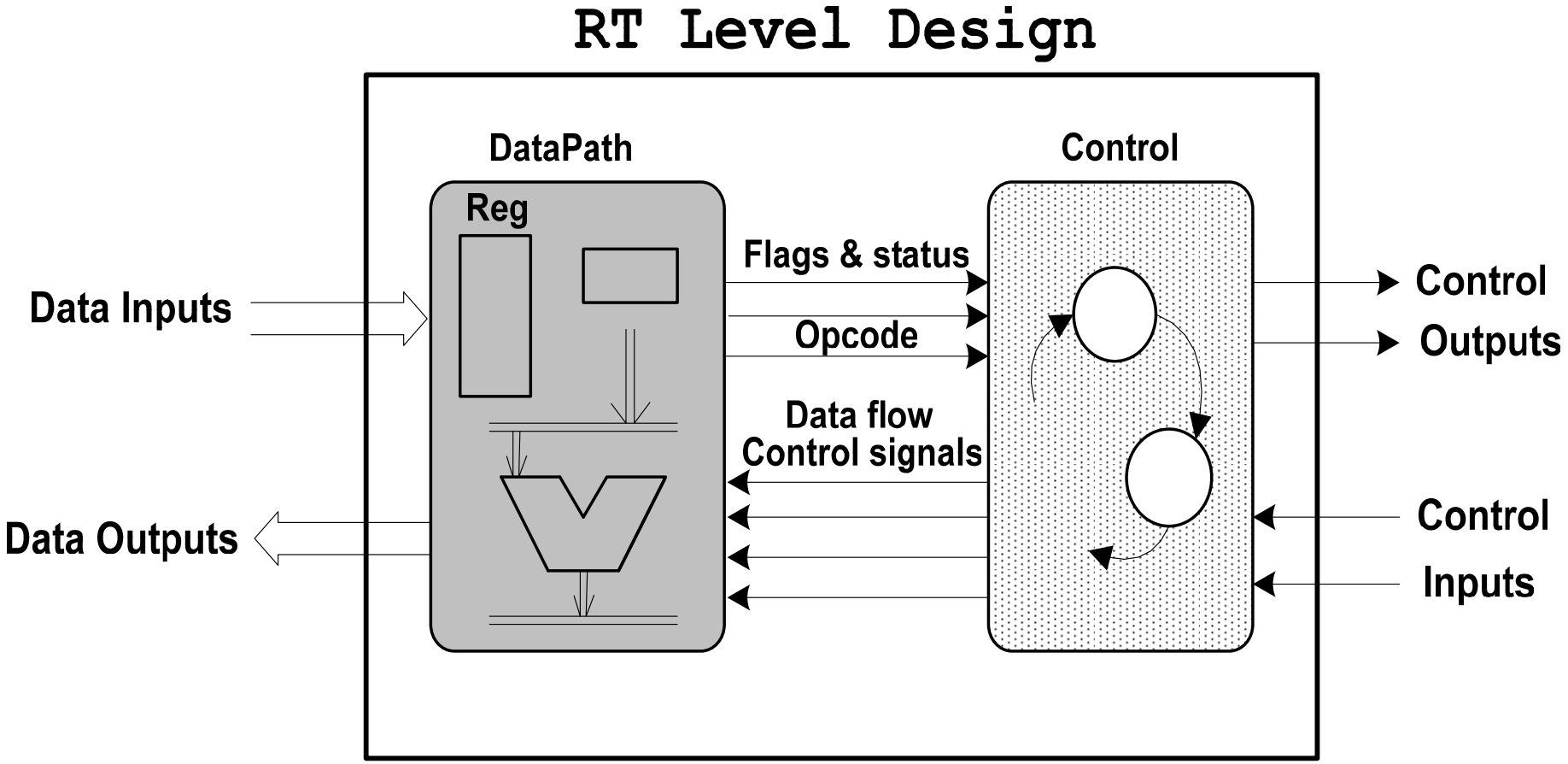
Chapter 7

Design and Synthesis of Datapath Controllers

RTL design of the digital system



Datapath & Controller

- The datapath portion of the circuit is the portion that **transforms the input signals into the corresponding output signals**.
- The controller is the portion that **governs the flow of the input signals through the various components comprising the datapath**.

Control-dominated & Data-dominated systems

- **Control-dominated systems** are reactive systems, responding to external events
- **Data-dominated systems** are shaped by the requirements of **high-throughput data computation and transport**, as in telecommunications and signal processing
- **Sequential machines** are commonly classified and **partitioned into datapath units and control units**

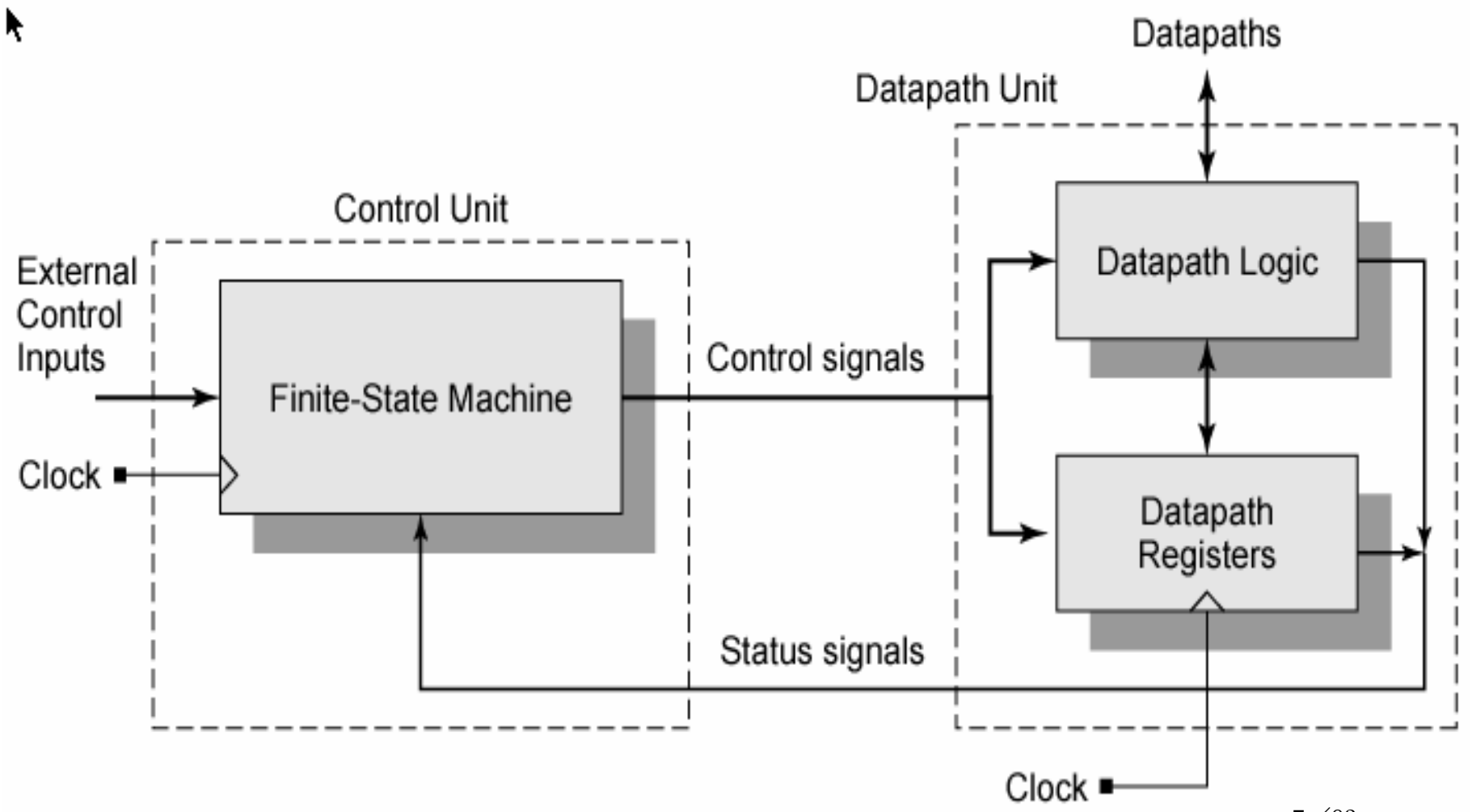
Data-dominated systems

- Most datapaths include **arithmetic units**, such as **arithmetic and logic units (ALUs), adders, multipliers, shifters, and digital signal processors**
- Datapath units are **characterized by repetitive operations on different sets of data, as in signal processing, image processing, and multimedia**

Control units

- Architectures that are dominated by control units will **generally have a significant amount of random (irregular) logic**, **together with some regular structures**, like **multiplexers** for steering signals, and **comparators**

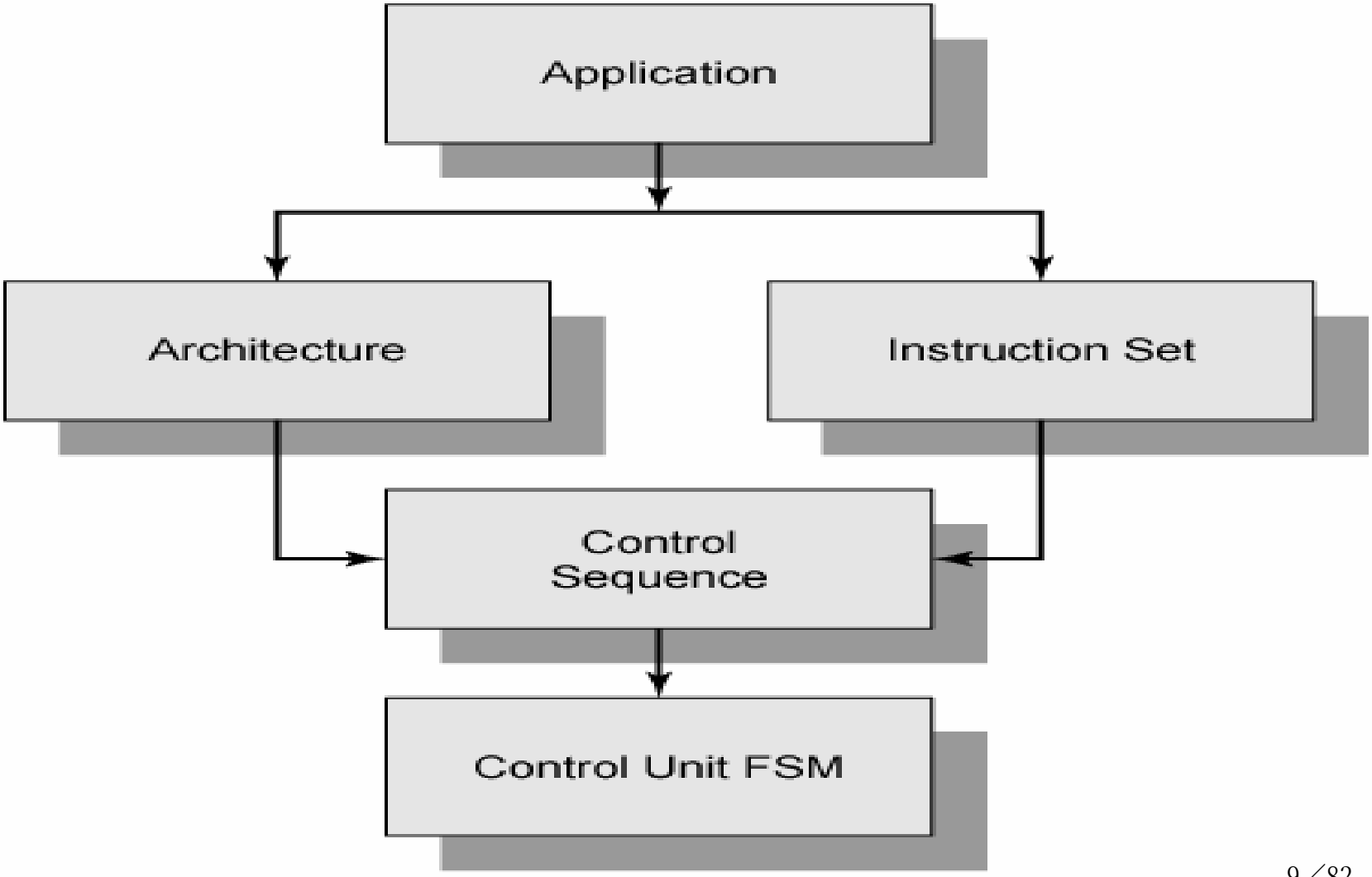
State-machine controller for a datapath



7.1 Partitioned sequential machines

- Partitioning a sequential machine into a datapath and a controller clarifies the architecture and simplifies the design of the system
- The process by which the machine is designed is said to be **application-driven**
- Ultimately, **the FSM that controls the datapath.**

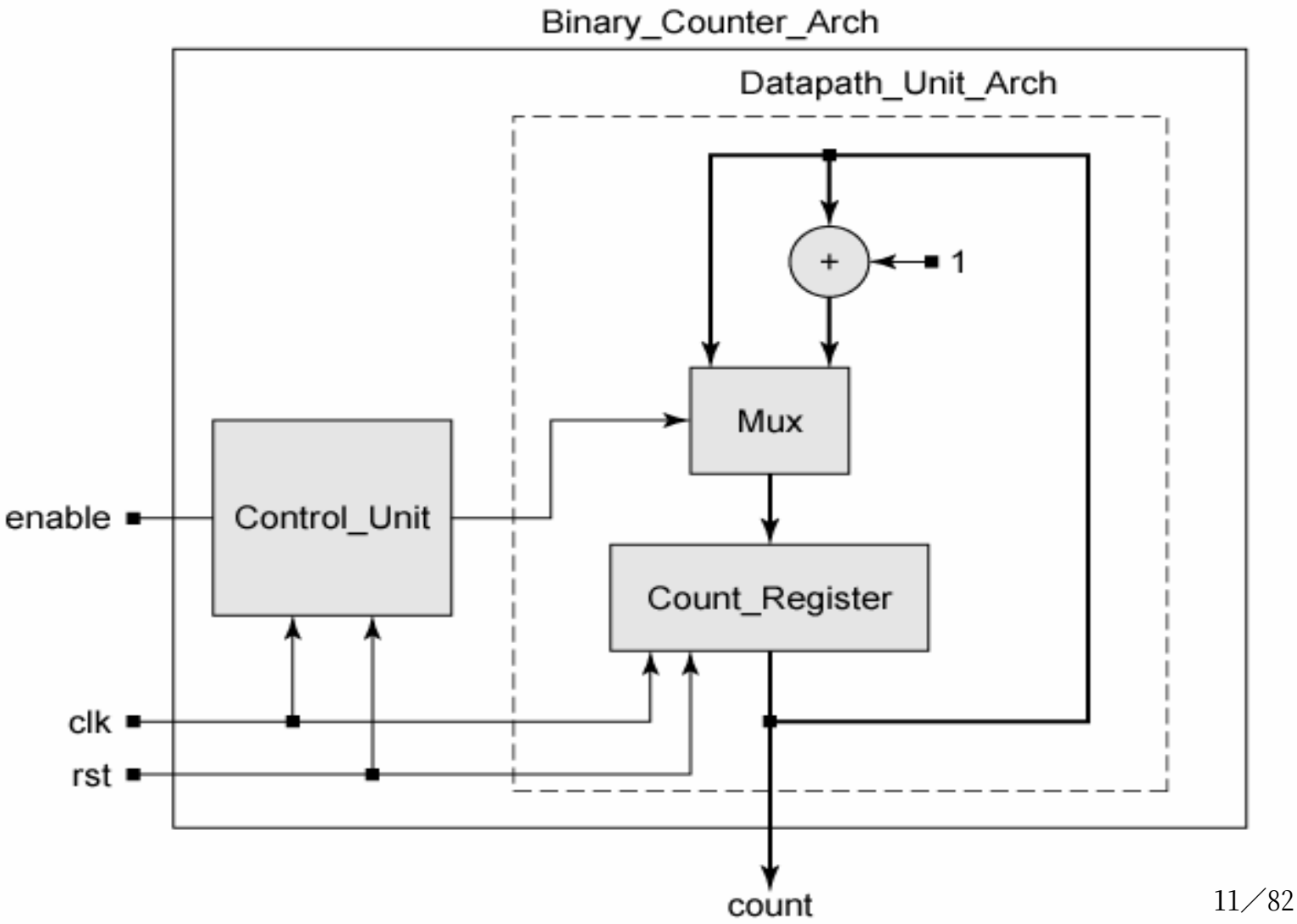
The sequence of steps in an application-driven design process



7.2 Design example: binary counter

- A synchronous 4-bit binary counter ,it is incremented by a count of 1 at each active edge of the clock
- To describe the counter **by an implicit state machine**
- Binary_Counter_Imp, executing a register transfer operation (count<=count+1) conditionally, in every clock cycle, depending on enable, and then synthesize a hardware realization directly
- **To partition the machine into an architecture of separate datapath and control units**

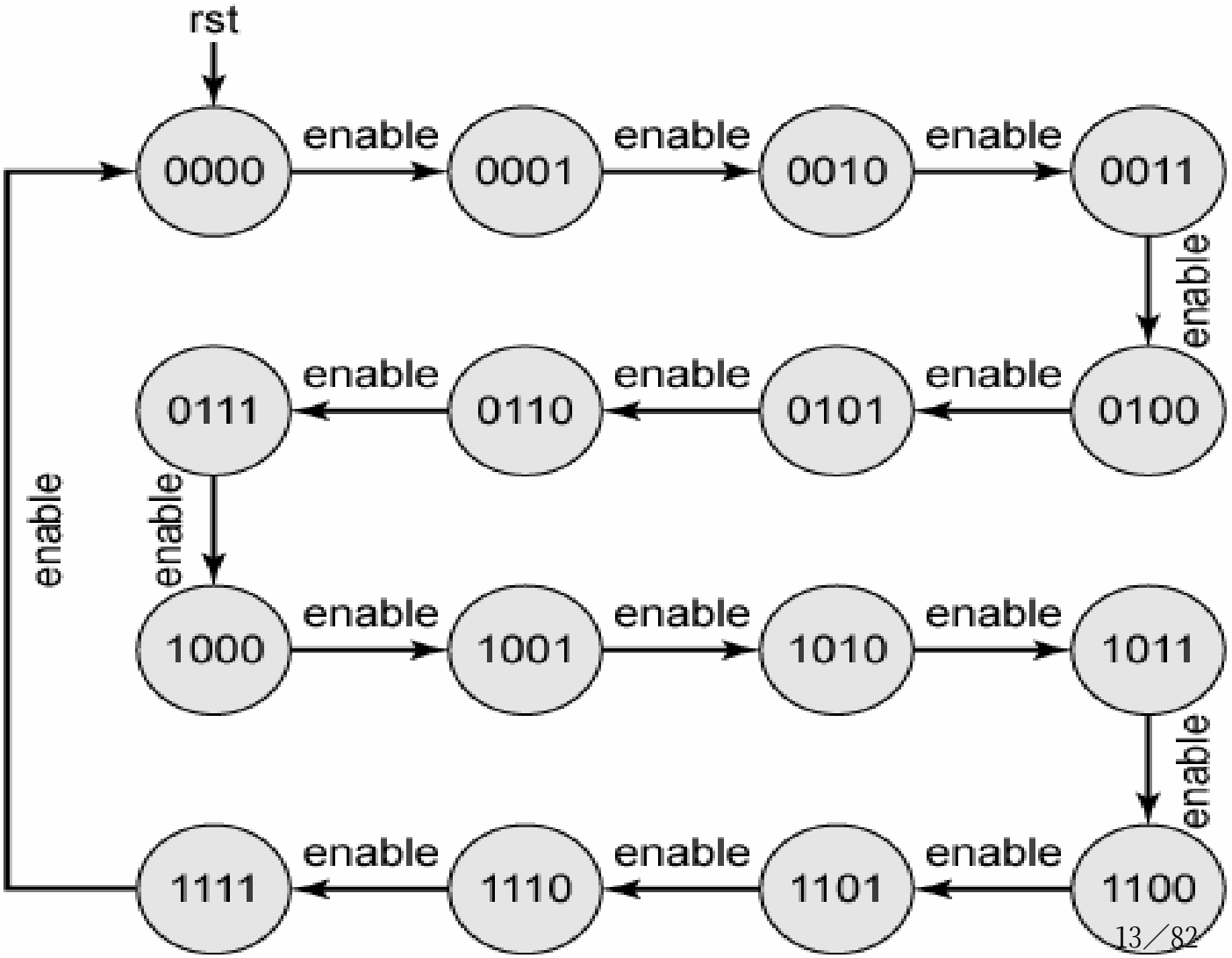
The architecture of separate datapath and control units



The functional elements

- (1) a 4-bit register to hold count
- (2) a mux that steers either count or the sum of count and to the input of the register
- (3) a 4-bit adder to increment count
- The signal **enable** must be asserted for counting to occur, and the signal **rst** overrides all activity and drives count to a value 0000_2

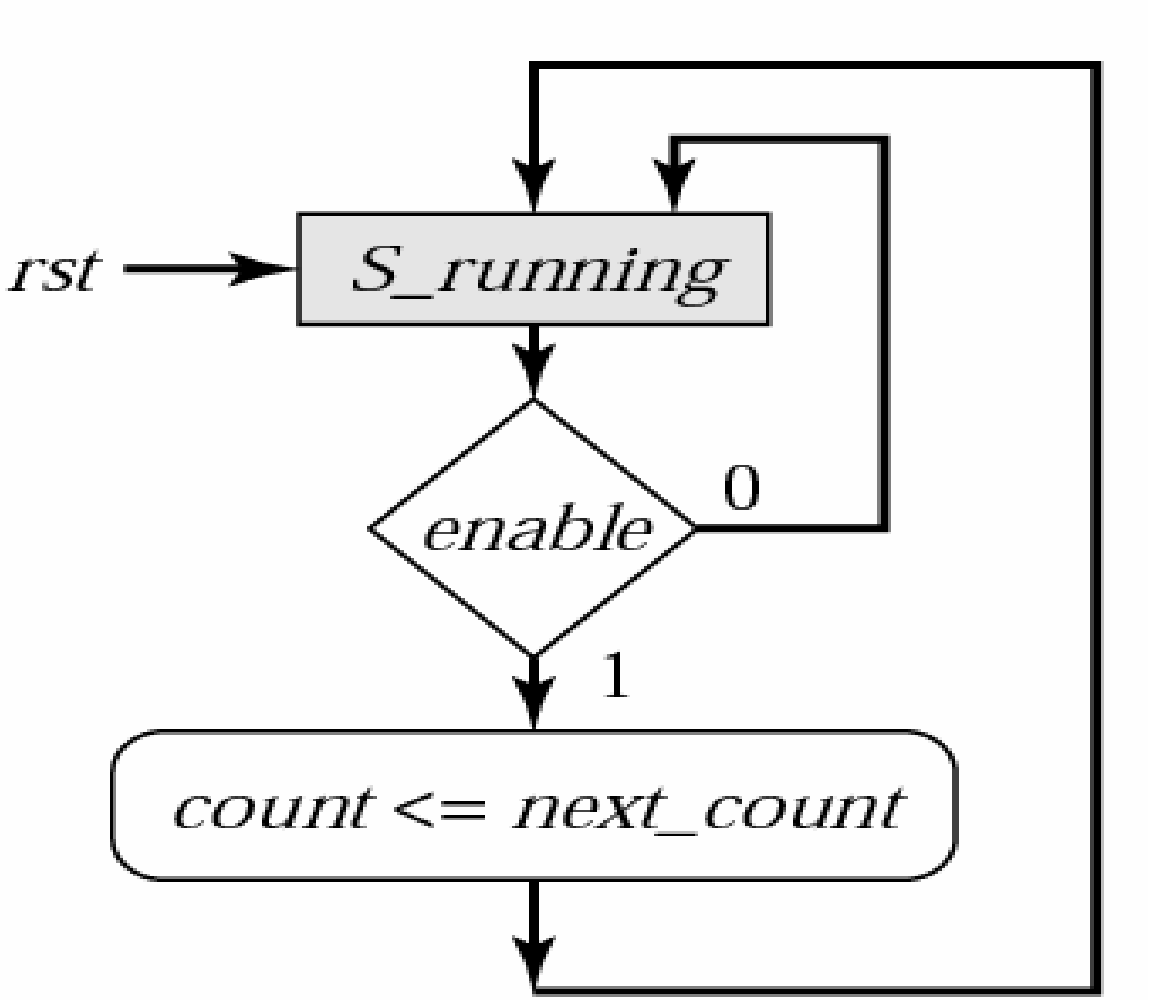
The simplified STG of the machine



Another view of the binary counter

- Another view of the binary counter **is based on the counter's activity**
- The machine **described by the ASM chart** in Figure 7-5, **Binary_Counter_ASM**, has one state, S_running.
- At every clock, enable is tested and a transition is made back to S_running;
- If enable is asserted, a conditional register operation that increments the counter is executed concurrently with the state transition

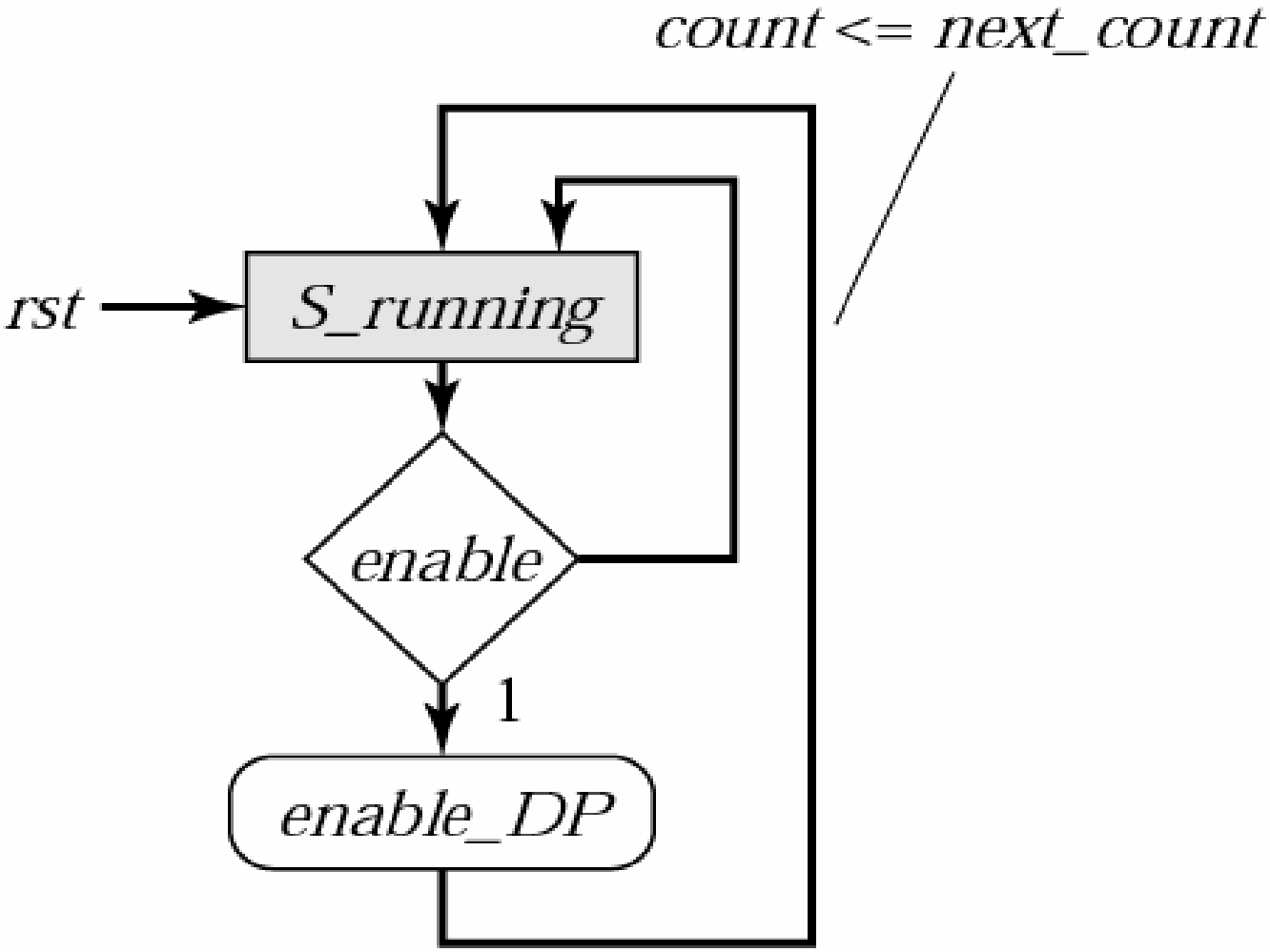
Figure 7-5 The binary counter: based on the counter's activity



