# Chapter 14

# Digital design using PicoBlaze

## 14.1 What is PicoBlaze?

PicoBlaze is an efficient 8-bit microcontroller architecture which can be synthesized in Spartan 3FPGAs (and also in other architectures of Xilinx FPGAs). PicoBlaze is similar to many microcontrollers architectures but it is specifically designed and optimized for Xilinx FPGAs. PicoBlaze is typically referred to as soft processor cores since they are synthesized from an HDL and use the programmable logic and routing resources of an FPGA for their implementation, as opposed to a dedicated processor hard core such as the PowerPC that is incorporated in some Xilinx's families.

The PicoBlaze design was originally named **KCPSM** which stands for **Constant(K) Coded Programmable State Machine** (formerly "Ken Chapman's PSM"). **Ken Chapman** is the Xilinx systems designer who devised and implemented the microcontroller.

## 14.2 Microprocessor Vs Microcontroller

Microprocessor is an IC which has only the CPU inside them i.e. only the processing powers such as Intel' Pentium 1,2,3,4, core 2 duo, i3, i5 etc. T hese microprocessors don't have RAM, ROM, and other peripheral on the chip. A system designer has to add them externally to make them functional. Application of microprocessor includes Desktop PC's, Laptops, notepads etc.

But this is not the case with Microcontrollers. Microcontroller has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals all embedded on a single chip. At times it is also termed as a mini computer or a computer on a single chip.

**Microprocessor Vs Micro Controller**

| Microprocessor | Micro Controller |
|---|---|
| Microprocessor is heart of Compute system. | Micro Controller is a heart of embedded system. |
| It is just a processor. Memory and I/O components have to be connected externally | Micro controller has external processor along with internal memory and i/O components |
| Since memory and I/O has to be connected externally, the circuit becomes large. | Since memory and I/O are present internally, the circuit is small. |
| Cannot be used in compact systems and hence inefficient | Can be used in compact systems and hence it is an efficient technique |
| Cost of the entire system increases | Cost of the entire system is low |
| Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries. | Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries. |
| Most of the microprocessors do not have power saving features. | Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further. |
| Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower. | Since components are internal, most of the operations are internal instruction, hence speed is fast. |
| Microprocessor have less number of registers, hence more operations are memory based. | Micro controller have more number of registers, hence the programs are easier to write. |
| Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module | Micro controllers are based on Harvard architecture where program memory and Data memory are separate |
| Mainly used in personal computers | Used mainly in washing machine, MP3 players |

