嵌入式系统设计与应用

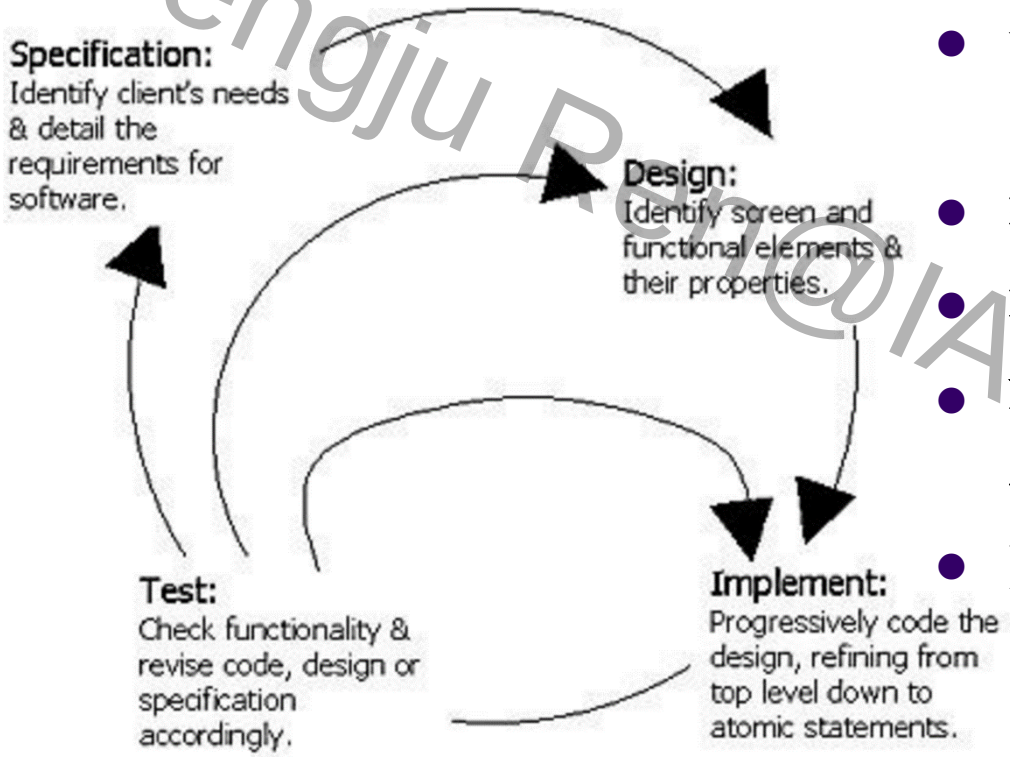
第五章 程序设计与分析（1）

西安交通大学电信学院

任鹏举

Pengju Ren@IAIR 2017-2018

本章主要内容



Software Design Cycle

- 嵌入式软件中的组件(状态机、循环缓存器、队列)
- 编程模型，如数据流和控制图
- 编译方法介绍
- 根据性能、大小和功耗来分析和优化程序
- 如何测试程序以验证其正确性

Pengju Ren@IAIR 2017-2018

1 嵌入式程序组件

- 状态机（**State machine**）

  用变量来表示内部的状态，根据输入完成状态的转移

  交通灯控制、**CPU design controller**

- 循环缓冲区（**Circular buffer**）

  I/O input buffer

- 队列（**Queue**）

Pengju Ren@XIR 2017-2018

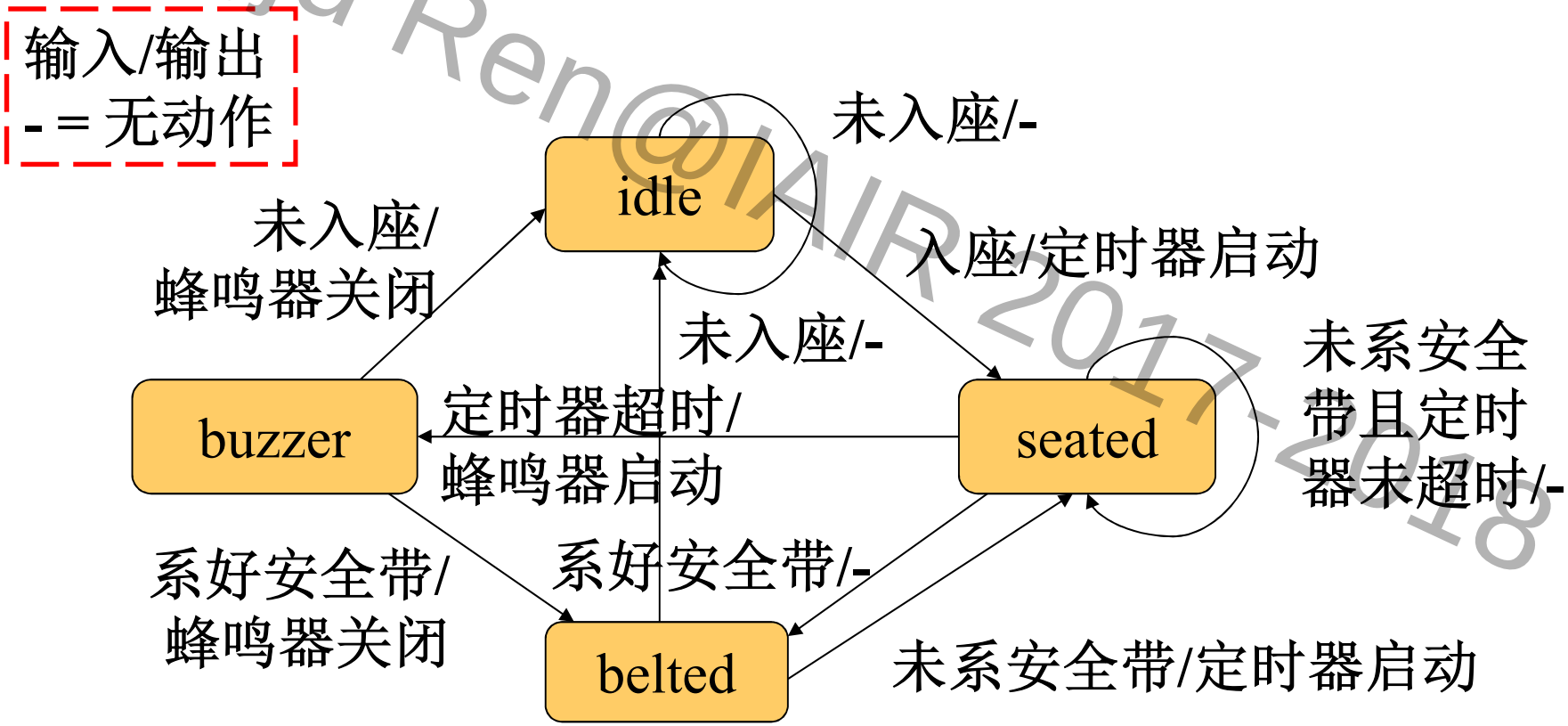
状态机（1）

- 反应系统（**reactive system**）：响应外部事件的系统。
  - 外部输入是间歇到达
  - 适合使用状态机描述
- 有限状态机是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。
  - **Moore**机：输出只由当前状态确定
  - **Mealy**机： 输出依赖于当前状态和输入

Peng Ren@IAIR 2017-2018

状态机（2）

- 例子：一个简单的座位安全带控制器



Pengju Ren@IAIR 2017-2018

状态机（3）

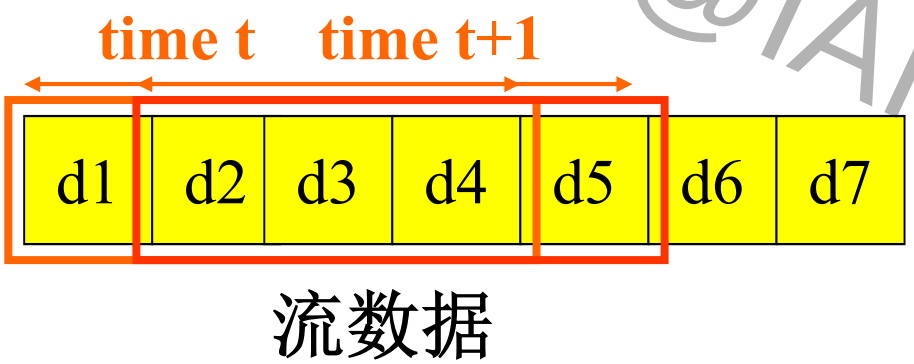
Inputs：seat, belt, timer
Outputs: buzzer

```
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3
switch (state) {
    case IDLE: if (seat) { state = SEATED; timer_on = TRUE; }
            break;
    case SEATED: if (belt) state = BELTED;
                    else if (timer) state = BUZZER;
            break;
    case BELTED: if (!seat) state = IDLE;
                    else if (!belt) state = SEATED;
            break;
    case BUZZER: if (belt) state = BELTED;
                    else if (!seat) state = IDLE;
            break;
}
```