A third method

- The designing of binary counter partitions the machine into a control unit and a datapath unit, but **designs a RTL behavioral model for the datapath unit, rather than a structural model**
- This style separates the design of the control unit from the design (and synthesis) of the datapath unit and simplifies the description of the datapath unit
- **It separates the unit that determines what happens from the unit that determines when it happens.**

A third method – ASMD (enable_DP linking the controller to the datapath)



Example 7.1

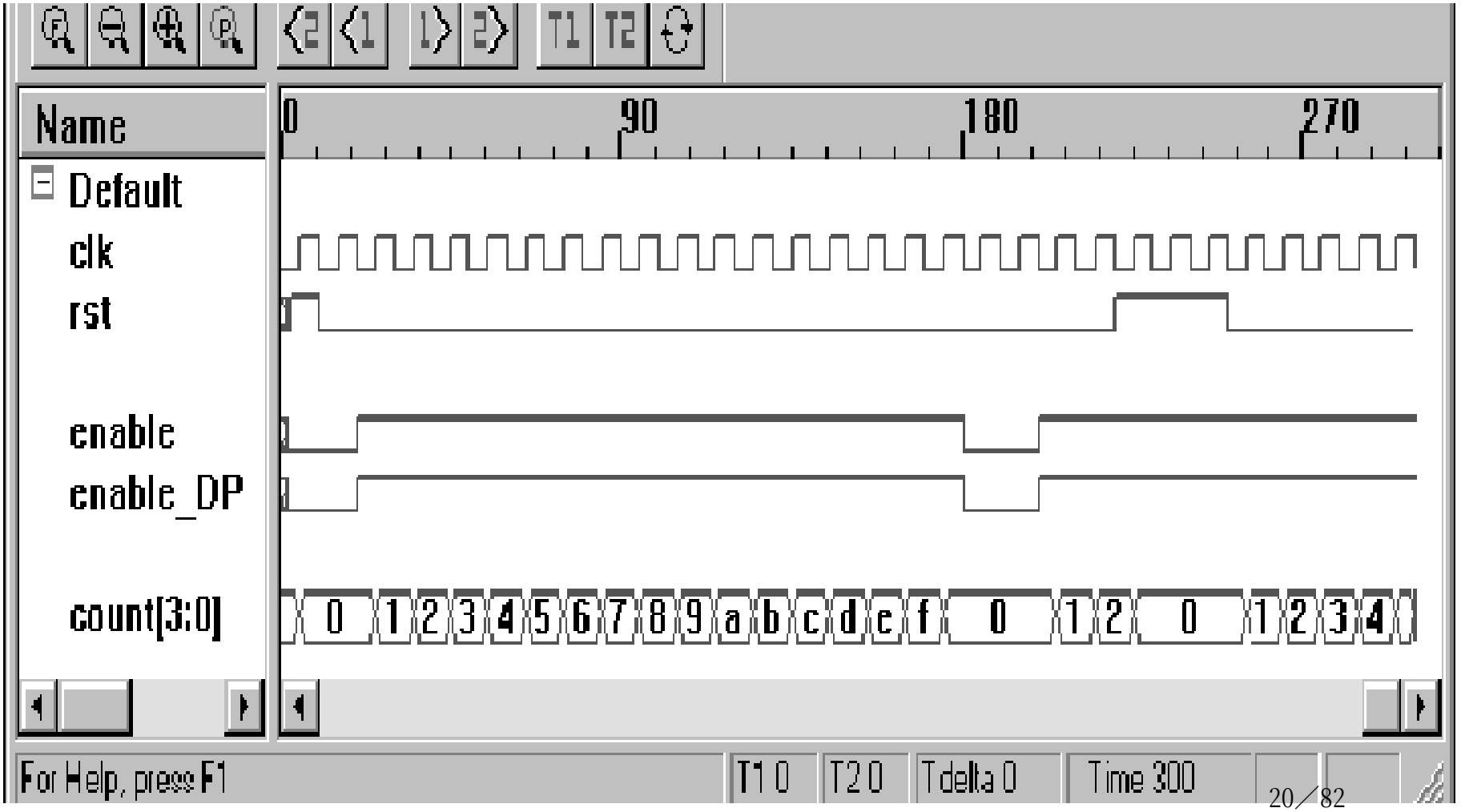
```
module Binary_Counter_Part_RTL (count, enable, clk, rst);
  parameter      size=4;
  output [size-1:0] count;
  input          enable;
  input          clk, rst;
  wire           enble_DP;
  Control_Unit M0 (enable_DP, enable, clk, rst);
  Datapath_Unit M1 (count, enable_DP, clk, rst);
endmodule

module Control_Unit (enable_DP, enable, clk, rst);
  output enable_DP;
  input enable;
  input clk, rst;                    // not needed
  wire enable_DP=enable;             // pass through
endmodule
```

```
module Datapath_Unit (count, enable, clk, rst);
  parameter           size=4;
  output [size-1:0]   count;
  input                       enable;
  input                       clk, rst;
  reg                 count;
  wire   [size-1:0]   next_count;
  always @ (posedge clk)
    if (rst==1) count<=0;
     else if (enable==1) count<=next_count(count);

  function [size-1:0] next_count;
  input [size-1:0] count;
   begin
    next_count=count+1;
   end
  endfunction
endmodule
```

Simulation result

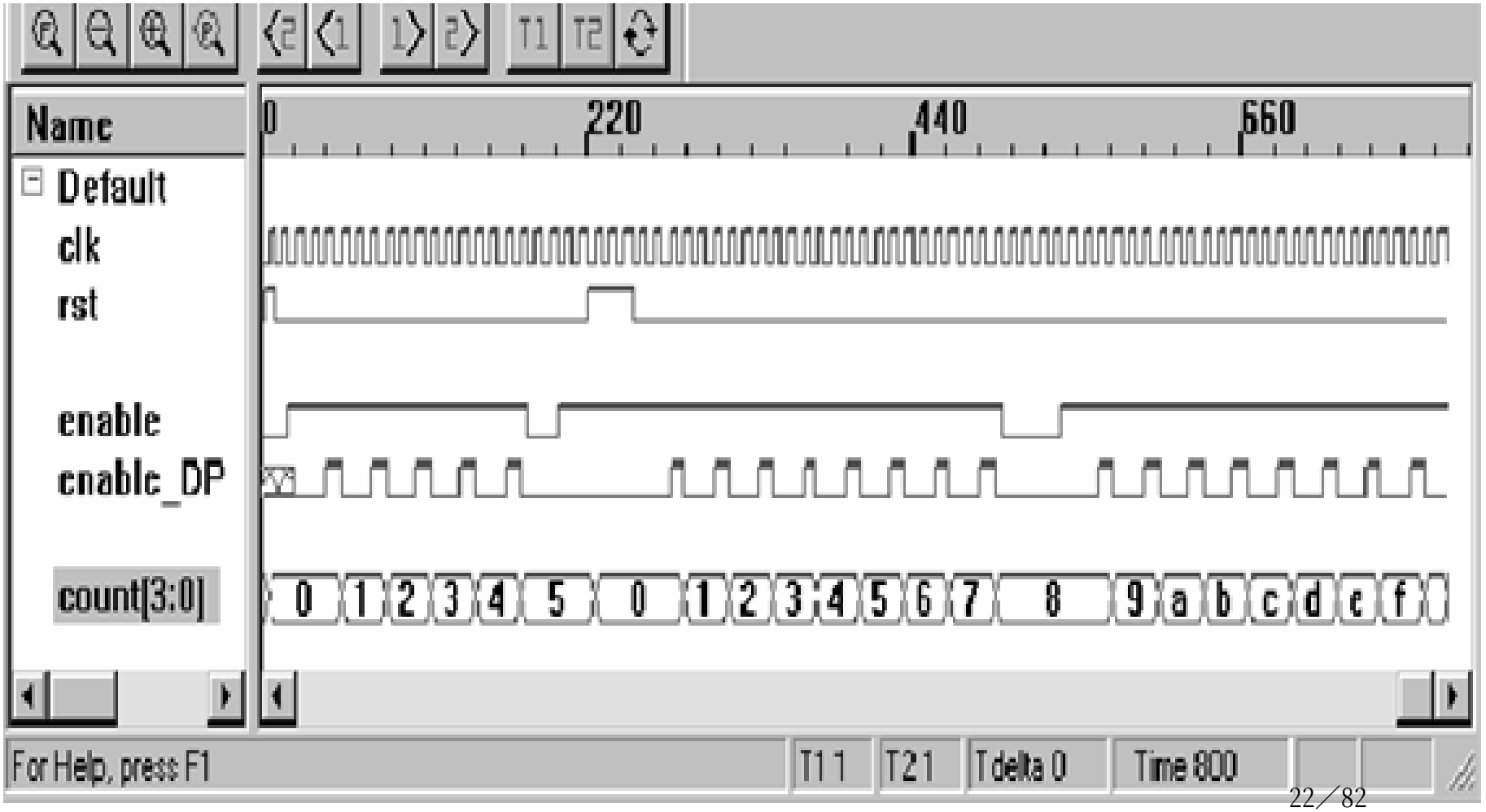


A counter which increments its count every third clock cycle:

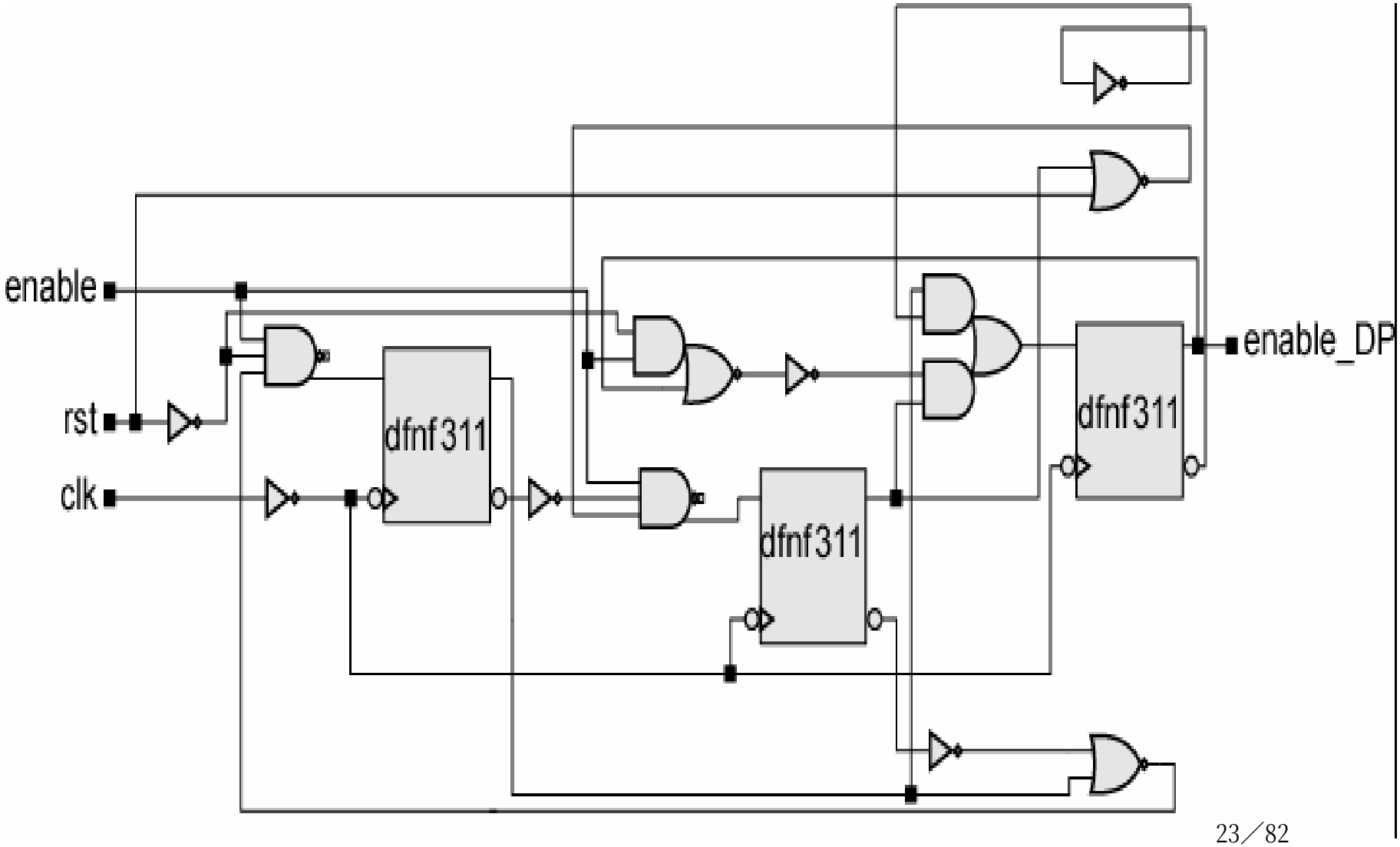
Only change the control unit

```
module Control_Unit_by_3  (enable_DP, enable, clk, rst);
      output            enable_DP;
      input enable;
      input clk, rst;                  // Not needed
      reg   enable_DP;
      always begin: Cycle_by_3
        @ (posedge clk) enable_DP  <= 0;
         if ((rst == 1) || (enable != 1)) disable Cycle_by_3; else
           @ (posedge clk)
            if ((rst == 1) || (enable != 1)) disable Cycle_by_3;
             else
               @ (posedge clk)
                 if ((rst == 1) || (enable != 1))
                      disable Cycle_by_3;
                 else
                     enable_DP <= 1;
       end // Cycle_by_3
endmodule
```

Simulation result



The synthesis results



7.3 RISC Architecture

- **Architectures of Microprocessors**
  - Random logic
  - Micro Code
  - Pipeline
- **Instruction Sets**
  - **Reduced instruction-set computers (RISC)**
  - **Complicated instruction-set computers (CISC)**

RISC Architecture

- **Reduced instruction-set computers (RISC)** are designed to have **a small set of instructions that execute in short clock cycles**, with a small number of cycles per instruction
- RISC machines are optimized to achieve efficient pipelining of their instruction streams

RISC Architecture

- Reduced Instruction Set Computer
  - Contains limited instruction set
  - Remove the complexity of micro-memory
  - Programs may be longer
- Examples:

  Power PC series, IBM, Motorola,ARM…

The machine consists of the functional units

- **a processor, a controller, and memory**
- Program instructions and data are stored in memory
- In program-directed operation, instructions are fetched synchronously from memory, decoded, and executed to:

(1) operate on data within the arithmetic and logic unit (ALU)

(2) change the contents of storage registers

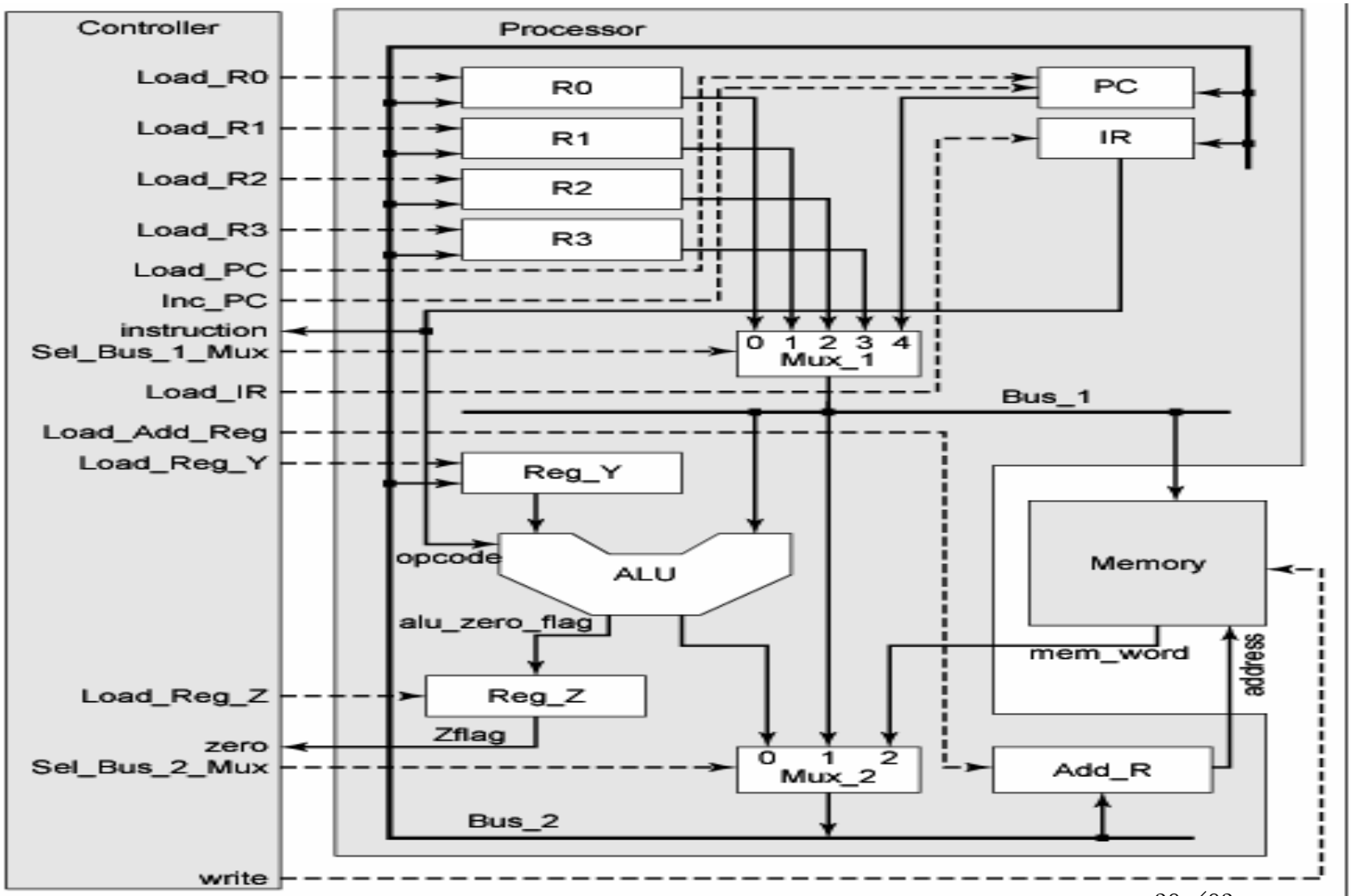
(3) change the contents of the program counter (PC), instruction register (IR) and the address register (ADD_R)

(4) change the contents of memory

(5) retrieve data and instructions from memory, and

(6) control the movement of data on the system busses

A stored-program RISC-architecture machine



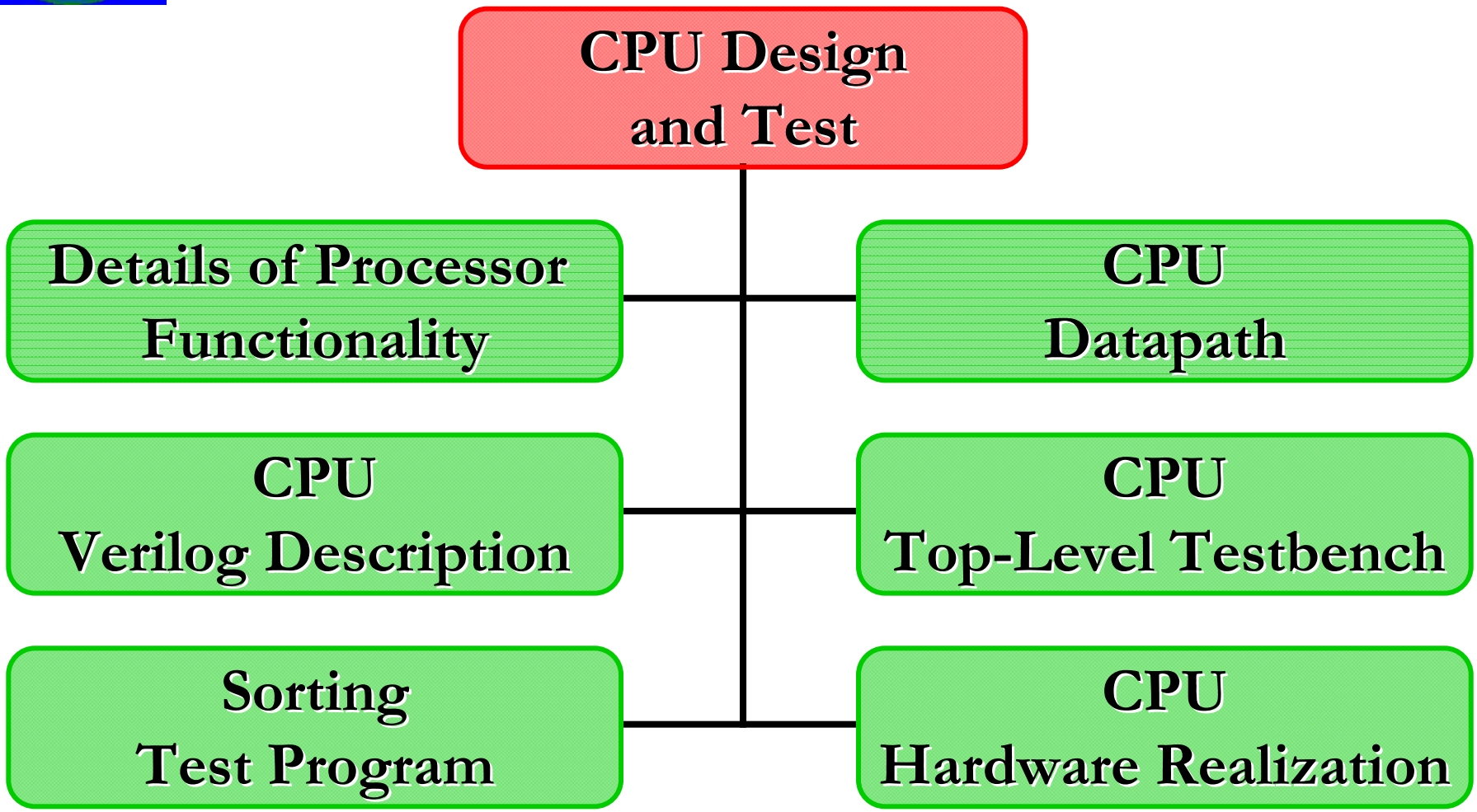
A stored-program RISC-architecture machine

Processor includes :

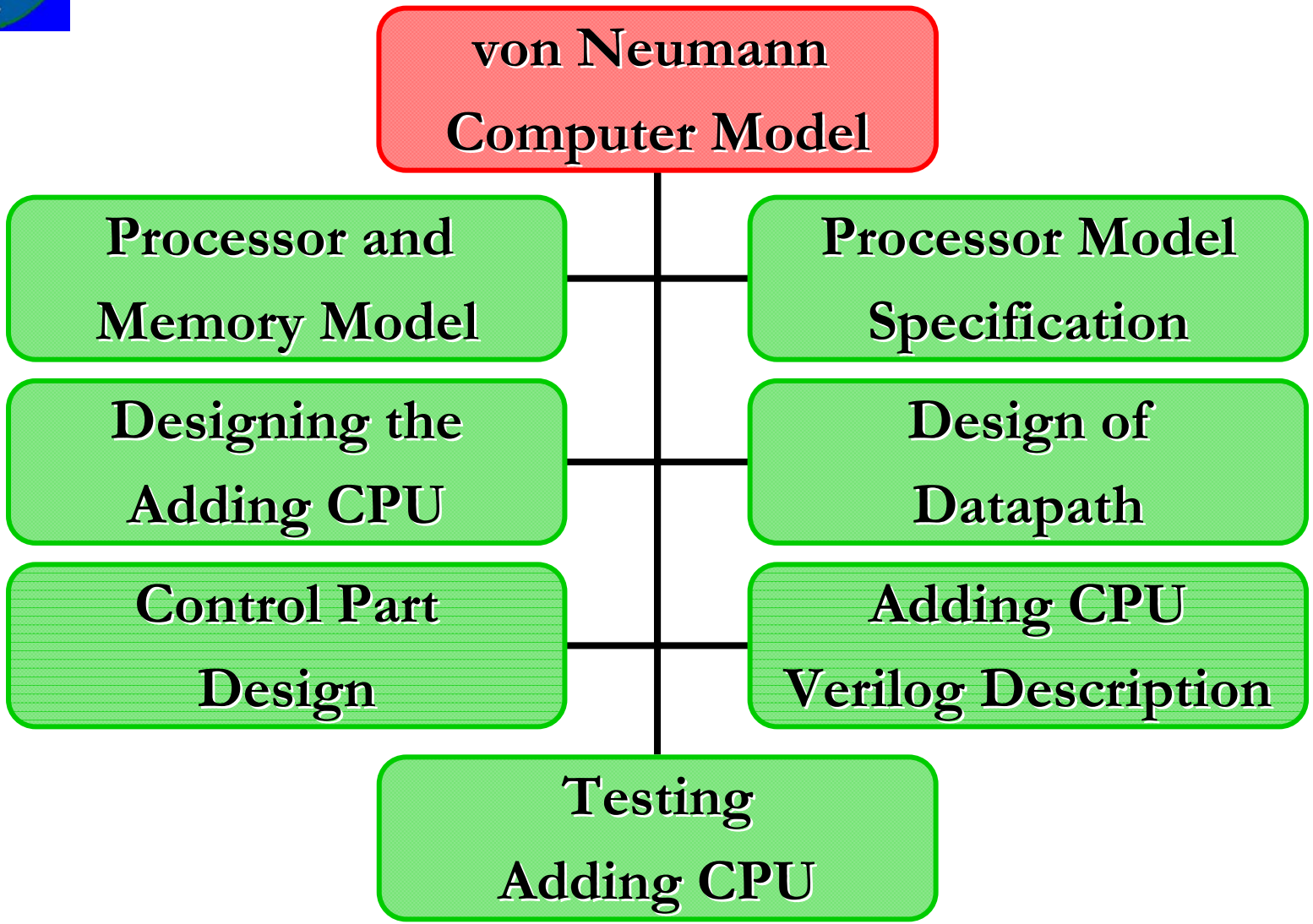
registers, datapaths, control lines, and an ALU ,...