Figure 14.1: Microprocessor Vs Microcontroller

## 14.3 Von Neumann architecture

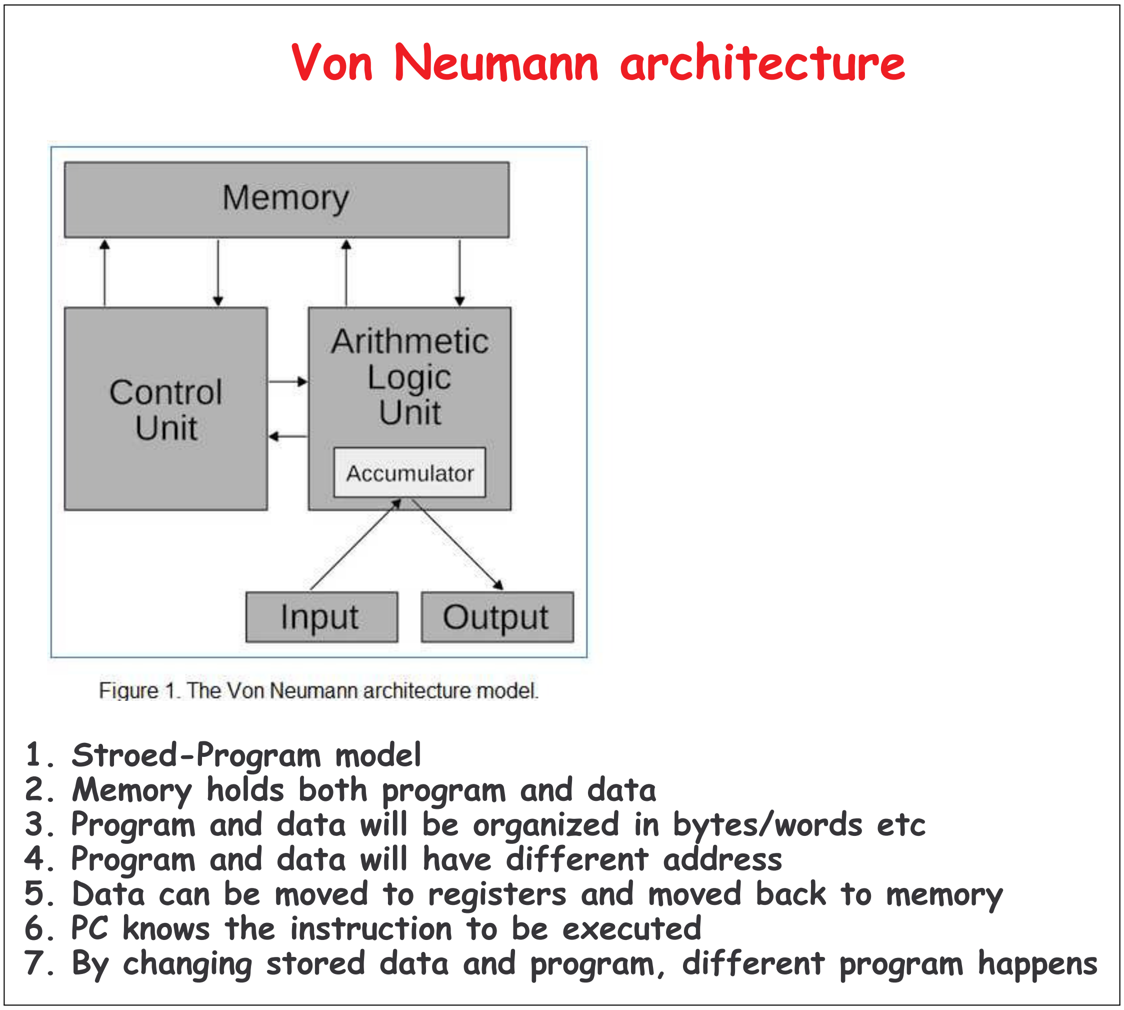



Figure 14.2: Von Neumann architecture

## 14.4 SAP1

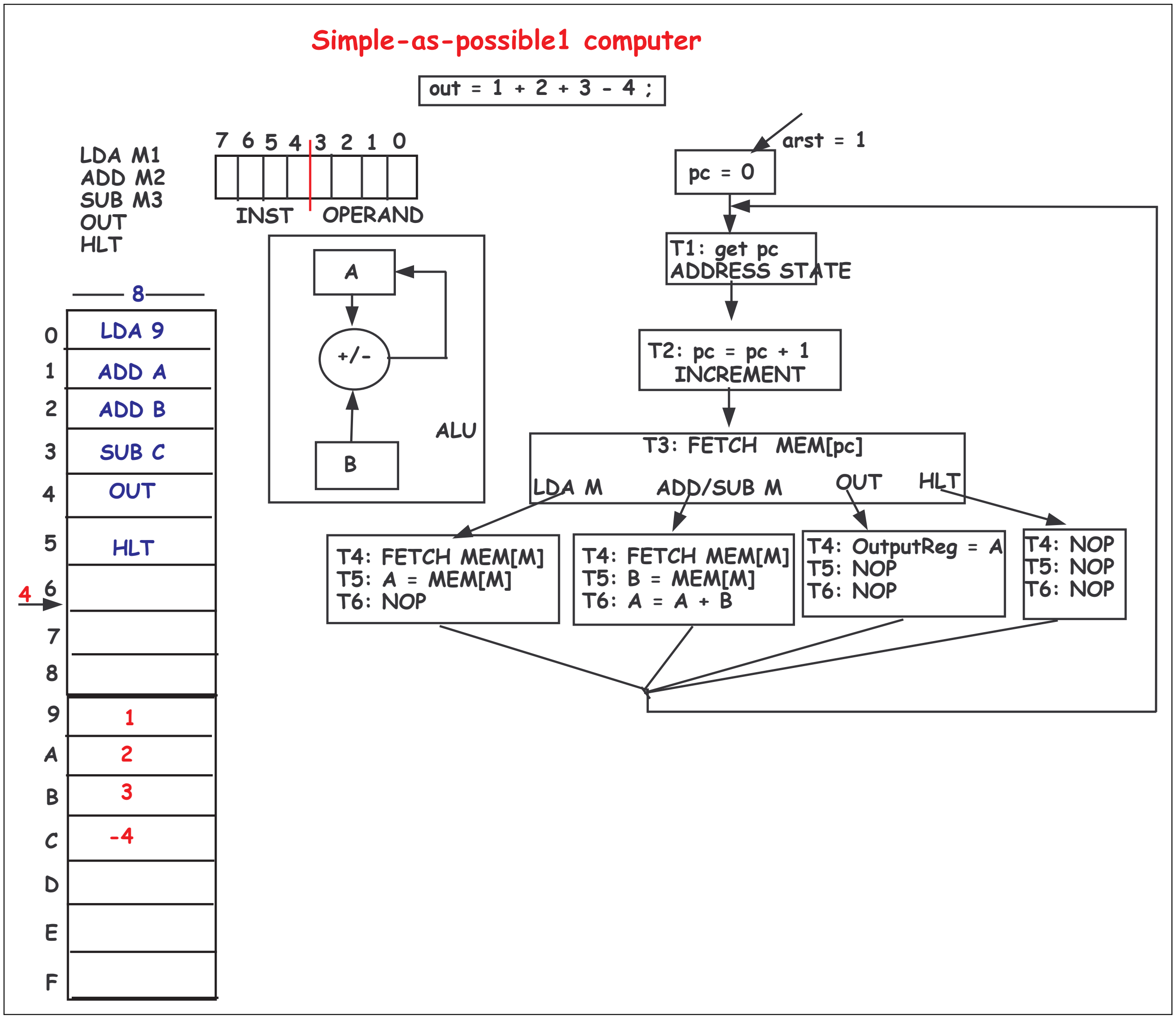



Figure 14.3: SAP1 architecture

## 14.5 Architecture of PicoBlaze

# Architecture of PicoBlaze

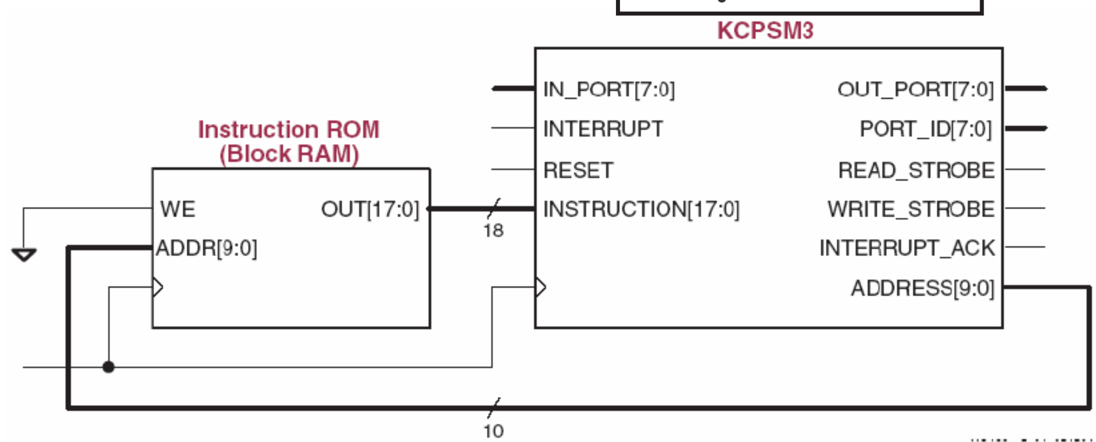
Has two parts:

1. Soft Processor Core: KCPSM3

   KCPSM3 – which stands
   for Ken Chapman Programmable State Machine version 3.

2) the program memory from which instructions are fetched and executed by the processor core

kcpsm3.v




embedded_kcpsm3.v

```
module embedded_kcpsm3(..)
    kcpsm3 processor
    prog_rom program
endmodule
```

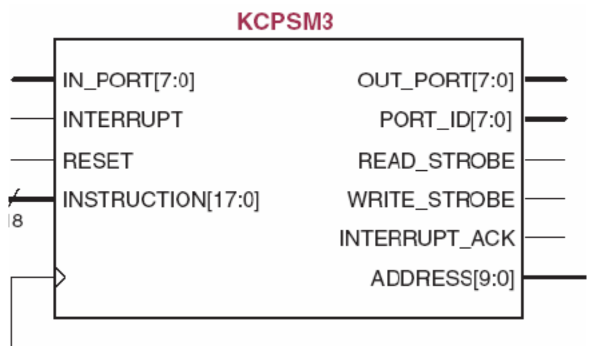
http://www.xilinx.com/ipcenter/processor_central/picoblaze/member/

Figure 14.4: Architecture of PicoBlaze

## 14.6 How big is KCPSM3?

**How Big is KCPSM3?**




Selected Device : xa3s100ecpg132-4

| | | | | |
|---|---|---|---|---|
| Number of Slices: | 100 | out of | 960 | 10% |
| Number of Slice Flip Flops: | 76 | out of | 1920 | 3% |
| Number of 4 input LUTs: | 181 | out of | 1920 | 9% |
| Number used as logic: | 113 | | | |
| Number used as RAMs: | 68 | | | |
| Number of IOs: | 58 | | | |
| Number of bonded IOBs: | 58 | out of | 83 | 69% |
| Number of GCLKs: | 1 | out of | 24 | 4% |

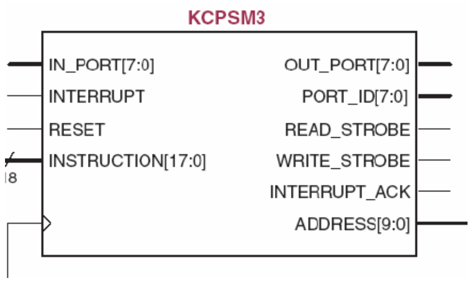
**USES 10% OF THE FPGA RESOURCE**

Figure 14.5: Area of soft processor

## 14.7 How fast is KCPSM3?

## How Fast is KCPSM3?




1. USES 10% OF THE FPGA RESOURCE
2. Requires TWO clock cycles to complete an instruction

If the system clock is 50MHZ, 25 MILLION Instructions can be executed in 1 second.

Delay: 7.147ns (Levels of Logic = 13)
Source: carry_flag_flop (FF)
Destination: pc_loop_register_bit_9 (FF)
Source Clock: clk rising
Destination Clock: clk rising

Data Path: carry_flag_flop to pc_loop_register_bit_9

| Cell:in->out | fanout | Gate Delay | Net Delay | Logical Name (Net Name) |
|---|---|---|---|---|
| FDRE:C->Q | 4 | 0.591 | 0.762 | carry_flag_flop (carry_flag) |
| LUT4:I0->O | 2 | 0.704 | 0.526 | condition_met_lut (condition_met) |
| LUT3:I1->O | 11 | 0.704 | 1.108 | normal_count_lut (normal_count) |
| LUT3:I0->O | 1 | 0.704 | 0.000 | value_select_mux_0 (pc_value<0>) |
| MUXCY:S->O | 1 | 0.464 | 0.000 | pc_value_muxcy_0 (pc_value_carry<0>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_1 (pc_value_carry<1>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_2 (pc_value_carry<2>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_3 (pc_value_carry<3>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_4 (pc_value_carry<4>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_5 (pc_value_carry<5>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_6 (pc_value_carry<6>) |
| MUXCY:CI->O | 1 | 0.059 | 0.000 | pc_value_muxcy_7 (pc_value_carry<7>) |
| MUXCY:CI->O | 0 | 0.059 | 0.000 | pc_value_muxcy_8 (pc_value_carry<8>) |
| XORCY:CI->O | 1 | 0.804 | 0.000 | pc_value_xor_high (inc_pc_value<9>) |
| FDRSE:D | | 0.308 | | pc_loop_register_bit_9 |
| Total | | 7.147ns (4.751ns logic, 2.396ns route) (66.5% logic, 33.5% route) | | |