Renqiu Ren@IAIR 2017-2018

循环缓冲区（1）

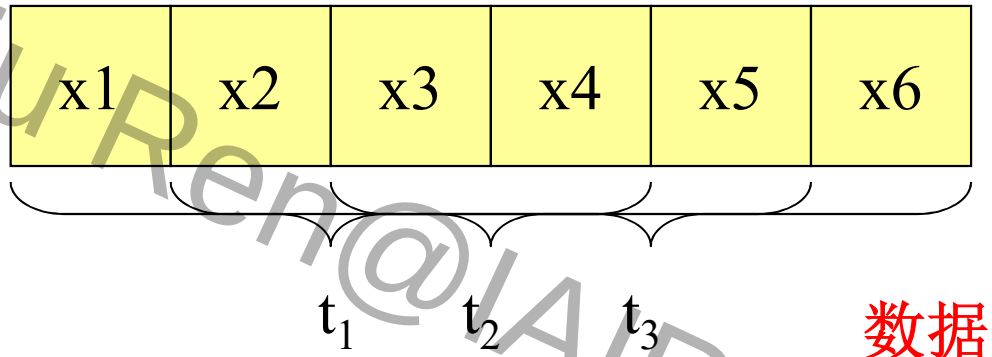
- 通常用于信号处理领域流数据的程序设计:
  - 新数据按一定速率持续到达;
  - 每个数据生命周期有限，需要立即处理。