- **ALU** : two operand datapaths, and its instruction set Instruction Action
- **Controller**: steer data to the proper destination, according to the instruction being executed ,fetch, decode, and execute
- **Instruction Set** :long or short, depending on

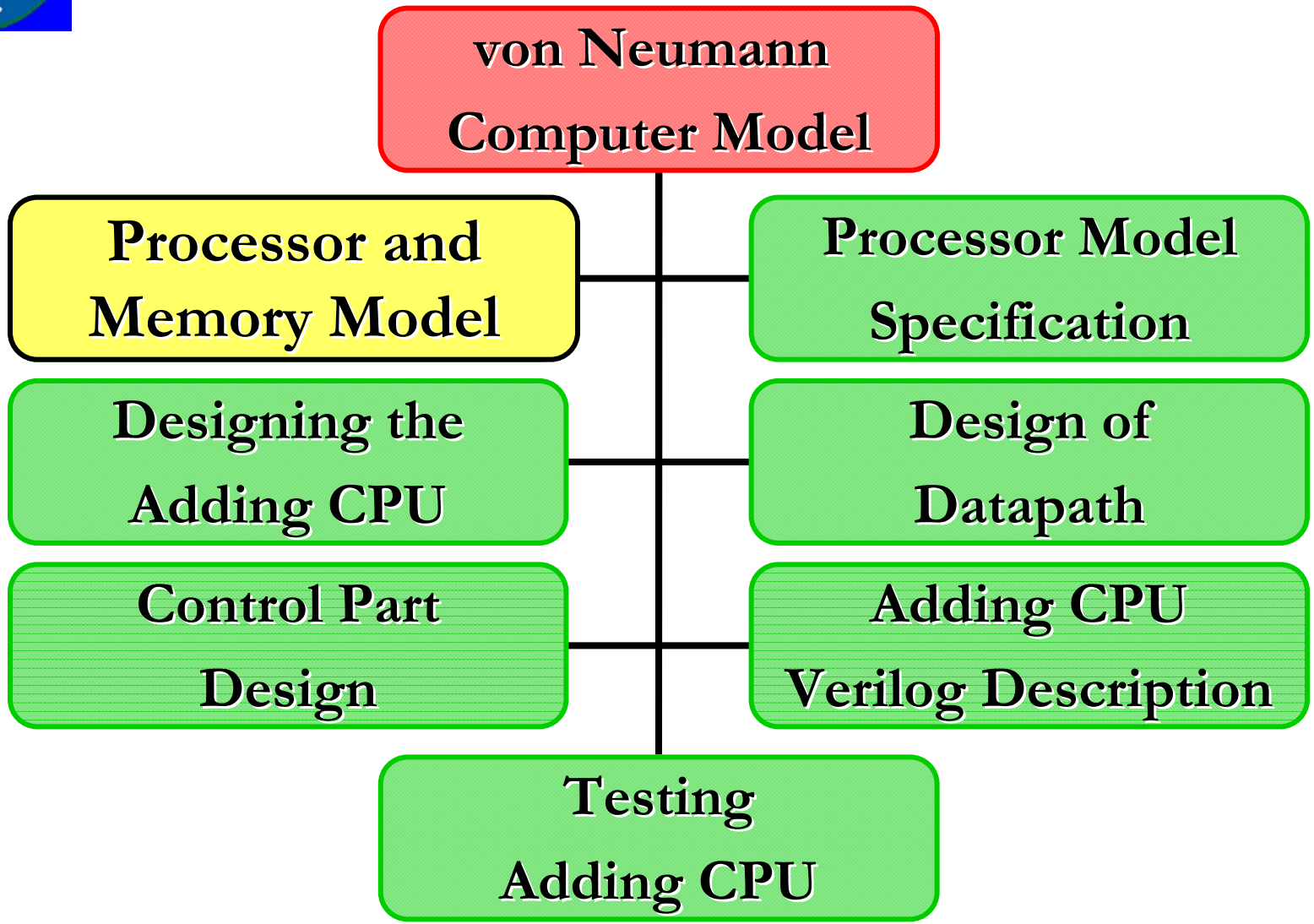
7.4 CPU Design and Test



von Neumann Computer Model

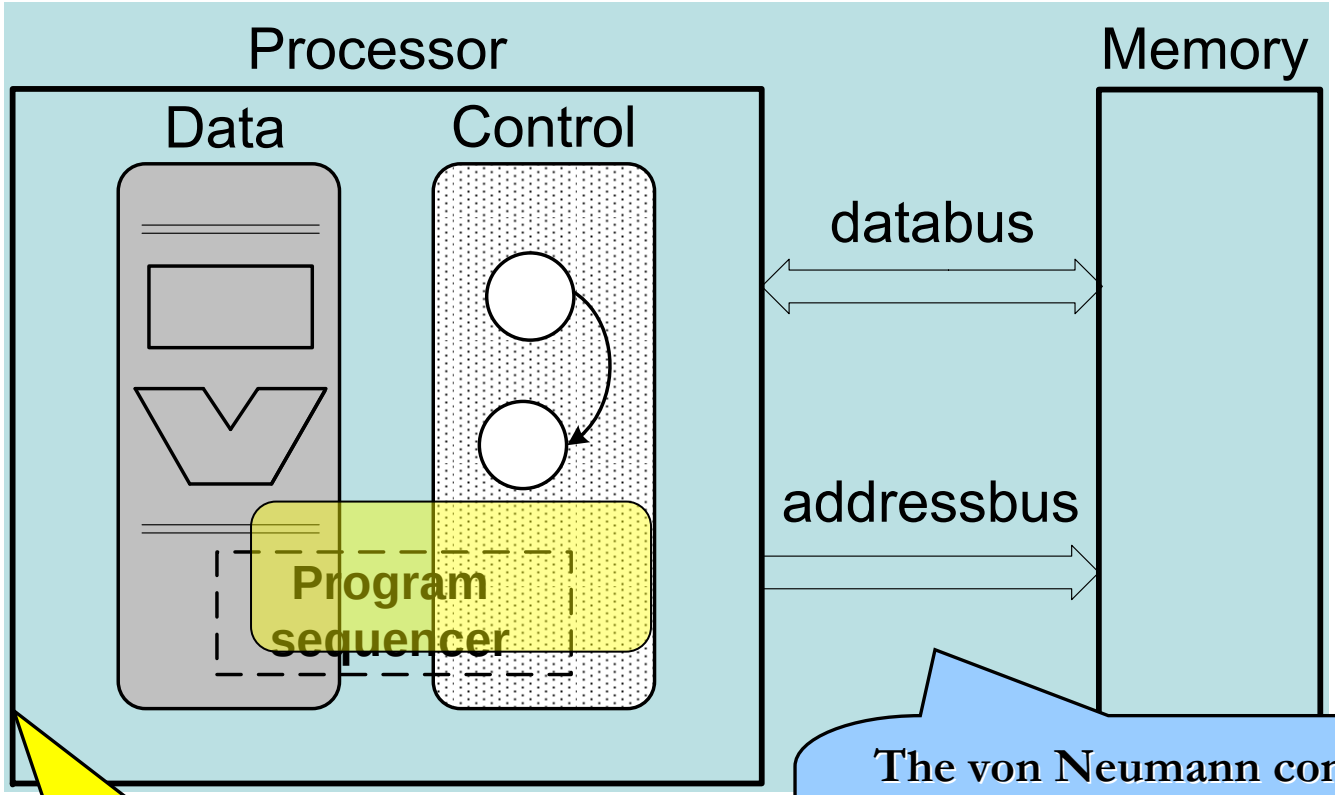


Processor and Memory Model



Processor and Memory Model

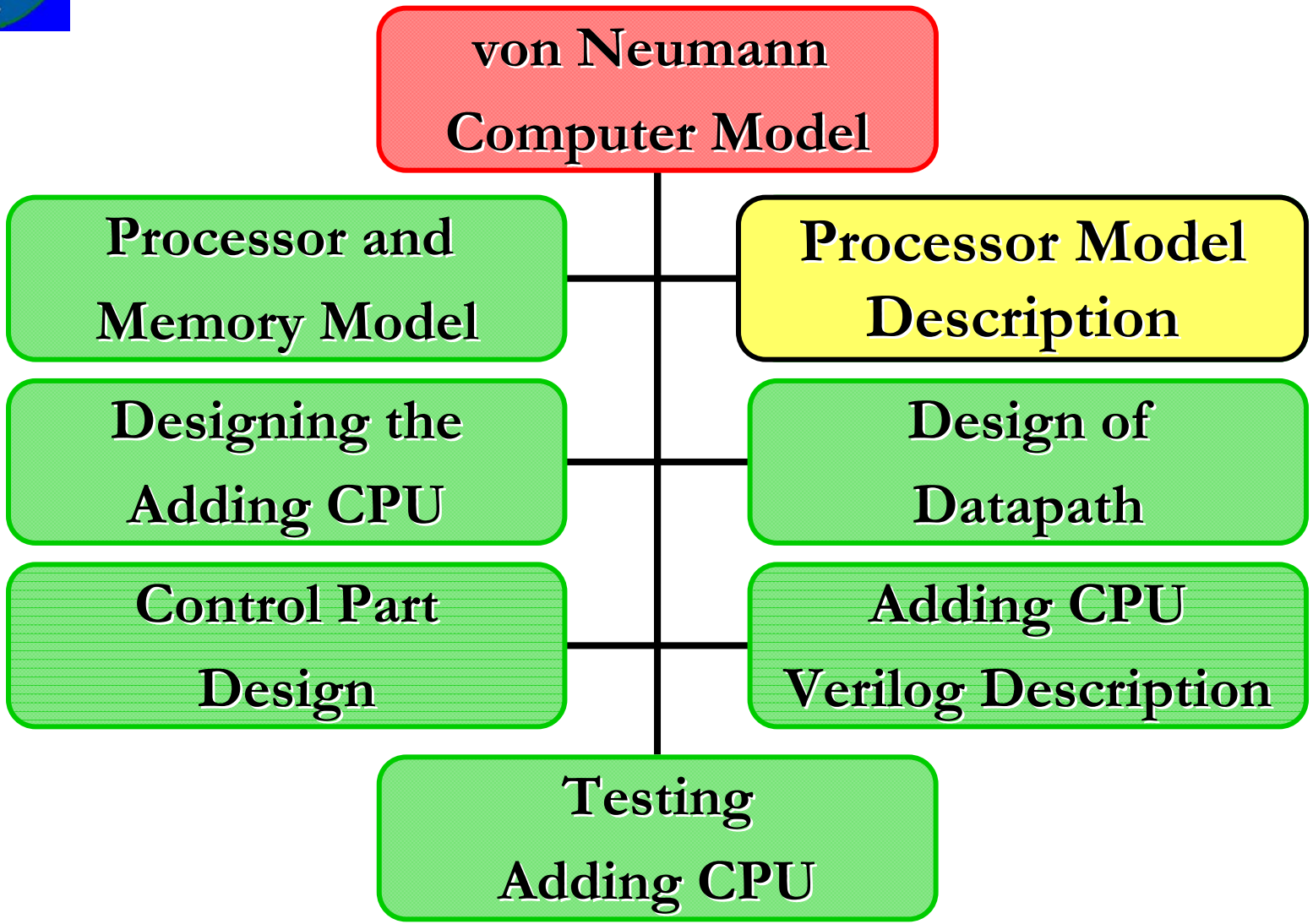
von Neumann Process Model



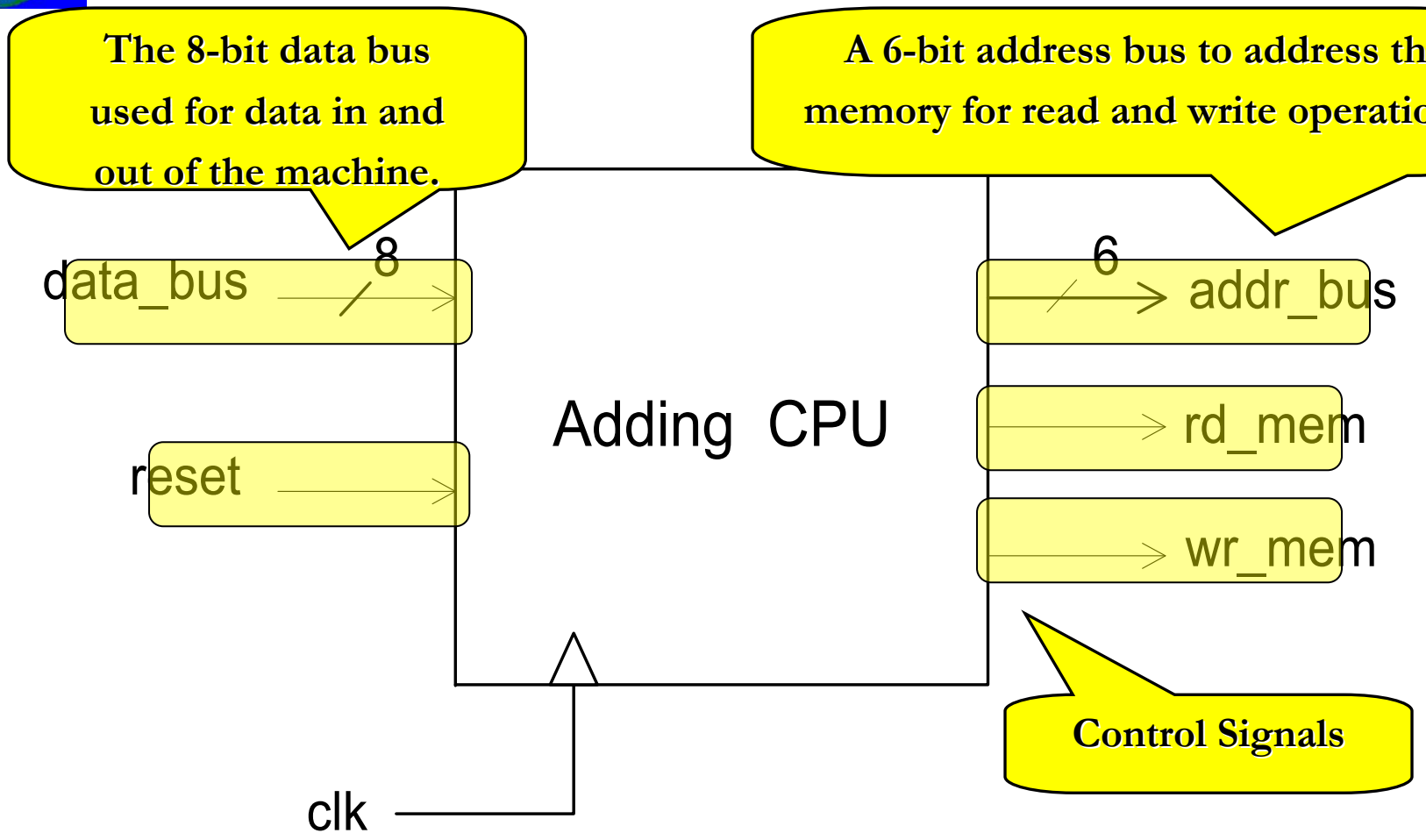
The von Neumann computer model is based on a processor using instructions and data from a single memory.

Using a sequencer, the processor fetches instructions from its memory.

Processor Model Description

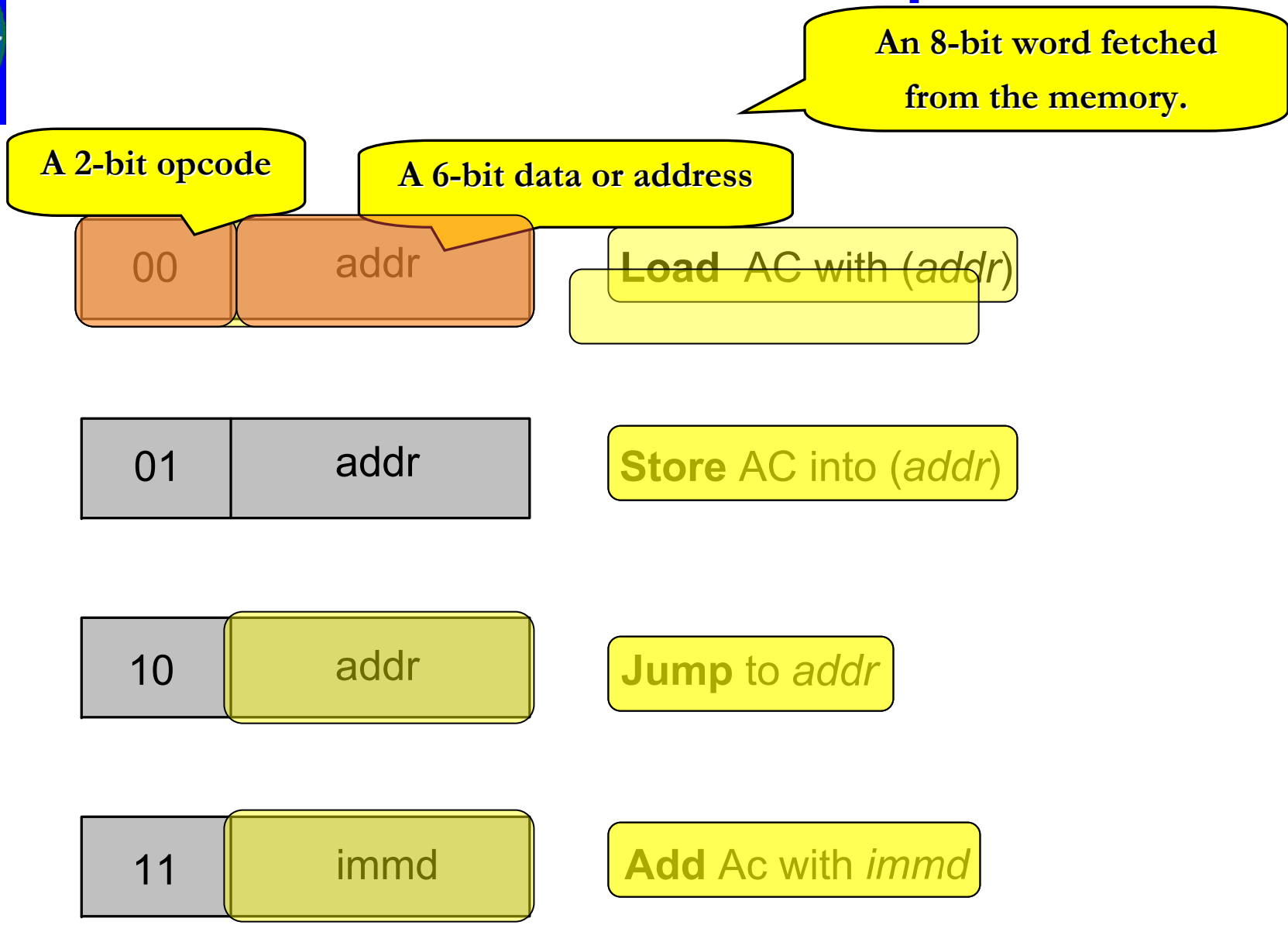


Processor Model Description



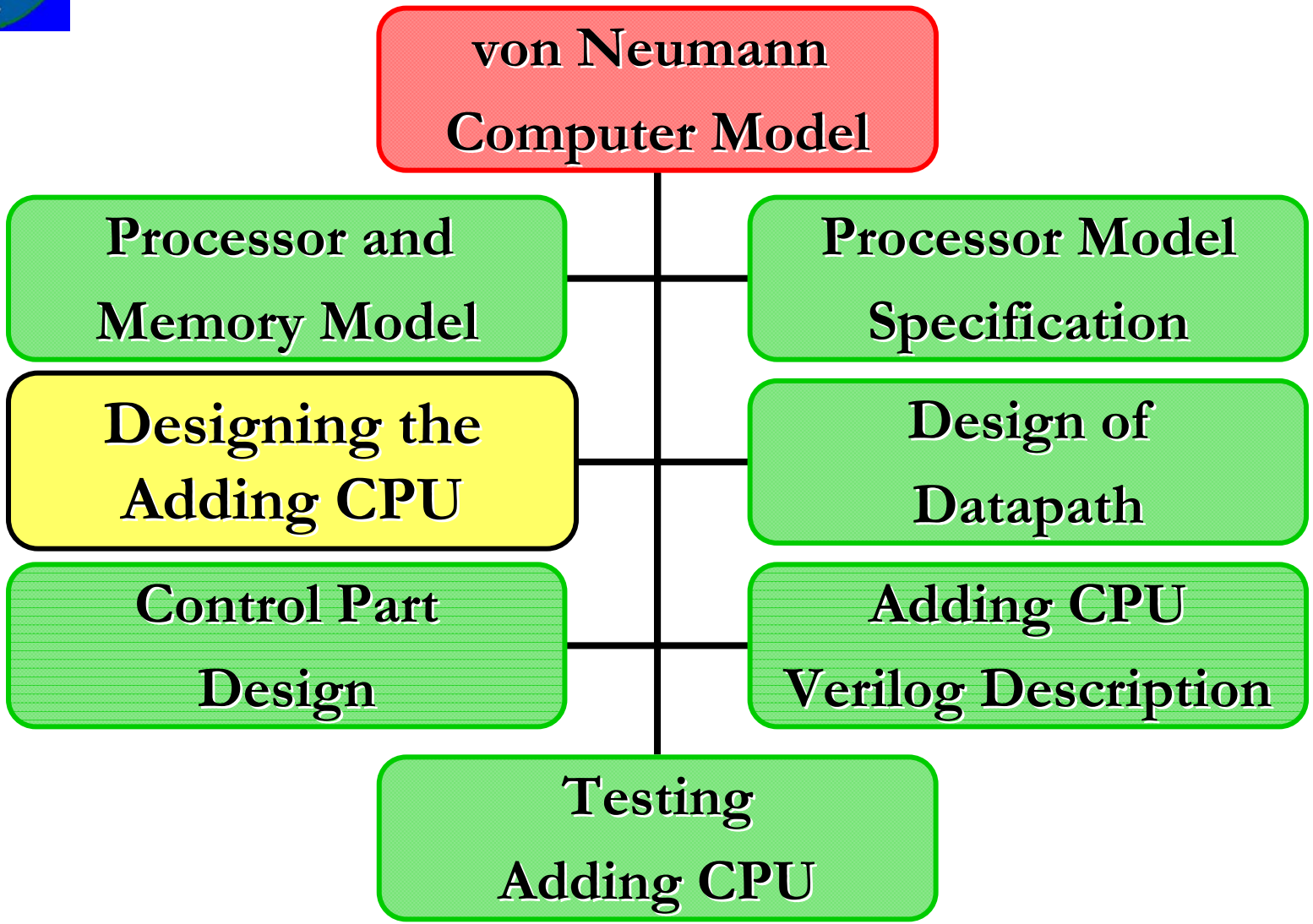
- **Interface of the Adding CPU Example**