Figure 14.6: Fmax of soft processor

## 14.8 KCPSM3 + ROM ≡ embedded_kcpsm3

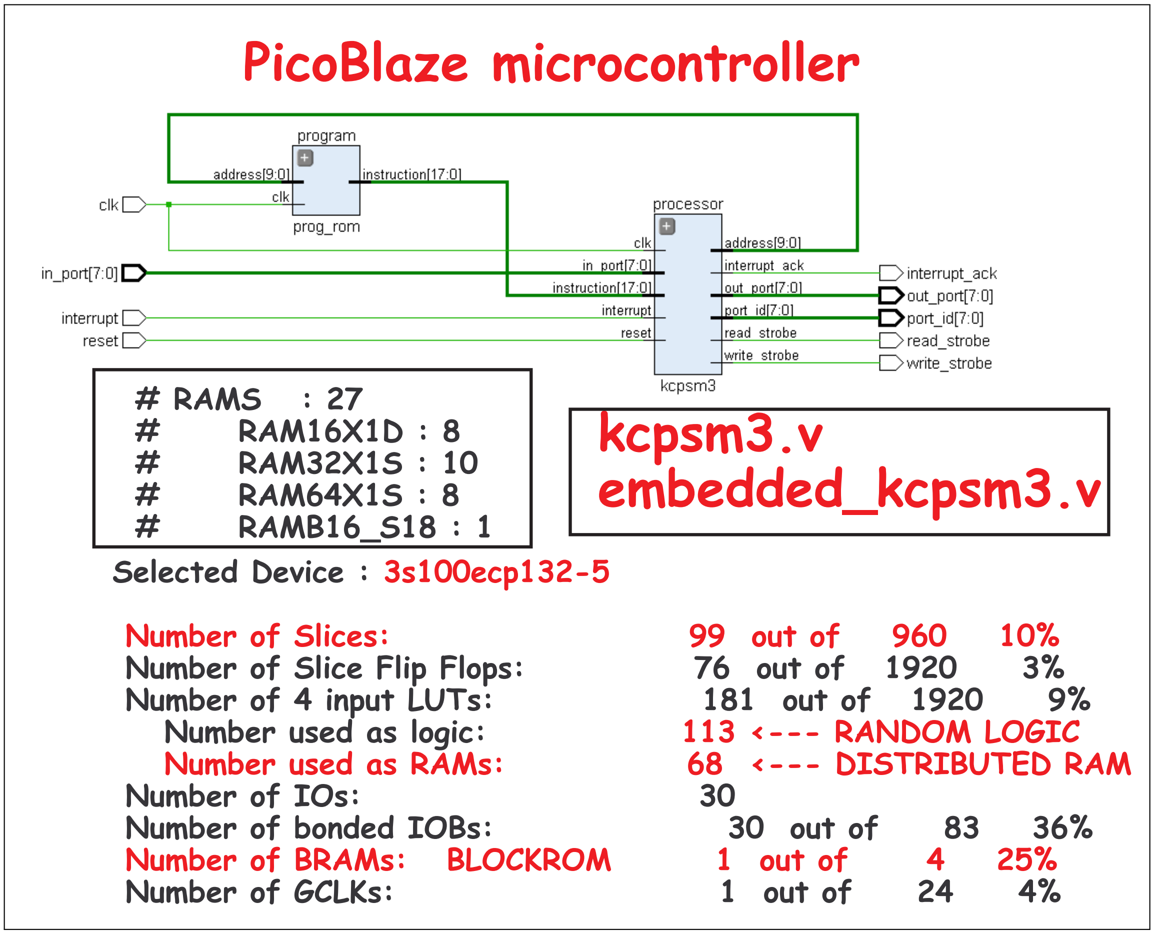



Figure 14.7: embedded_kcpsm3

## 14.9 Software view of architecture of PicoBlaze

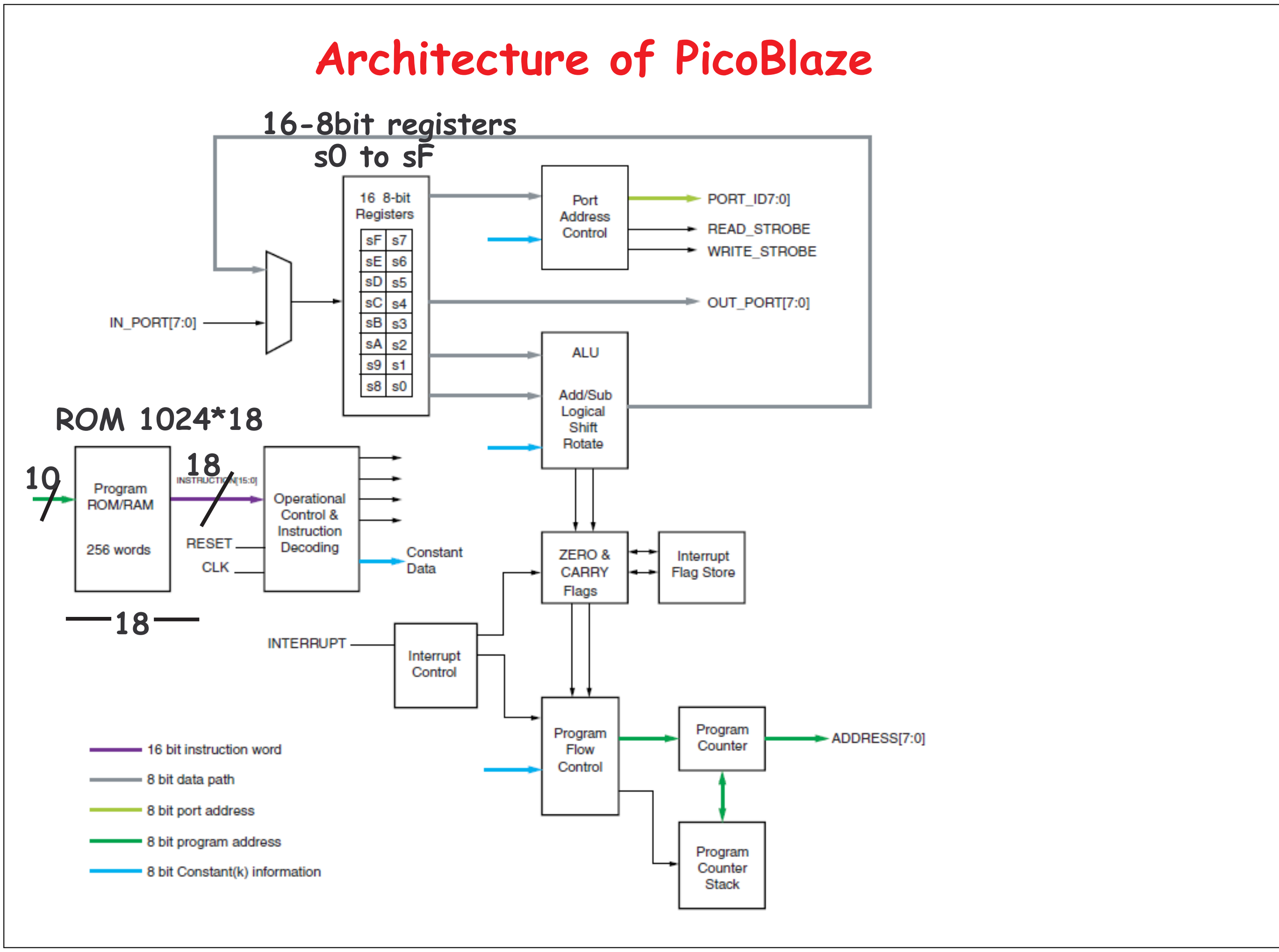



Figure 14.8: Block diagram PicoBlaze

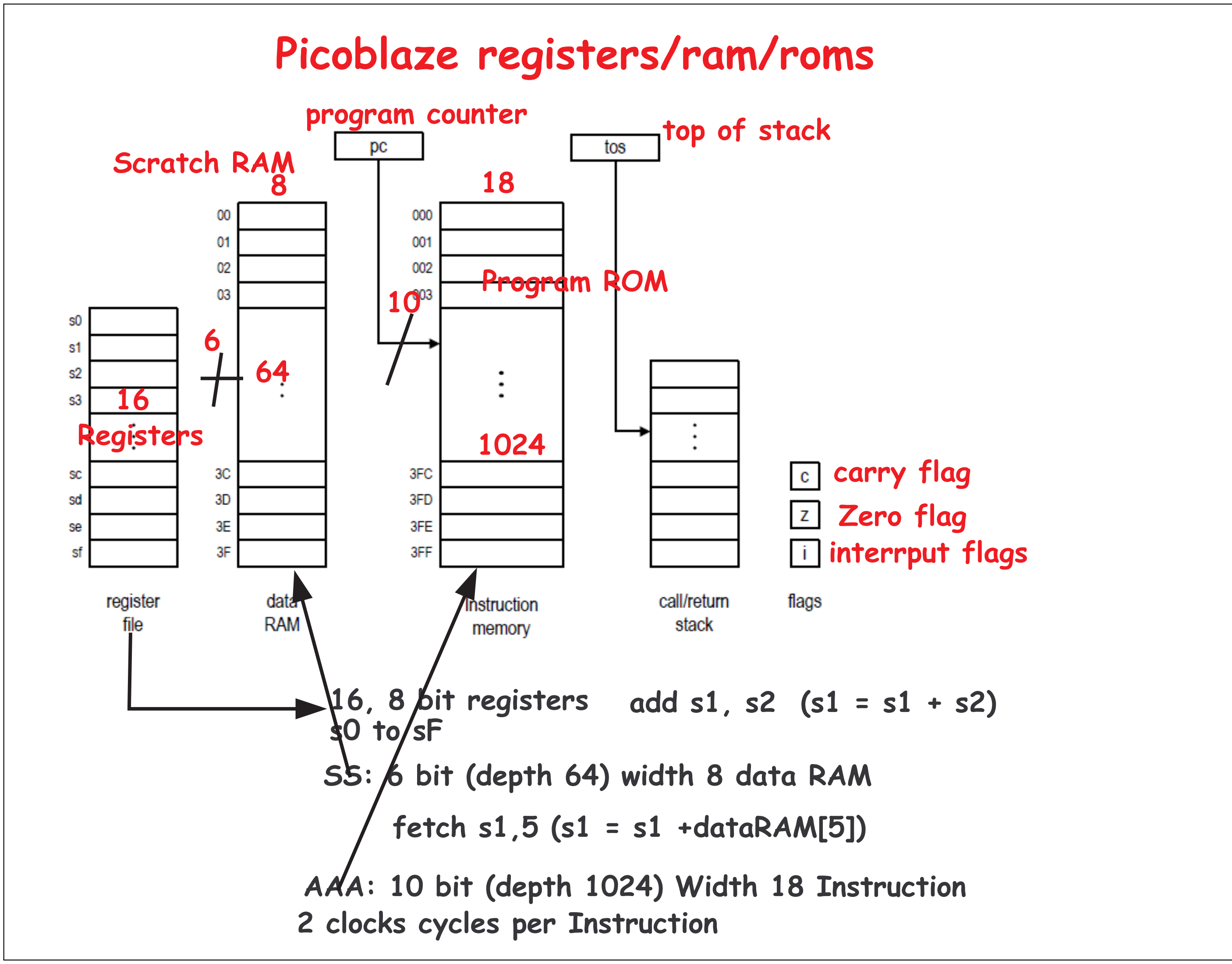



Figure 14.9: Various registers and ROMs

## 14.10 Instruction format

### 14.10.1 Arithmetic Instructions

**ARITHMETIC INSTRUCTIONS**

**All arithmetic instructions effect ZERO and CARRY**

| Instruction | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD sX,kk | 0 | 1 | 1 | 0 | 0 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| ADD sX,sY | 0 | 1 | 1 | 0 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |
| ADDCY sX,kk | 0 | 1 | 1 | 0 | 1 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| ADDCY sX,sY | 0 | 1 | 1 | 0 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

**Registers with Registers**

**ADD sXX, sYY Addition of sXX and sYY** **sXX = sXX + sYY**
**ADDCY sXX, sYYAddition of sXX, sYY and CARRY**
**SUB sXX, sYY Subtraction of sXX and sYY**
**SUBCY sXX, sYYSubtraction of sXX, sYY and CARRY**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SUB sX,kk | 0 | 1 | 1 | 1 | 0 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| SUB sX,sY | 0 | 1 | 1 | 1 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |
| SUBCY sX,kk | 0 | 1 | 1 | 1 | 1 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| SUBCY sX,sY | 0 | 1 | 1 | 1 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

**Registers with constants**

**ADD sXX, kk Addition of sXX and kk** **sXX = sXX + kk(8 bits)**
**ADDCY sXX, kk Addition of sXX, kk and CARRY**
**SUB sXX, kk Subtraction of sXX and kk**
**SUBCY sXX, kk Subtraction of sXX, kk and CARRY**

Figure 14.10: Arithmetic Instructions

### 14.10.2 Logical Instructions

**LOGICAL INSTRUCTIONS**

**CARRY is 0 after Logical operations**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AND sX,kk | 0 | 0 | 1 | 0 | 1 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| AND sX,sY | 0 | 0 | 1 | 0 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

**AND sXX, sYY Bit-wise AND between sXX and sYY**
**AND sXX, kk Bit-wise AND between sXX and kk**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OR sX,kk | 0 | 0 | 1 | 1 | 0 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| OR sX,sY | 0 | 0 | 1 | 1 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

**OR sXX, sYY Bit-wise OR between sXX and sYY**
**OR sXX, kk Bit-wise OR between sXX and kk**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XOR sX,kk | 0 | 0 | 1 | 1 | 1 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| XOR sX,sY | 0 | 0 | 1 | 1 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

**XOR sXX, sYY Bit-wise XOR between sXX and sYY**
**XOR sXX, kk Bit-wise XOR between sXX and kk**

Figure 14.11: Logical Instructions

### 14.10.3 Compare Instructions

**COMPARE INSTRUCTIONS**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| COMPARE sX,kk | 0 | 1 | 0 | 1 | 0 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| COMPARE sX,sY | 0 | 1 | 0 | 1 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

```
compare sXX,sYY

 if (sXX == sYY) {
       ZERO = 1
 }else {
       ZERO = 0
 }
 if (sYY > sXX) {
       CARRY = 1
 }else {
       CARRY = 0
 }
```

```
compare sXX,kk
 if (sXX == kk) {
       ZERO = 1
 }else {
       ZERO = 0
 }
 if (kk > sXX) {
       CARRY = 1
 }else {
       CARRY = 0
 }
```

