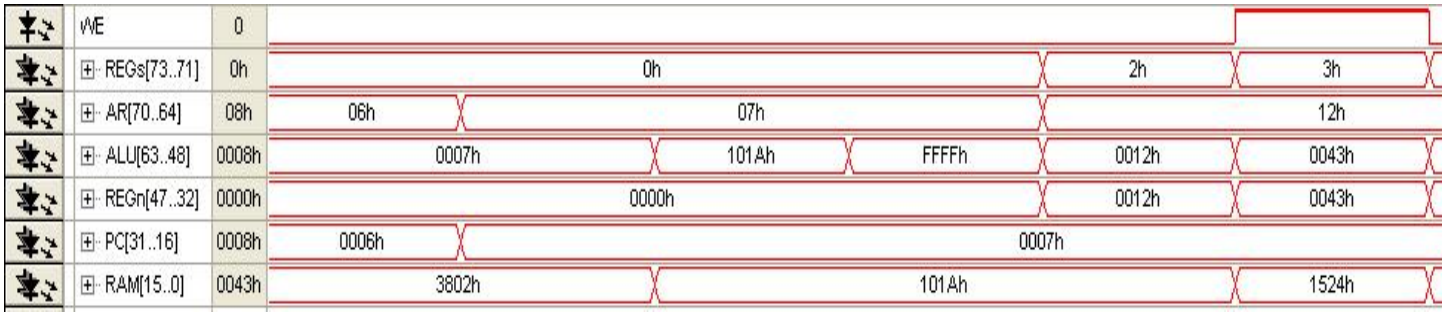

Figure: The real-time testing waveform of the in-system S/P module executing STA data writing instructions from RAM for KX9016