Processor Model Description



- **Instruction Format**

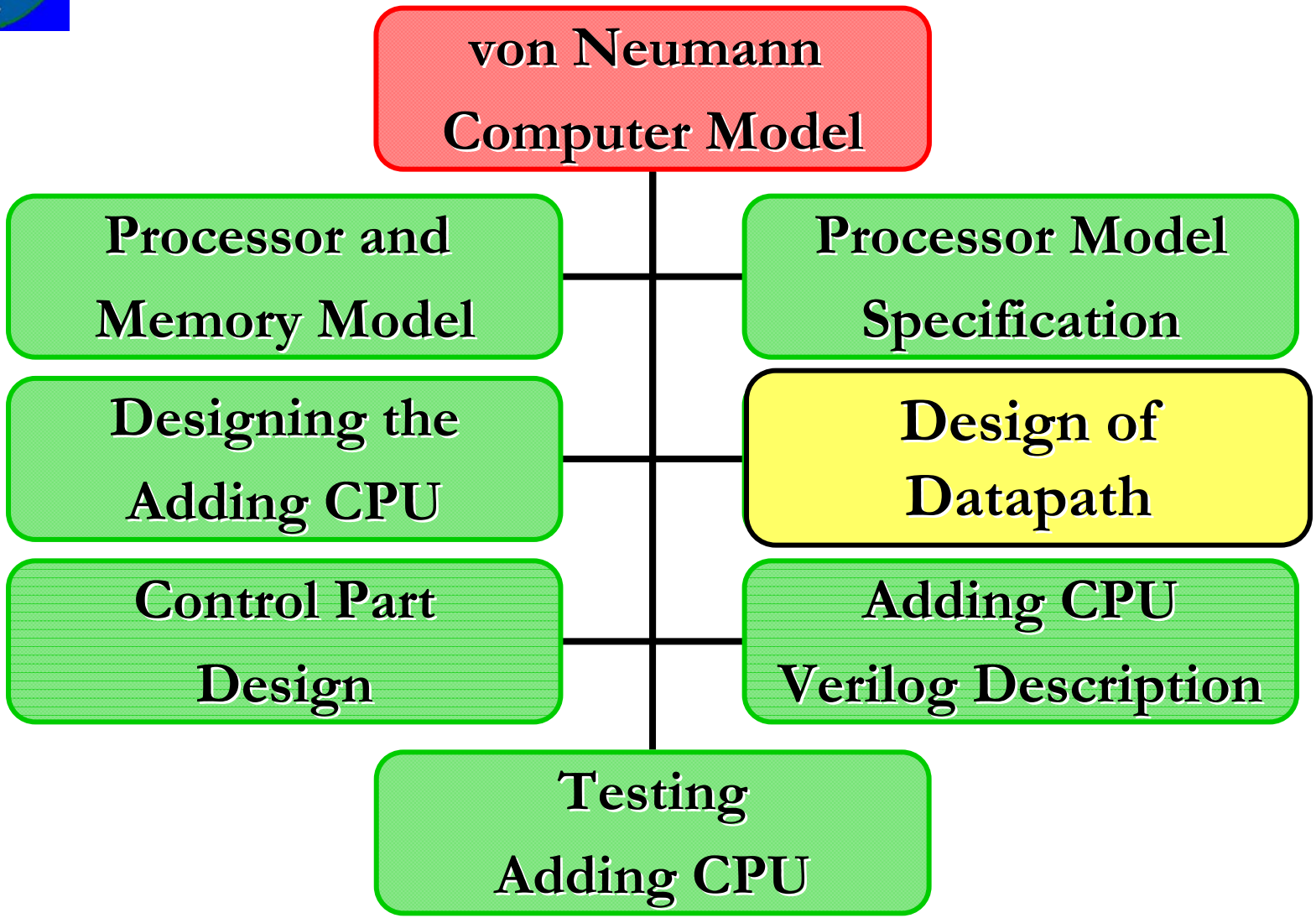
Designing the Adding CPU



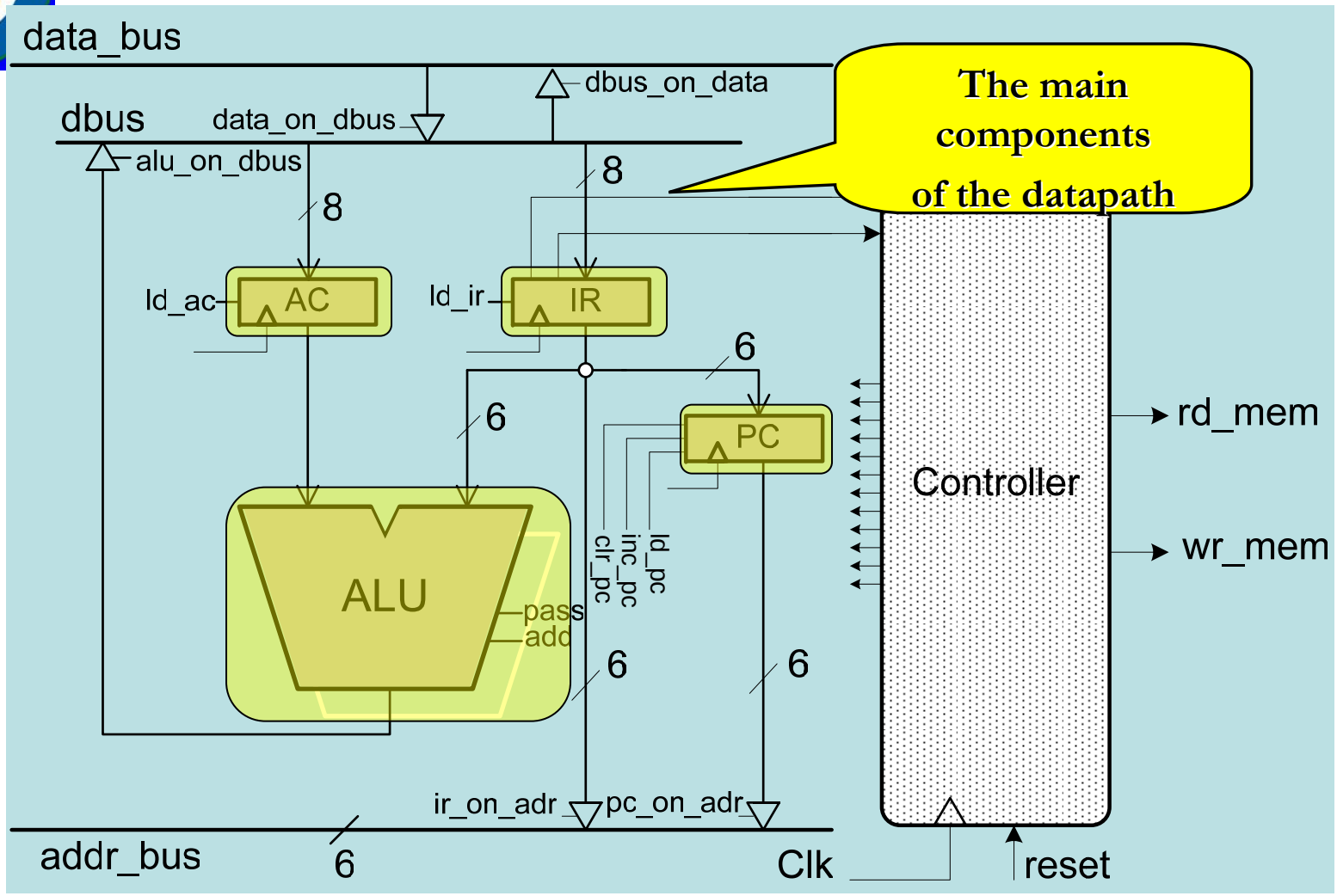
Designing the Adding CPU

- The first step in the design of our adding machine:
  - Decide on its **data and control partitioning**
  - Decide what goes into its data part and what behavior is expected from its controller
- **The datapath of the design has:**
  - The *AC* register for keeping data to operate on
  - The *PC* register to keep track of the address being fetched
  - An adder unit to perform the addition
  - An instruction register (*IR*) for storing the most recent instruction fetched
- **The controller part** is a state machine that **looks at the opcode** of the instruction in IR and **decides** on how data is to be routed.

Design of Datapath



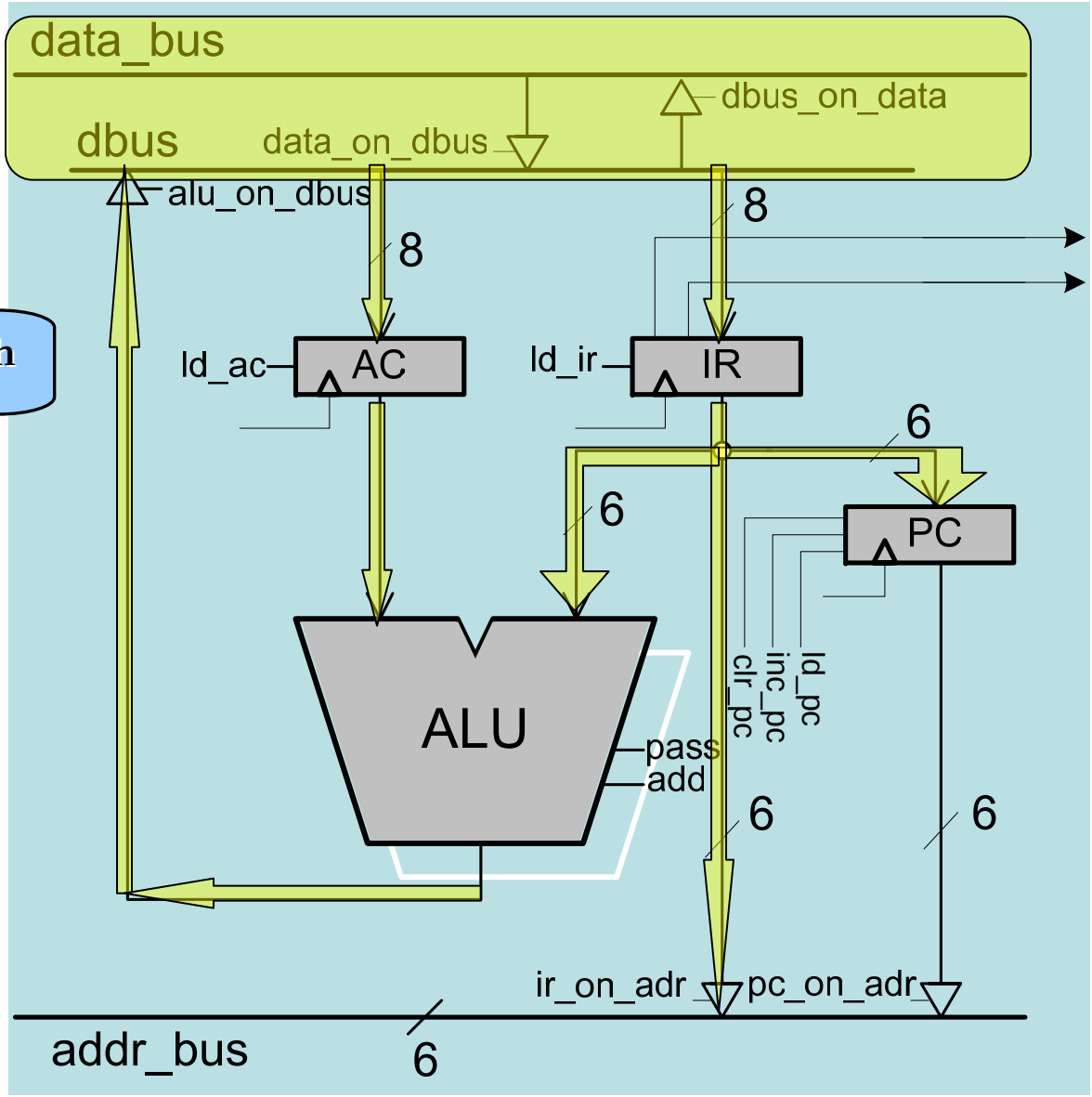
Design of Datapath



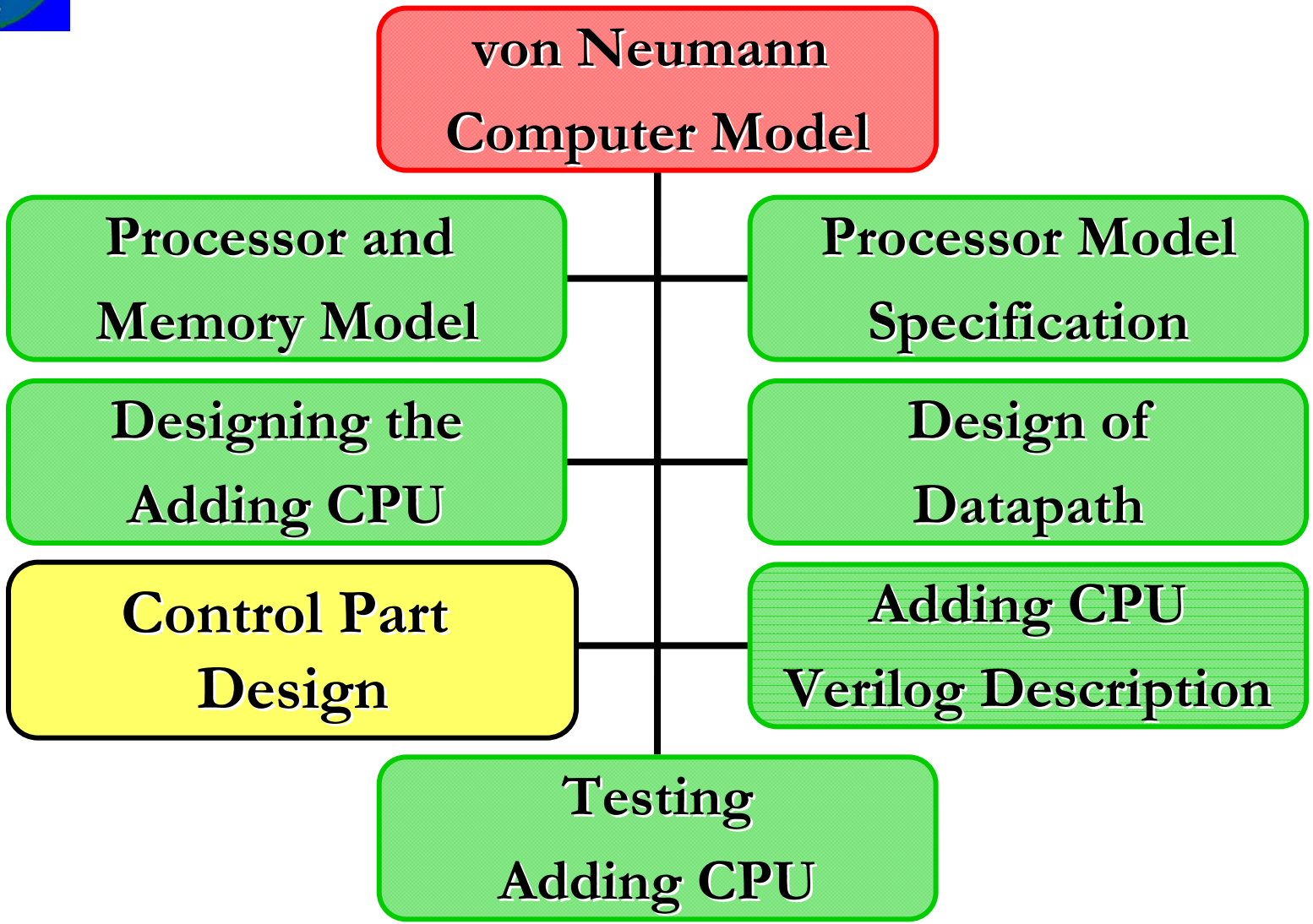
- **Architectural Design of our Adding Machine**

Design of Datapath

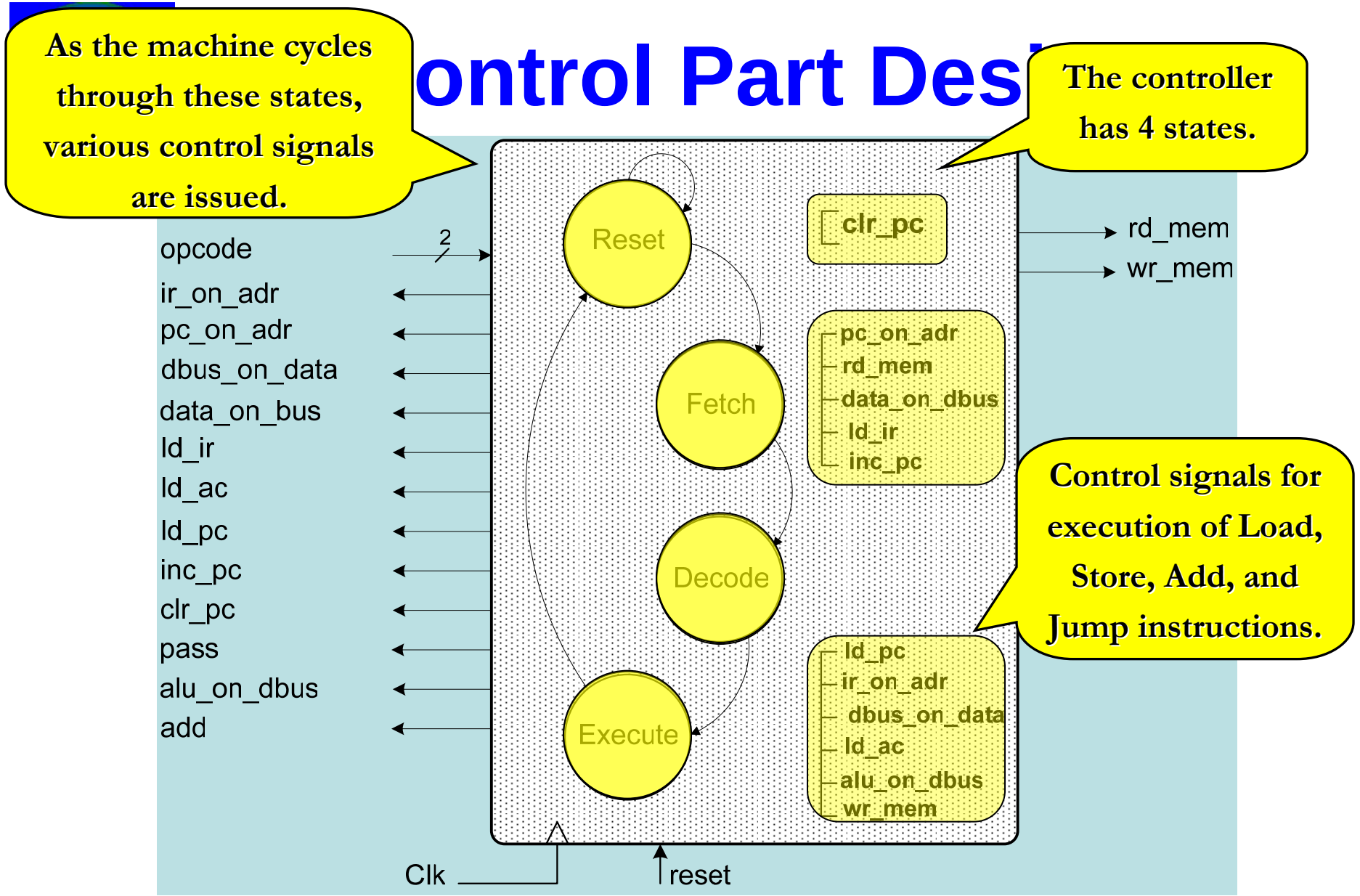
The Datapath



Control Part Design

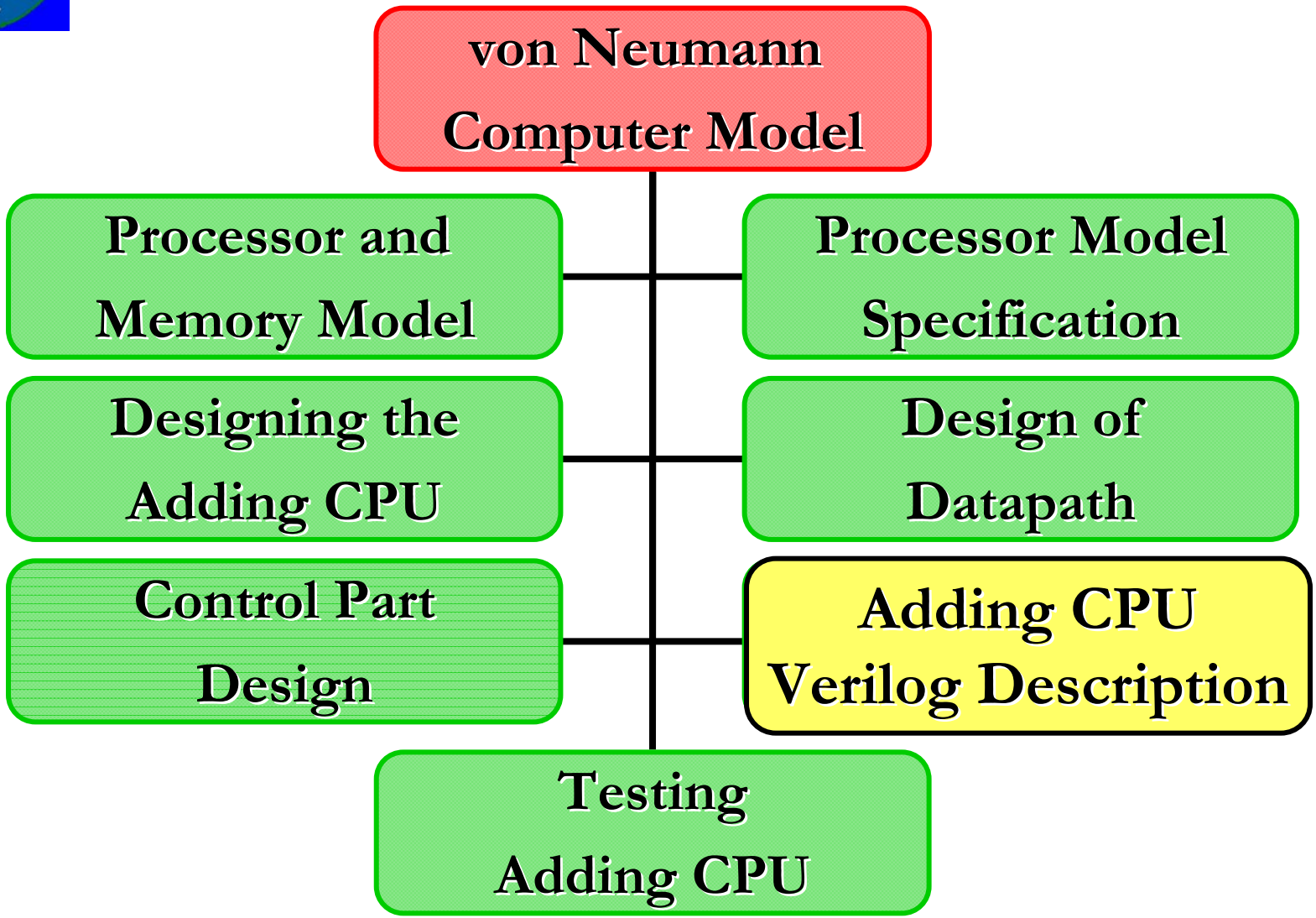


ontrol Part Desi



- **Controller of adding CPU**

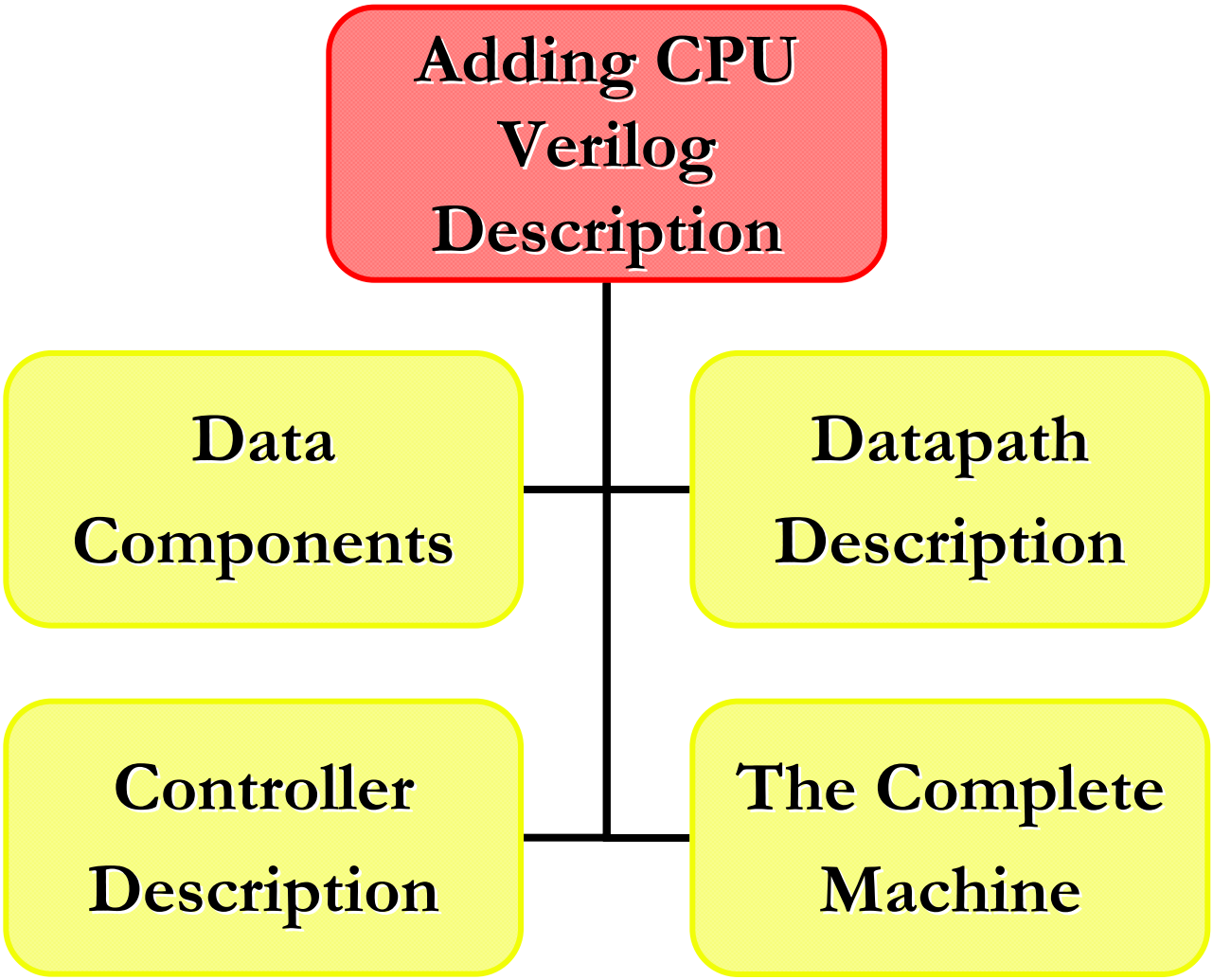
Adding CPU Verilog Description



Adding CPU Verilog Description

- Developing the complete Verilog code of our adding machine by:
  - Describing components of the datapath
  - Forming the Verilog code of the datapath by instantiating and wiring these components
  - Describing the controller using a state machine coding style
  - At the end, wiring datapath and controller in a top-level Verilog module