- 循环缓冲区是便于有效处理流数据的一种数据结构。

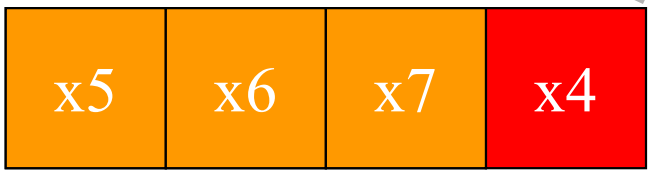
Dengju Ren@IAIR 2017-2018

循环缓冲区（2）



数据流

数据窗口尺寸不变，只是随时间滑动。

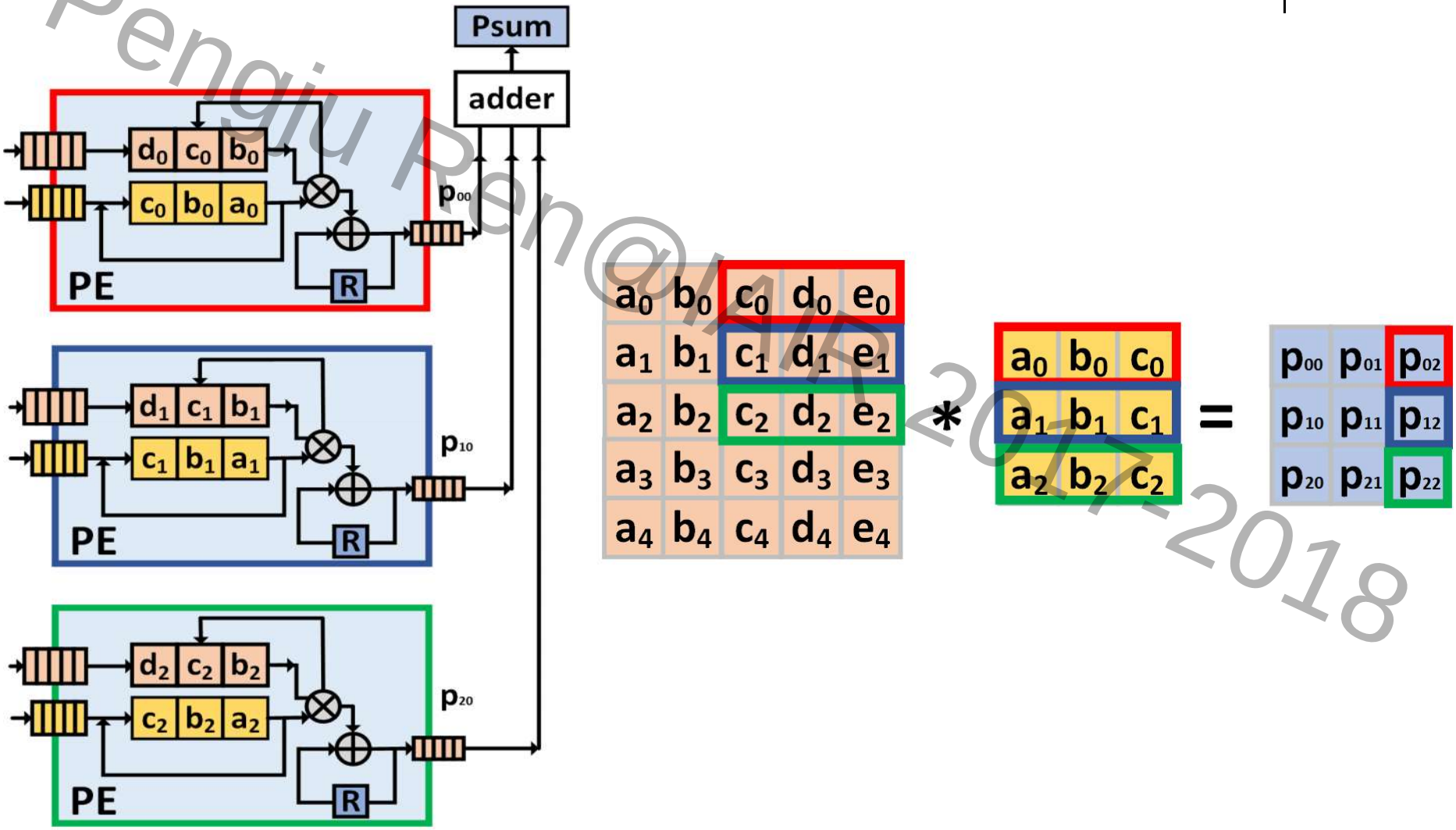


循环缓冲区

$$y(i) = \sum_{n=0}^{N} a(n)x(i-n)$$

Pengju Ren@IAIR 2017-2018

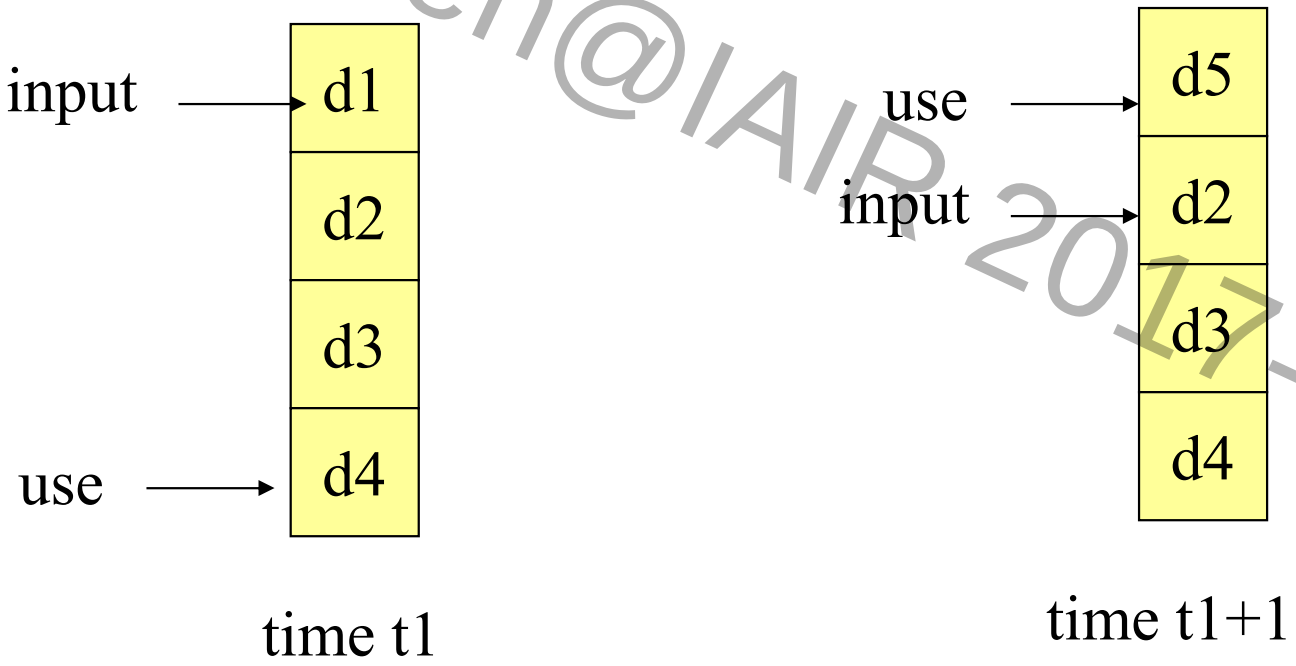
循环缓存区在卷积运算中示例



Pengju Ren@IAIR 2017-2018

循环缓冲区（3）

- 索引指向循环缓冲区的下一个替换数据位置和当前使用数据位置。



Pengju Ren@IAIR 2017-2018

循环缓冲区（4）

```
#define CMAX 6 /* filter order, the size is 6 */
int circ[CMAX]; /* circular buffer */
int pos; /* position of current sample */

void circ_update(int xnew) {
    /* compute the new head value with wraparound; the pos pointer
    moves from 0 to CMAX-1 */
    pos = ((pos == CMAX-1) ? 0 : (pos+1));
    /* insert the new value at the new head */
    circ[pos] = xnew;
    }
```

Pengju Ren@IAIR 2017-2018

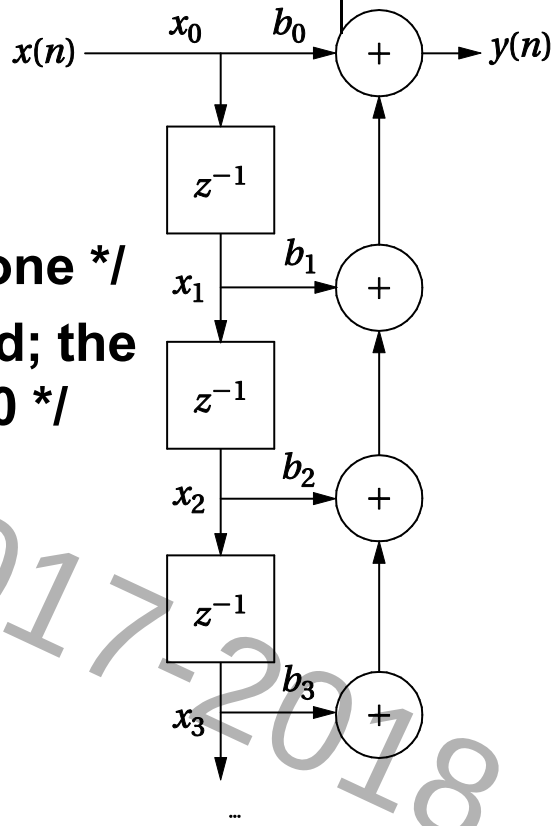
循环缓冲区（5）

```
void circ_init() {
    int i;
    for (i=0; i<CMAX; i++) /* set values to 0 */
        circ[i] = 0;
    pos=CMAX-1; /* start at tail so first element will be at 0 */
    }
int circ_get(int i) {
    int ii;
    /* compute the buffer position */
    ii = (pos-i)%CMAX
    return circ[ii]; /* return the value */
    }
```

Pengju Ren@IAIR 2017-2018

循环缓冲区应用：FIR

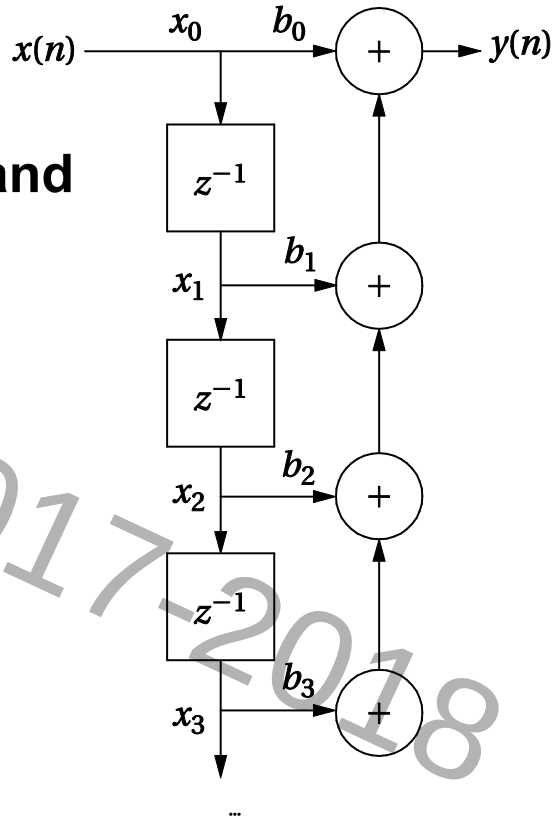
```
void circ_update(int xnew) {
    /* add the new sample and push off the oldest one */
    /* compute the new head value with wraparound; the
          pos pointer moves from CMAX-1 down to 0 */
    pos = ((pos == 0) ? CMAX-1 : (pos-1));
    /* insert the new value at the new head */
    circ[pos] = xnew;
    }
```



Pengju Ren@IAR 2017-2018

循环缓冲区应用：FIR

```
int fir(int xnew) {
    /* given a new sample value, update the queue and
          compute the filter output */
    int i;
    int result; /* holds the filter output */
    circ_update(xnew); /* put the new value in */
    for (i=0, result=0; i<CMAX; i++)
          result += b[i] * circ_get(i);
    return result;
    }
```



Pengju Ren@IAIR 2017-2018

队列（1）

- 队列也被看作是**弹性缓冲区**：数据不规则的到达。
- 构建队列的方法
  - 链表：队列大小没有限制，但需要动态分配内存
  - 数组：缓冲区

Pengju Ren@IAIR 2017-2018

队列（2）基于缓冲区的队列

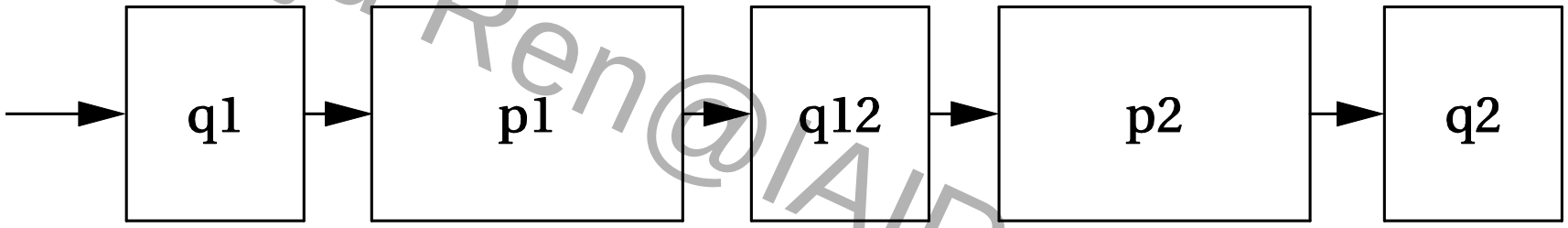
```
#define Q_SIZE 32
#define Q_MAX (Q_SIZE-1)
int q[Q_SIZE], head, tail;
void initialize_queue() {
        head = 0; tail = Q_MAX; }
Int wrap (int i) {
         return ((i+1) % Q_SIZE; }

void enqueue(int val) {
     if ((wrap(tail) == head)
        error("enqueuer onto full queue");
     tail = wrap(tail);
     q[tail] = val;
}
```

```
int dequeue() {
     int returnval;
     if (head == tail) error
     ("dequeuer from empty
     queue");
     returnval = q[head];
     head = wrap(head);
     return returnval;
}
```