Figure 14.12: Compare Instructions

### 14.10.4 Data movement Instructions

**DATA MOVEMENT INSTRUCTIONS**

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LOAD sX,kk | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | k | k | k | k | k | k | k | k |
| LOAD sX,sY | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

```
load sXX,sYY   sXX <- sYY
load sXX,kk    sXX <-- kk
```

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FETCH sX, ss | 0 | 0 | 0 | 1 | 1 | 0 | x | x | x | x | 0 | 0 | s | s | s | s | s | s |
| FETCH sX,(sY) | 0 | 0 | 0 | 1 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

```
fetch sXX,SYY   sXX = RAM[(sYY)] ;
fetch sXX,ss    sXX = RAM[ss] ;
```

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| STORE sX, ss | 1 | 0 | 1 | 1 | 1 | 0 | x | x | x | x | 0 | 0 | s | s | s | s | s | s |
| STORE sX,(sY) | 1 | 0 | 1 | 1 | 1 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |

```
store sXX,SYY   RAM[(sYY)] = sXX ;
store sXX,ss    RAM[ss] = sXX;
```

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT sX,(sY) | 0 | 0 | 0 | 1 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |
| INPUT sX,pp | 0 | 0 | 0 | 1 | 0 | 0 | x | x | x | x | p | p | p | p | p | p | p | p |

```
input sXX,SYY            input sXX,pp
    port_id = sYY;           port_id = pp;
    sXX = in_port            sXX = in_port
```

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OUTPUT sX,(sY) | 1 | 0 | 1 | 1 | 0 | 1 | x | x | x | x | y | y | y | y | 0 | 0 | 0 | 0 |
| OUTPUT sX,pp | 1 | 0 | 1 | 1 | 0 | 0 | x | x | x | x | p | p | p | p | p | p | p | p |

```
output sXX,SYY           output sXX,pp
    port_id = sYY;           port_id = pp;
    out_port= sXX            out_port = sXX
```