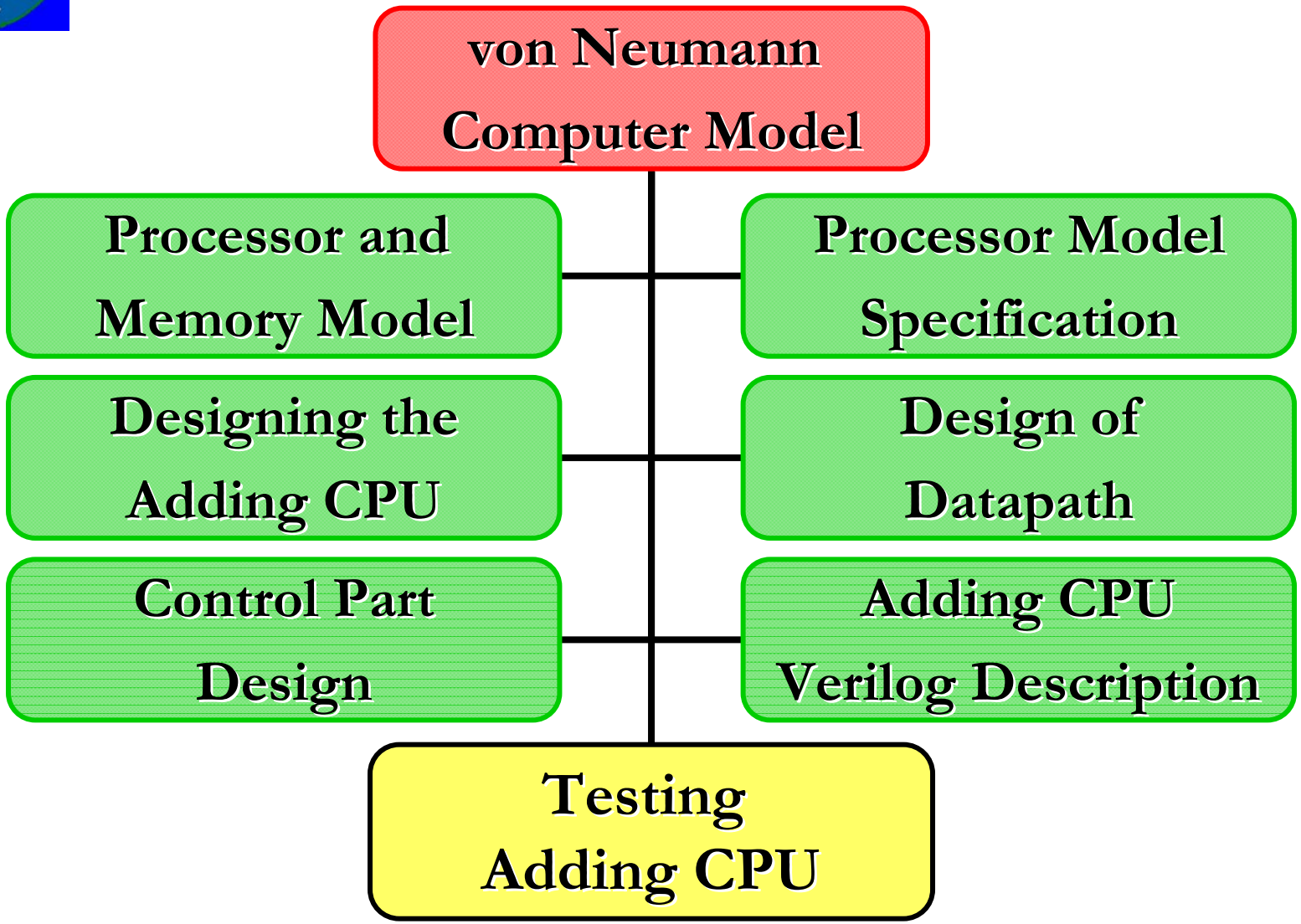
Adding CPU Verilog Description



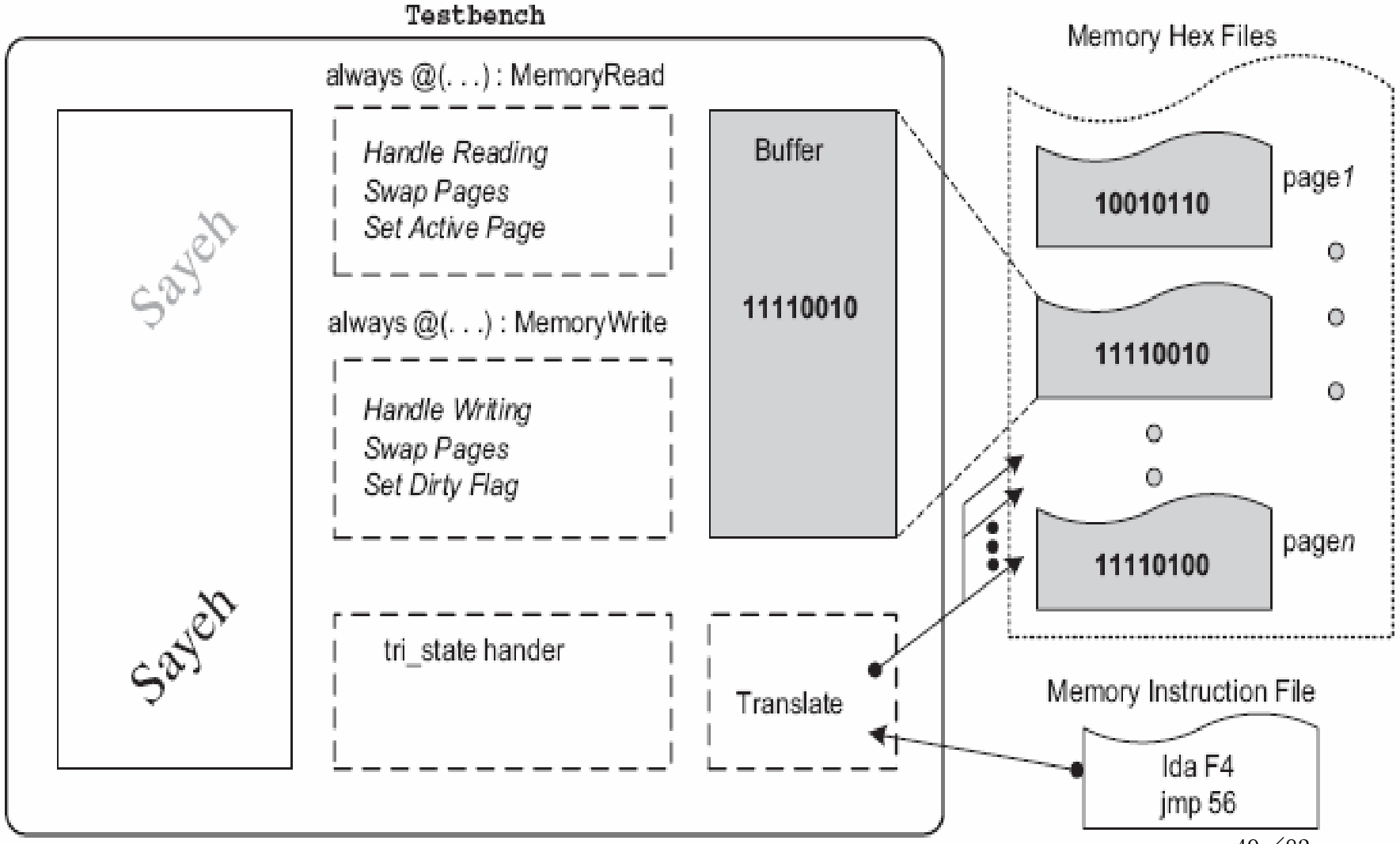
Datapath Components of Adding Machine

```
module AC ( input [7:0] data_in, input load, clk,
            output reg [7:0] data_out );
    always @( posedge clk )
       if( load ) data_out <= data_in;
endmodule
//
//
module PC ( input [5:0] data_in, input load, inc, clr, clk,
            output reg [5:0] data_out );
    always @( posedge clk )
       if( clr ) data_out <= 6'b000_000;
       else if( load ) data_out <= data_in;
       else if( inc ) data_out <= data_out + 1;
endmodule
//
//
module IR ( input [7:0] data_in, input load, clk,
            output reg [7:0] data_out );
    always @( posedge clk )
       if ( load ) data_out <= data_in;
endmodule
//
//
module ALU ( input [7:0] a, b, input pass, add,
             output reg[7:0] alu_out );
    always @(a or b or pass or add)
        if (pass) alu_out = a;
        else if (add) alu_out = a + b;
        else alu_out = 0;
endmodule
```

Testing Adding CPU



Graphical Representation of CPU Testbench



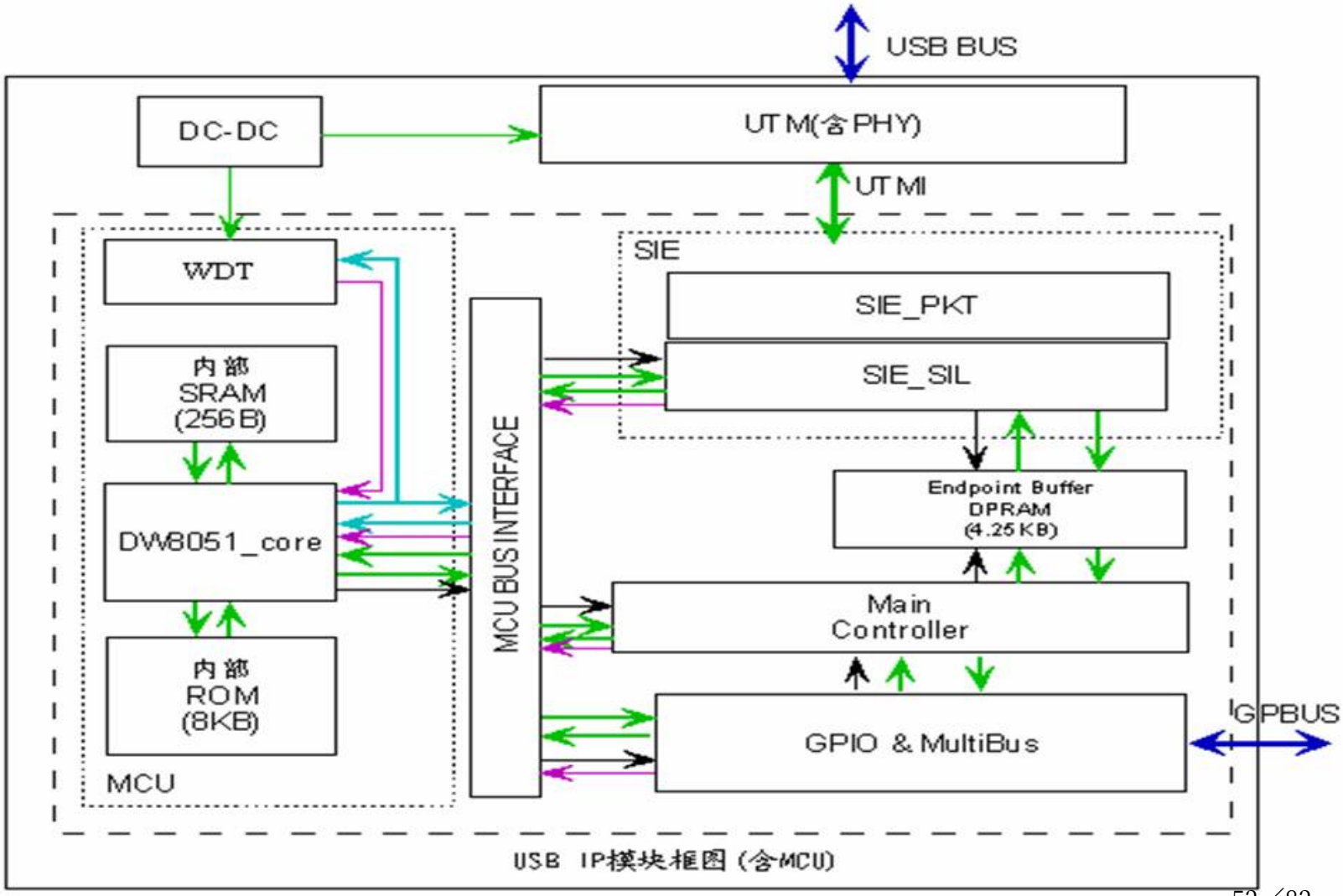
CPU test

- Instruction translation.
- Memory read procedure.
- Memory write procedure.
- Sorting test program
- CPU hardware realization

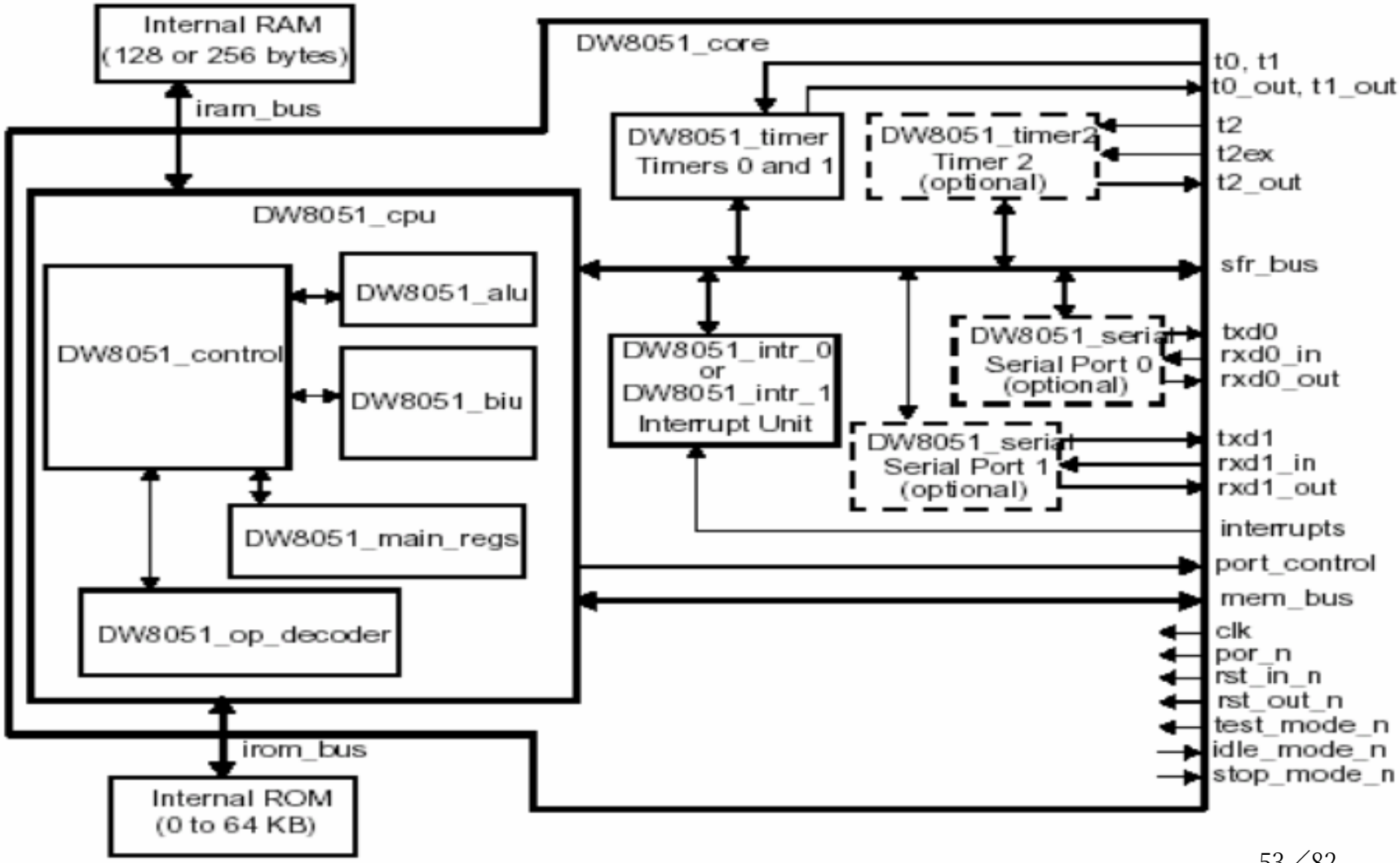
7.5 Design examples

- USB2.0 interface IP CORE
- Wireless Sensor Network node controller

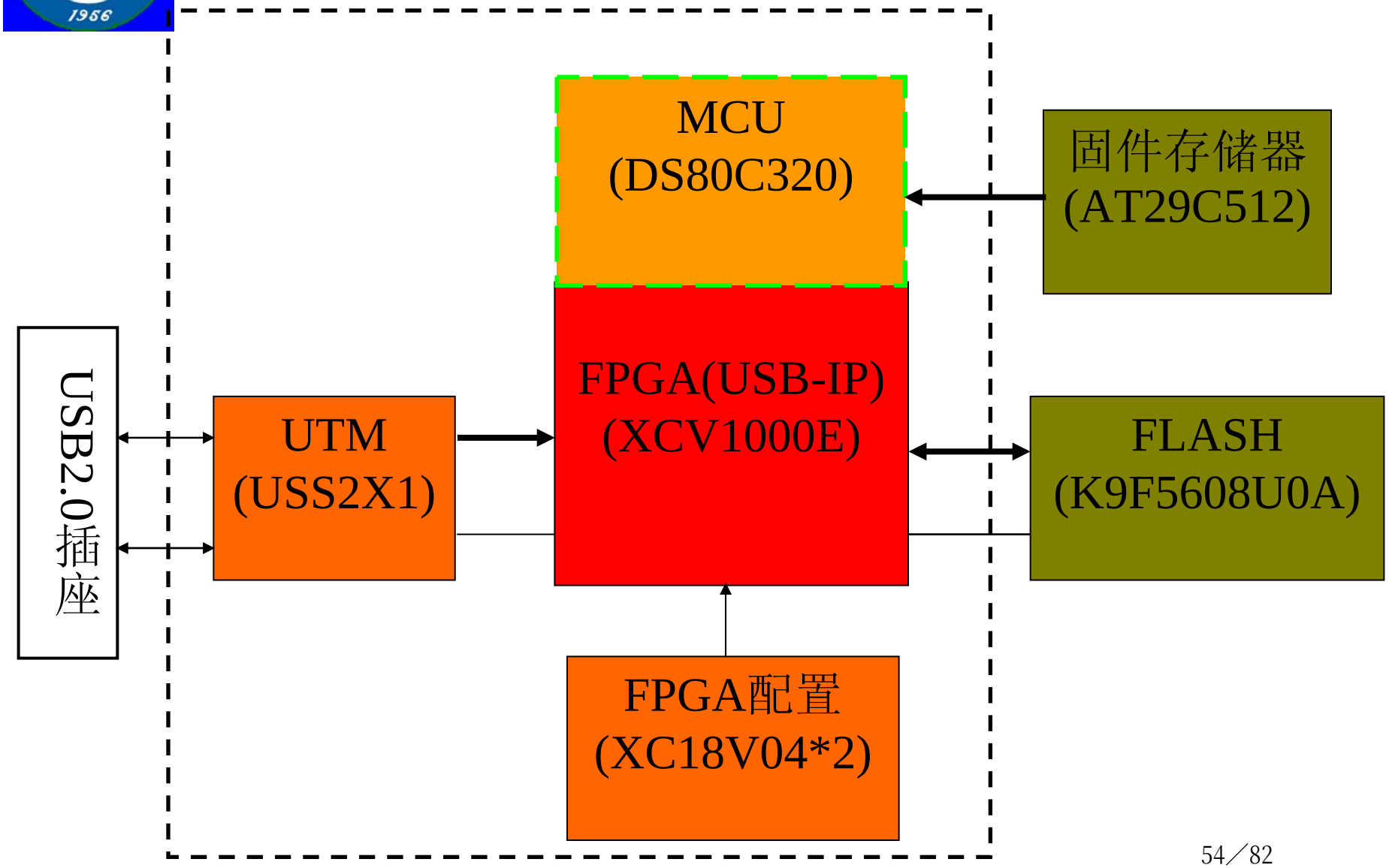
1. USB2.0 IP CORE



8051 CPU CORE

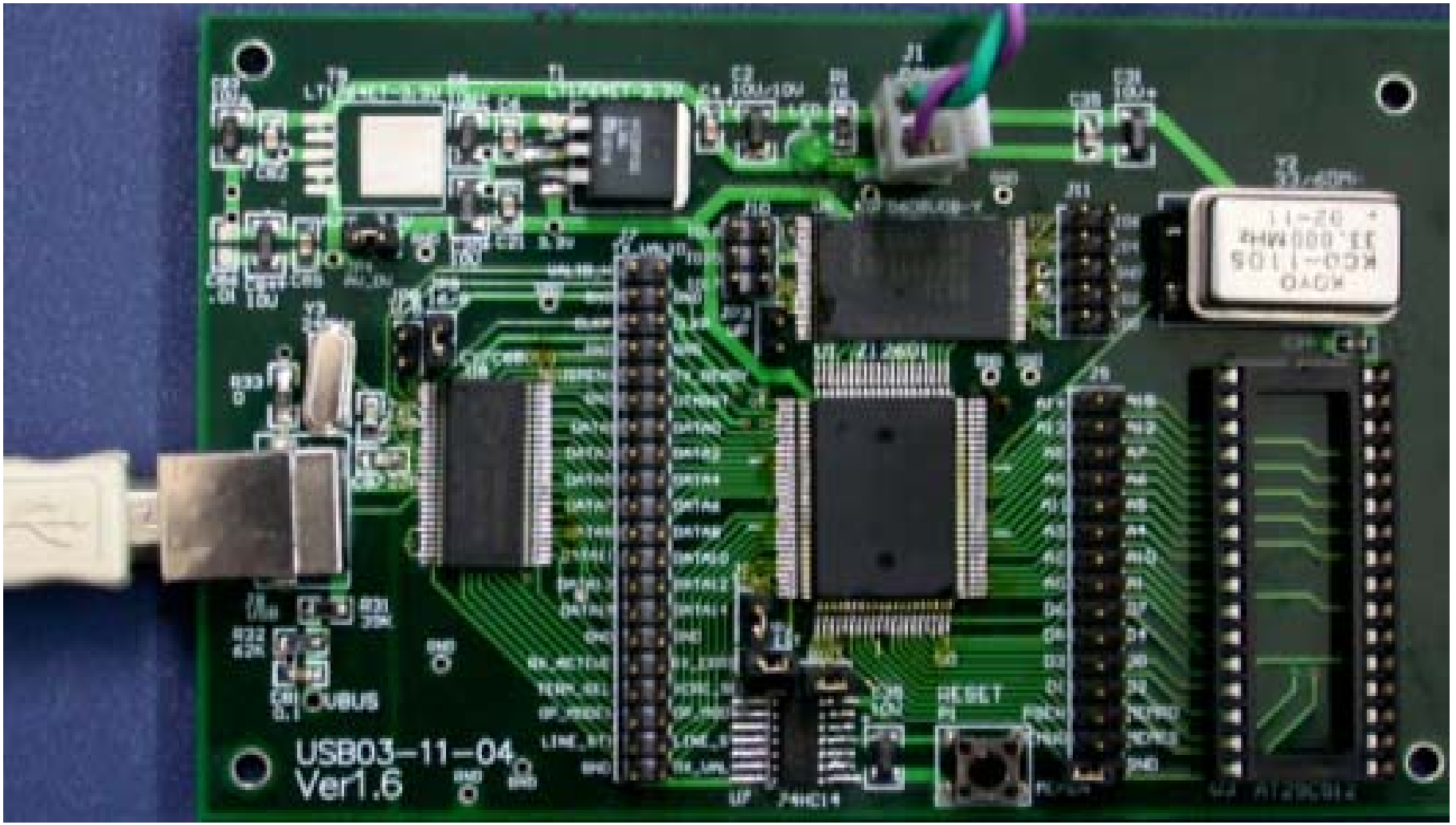


The Demo of USB2.0 IP CORE

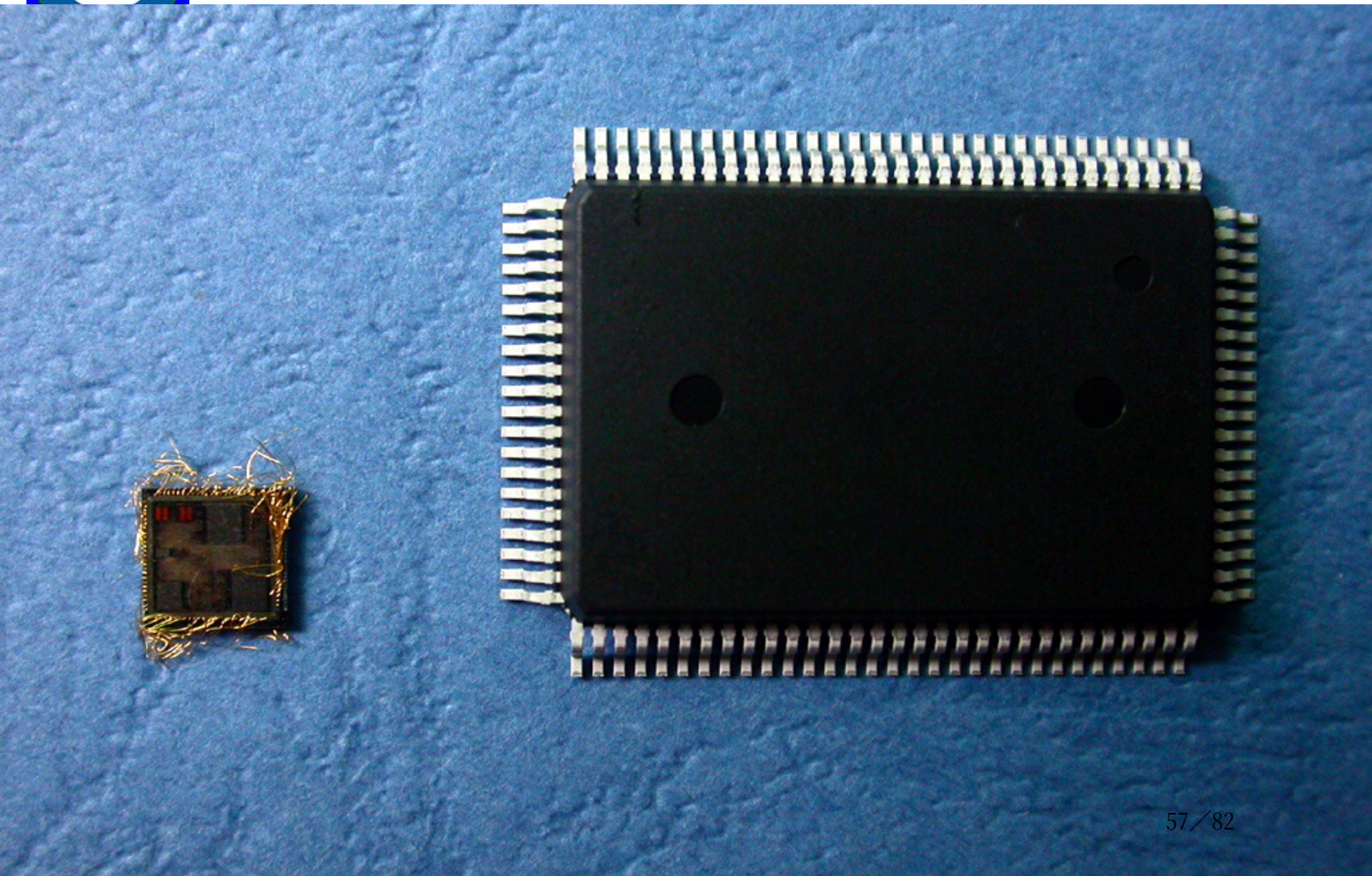


USB2.0 DEMO Baesd on FPGA

USB2.0 DEMO Baesd on ASIC



USB2.0 interface chip(ASIC)