Pengju Ren@IAIR 2017-2018

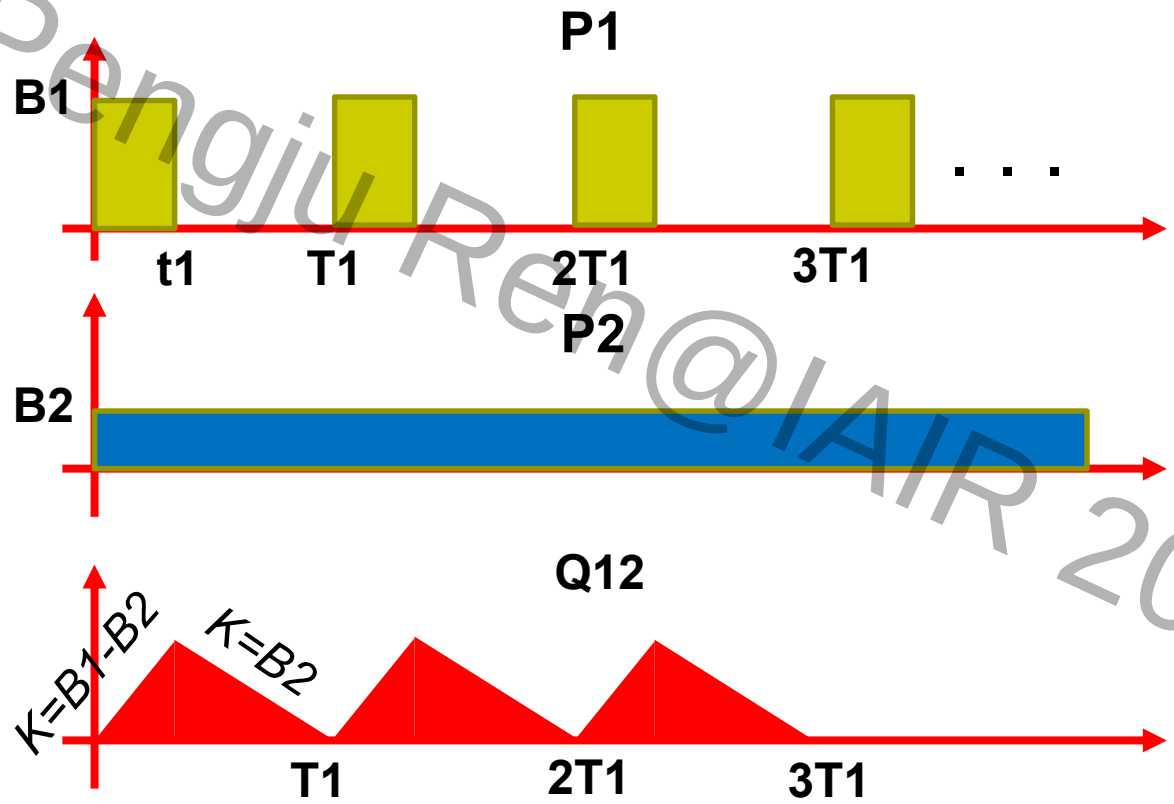
生产者/消费者队列



用队列来满足不同的输入和输出速率

Pengju Ren@IAIR 2017-2018

生产者/消费者队列



B2=？

B2=B1*t1/T1

Q12的大小？

Q12=t1*(B1-B2)/2

如果两个不同输入队列 P1,P1’（生产者），一个输出P2（消费者），如何设计？

1） P1，P1’周期相同，相位不同？B2为恒定带宽
2） P1，P1’周期不同，相位不同？B2为恒定带宽
3） P1，P1’周期相同，相位不同？B2为周期Burst传输
4） P1，P1’周期不同，相位不同？B2为周期Burst传输

Pengju Ren@IAIR 2017-2018

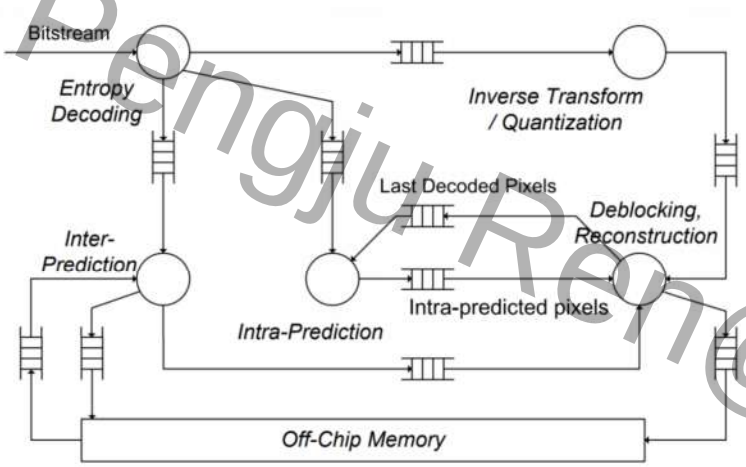
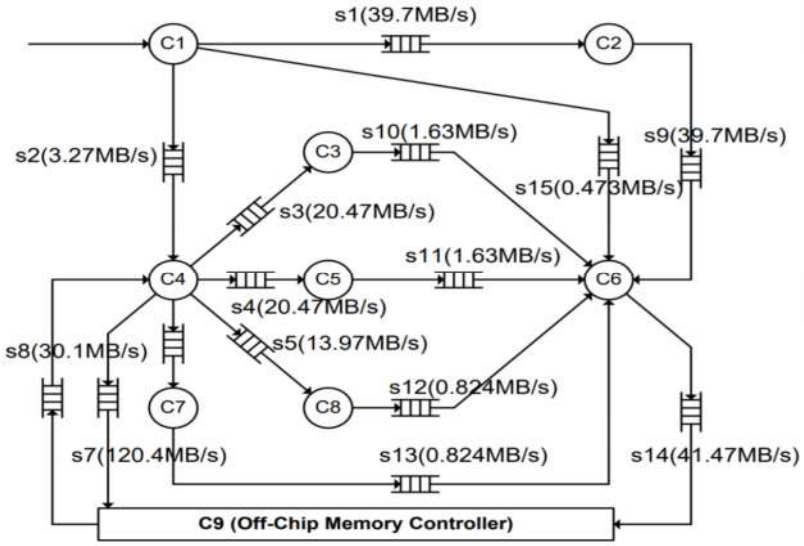
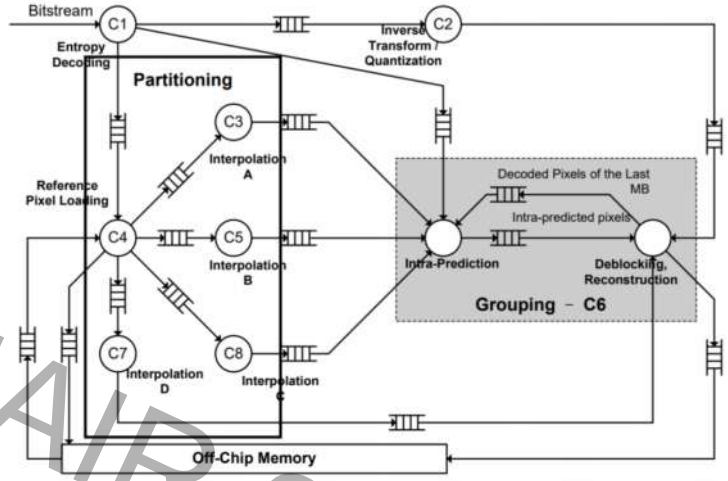
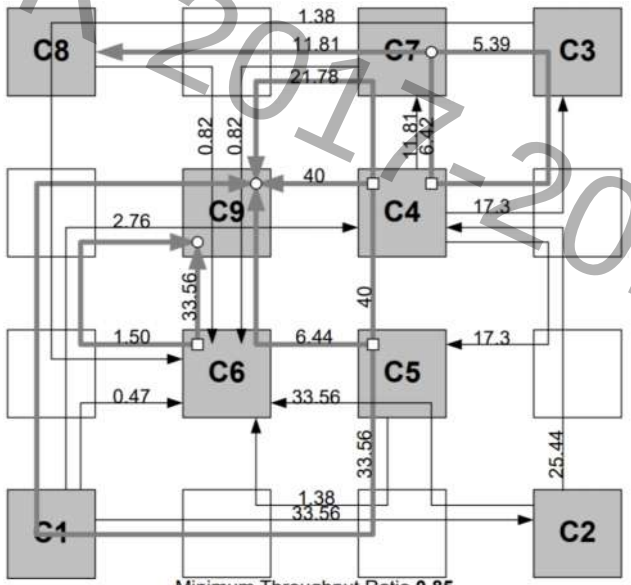


Fig. 1. High-level module description of H.264 decoder.