Figure 14.13: Data movement Instructions

### 14.10.5 INPUT Instruction

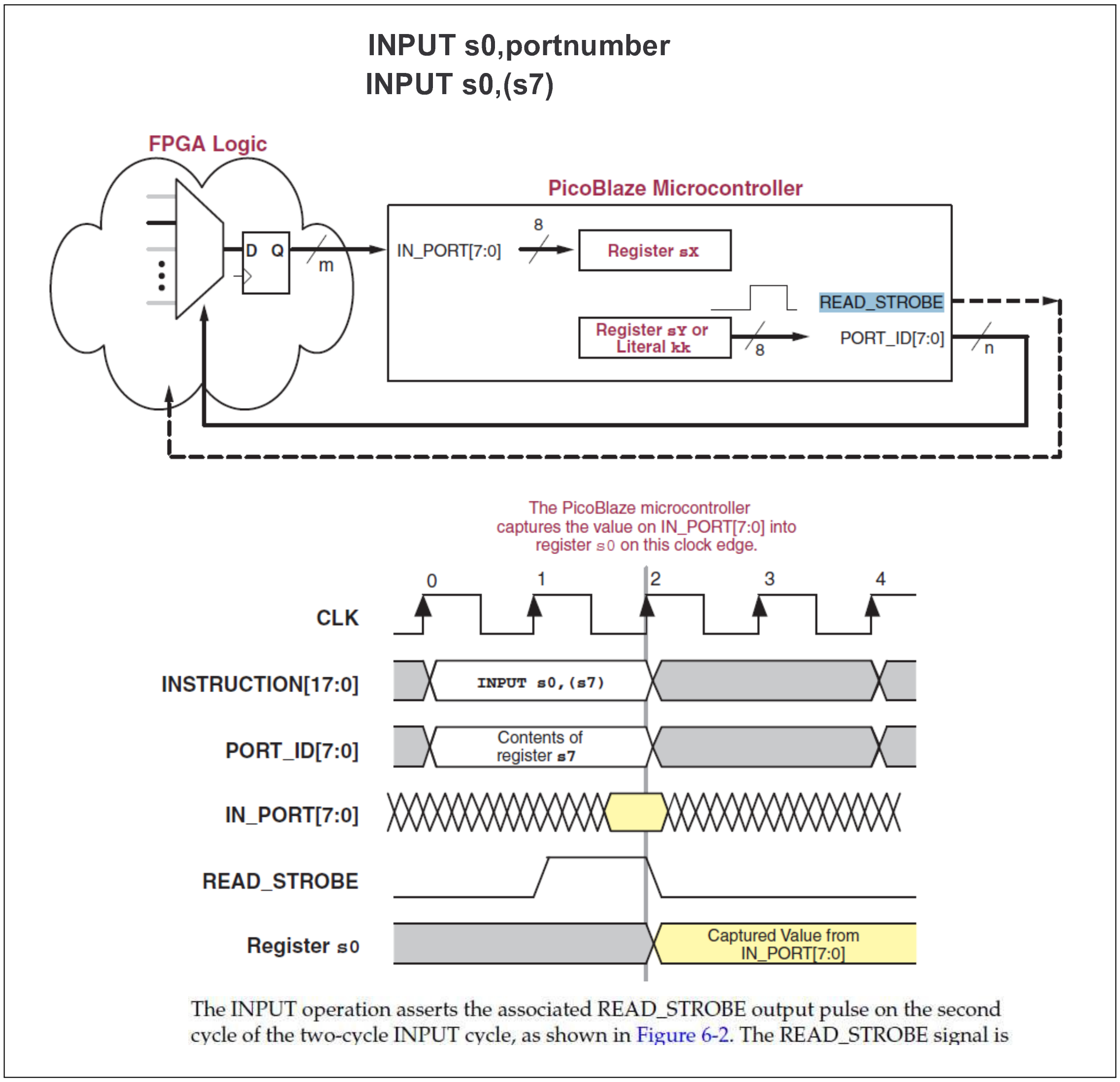



Figure 14.14: Input Instruction

### 14.10.6 Shift and rotate Instructions

### 14.10.7 Program flow Instructions

## 14.11 PicoBlaze programming model

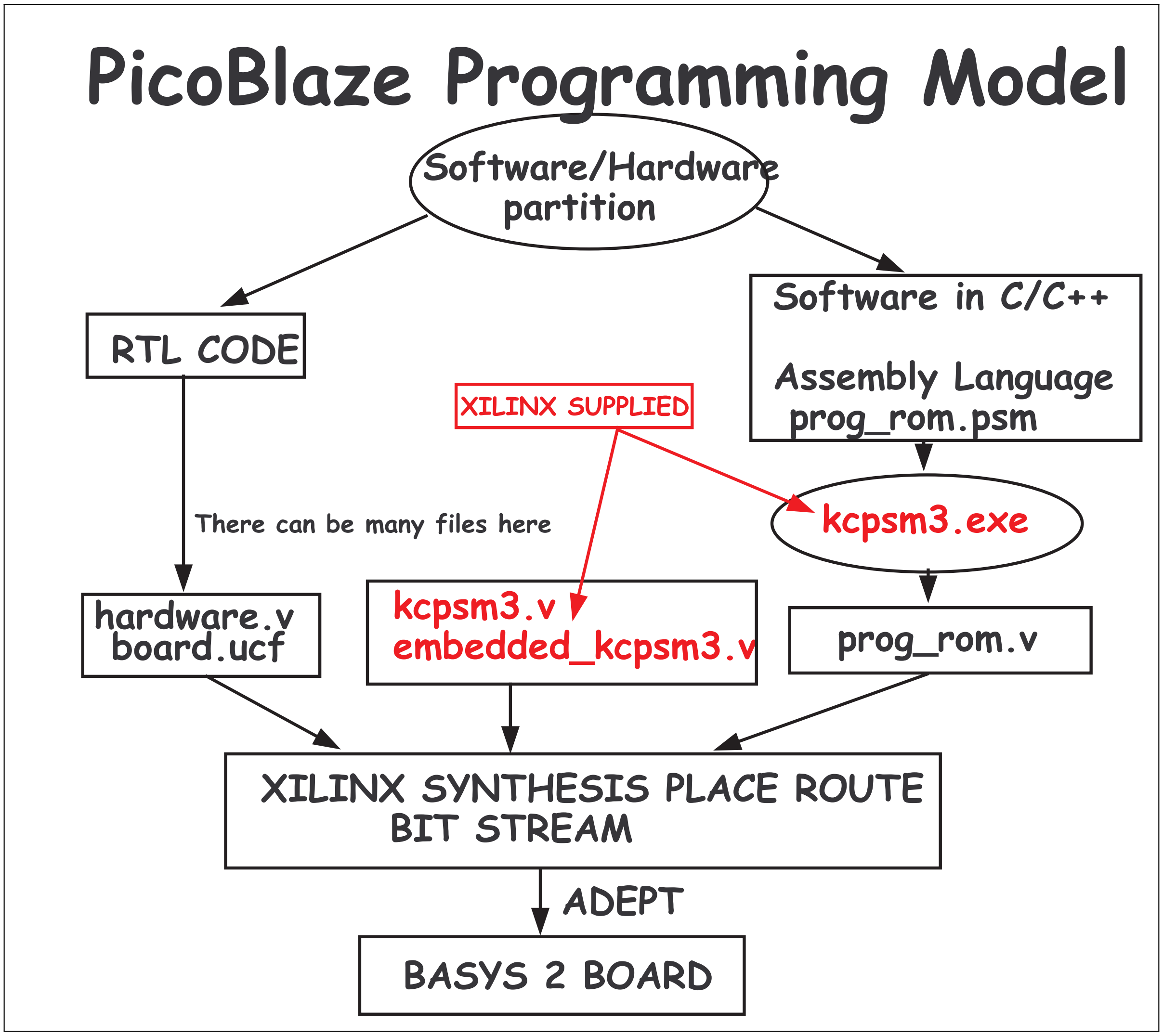


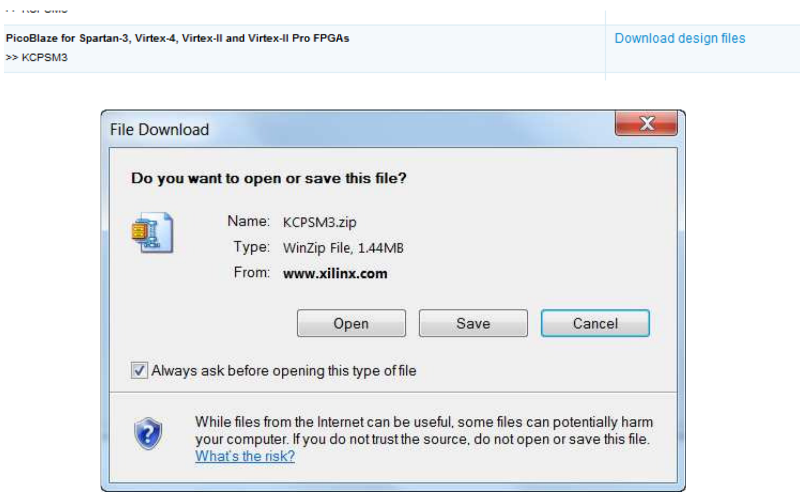

Figure 14.15: PicoBlaze programming model

## 14.12 Required files and executables to use MicroBlaze

# Required files and executable

http://www.xilinx.com/ipcenter/processor_central/picoblaze/member/




From the above download, you get **KCPSM3.zip**

When you unzip, you get:

Directory **Assembler** which has:
- **KCPSM3.EXE** -- Assembler that converts your assembly language **prog_rom.psm to prog_rom.v**
- **ROM_form.v** -- Uses this form

Directory **Verilog** which has
- **kcpsm3.v** -- Microcontroller in structural Verilog
- **embedded_kcpsm3.v**- ROM + instantiation of kcpsm3

```
module embedded_kcpsm3() ;
    kcpsm3 processor
    prog_rom program
endmodule
```

DO NOT CHANGE ANYTHING HERE.
ALWAYS enter your assembly language in **prog_rom.psm**

Figure 14.16: Required files and executables

## 14.13 Example 1

### 14.13.1 Problem description

1. Read 8 switches of the board using Input port of the processor.
2. Output the read value (0 to 255) on to the LEDS of the board.
3. Convert the lower 4 bits of the read values (SW3 to SW0) (15 to 0) to hexadecimal and display on the last segment of the seven segment display.
4. When switches are changed, both the LEDS and seven segment display should change correctly.

Figure 14.17: Problem description

### 14.13.2 Assembly language to Verilog ROM

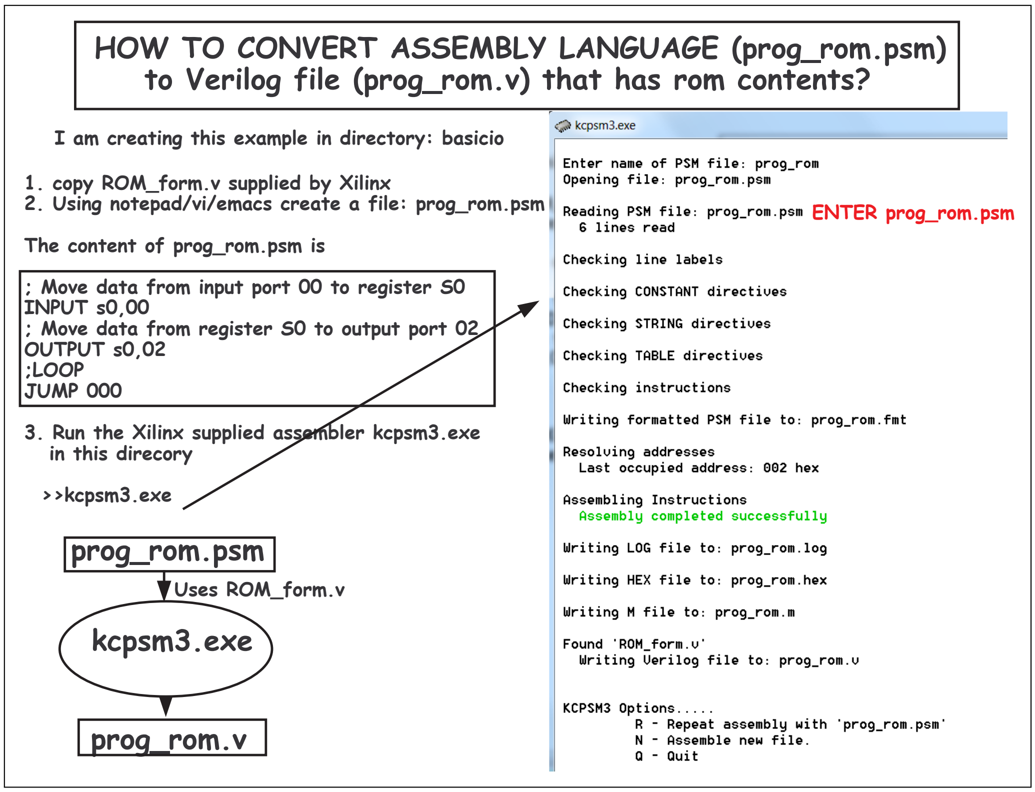



Figure 14.18: Assembly to Verilog

### 14.13.3 Hardware Design

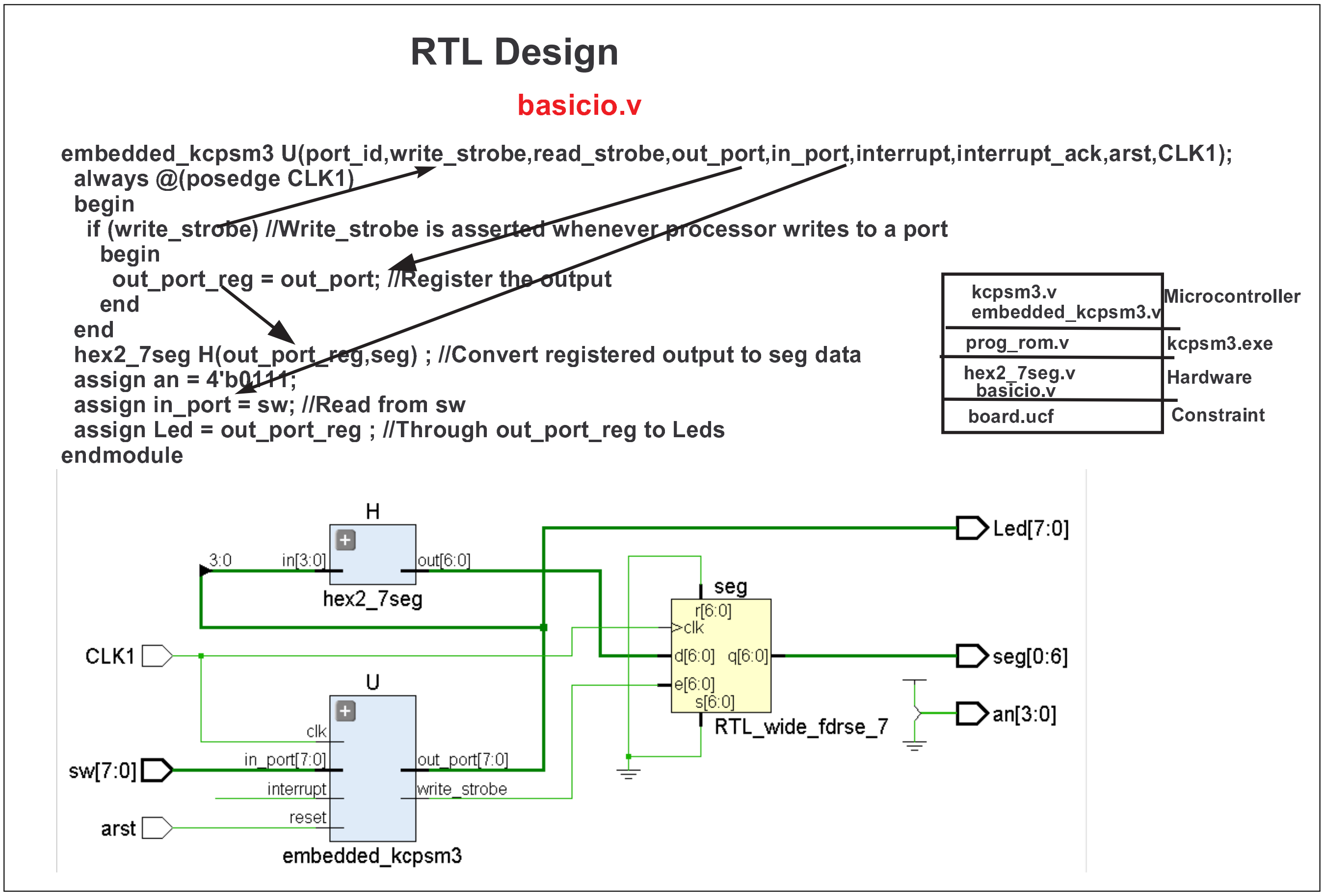



Figure 14.19: Hardware design

```
;prog_rom.psm
; Move data from input port 00 to register S0
INPUT s0,00
; Move data from register S0 to output port 02
OUTPUT s0,02
;LOOP
JUMP 000