2. Low power MCU design for Wireless Sensor Network

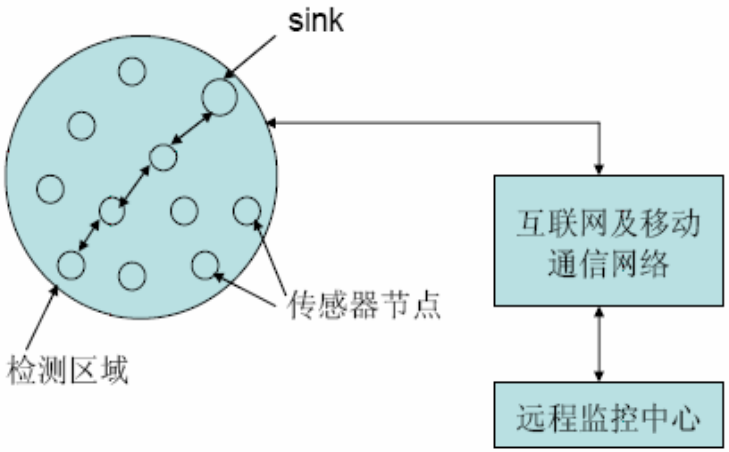


图1 无线传感器网络体系结构图

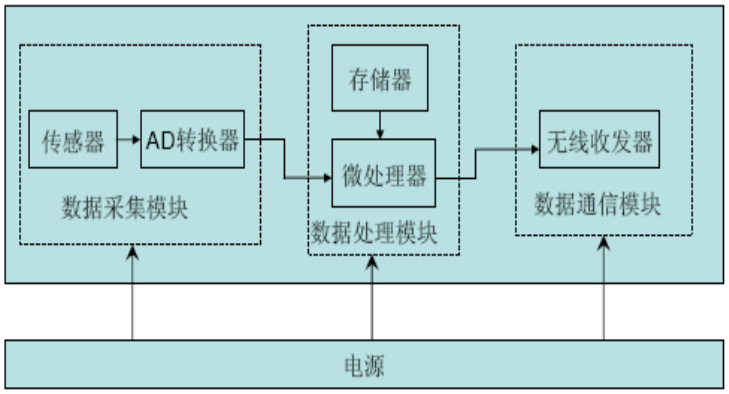
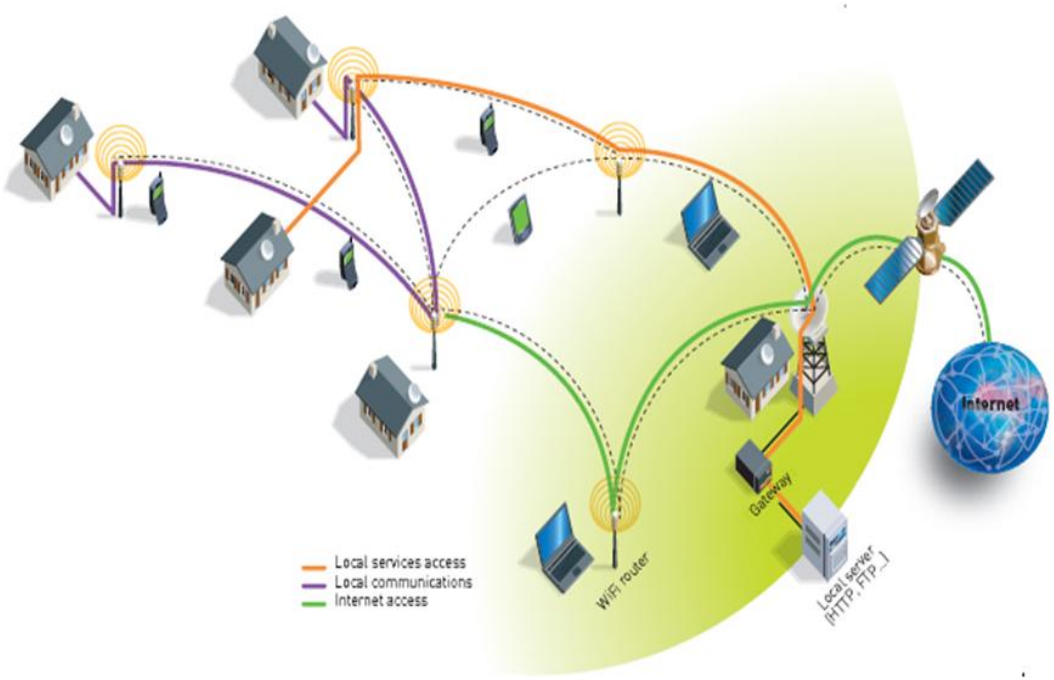
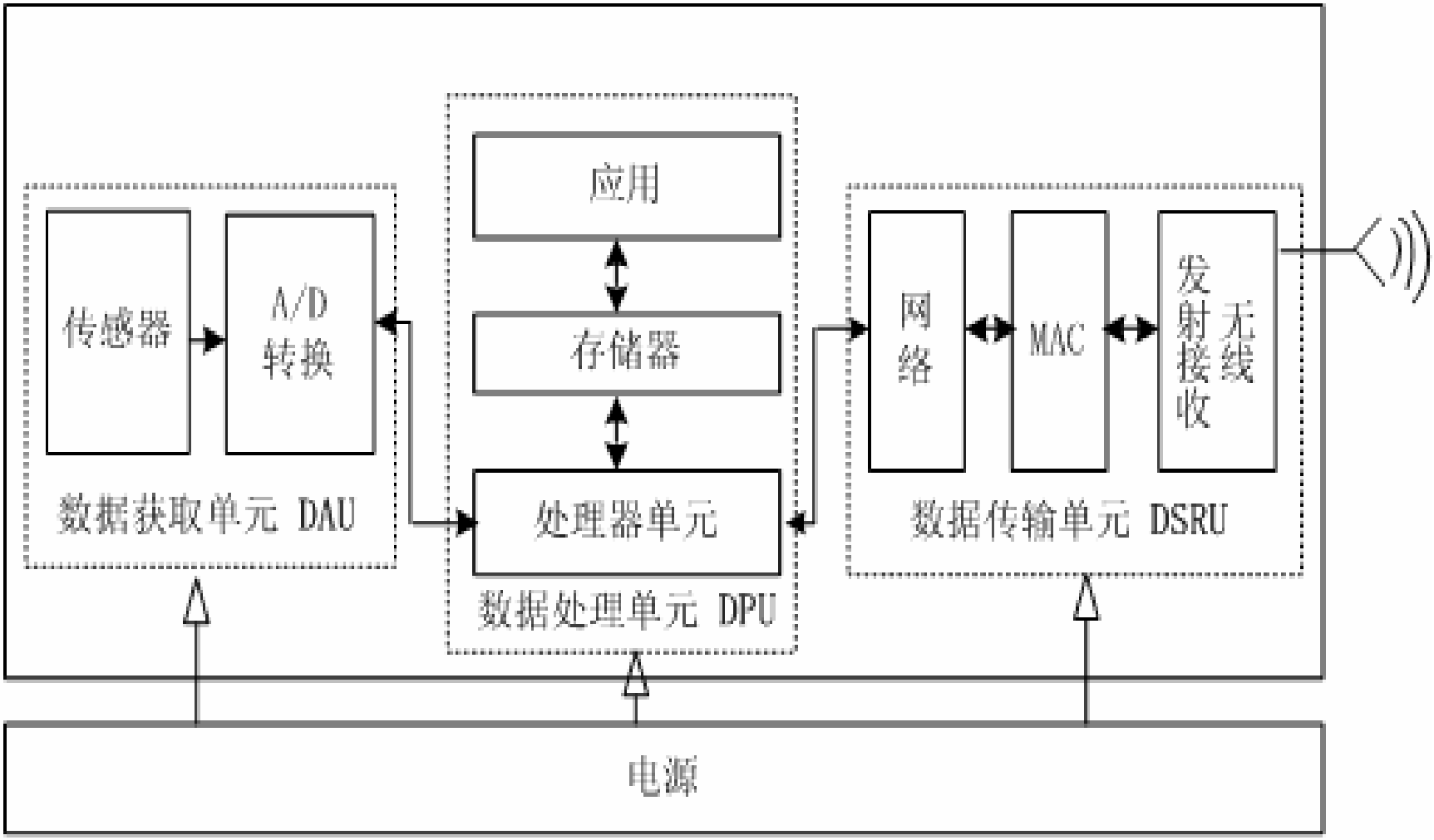


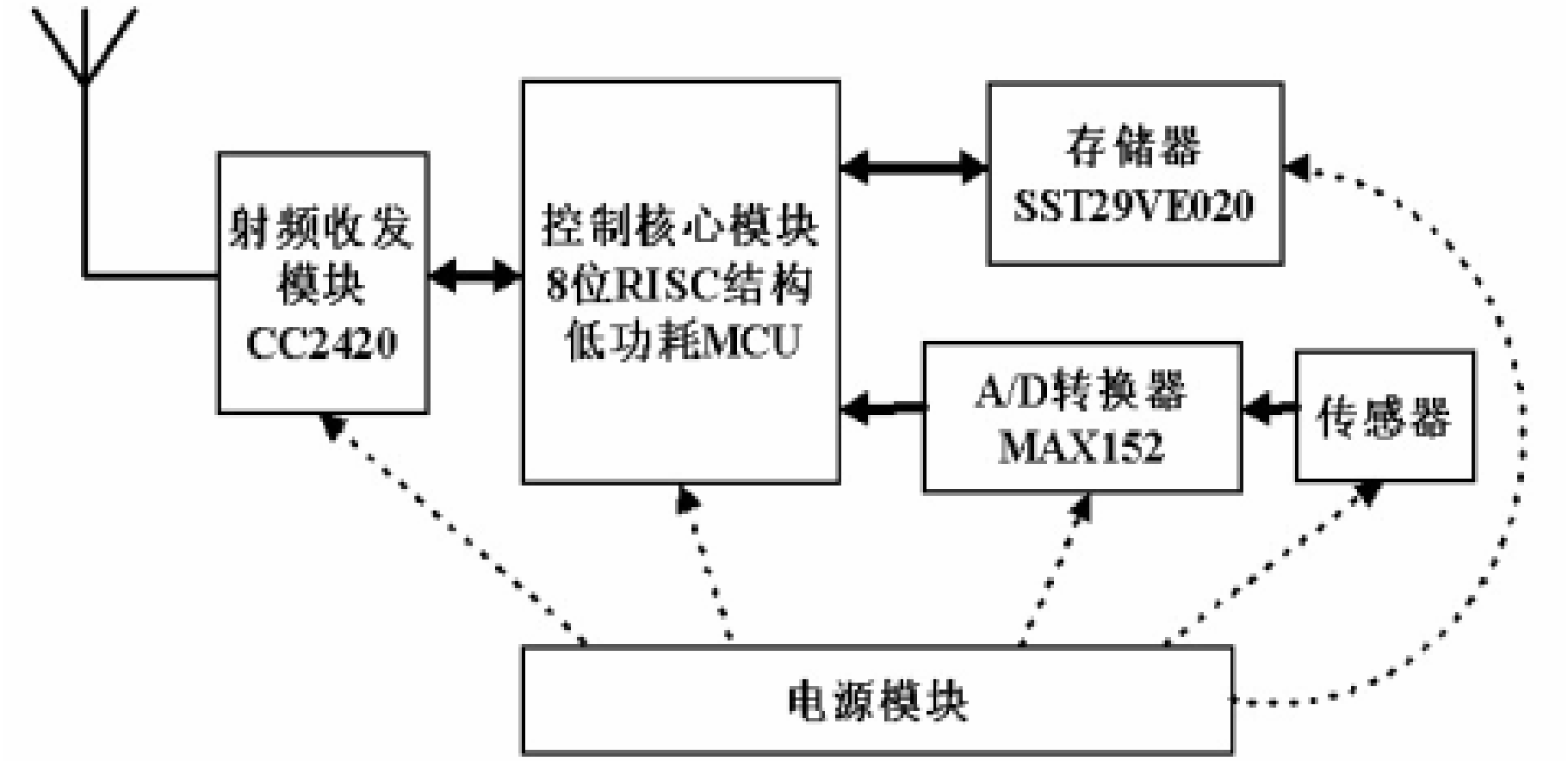
图2 节点的组成结构图



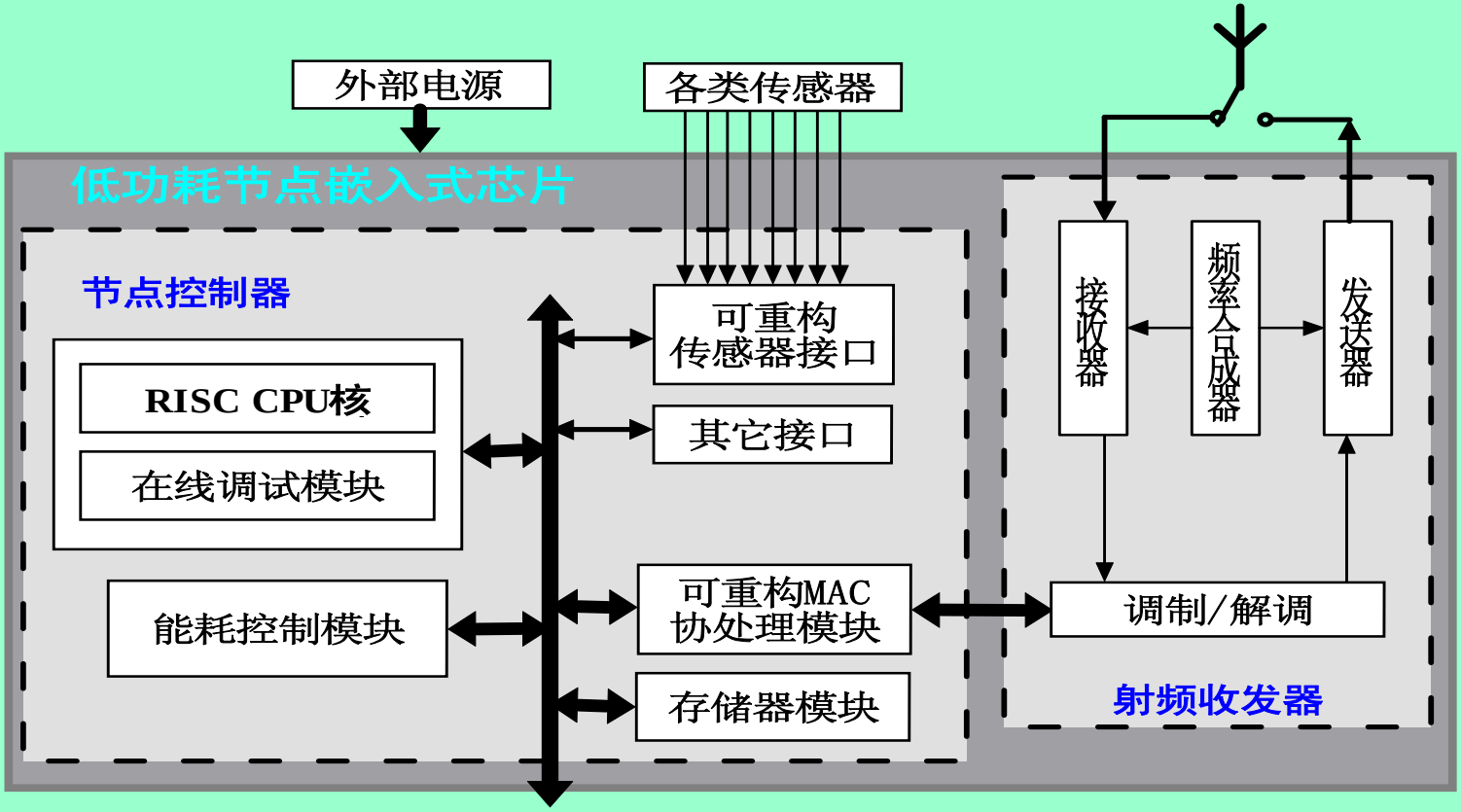
The Node Controller of Wireless Sensor Network