Minimum Throughput Ratio **0.85**
20 Total Paths (split path in thick and gray)

Pengju Ren@IAR 2017-2018

2 程序模型

- 源代码并不是好的程序表示方式:
  - 复杂难懂;
  - 隐含着很多信息。
- 编译器提取中间表示来处理和优化程序。

- 控制/数据流图(control/data flow graph, CDFG)
  - 数据操作（算法和其他计算）
  - 控制操作（条件式conditionals）

Pengju Ren@XIR 2017-2018

数据流图

- 数据流图(**data flow graph, DFG**)：无条件式的程序模型。
  - 不能表示控制
- 基本语句块：一个无条件式的代码段，准确的说：代码只有一个入口和一个出口。
  - 程序从开始处进入该基本语句块并执行全部语句

Peng Ren@IAIR 2017-2018

单赋值形式

```
x = a + b;
y = c - d;
z = x * y;
y = b + d;
```

基本语句块

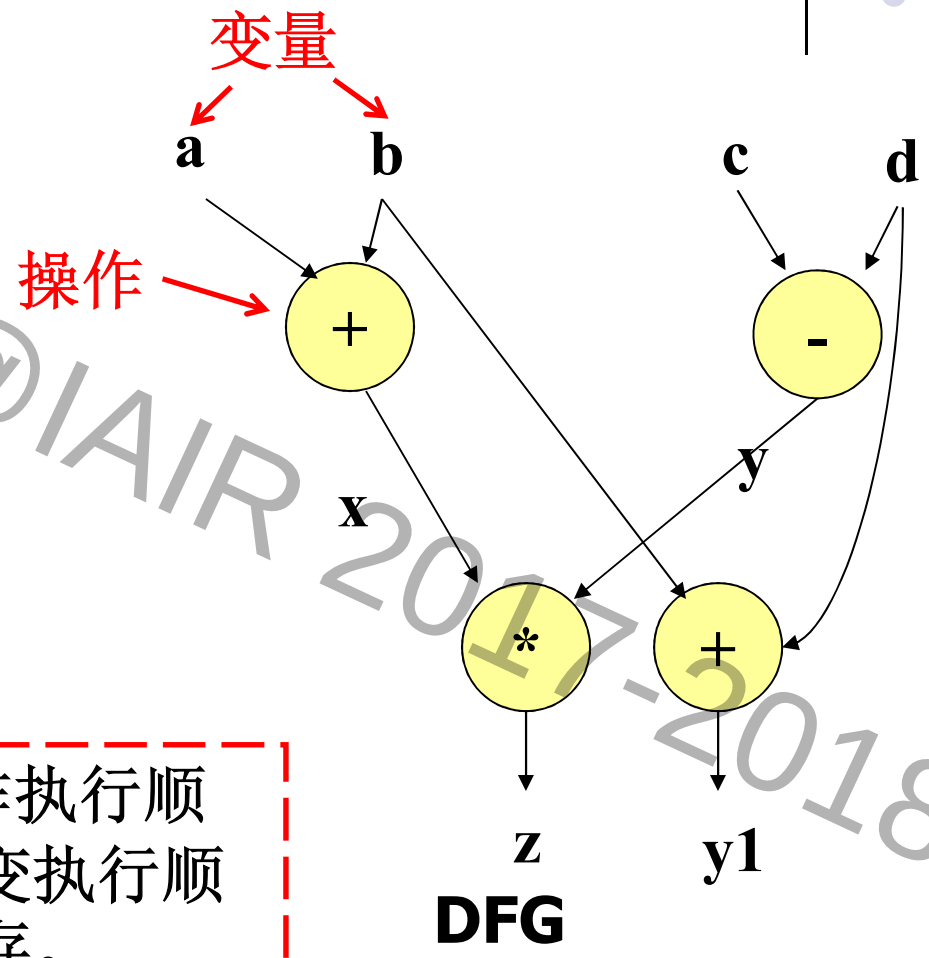
```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

单赋值形式

Pengju Ren@IAIR 2017-2018

数据流图

```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```



单赋值形式

数据流图使C代码中的操作执行顺序不清晰。优势，可以改变执行顺序，优化流水线或高速缓存。

Pengju Ren@IAIR 2017-2018

控制/数据流图

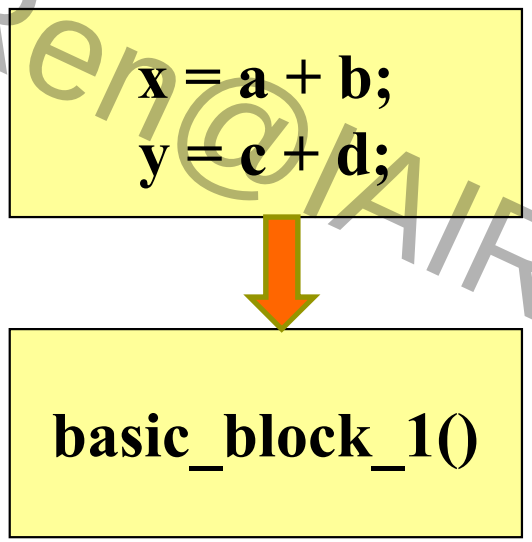
- **CDFG**用数据流图作为一个元素，添加描述控制的结构。

- 基本的**CDFG**有两种节点：
  - 判定节点**(decision node)**
  - 数据流节点 **(data flow node)**：包括一个完整的表示基本语句块的数据流图