```

```

//C:\work\fpga\course\v\picoblazeapplications\gold\kcpsm3.v
//C:\work\fpga\course\v\picoblazeapplications\gold\embedded_kcpsm3.v
//C:\work\fpga\course\v\ch7seg\hex2_7seg.v
//C:\work\fpga\course\v\ch7seg\board.ucf
//C:\work\fpga\course\v\picoblazeapplications\basicio\prog_ram.v
//C:\work\fpga\course\v\picoblazeapplications\basicio\basicio.v

module basicio(CLK1,sw, arst, seg, an,Led) ;
        parameter N = 7 ;
        parameter W = 4 ;
        input CLK1, arst ;
        input [N:0] sw ; /* 3210 */
        output [0:N-1] seg;
        output [W-1:0] an ;
        output [N:0] Led ;

        wire[9:0] address;
        wire[17:0] instruction;
        wire [7:0] port_id,in_port,out_port;
        wire write_strobe ;
        wire interrupt_ack;
        wire read_strobe;
        reg [7:0] out_port_reg ;

        embedded_kcpsm3 U(port_id,write_strobe,read_strobe,out_port,in_port,interrupt,interrupt_ack,arst,CLK1);
        always @(posedge CLK1)
        begin
                if (write_strobe) //Write_strobe is asserted whenever processor writes to a port
                        begin
                                out_port_reg = out_port; //Register the output
                        end
        end
        hex2_7seg H(out_port_reg,seg) ; //Convert registered output to seg data
        assign an = 4'b0111;
        assign in_port = sw; //Read from sw
        assign Led = out_port_reg ; //Through out_port_reg to Leds
endmodule

```

# 14.14 Example 2

## 14.14.1 Problem description

```
1. Read 8 switches of the board using Input port of the processor.
2. Output the read value (0 to 255) on to the LEDS of the board.
3. Sum the lower 4 bits of the read values (SW3 to SW0)
    (15 to 0) to 1
    example: If switch is 1010(10)
    sum is: 10 + 9 + 8 +  .. + 1 = 55
4. Convert binary value of sum to hexadecimal and display on the
    seven segment display.
5. When switches are changed, both the LEDS and seven segment
    display should change correctly.
6. Note that, this program produces wrong results, if the sum is
   greater than 255. This is because the picoblaze processor is 8 bits.
7. Modify the code to produce correct results.
```