The Node Hardware Architecture of Wireless Sensor Network



The Node Controller of Wireless Sensor Network



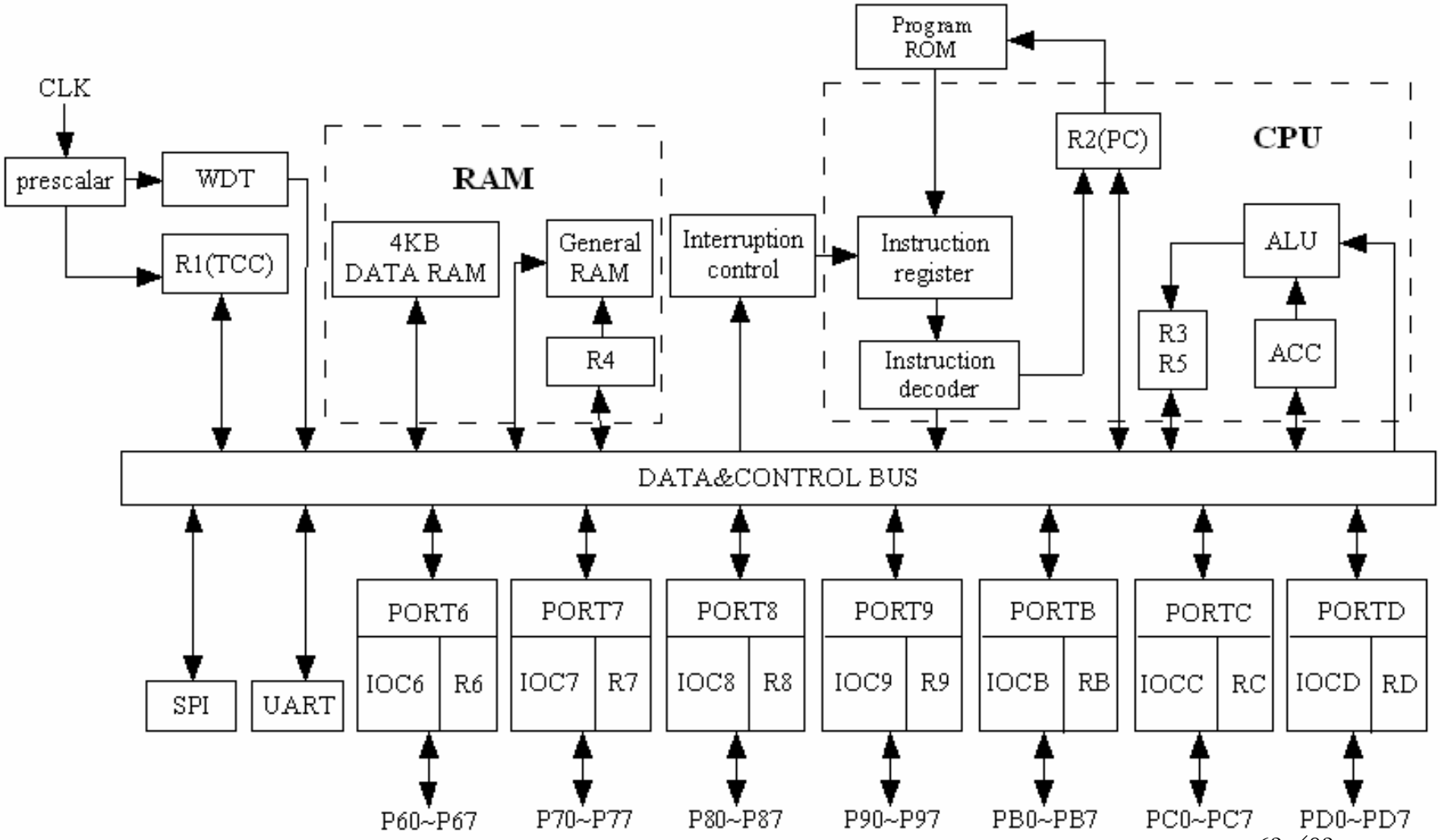
(1) WSN节点系统架构及其关键技术

(2) WSN节点芯片设计与测试验证

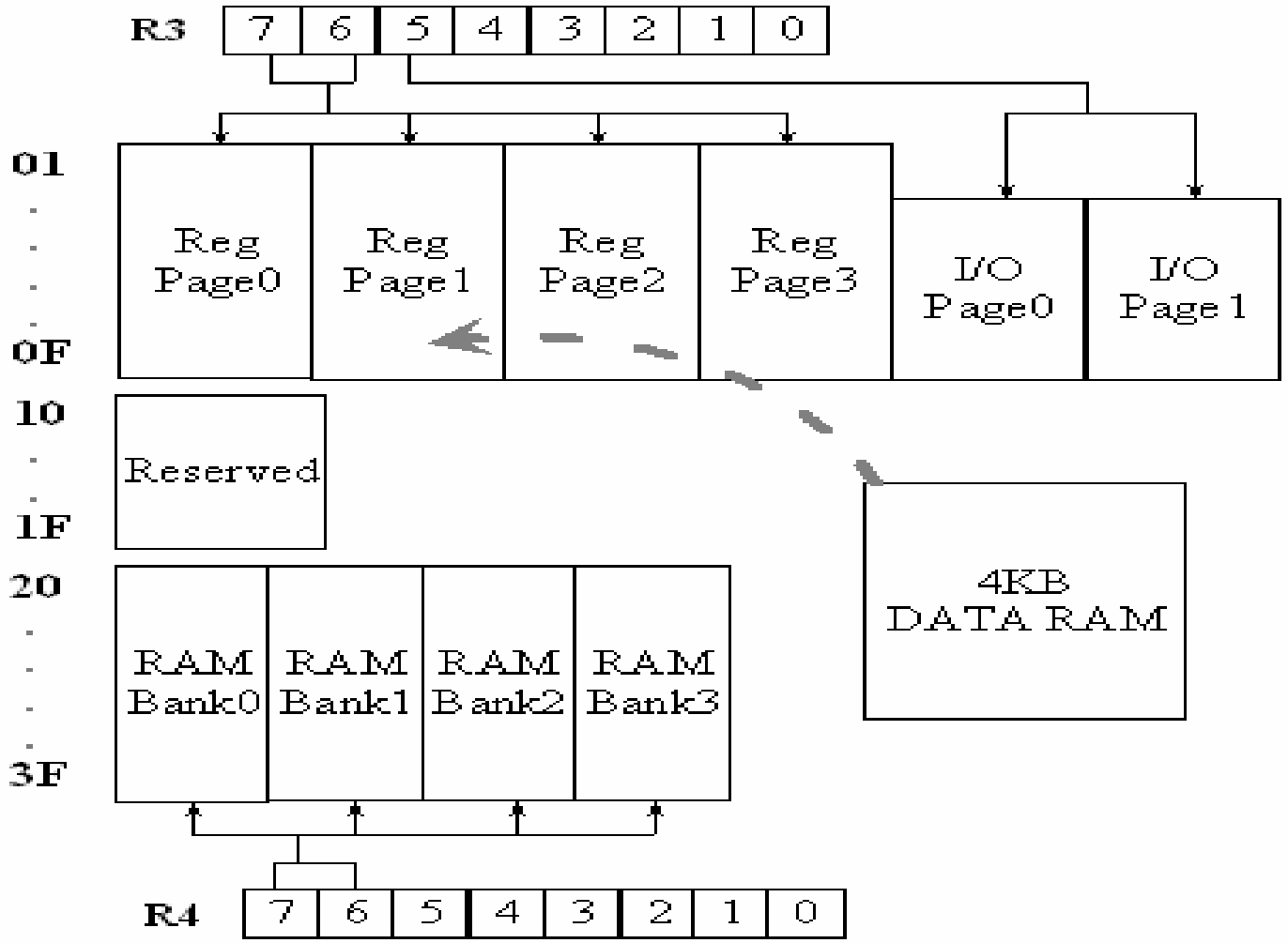
(3) 芯片软硬件集成调试环境

(4) MAC及组网协议及WSN典型应用验证系统

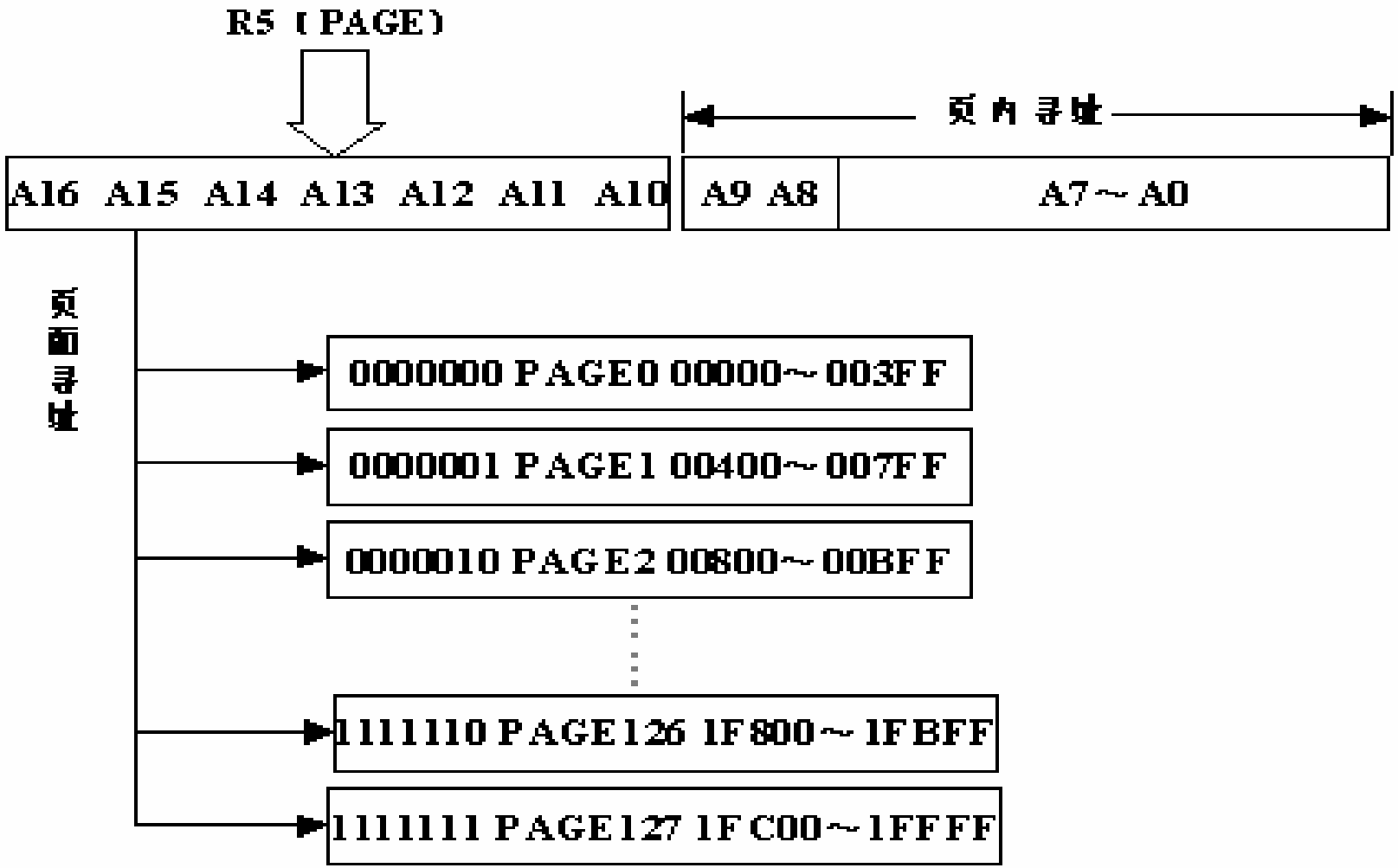
Low power MCU design



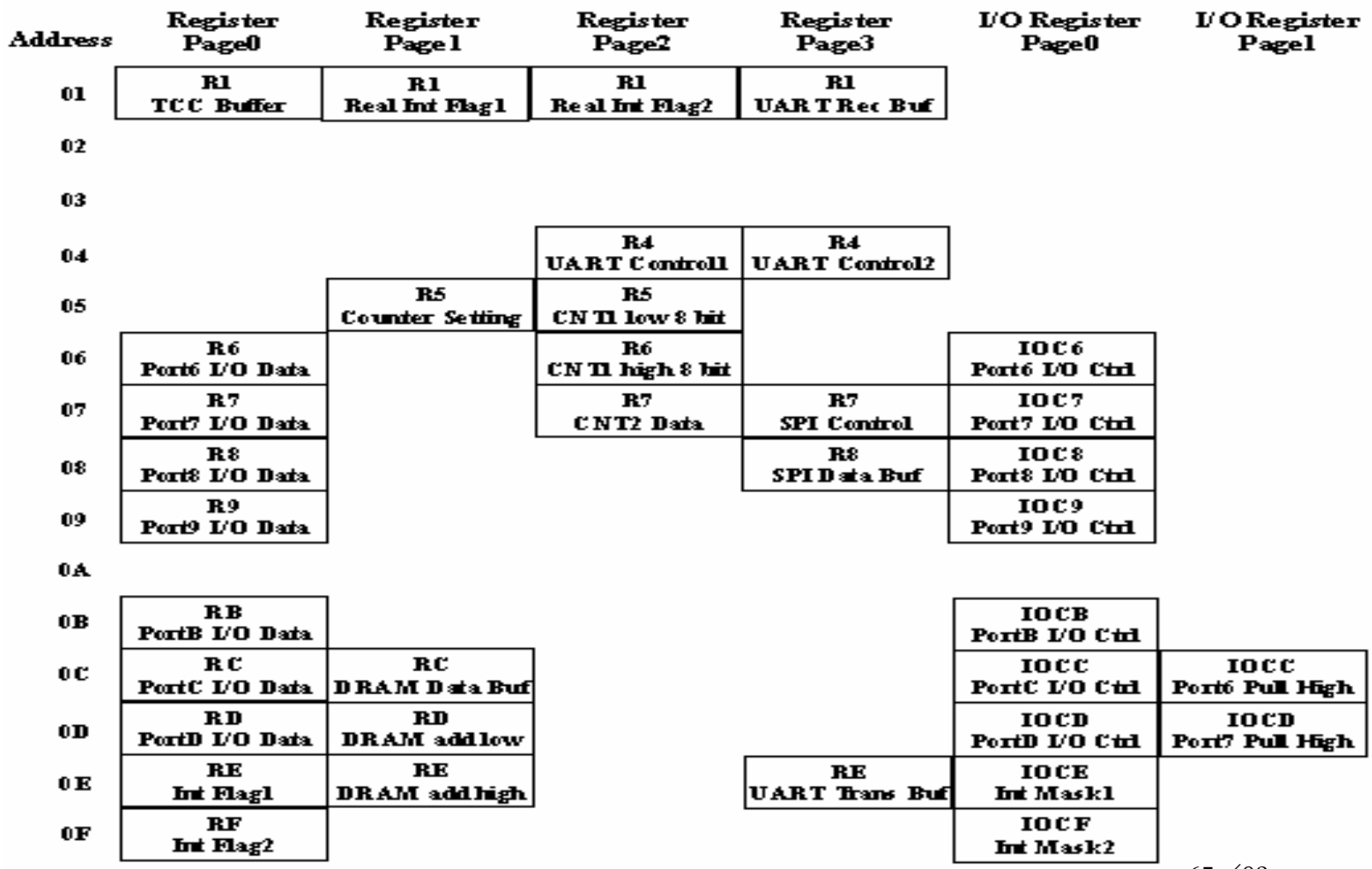
The Memory Organization



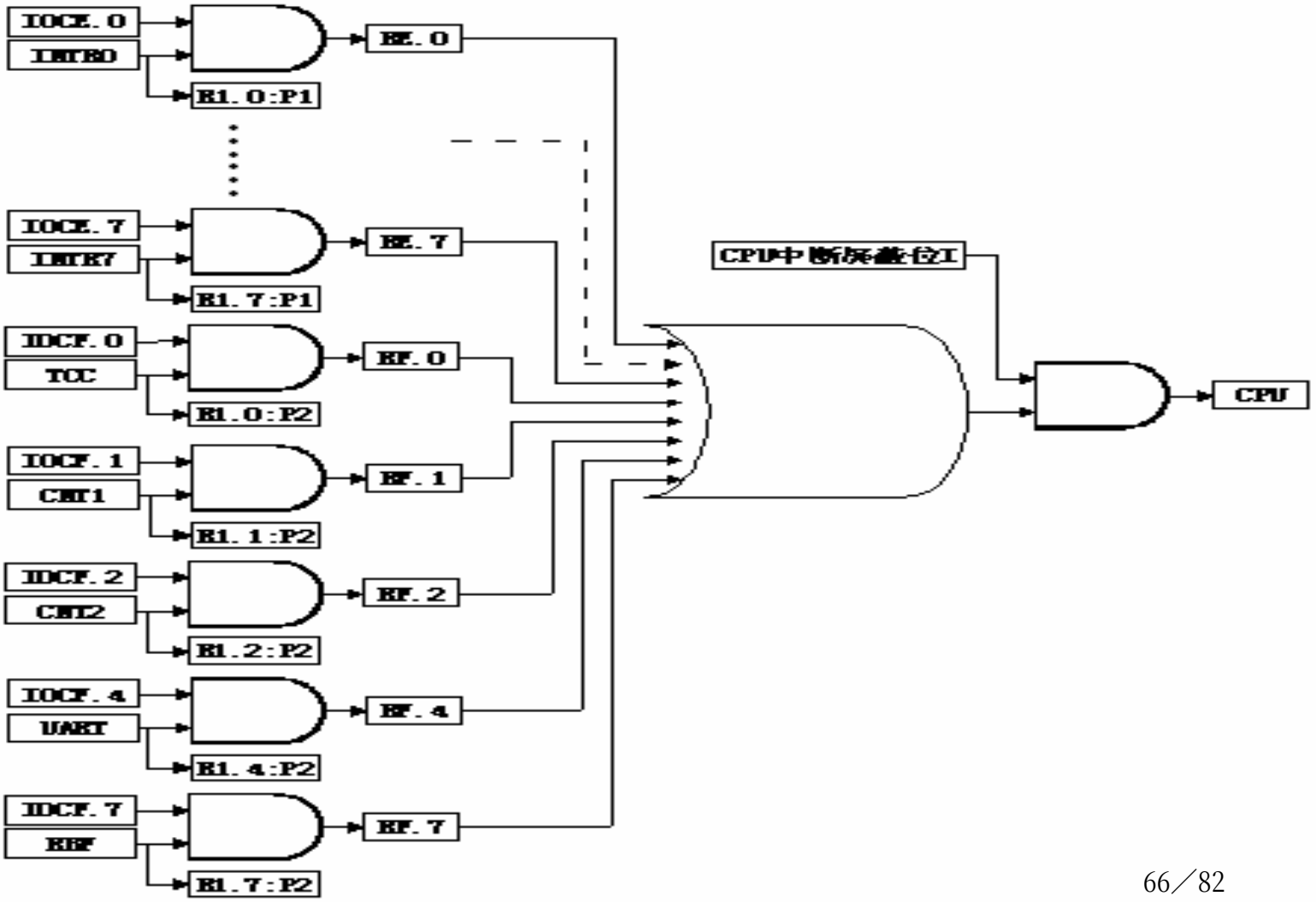
The Program Counter



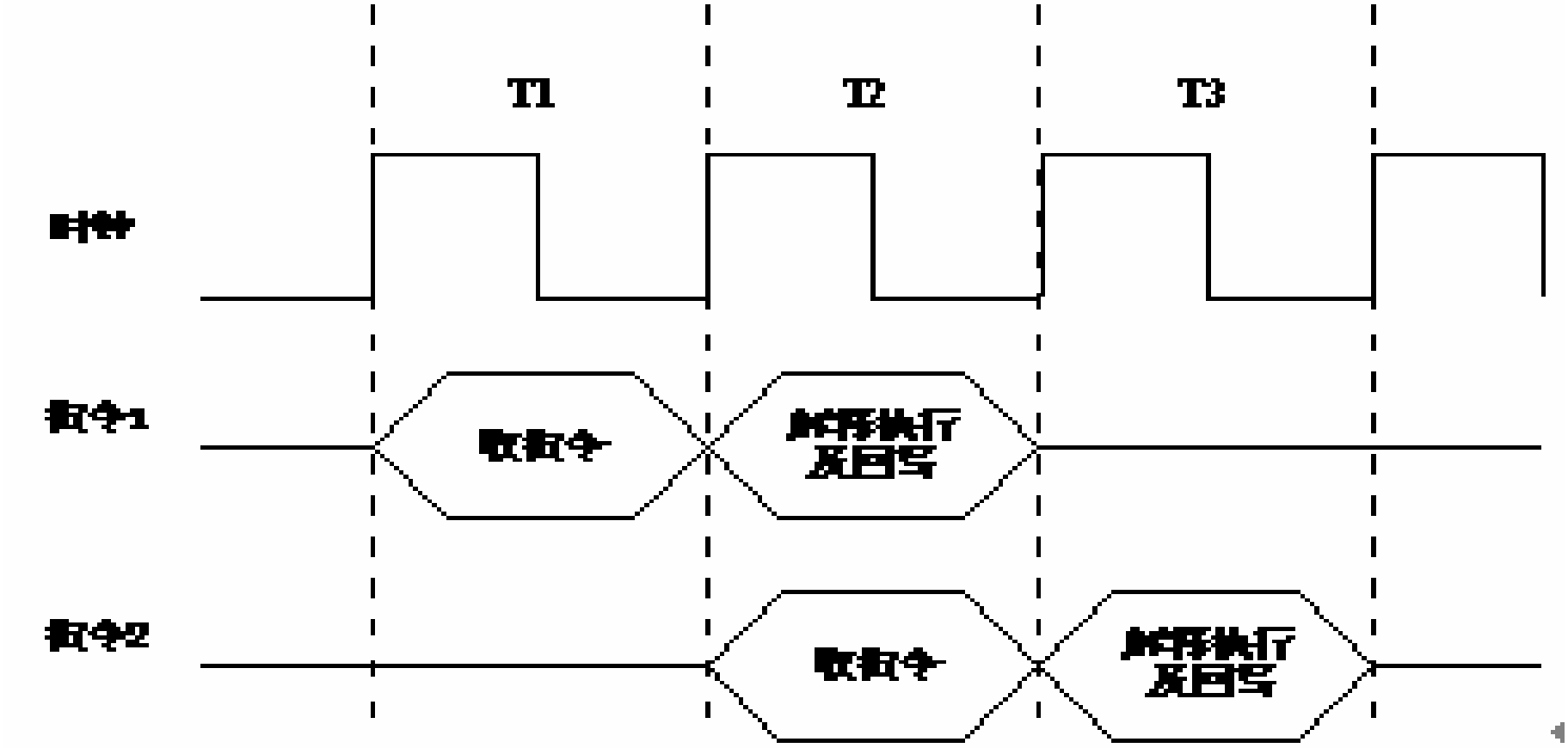
The Function Module Registers Sets



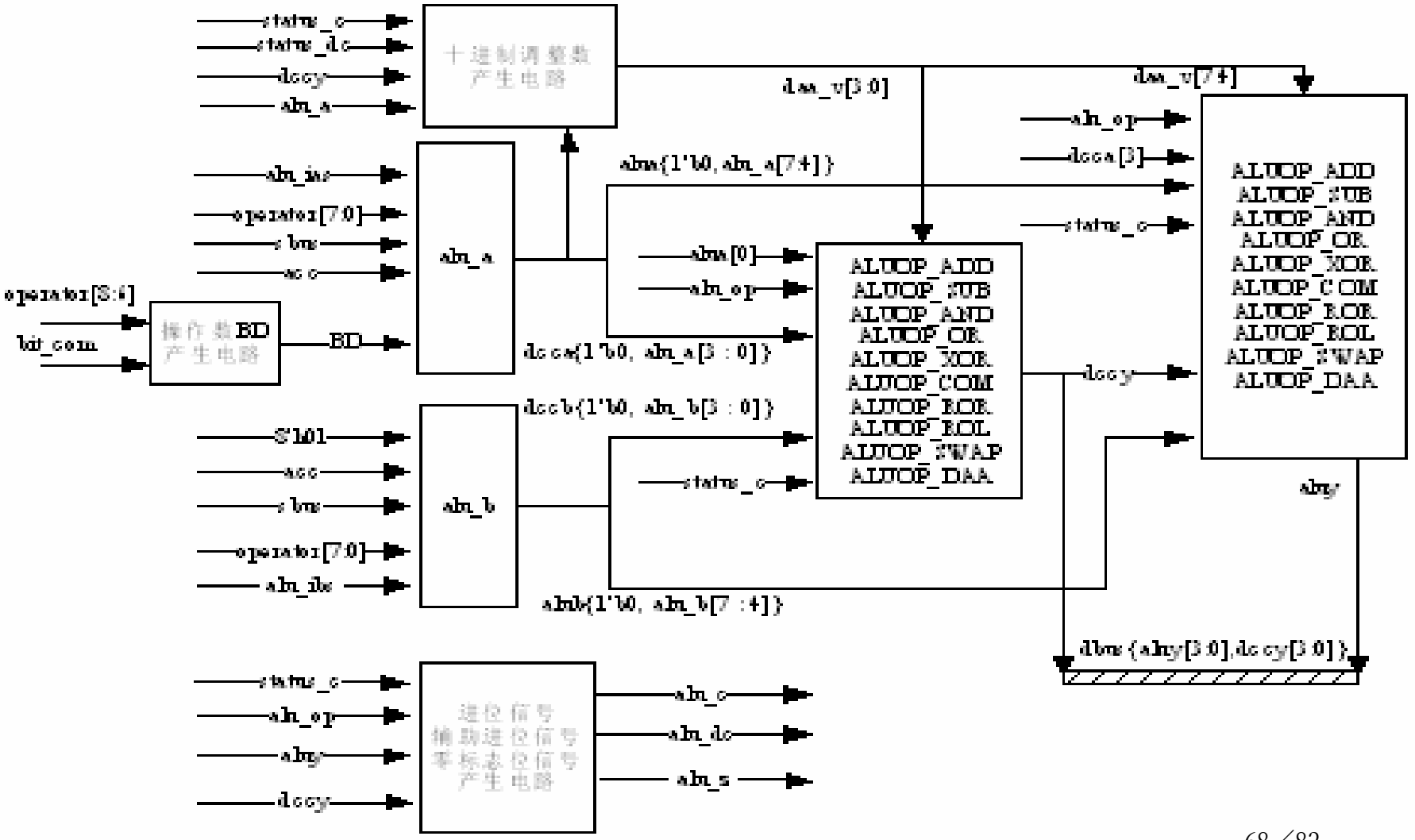
The Interrupt System



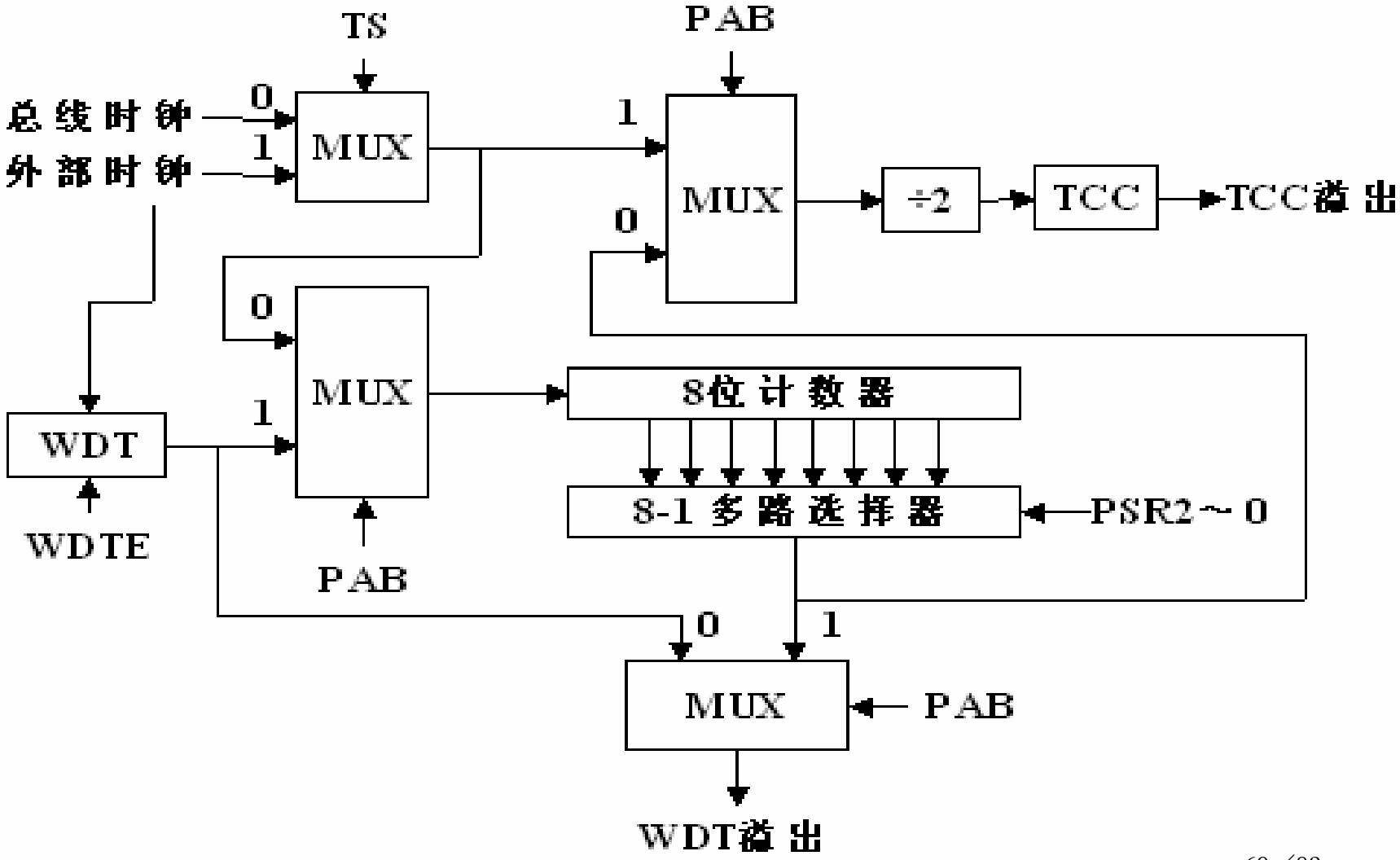
Execution of the single cycle instruction



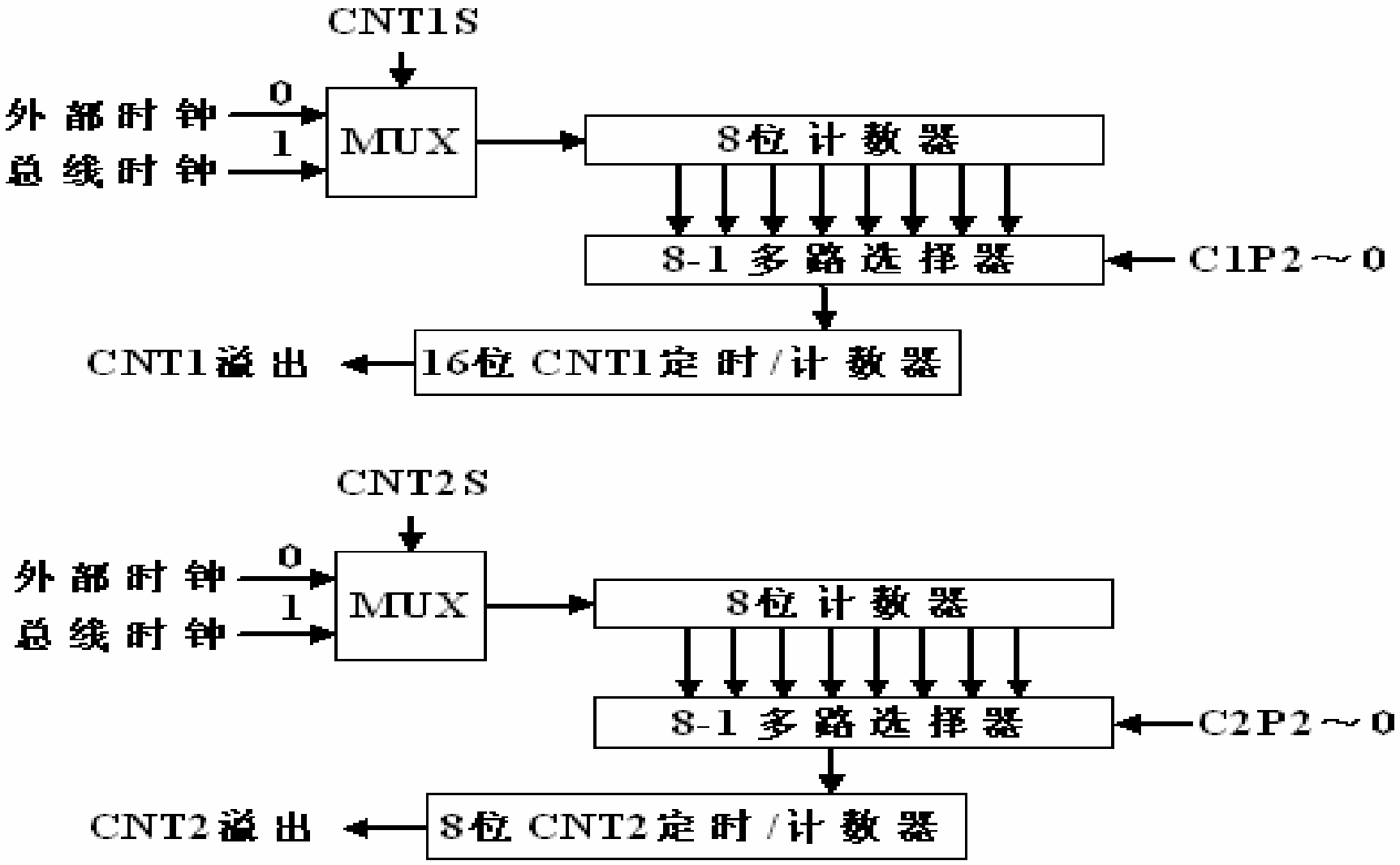
The ALU


status_c
status_dc
dccy
alu_s
十进制调整数产生电路
daa_v[3:0]
daa_v[7:4]
alu_op
dcca[3]
status_c
ALUOP_ADD
ALUOP_SUB
ALUOP_AND
ALUOP_OR
ALUOP_XOR
ALUOP_COM
ALUOP_RCR
ALUOP_RCL
ALUOP_SWAP
ALUOP_DAA
alu_ins
operator[7:0]
s_bus
acc
alu_a
alua{1'b0,alu_s[7:4]}
operator[3:0]
bit_com
操作数BD产生电路
BD
alua[0]
alu_op
dccs{1'b0, alu_s[3 : 0]}
dccb{1'b0, alu_b[3 : 0]}
S'b01
acc
s_bus
operator[7:0]
alu_ins
alu_b
status_c
dccy
alub{1'b0, alu_b[7 :4]}
aluy
dbus{aluy[3:0],dccy[3:0]}
status_c
alu_op
aluy
dccy
进位信号
辅助进位信号
零标志位信号
产生电路
alu_c
alu_dc
alu_z

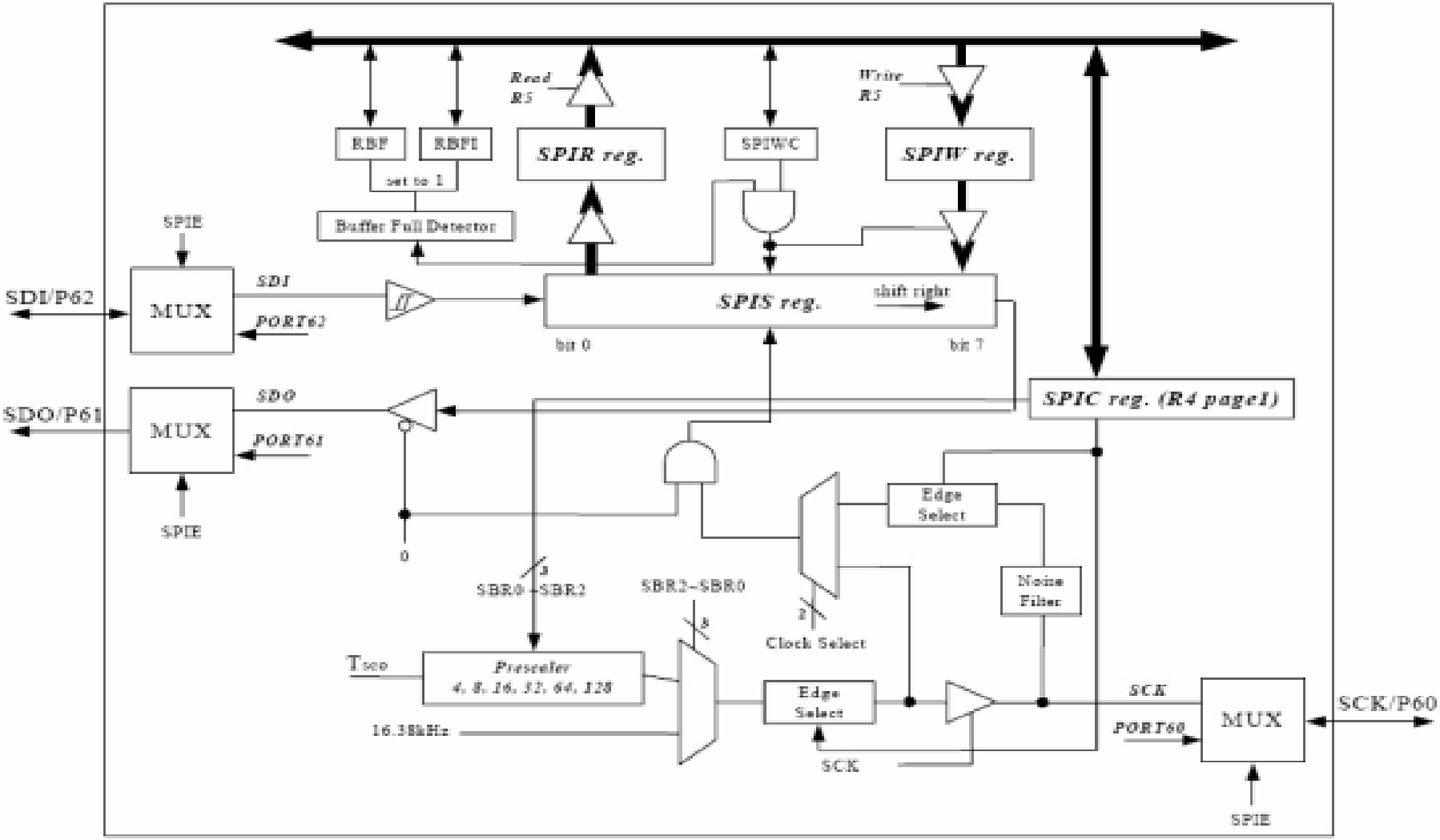
The WDT & TCC


TS
PAB
总线时钟
外部时钟
0
1
MUX
1
0
MUX
÷2
TCC
TCC溢出
0
1
MUX
8位计数器
WDT
WDTE
8-1 多路选择器
PSR2～0
PAB
0
1
MUX
PAB
WDT溢出