Pengju Ren@IAIR 2017-2018

数据流节点

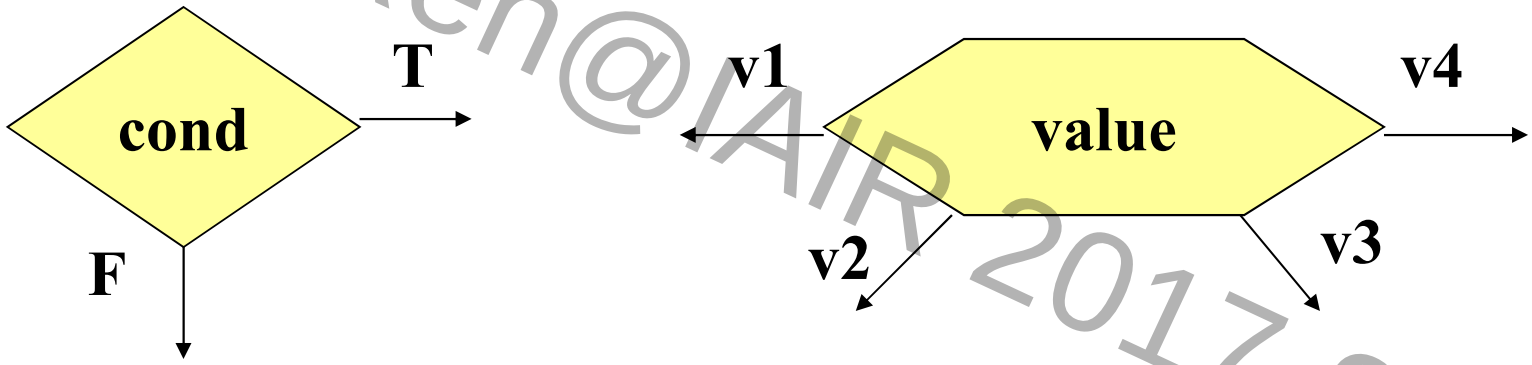
用矩形节点表示基本语句块



为了简便，C代码中的基本语句块被表示为函数调用的形式。

Penoju Ren@IAIR 2017-2018

判定节点

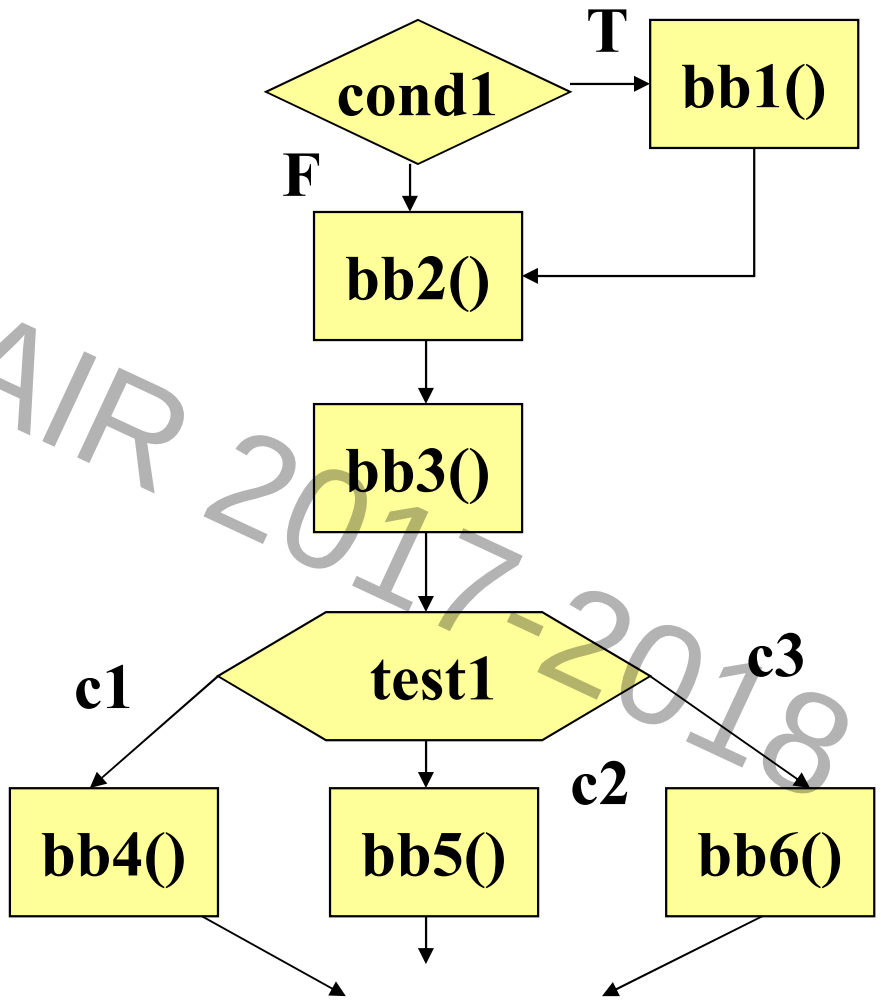


Equivalent forms

Pengju Ren@IAIR 2017-2018

CDFG示例

```
if (cond1) bb1();
else bb2();
bb3();
switch (test1) {
    case c1: bb4(); break;
    case c2: bb5(); break;
    case c3: bb6(); break;
}
```



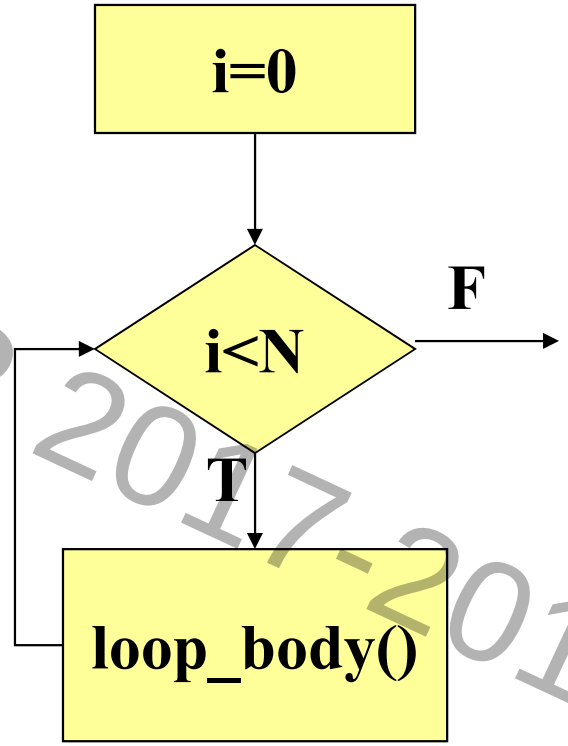
Pengju Ren@IAIR 2017-2018

For循环的While等价形式

```
for (i=0; i<N; i++)
    loop_body();
```

for loop

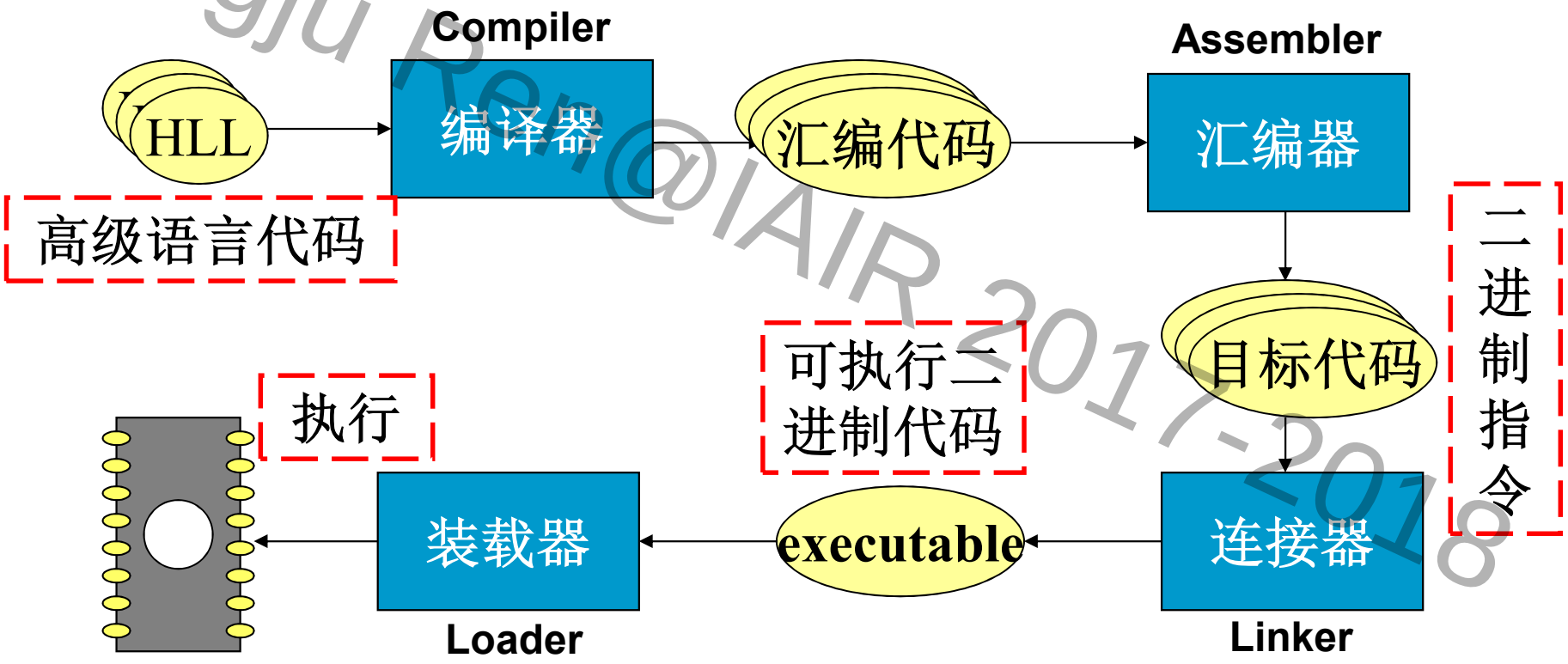
```
i=0;
while (i<N) {
    loop_body(); i++; }
```