Figure 14.20: Problem description

### 14.14.2 Software and hardware partition

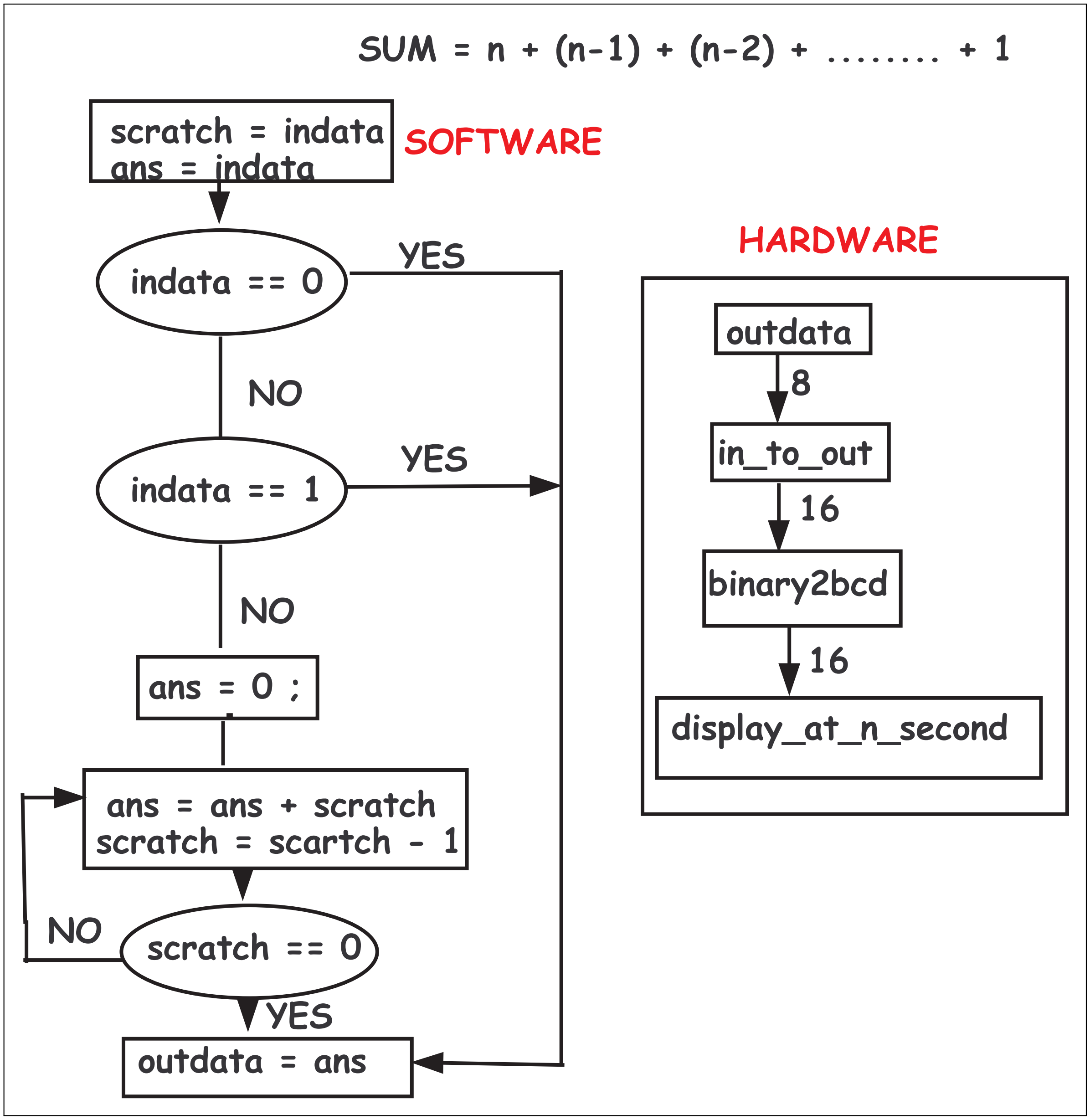



Figure 14.21: Software and hardware partition

### 14.14.3 Assembly language to Verilog ROM

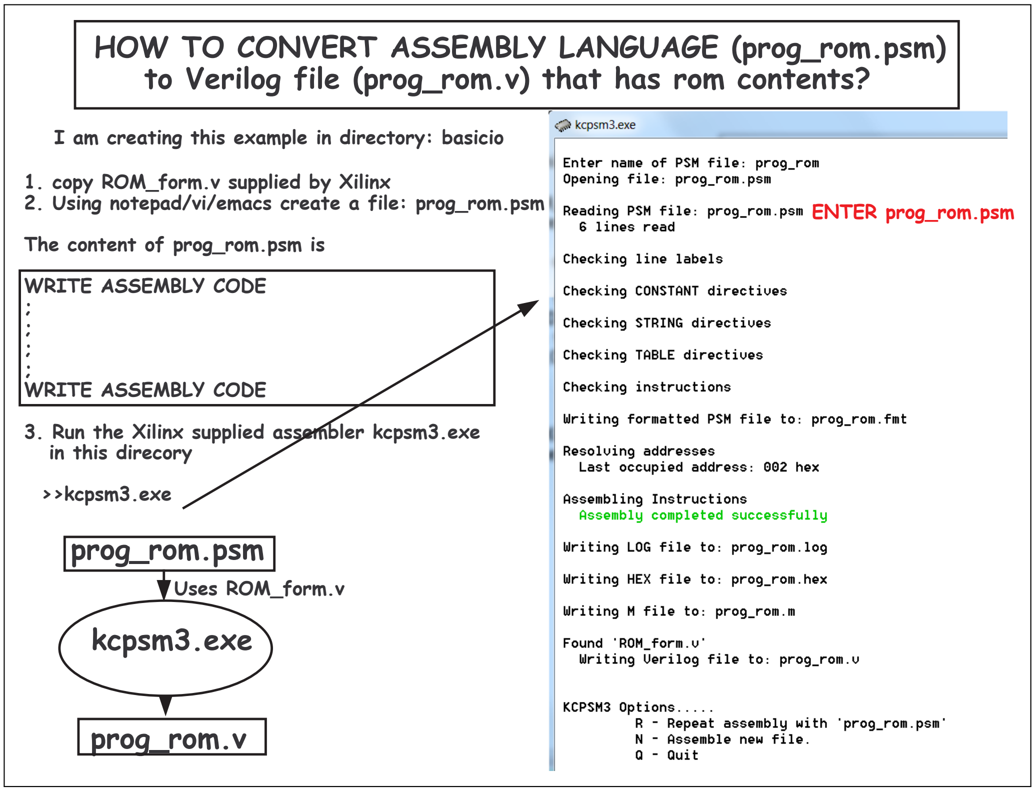



Figure 14.22: Assembly to Verilog

### 14.14.4 Hardware Design

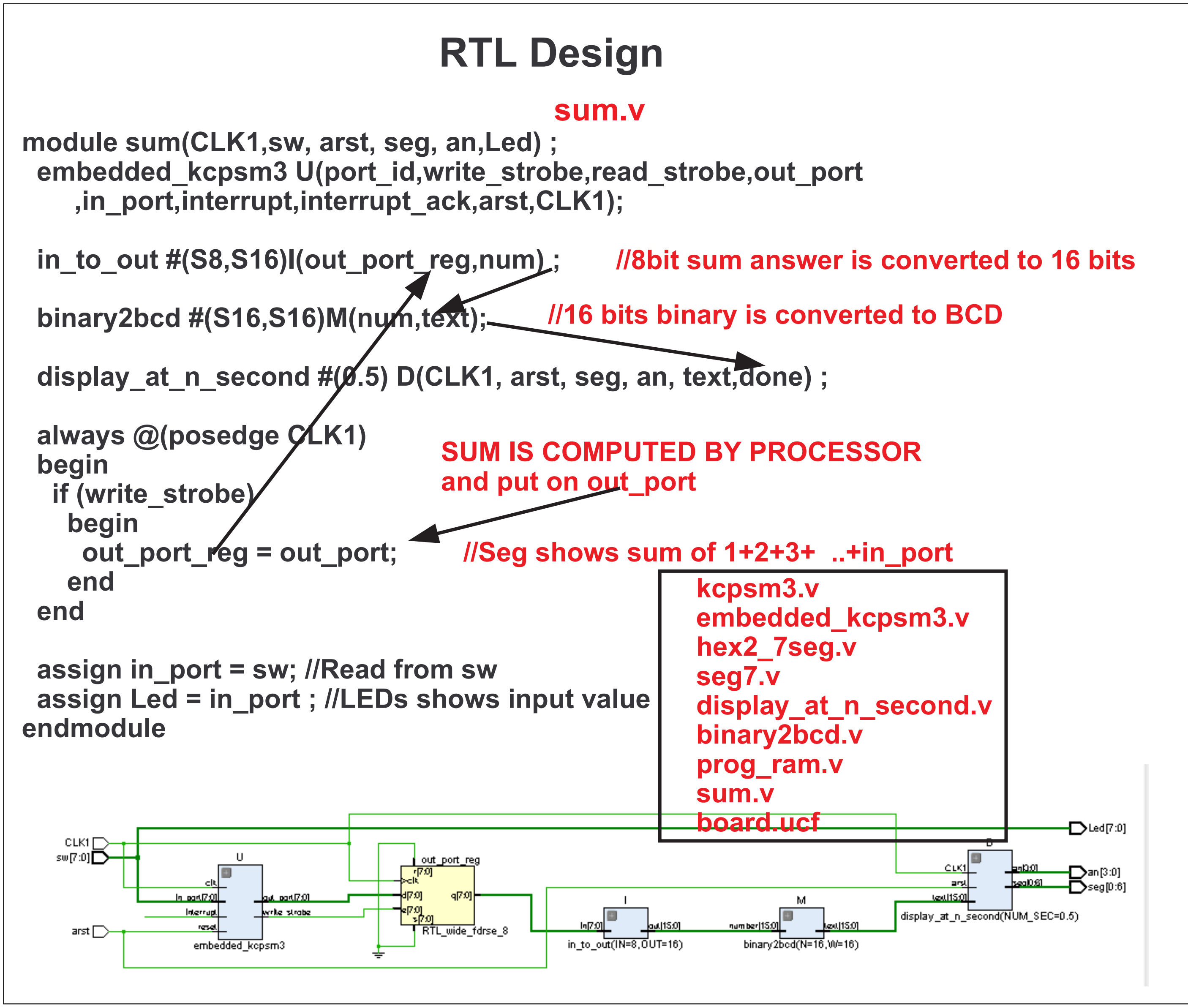



Figure 14.23: Hardware design

```
;FILE: prog_rom.psm
; Move data from input port 00 to register S0
;INPUT s0,00
; Move data from register S0 to output port 02
;OUTPUT s0,02
;LOOP
;JUMP 000

;======================================================
; input and output port
constant INPUTPORT, 00
constant OUTPUTPORT, 02
;======================================================

;======================================================
; register alias
namereg s0, indata ;//s0 reg has indata, switch contents (0 to 255)
namereg s1, scratch ; //Use s1 reg as a scratch
namereg s2, ans; //Use s2 for final answer
;======================================================
; Main Program
loop:
    call read_switch
    call compute_sum
    call display
    jump loop

read_switch:
    input indata,INPUTPORT ; //Read INPUTPORT to indata reg
    return

display:
    output ans,OUTPUTPORT ; //Output ans reg value to OUTPUTPORT
    return

compute_sum:
    load ans, indata ;//Copy indata to ans
    load scratch, indata ;//Copy indata to scratch
    compare indata, 00; //is indata == 0
    jump nz, start1 ; //Goto start1 if indata != 0 ;
    return ; //Return if indata == 0
    start1: compare indata, 01; //is indata == 1
    jump nz, start ; //Goto start if indata != 1 ;
    return ; //Return if indata = 1

    start: load ans, 00 ;//ans = 0 ;
    until:
            add ans, scratch; //ans = ans + scratch
            sub scratch, 01 ;//scratch = scratch - 1;
            jump nz, until ; //repeat until scratch = 0 ;
    return ;

```

```
//FILE: sum.v

//C:\work\fpga\course\v\picoblazeapplications\gold\kcpsm3.v
//C:\work\fpga\course\v\picoblazeapplications\gold\embedded_kcpsm3.v
//hex2_7seg.v seg7.v display_at_n_second.v binary2bcd.v board.ucf

//C:\work\fpga\course\v\picoblazeapplications\sum\prog_ram.v
//C:\work\fpga\course\v\picoblazeapplications\sum\sum.v

module in_to_out(in,out) ;
    parameter IN = 8 ;
    parameter OUT = 16 ;
    input [IN-1:0] in ;
    output [OUT-1:0] out ;

    assign out = in ;
endmodule

module sum(CLK1,sw, arst, seg, an,Led) ;
    parameter N = 7 ;
    parameter W = 4 ;
    parameter S8 = 8 ;
    parameter S16 = 16 ;
    input CLK1, arst ;
    input [N:0] sw ; /* 3210 */
    output [0:N-1] seg;
    output [W-1:0] an ;
    output [N:0] Led ;

    wire[9:0] address;
    wire[17:0] instruction;
    wire [7:0] port_id,in_port,out_port;
    wire write_strobe ;
    wire interrupt_ack;
    wire    read_strobe;
    wire [N-1:0] seg_out ;
    reg [7:0] out_port_reg;
    wire [S16-1:0] num ;
    wire [S16-1:0] text ;
    wire done ;

    embedded_kcpsm3 U(port_id,write_strobe,read_strobe,out_port,in_port,interrupt,interrupt_ack,arst,CLK1);
    in_to_out #(S8,S16)I(out_port_reg,num) ;
    binary2bcd #(S16,S16)M(num,text);
    display_at_n_second #(0.5) D(CLK1, arst, seg, an, text,done) ;

    always @(posedge CLK1)
    begin
        if (write_strobe)
            begin
                out_port_reg = out_port; //Seg shows sum of 1+2+3+
```

```
..+in_port
            end
    end

    assign in_port = sw; //Read from sw
    assign Led = in_port ; //LEDs shows input value

endmodule
```

## 14.15 Example 3

### 14.15.1 Problem description

1. Start with a number,start, say 4.
2. Increment the number by a number, incr, say 20
3. Addition is done using microblaze
4. Display the number on the board at a time interval of 'n' second
   Example 4,24,44,64 etc
5. Output the above results on LED's also
6. What happens after the sum > 255
7. Modify the code, both hardware and software code, to
   handle 16 bits.