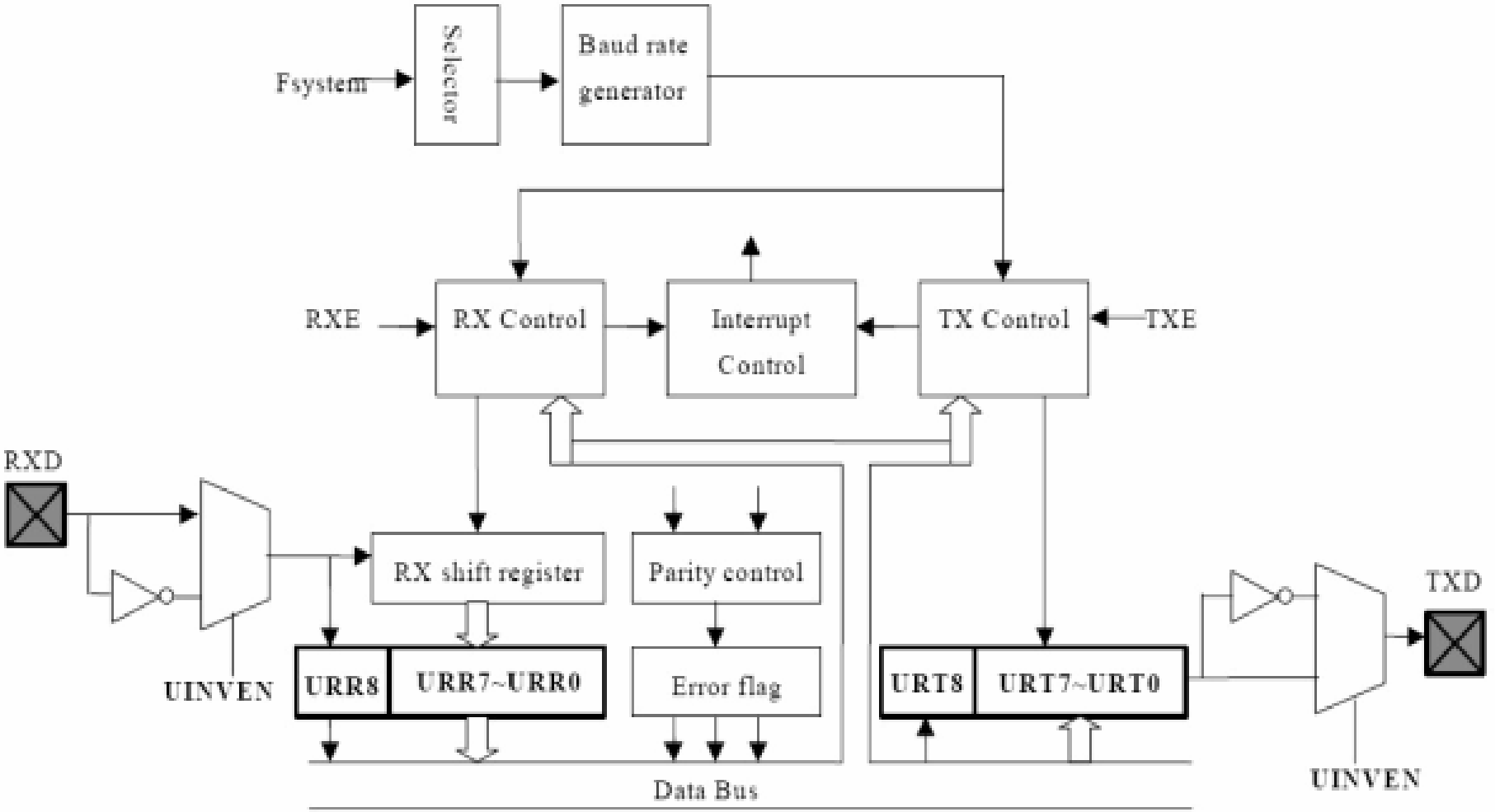
The Timer and Counter



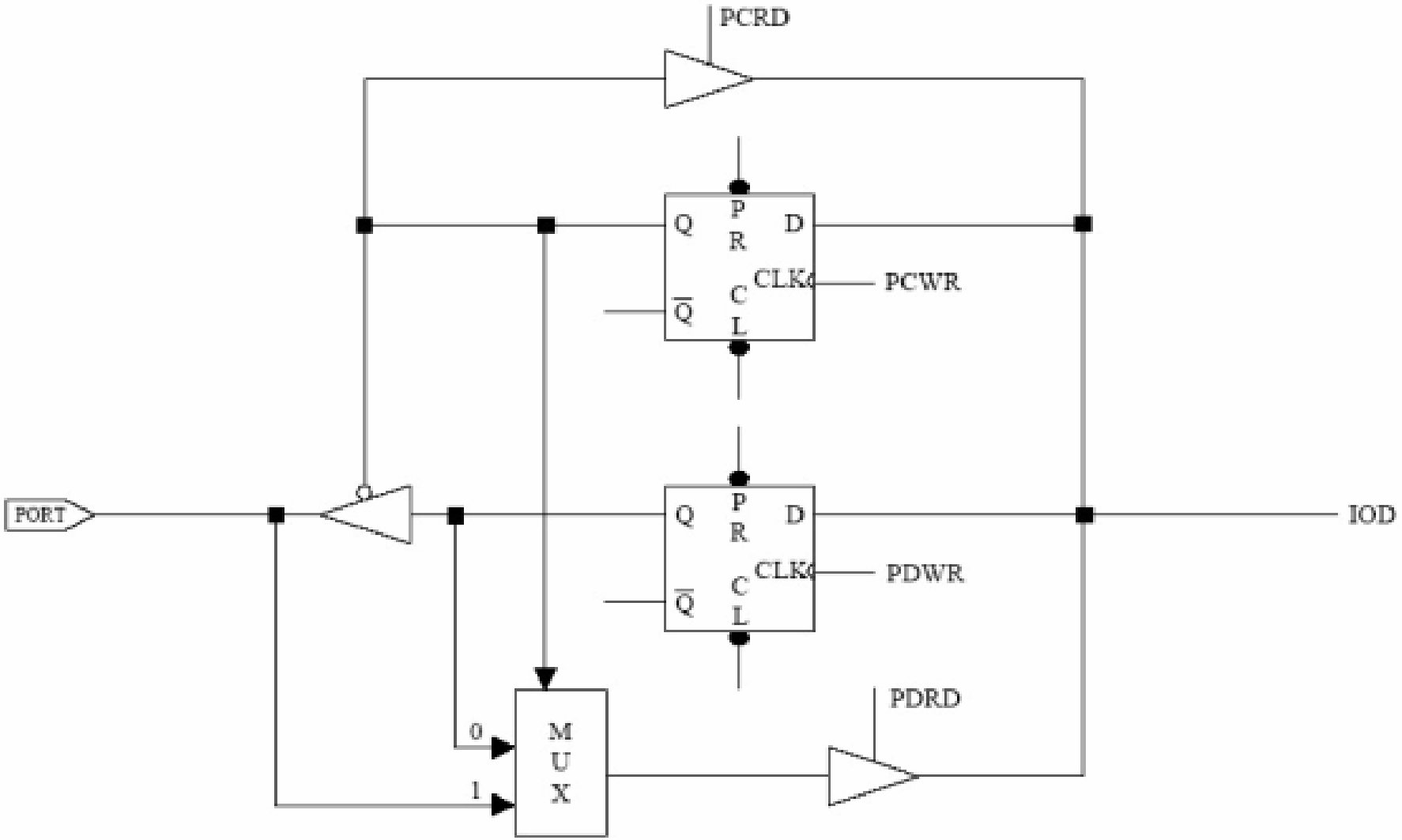
The SPI


Read
R5
Write
R5
RBF
RBFI
SPIR reg.
SPIWC
SPIW reg.
set to 1
Buffer Full Detector
SPIE
SDI/P62
MUX
SDI
PORT62
SPIS reg.
shift right
bit 0
bit 7
SPIC reg. (R4 page1)
SDO/P61
MUX
SDO
PORT61
SPIE
0
Edge
Select
Noise
Filter
SBR0 - SBR2
SBR2-SBR0
3
8
2
Clock Select
Tsco
Prescaler
4, 8, 16, 32, 64, 128
16.38kHz
Edge
Select
SCK
SCK
PORT60
MUX
SCK/P60
SPIE

The UART

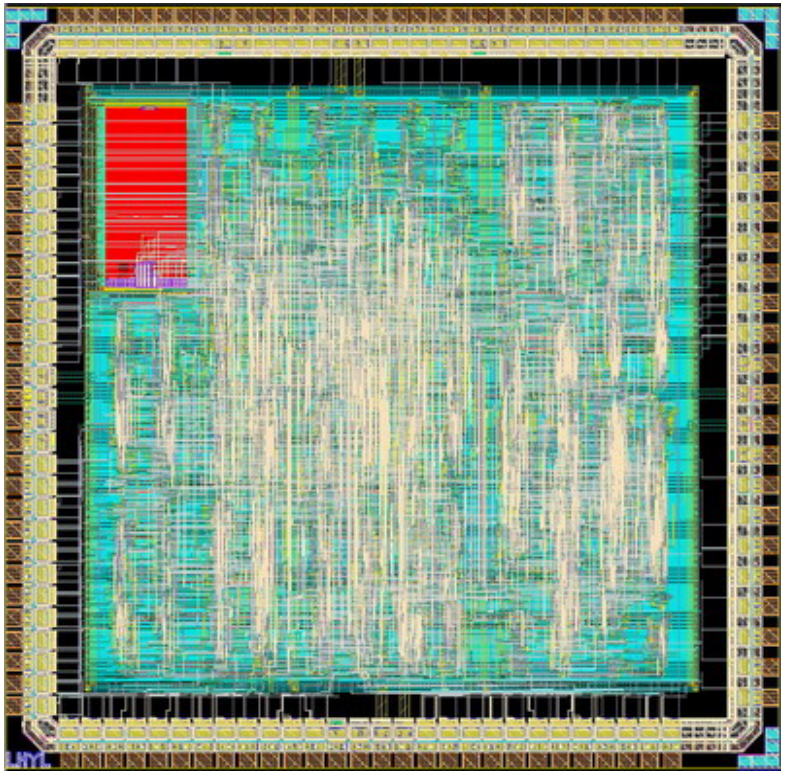

Fsystem
Selector
Baud rate generator
RXE
RX Control
Interrupt Control
TX Control
TXE
RXD
RX shift register
Parity control
UINVEN
URR8
URR7~URR0
Error flag
URT8
URT7~URT0
TXD
UINVEN
Data Bus

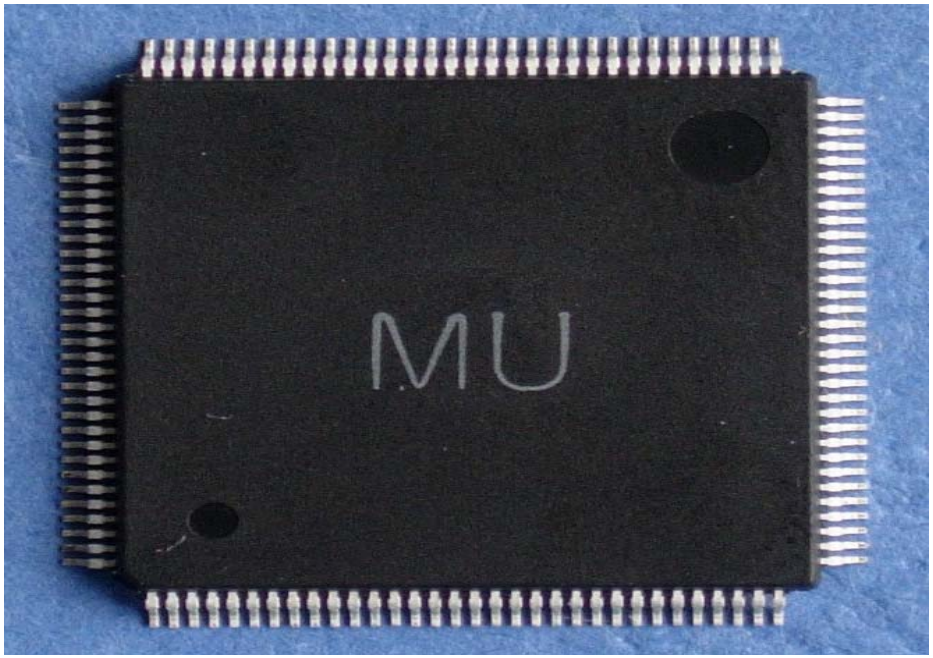
The Architecture of I/O Ports



The Node Controller Demo Circuit of Wireless Sensor Network

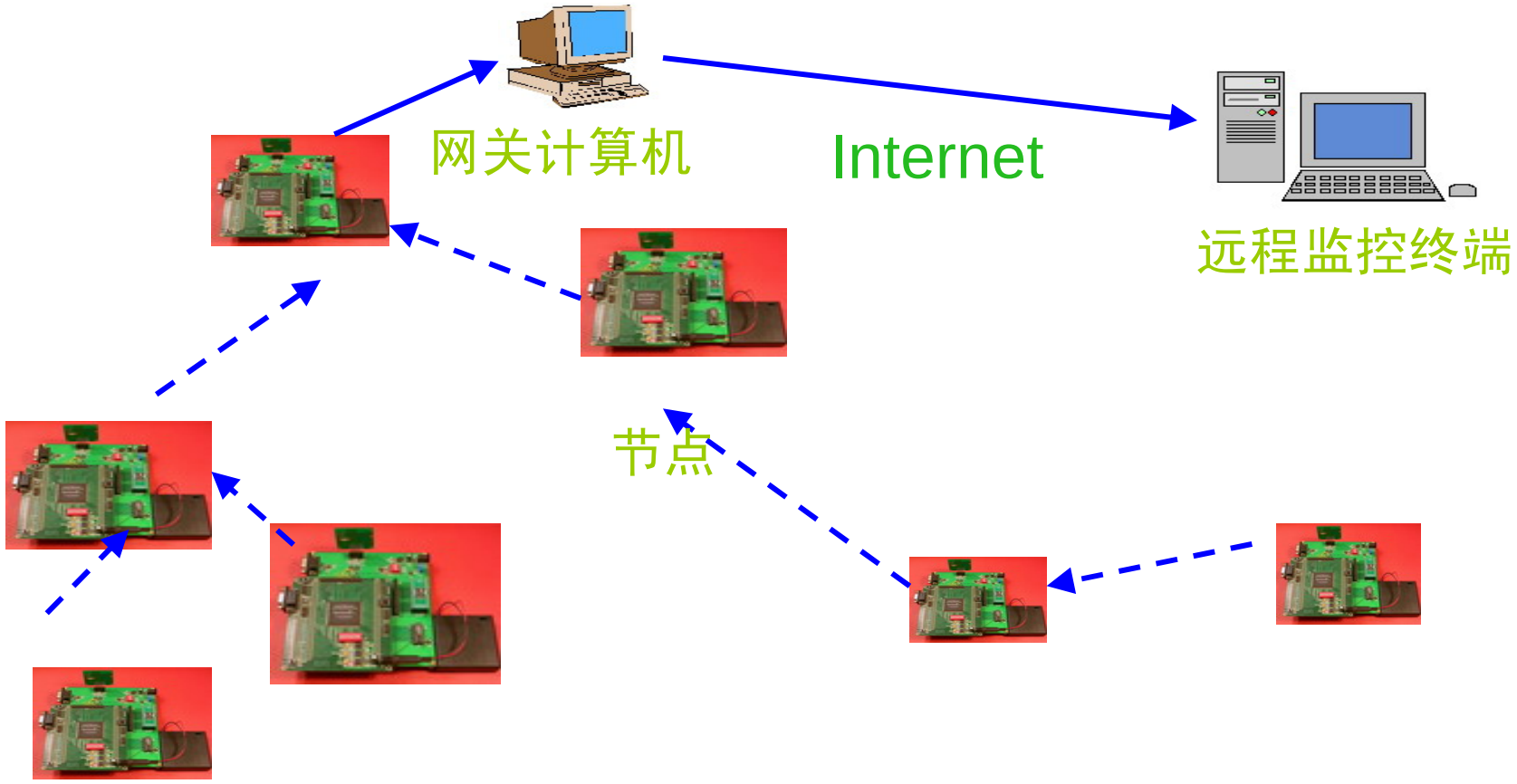
Wireless Sensor Network node controller





Wireless Sensor Network node controller

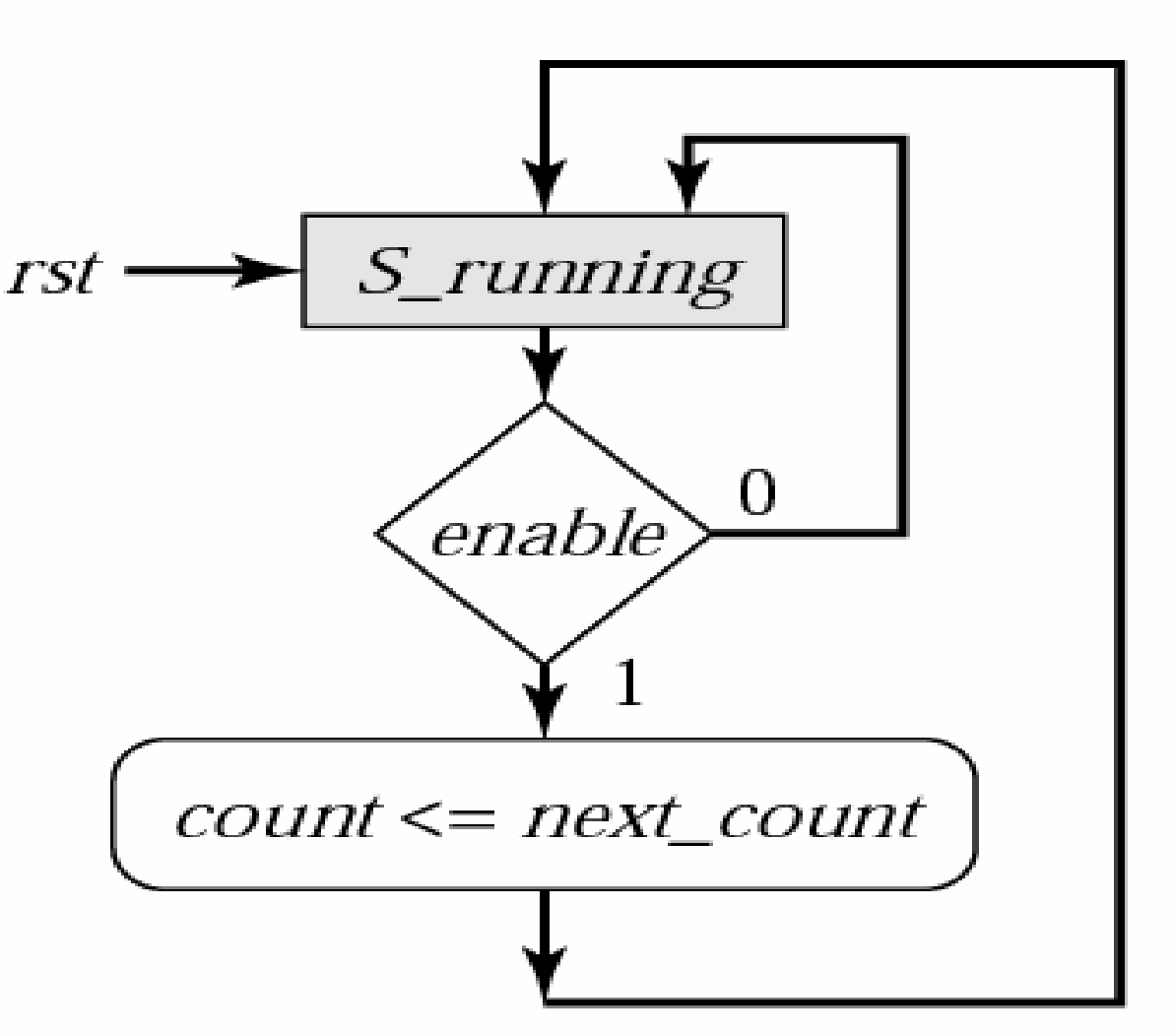
The Node Controller Demo of Wireless Sensor Network



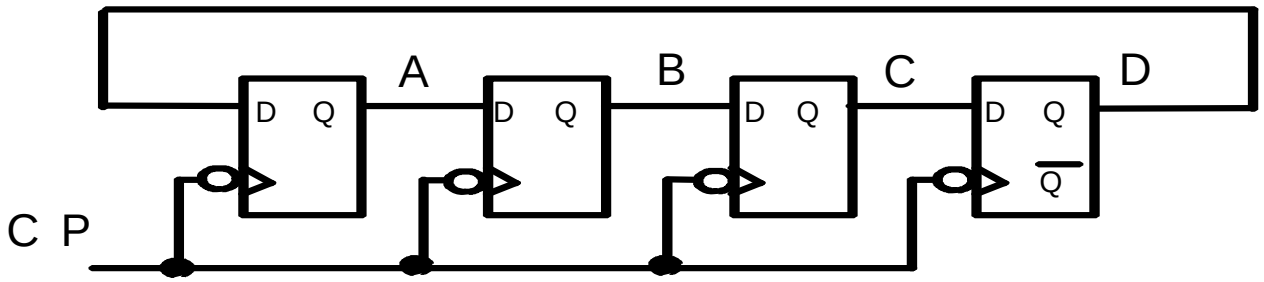
Problem

- 7-1 Develop, verify, and synthesize Johnson_Counter_ASM, a Verilog behavioral module of based on a direct implementation of the ASM chart in Figure 7.5.
- Hint: Use a Verilog function to described next_count.

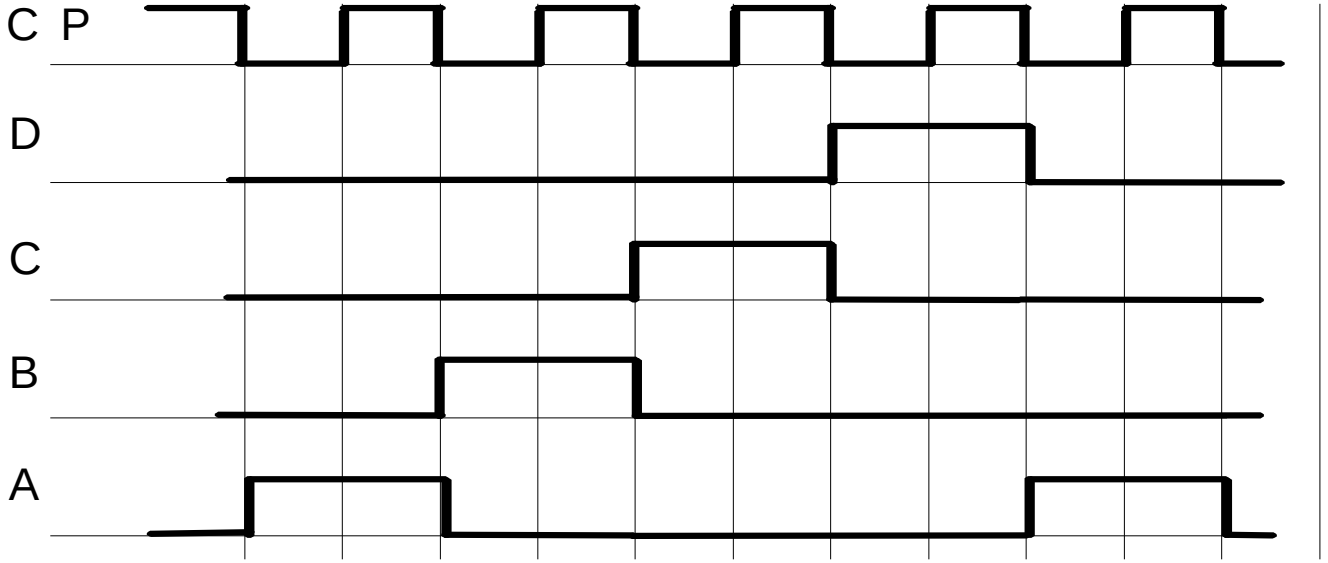
Figure 7.5



Ring Counter

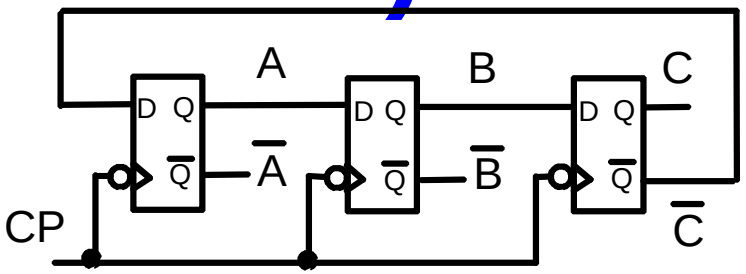


IF A B C D = 1 0 0 0 At Start:

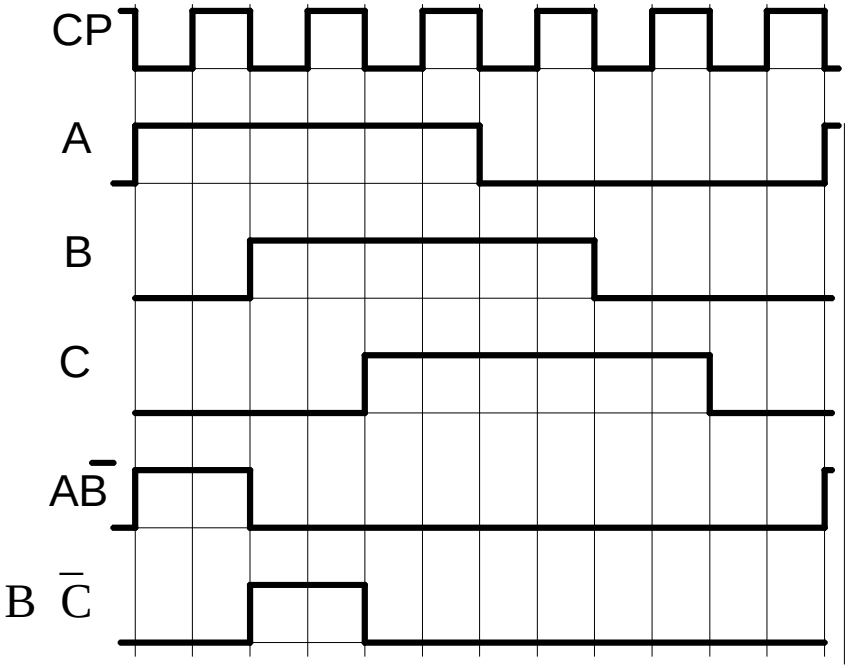


Johnson Counter (Switch-Tail)

- **A Johnson counter is also a shift register with feedback. Assuming the initial state ABC is 000 we get the timing diagram shown:**
- **The state must be initialized to operate correctly. Use logic to assure this.**



IF ABCD = 000 At Start:



Problem

- **7-18**
- Describe the differences between the circuits that will be synthesized from the following Verilog cyclic behaviors:

- always @ (a or b or c or d) y = a + b + c + d;

- always @ (a or b or c or d) y = (a + b) + (c + d);