Pengju Ren@IAIR 2017-2018

3 汇编、连接和装载

- 汇编和连接是编译处理过程的最后几步:



从编译到装载的程序创建过程

Pengju Ren@IAIR 2017-2018

多模块汇编程序

- 对于有多个组成文件的汇编程序，不希望在汇编前为所有模块指定起始地址，否则需要确定程序长度和连接顺序。
- 汇编程序的地址有两种类型:
  - 绝对地址（**absolute addresses**）相对于**CPU**的起始地址计算。
  - 相对地址（**relative addresses**）相对于模块的起始地址计算。
    - 连接器负责把相对地址译成绝对地址！！！

Pengju Ren@XIR 2017-2018

汇编器

- 汇编程序的主要任务：
  - 为符号化指令生成二进制码
  - 将标记翻译成地址
  - 处理伪操作（伪指令）
- 汇编程序的难点主要在于处理标记。
- 汇编语言标记:

```
            ORG 100
label1      ADR r4,c
```

Pengju Ren@IAIR 2017-2018

标记处理过程

- 标记处理要求对汇编源代码进行两趟扫描处理
  - 第一趟扫描处理：决定每个标记的地址
  - 第二趟扫描处理：用第一趟计算的标记值汇编指令
- 汇编器在第一趟扫描过程中创建符号表

PLC， Program Location Counter
程序位置计数器

	ADD r0,r1,r2	
→ xx	ADD r3,r4,r5	
	CMP r0,r3	
yy	SUB r5,r6,r7	

汇编代码

标记名称	标记值
xx	0x4
yy	0xC

符号表
symbol table

Pengju Ren@XIR 2017-2018

创建符号表

- 如何知道PLC的起始值？
  - 程序起点　ORG语句
  - ORG　2000
  - 汇编程序一般允许一个程序有多个ORG语句

PLC=100	ORG 100	xx	0x104
PLC=100	ADD r0,r1,r2	yy	0x10C
PLC=104	ADD r3,r4,r5		
PLC=108	CMP r0,r3		
PLC=10C	SUB r5,r6,r7		
yy			

Pengju Ren@XIR 2017-2018

伪操作

- 汇编中对伪操作（伪指令）的处理:
  - **ORG** 设置语句的起点。
  - **EQU** 允许将标记添加到符号表，而不增加**PLC**。
  - **DCD** 数据声明，定义数据块。
  - **ADR** 在寄存器中创建地址。

```
        ADD r0,r1,r2
FOO     EQU 5
BAZ     SUB r3,r4,#FOO
```

汇编代码

Pengju Ren@IAIR 2017-2018

Review：DCD的使用

- C程序：switch (test) { case 0: … break; case 1: … }
- 汇编程序：

```
        ADR  r2, test            ; get address for test
        LDR  r0, [r2]            ; load value for test
        ADR  r1, switchtab       ; load address for switch table
        LDR  r15, [r1, r0 LSL #2] ; index switch table
switchtab  DCD case0
           DCD case1
           ...
case0      …            ; code for case 0
           …
case1      …            ; code for case 1
```

DCD伪指令用于分配一片连续的字存储单元并用伪指令中指定的表达式初始化。其中，表达式可以为程序标号或数字表达式。

Pengju Ren@IAIR 2017-2018

Review: ADR的使用

- 使用ADR伪指令：在汇编源代码中，伪指令就像一般的指令，只是它没有直接对应于特定的ARM指令。使用时，汇编器都有一定的规则选择最适当的指令或几条相连指令。ADR->ADD/SUB

```
copy        ADR     r1, TABLE1      ; r1 points to TABLE1
            ADR     r2, TABLE2      ; r2 points to TABLE2
            LDR     r0, [r1]        ; load first value ....
            STR     r0, [r2]        ;    and store it in TABLE2
            .......
TABLE1      ......                  ; <source of data>
            ......
TABLE2      ......                  ; <destination of data>
```

Pengju Ren@IAIR 2017-2018

相对地址生成

- 一些标记的值在汇编的时候可能并不知道
- 模块内的标记可能被保留成相对的形式
- 必须跟踪外部标记 – 不能对那些使用外部标记的指令生成二进制代码

Pengju Ren@IAIR 2017-2018

连接

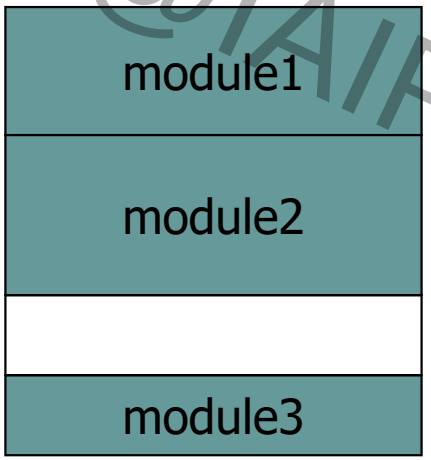
- 连接是将多个汇编程序生成的目标代码模块组成一个可执行文件的过程。

- 连接完成的工作:
  - 确定模块间的顺序
  - 解决跨模块跨文件的标记问题

Pengju Ren@IAIR 2017-2018

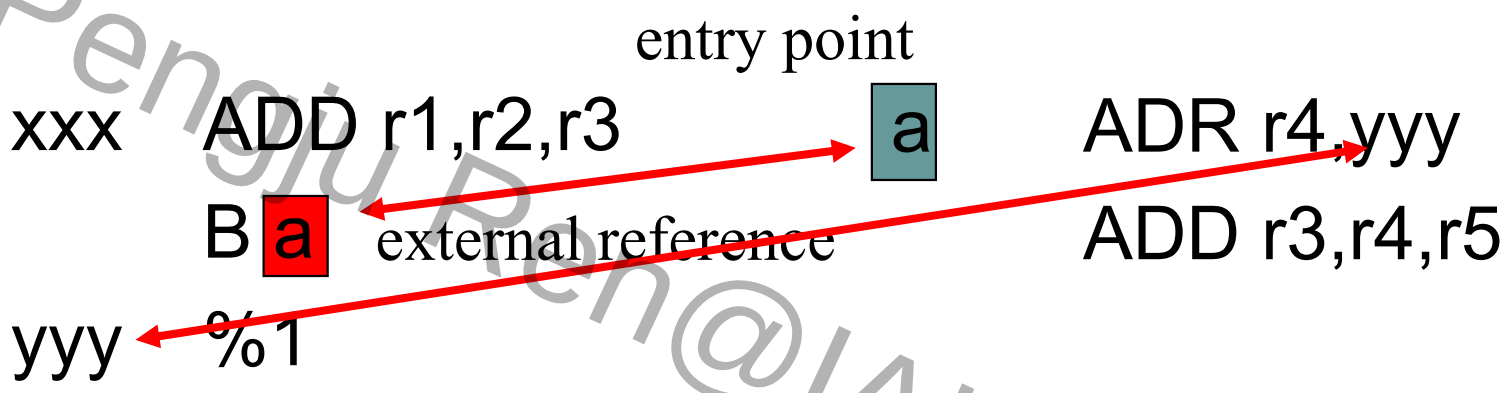
确定模块间的顺序

- 代码段必须被放置在存储空间的绝对位置.
- 装载图或链接标志会控制模块的放置顺序。



Pengju Ren@IAIR 2017-2018

入口点与外部引用（1）



- 程序中定义标记的地方称为入口点（**entry point**）
- 程序中使用标记的地方称为外部引用（**external reference**）

Pengju Ren@IAIR 2017-2018

入口点与外部引用（2）

```
label1   LDR r0,[r1]
         ...
         ADR a
         ...
         B label2
         ...
var1     % 1
```

```
label2   ADR var1
         ...
         B label3
         ...
x        % 1
y        % 1
a        % 10
```

External references	Entry points
a	label1
label2	var1

File 1

External references	Entry points
var1	label2
label3	x
	y
	a

File 2

Pengju Ren@IAIR 2017-2018

连接的过程

- 连接分两个阶段：
  - 第一个阶段，要确定每个目标文件的开始地址。
    - 目标文件的装入次序：用户通过制定参数或创建装载映像文件来给定次序。
    - 每个目标文件的长度
  - 第二个阶段，将来自全部目标文件的符号表合并成一个大的符号表，然后编辑目标文件，将相对地址转成绝对地址。
- 嵌入式系统中，代码模块载入内存的位置控制很重要！