Examples of Application Program Design and System Optimization of KX9016

Multiplication Algorithm and Its Hardware Implementation

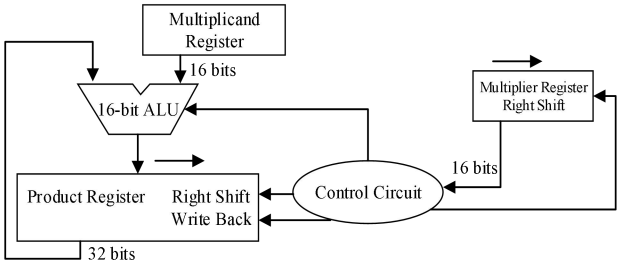

Figure: The hardware implementation of multiplication algorithm 1

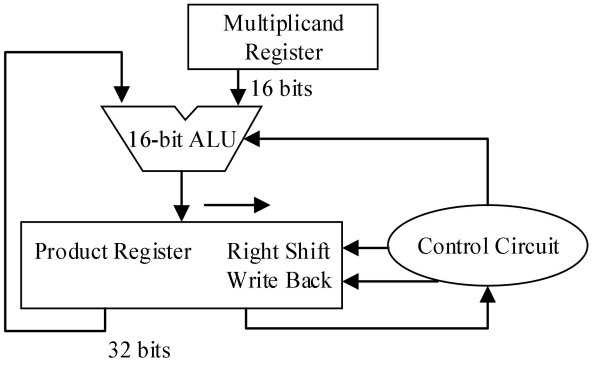

Figure: The hardware implementation of the improved multiplication algorithm 2

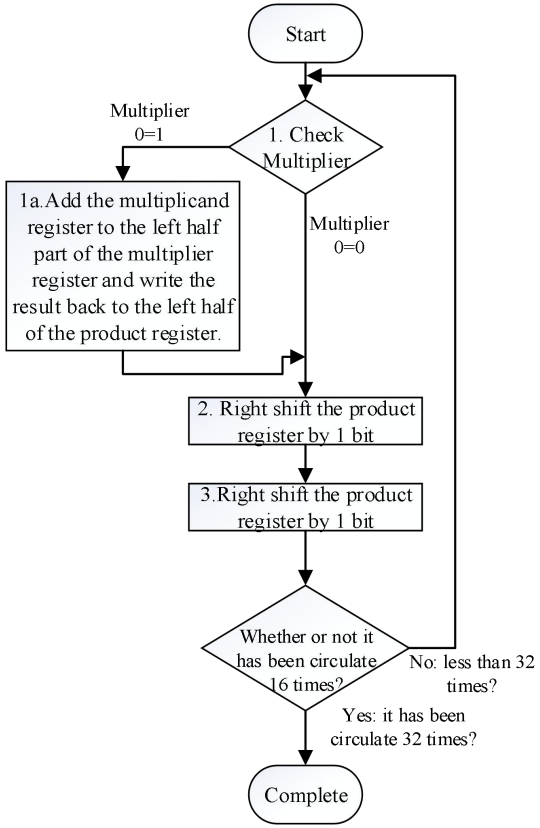

Figure: The flow chart of the multiplication algorithm 1

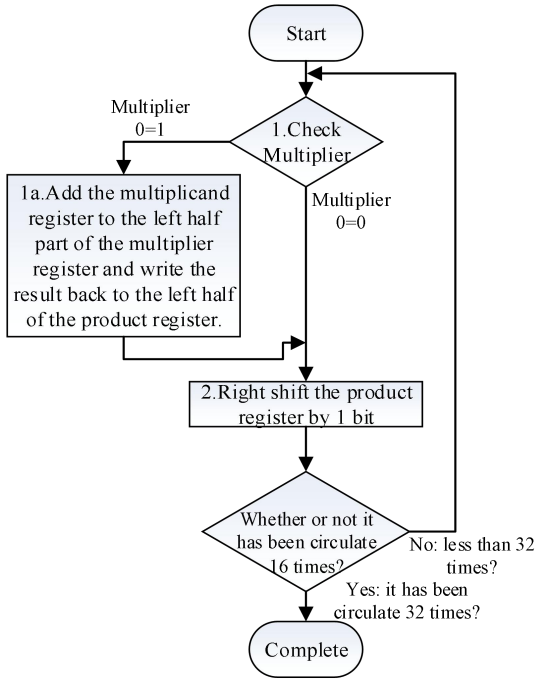

Figure: The flow chart of the multiplication algorithm 2

Division Algorithm and Its Hardware Implementation

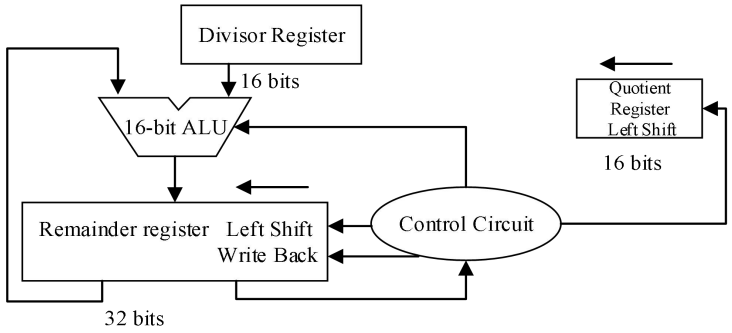

Figure: The hardware structure of division algorithm 1

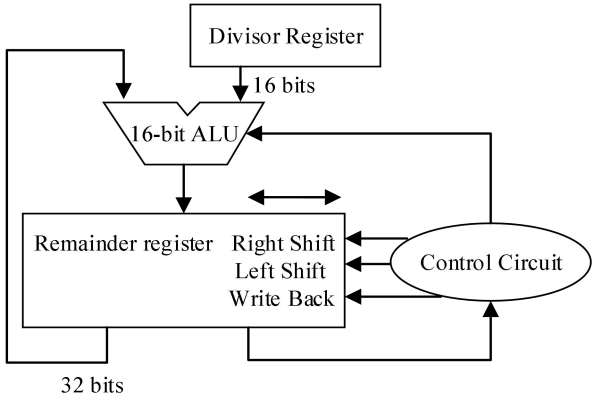

Figure: The hardware structure of division algorithm 2

Optimization of KX9016v1 Hardware System

Design of 32-bit RISC-V Processor

RISC-V Basic Structure and Basic Integer Instruction Set RV32I

Table Types and formats of RV32I instructions

Format	Meaning	31~26	25~20	19~15	14~12	11~7	6~0
R-type	Register	funct7	rs2	rs1	funct3	rd	opcode
I-type	Immediate	imm[11:0]		rs1	funct3	rd	opcode
S-type	Store	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B-type	Branch	imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1] imm[11]	opcode
U-type	Upper Immediate	imm[31:12]				rd	opcode
J-type	Jump	imm[20] imm[10:1]	imm[11]	imm[19:12]		rd	opcode

Table RV32I general Register

Index	Name	Alias	Function
R0	x0	zero	Hardwired zero
R1	x1	ra	Return address
R2	x2	sp	Stack pointer
R3	x3	gp	Global pointer
R4	x4	tp	Thread pointer
R5~R7	x5~x7	t0~t2	Temporary
R8	x8	s0/fp	Saved register, frame pointer
R9	x9	s1	Saved register
R10~R11	x10~x11	a0~a1	Function argument, return value
R12~R17	x12 ~x17	a2~a7	Function argument
R18~R27	x18 ~x27	s2~s11	Saved register
R28~R31	x28~x31	t3~t6	Temporary