Figure 14.24: Problem description

### 14.15.2 Software and hardware partition

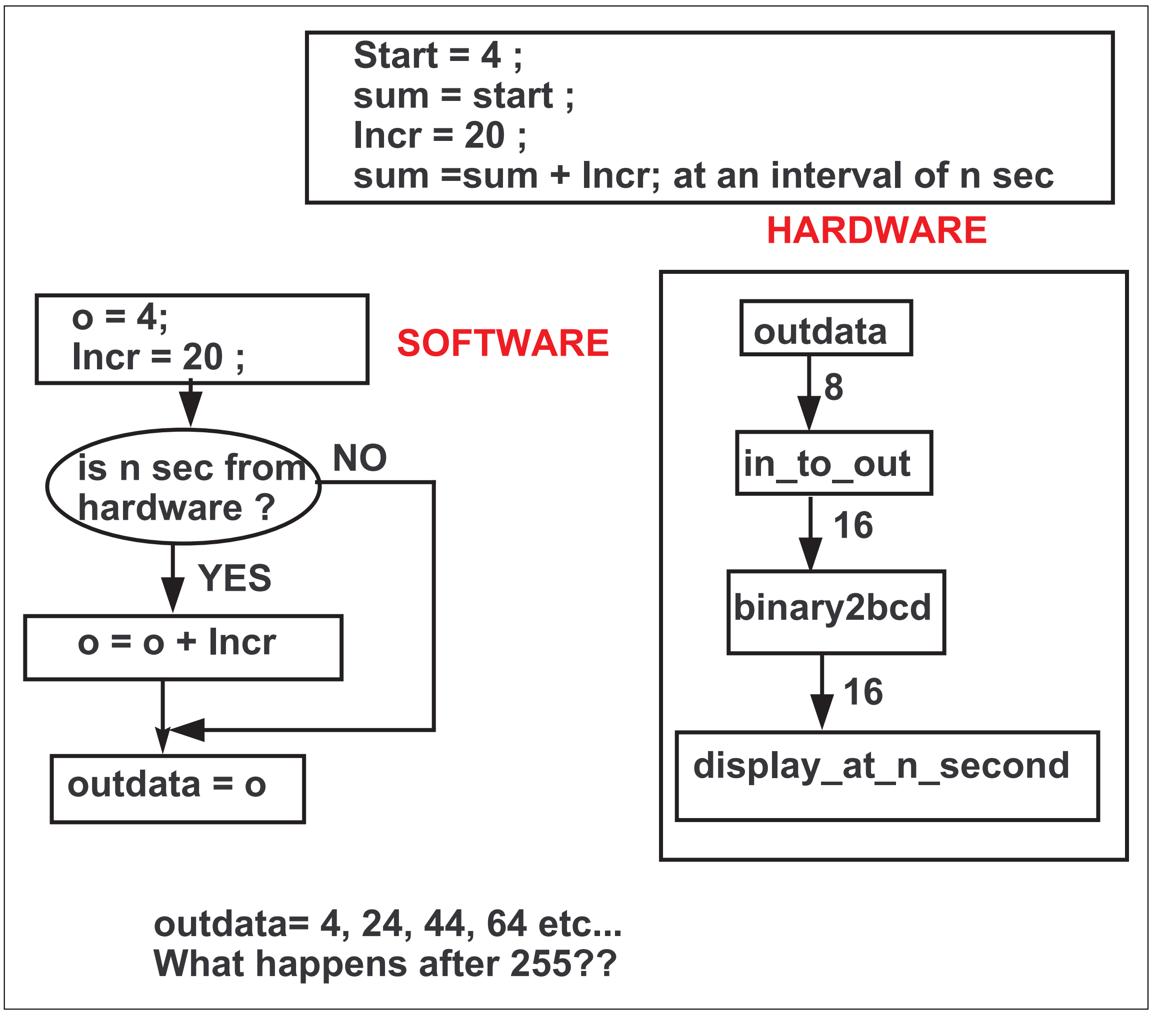



Figure 14.25: Software and hardware partition

### 14.15.3 Assembly language to Verilog ROM

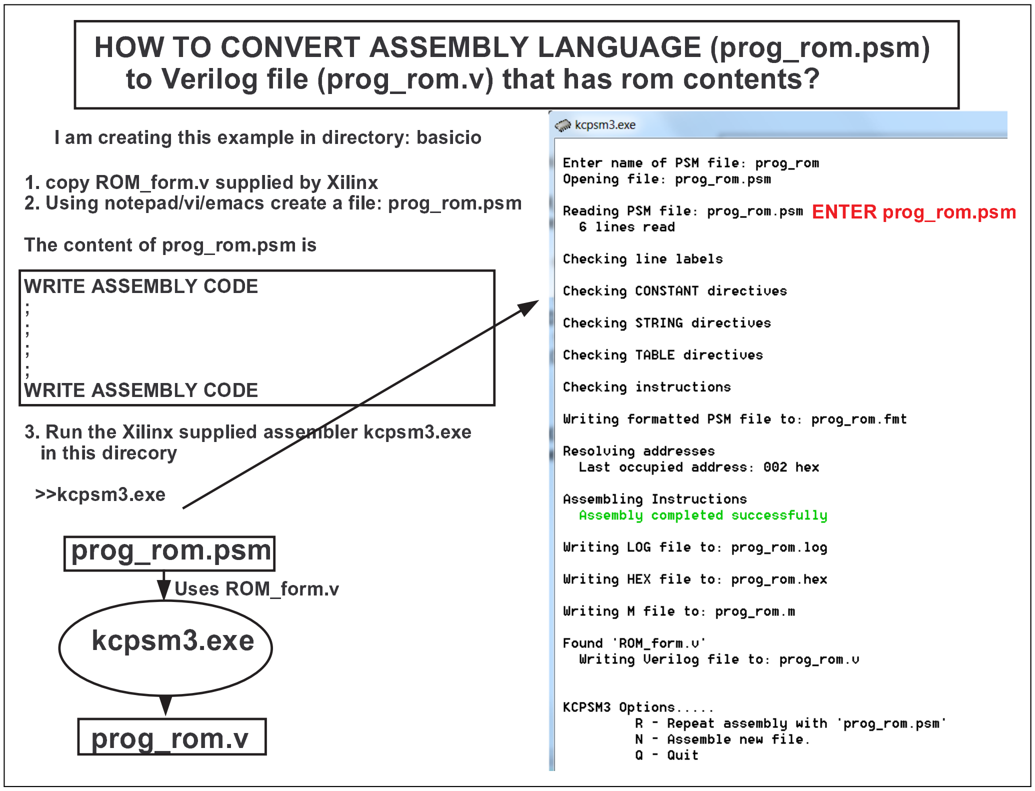



Figure 14.26: Assembly to Verilog

### 14.15.4 Hardware Design

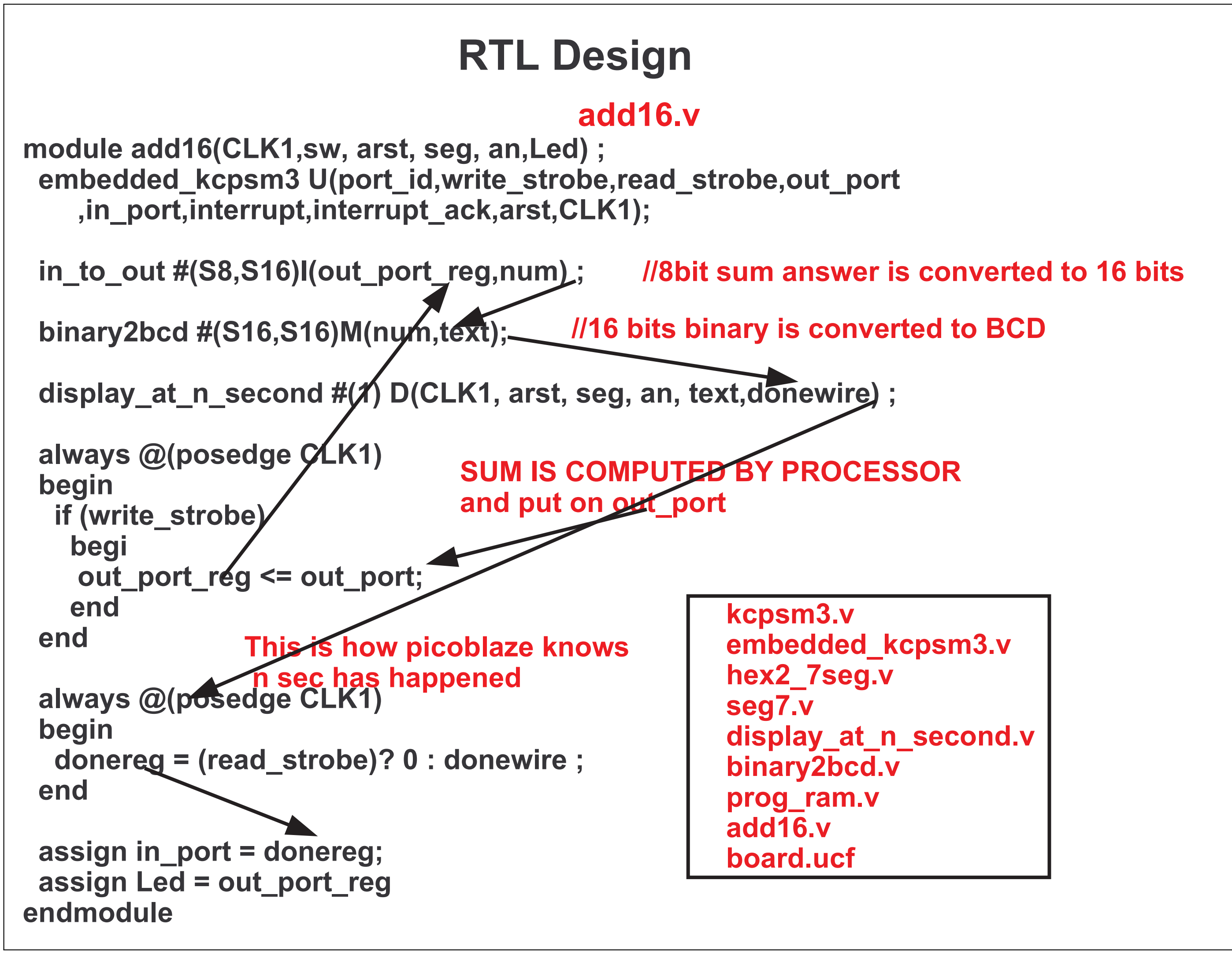



Figure 14.27: Hardware design

```
;prog_rom.psm

;======================================================
; input and output port
constant INPUTPORT, 00
constant OUTPUTPORT, 02
;======================================================

;======================================================
; register alias
namereg s0, indata ; //Read from switch. Will be 1 when 1sec is passed else 0
namereg s1, startdata ; //s1 has initial data, say 05
namereg s2, ans; //s2 has answer, ans = ans + s1
;======================================================

;======================================================
; Main Program
;======================================================
loop:
        call init
        call add_num

;======================================================
; read switch
;======================================================
 read_switch:
        input indata,INPUTPORT ; //Read INPUTPORT to indata reg
        return

;======================================================
; send data to external world
;======================================================
display:
        output ans,OUTPUTPORT ; //Output ans reg value to OUTPUTPORT
        return

;======================================================
; initialize
;======================================================
 init:
        load ans, 01 ; //start with 04 HEXNUMBER
        load startdata, 01; //increment at a distance of 35. 23 is HEXNUMBER
        call display
        return

;======================================================
; add numbers
;======================================================
add_num:
        call read_switch; //Read in_port
        compare indata, 00; //is indata == 0? when done is 1 indata is 1.
```

```
That means 1 sec has happened
        jump nz, accumulate ; //Goto accumulate if indata != 0 ;
        jump add_num ; //LOOP
        accumulate: add ans, startdata; //s2 = s2 + s1
                    call display
                    jump add_num
```

```
//MICROCONTROLLER
//C:\work\fpga\course\v\picoblazeapplications\gold\kcpsm3.v
//C:\work\fpga\course\v\picoblazeapplications\gold\embedded_kcpsm3.v

//CODE FOR DISPLAY
//C:\work\fpga\course\v\ch7seg\hex2_7seg.v seg7.v display_at_n_second.v binary2bcd.v board.ucf

//SOFTWARE CODE
//C:\work\fpga\course\v\picoblazeapplications\add16\prog_ram.v

//MAIN
//C:\work\fpga\course\v\picoblazeapplications\add16\add16.v

//THIS WORKS PERFECTLY AT A DISTANCE OF 1 Sec

module in_to_out(in,out) ;
  parameter IN = 8 ;
  parameter OUT = 16 ;
  input [IN-1:0] in ;
  output [OUT-1:0] out ;

  assign out = {8'b0,in};
endmodule

module add16(CLK1,sw, arst, seg, an,Led) ;
  parameter N = 7 ;
  parameter W = 4 ;
  parameter S8 = 8 ;
  parameter S16 = 16 ;
  input CLK1, arst ;
  input [N:0] sw ; /* 3210 */
  output [0:N-1] seg;
  output [W-1:0] an ;
  output [N:0] Led ;

  wire[9:0] address;
  wire[17:0] instruction;
  wire [7:0] port_id,in_port,out_port;
  wire write_strobe ;
  wire interrupt_ack;
  wire     read_strobe;
  wire [N-1:0] seg_out ;

  reg [7:0] out_port_reg;
  wire [S16-1:0] num ;
  wire [S16-1:0] text ;
  wire donewire;
  reg donereg ;

  embedded_kcpsm3 U(port_id,write_strobe,read_strobe,out_port,in_port,interrupt,interrupt_ack,arst,CLK1);
```

```
    in_to_out #(S8,S16) I(out_port_reg,num) ;
    binary2bcd #(S16,S16) B2B(num,text);
    display_at_n_second #(1) D(CLK1, arst, seg, an, text,donewire) ;

    always @(posedge CLK1)
    begin
      if (write_strobe)
        begin
         //All the work is done by microblaze.
         //It gives answer in out_port. But we sample at edge of clock
         out_port_reg <= out_port;
        end
    end

    always @(posedge CLK1)
    begin
      donereg = (read_strobe)? 0 : donewire ;
    end

    assign in_port = donereg;
    //in_port will have value of '1' for every 1 second
    assign Led = out_port_reg ;
  endmodule

```