Pengju Ren@IAIR 2017-2018

动态链接库

- 动态链接库（**dynamically linked library**）一般用于工作站和**PC**，有些嵌入式计算环境也支持。
- 动态链接库不是将一个通用例程连接到系统中的每一个可执行程序，而是允许他们在程序执行的开始连接
- **Pros:**
  - 一个库文件可以在多个可执行文件间共享，节省存储空间；
  - 使得使用这些库的程序易于更新。
- **Cons:**
  - 在程序开始执行前引入一个延迟。

Peng Ren@IAIR 2017-2018

目标代码设计

- 连接器控制目标代码模块在存储器中的位置，但是有些数据类型的放置点需要控制
  - I/O设备的中断向量和其他信息必须被放置在特殊位置
  - 必须设置内存管理表（下页）
  - 用于进程间通信的全局变量必须放置在所有该数据的用户都可访问的位置。

Peng Ren@IAIR 2017-2018

Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Nuvoton NUC710

Xtal (MHz): 33.0

Operating system: None

System-Viewer File (.Sfr):

Code Generation

ARM-Mode

Use Cross-Module Optimization

Use MicroLIB

Big Endian

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☑	ROM1:	0x0000	0x1000	◉
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
☐	IROM1:			○
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
☑	IRAM1:	0xFFE00000	0x2000	☐
☐	IRAM2:			☐

OK | Cancel | Defaults | Help

Pengju Ren@IAR 2017-2018

目标代码设计

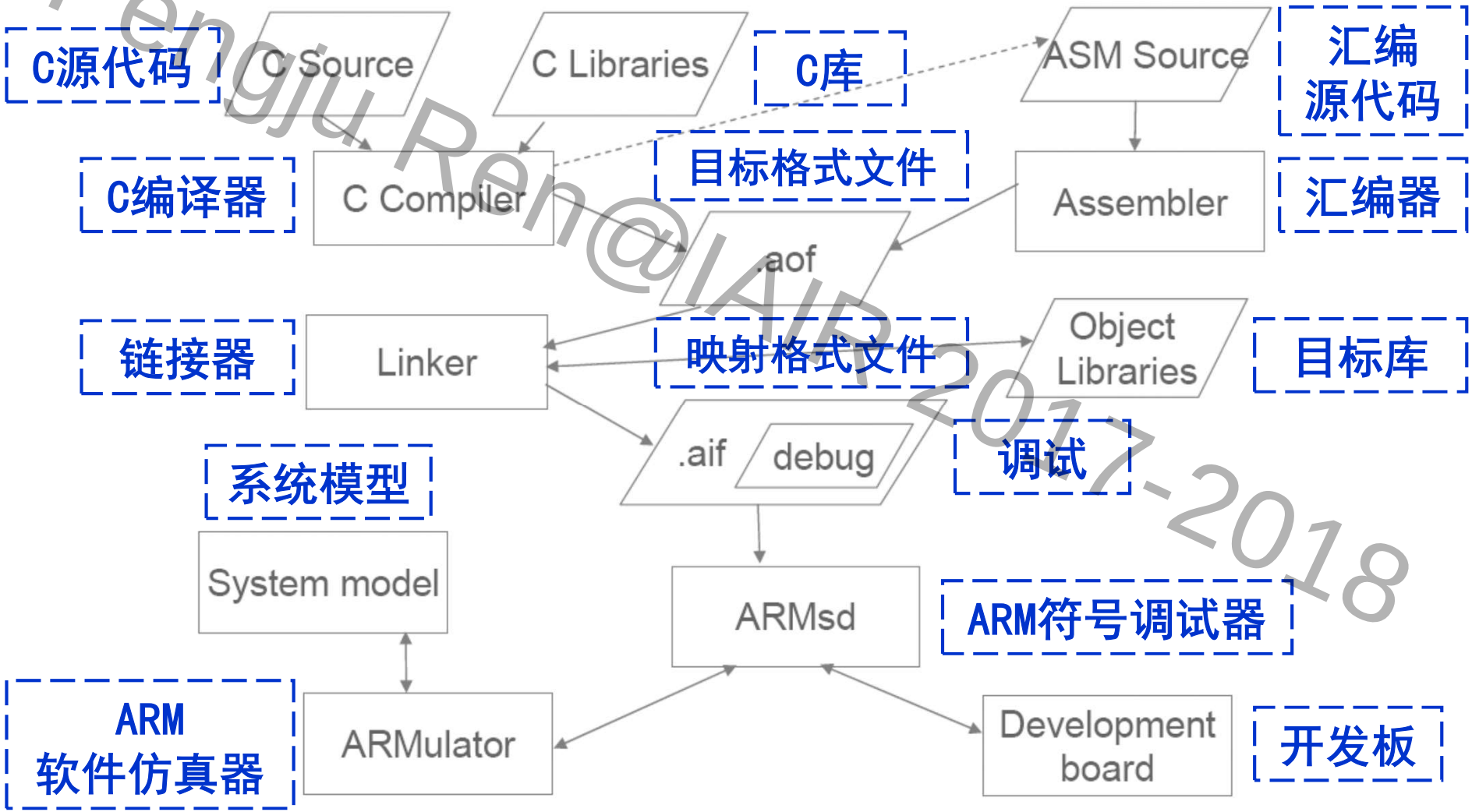
- **可重入**：如果程序执行一个函数时被另一个执行该函数的调用中断而不改变函数结果，则该函数是可重入的。否则，则不是；

```
int foo = 1;
int task1() {                    int task1(int foo){
        foo = foo + 1;                   return foo+1;
        return foo;}             }
```

- **可重定位**：若一个程序置于内存不同位置均可执行，则称之为可重定位。有时候需要使用不可重定位地址，如I/O设备寻址时。

Pengju Ren@IAIR 2017-2018

3 ARM开发工具



Pengju Ren@IAIP 2017-2018

ARM交叉开发工具链（1）

- 将C或汇编语言源文件编译或汇编成ARM的目标格式（.aof）文件，进而链接为ARM映射格式（.aif）文件。
- 映射格式文件可被构造为包括ARM符号调试器（ARMsd）所需的调试表，用于硬件开发板或软件仿真（ARMulator）
- ARM C编译器：通ANSI的C标准兼容，并得到适当的标准函数库支持，产生汇编源程序输出。
- ARM汇编器：是完全宏汇编器，产生ARM目标格式输出，输入可以与C编译器的输出链接。
- 链接器：链接器将一个或多个目标文件组合为可执行的程序，它解决目标文件之间的符号引用，以及在程序中需要时从库中提取目标模块。

Pengju Ren@IWR 2017-2018

ARM交叉开发工具链（2）

- **ARMsd**：ARM符号调试器是协助调试程序的前端接口。调试程序既可以再仿真环境下（ARMulator）运行，又可以在远程目标系统（ARM开发板）上运行。远程系统必须通过串行线或JTAG测试接口支持相应的远程调试协议。ARMsd支持设置断点和观察点。
- **ARMulator**：是一套在主系统上用软件模拟各种ARM处理器核行为的程序。它可以在不同精度级别上运行：
  - 指令精度模型给出系统状态的确切行为，而不考虑处理器的精确时序特性。
  - 周期精度模型逐个周期地给出处理器的确切行为，能准确给出程序所需的时钟周期数。
  - 时序精度模型给出信号在周期内的准确时序，能够考虑逻辑延时。

Pengju Ren@AIP 2017-2018

ARM交叉开发工具链（3）

- **ARMulator续：**
  - 简单地说，**ARMulator**使得用**C**编译器或汇编器开发的**ARM**程序能够在没有**ARM**处理器的主机上进行测试和调试，使得程序执行所用的时钟周期得以精确测量，目标系统的性能得以评估。
  - 复杂地说，**ARMulator**可以作为目标系统的完全的、具有时序精度的**C**模型来使用，它具有全部的**Cache**细节和存储器管理功能，运行某一操作系统。
- **ARM开发板**
- 软件工具套件：**ARM ADS**是**ARM**公司推出的新一代**ARM**集成开发工具，支持**Windows**和**Linux**操作系统。

Pengju Ren@IAIR 2017-2018