Table RV32I basic integer instruction set

RV32I Base	Format	Opcode	funct7	funct3	Other	Function
SLL rd,rs1,rs2	R	0110011	0000000	001	--	Shift Left Logical
SLLI rd,rs1,shamt	I	0010011	0000000	001	--	Shift Left Log. Imm.
SRL rd,rs1,rs2	R	0110011	0000000	101	--	Shift Right Logical
SRLI rd,rs1,shamt	I	0010011	0000000	101	--	Shift Right Log. Imm.
SRA rd,rs1,rs2	R	0110011	0100000	101	--	Shift Right Arithmetic
SRAI rs,rs1,shamt	I	0010011	0100000	101	--	Shift Right Arith. Imm.
ADD rd,rs1,rs2	R	0110011	0000000	000	--	ADD
ADDI rd,rs1,imm	I	0010011	--	000	--	ADD Immediate
SUB rd,rs1,rs2	R	0110011	0100000	000	--	SUBtract
LUI rd,imm	U	0110111	--	--	--	Load Upper Imm
AUIPC rd,imm	U	0010111	--	--	--	Add Upper Imm to PC
XOR rd,rs1,rs2	R	0110011	0000000	100	--	XOR
XORI rd,rs1,imm	I	0010011	--	100	--	XOR Immediate
OR rd,rs1,rs2	R	0110011	0000000	110	--	OR
ORI rd,rs1,imm	I	0010011	--	110	--	OR Immediate
AND rd,rs1,rs2	R	0110011	0000000	111	--	AND
ANDI rd,rs1,imm	I	0010011	--	111	--	AND Immediate
SLT rd,rs1,rs2	R	0110011	0000000	010	--	Set <
SLTI rd,rs1,imm	I	0010011	--	010	--	Set < Immediate
SLTU rd,rs1,rs2	R	0110011	0000000	011	--	Set < Unsigned
SLTIU rd,rs1,imm	I	0010011	--	011	--	Set < Imm Unsigned
BEQ rs1,rs2,imm	B	1100011	--	000	--	Branch =
BNE rs1,rs2,imm	B	1100011	--	001	--	Branch ?
BLT rs1,rs2,imm	B	1100011	--	100	--	Branch <
BGE rs1,rs2,imm	B	1100011	--	101	--	Branch ≥
BLTU rs1,rs2,imm	B	1100011	--	110	--	Branch < Unsigned
BGEU rs1,rs2,imm	B	1100011	--	111	--	Branch = Unsigned
JAL rd,imm	J	1101111	--	--	--	J&L
JALR rd,rs1,imm	I	1100111	--	000	--	Jump & Link Register
FENCE	I	0001111	--	000	0	Synch thread
FENCE.I	I	0001111	--	001	1	Synch Instr & Data
ECALL	I	1110011	--	000	0	CALL
EBREAK	I	1110011	--	000	1	BREAK

32-bit Multiplication Instruction Set RV32M

Table RV32M integer multiplication and division instruction set

Category	RV32M(Multiply-Divide)	Format	Opcode funct	Fuction
Multiply	MUL rd,rs1,rs2	R		MULtiply
	MULH rd,rs1,rs2	R		MULtiply High
	MULHSU rd,rs1,rs2	R		MULtiply High Sign/Uns
	MULHU rd,rs1,rs2	R		MULtiply High Uns
Divide	DIV rd,rs1,rs2	R		DIVide
	DIVU rd,rs1,rs2	R		DIVide Unsigned
Remainder	REM rd,rs1,rs2	R		REMainder
	REMU rd,rs1,rs2	R		REMainder Unsigned

16-bit Compressed Instruction Set RVC

Table Types and Formats of RVC Instructions

Format	Meaning	15~13	12	11	10~6	5	4~2	1~0
CR	Register	funct4		rd/rs1		rs2		op
CI	Immediate	funct3	imm	rd/rs1		imm		op
CSS	Stack-relative Store	funct3	imm			rs2'		op
CIW	Wide Immediate	funct3	imm				rd'	op
CL	Load	funct3	imm		rs1'	imm	rd'	op
CS	Store	funct3	imm		rs1'	imm	rs2'	op
CB	Branch	funct3	offset		rs1'	offset		op
CJ	Jump	funct3	Jump target					op

Table **Part of RVC and the comparison with 32-bit instructions**

RVC		RISC-V equivalent		Funciton
C.LW	rd',rs1',imm	LW	rd',rs1',imm*4	Load Word
C.LWSP	rd,imm	LW	rd,sp,imm*4	Load Word SP
C.SW	rs1',rs2',imm	SW	rs1',rs2',imm*4	Store Word
C.SWSP	rs2,imm	SW	rs2,sp,imm*4	Store Word SP
C.ADD	rd,rs1	ADD	rd,rd,rs1	ADD
C.ANDI	rd,imm	ANDI	rd,rd,imm	AND Immediate
C.OR	rd,rs1	OR	rd,rd,rs1	OR
C.XOR	rd,rs1	AND	rd,rd,rs1	Exclusive OR