CH0

-ALU: Arithmetic-Logic Unit; **-FPGA:** Field Programmable Gate Array **-HDL:** Hardware Description Language **-VHDL:** Very-High-Speed Integrated Circuit Hardware Description Language **-Verilog:** Verify Logic **-VHSIC**: Very-High-Speed Integrated Circuit

CH1

-Combinational logic: 当前输出只与当前输入有关

-Sequential logic: present output depends on the present input and past sequence of inputs

-Boolean Algebra and Algebra Simplification:
分配律 X+YZ=(X+Y)(X+Z) 化简定理 X(X+Y)=X XY'+Y=X+Y XY+YZ+X'Z=XY+X'Z

-DeMorgan: [f(X1,X2,...,,0,1,+,•)]'=f(X1',X2',...,Xn',1,0,•,+)

-Duality: $[f(X1,X2,...,Xn,0,1,+,\bullet)]^D$=f(X1,X2,...,Xn,1,0,•,+)

注意替换顺序，先取反后变号 (A+BC)'=A'(B'+C')

-Karnaugh maps: 相邻的两个方块之间只有一个变量不同；最上面一行和最下面一行是相连的，左右也是；

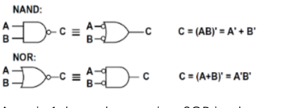
-don't care: 不会出现或者该组合未定义。

-prime implicant 质蕴涵项: 1,2,4,8 个相连的 1;基本质蕴涵项是至少包含一个其他质蕴涵项没有的 1 的项

MinSOP 最小与或: Σmi MinPOS 最小或与表达: ΠMi

MinSOP = MinPOS' 利用 DeMorgan laws 转化

- NAND and NOR Gates



-Static 1/0-hazard:本应该是保持 1/0 但有时候变成 0/1

A static 1-hazard occurs in a SOP implementation when two minterms differing by only one input variable are not covered by the same product term

-Dynamic hazard: 当输出应该从 1 到 0 或者反过来的时候，the output may change three or more times

触发器只有在使能信号上升和下降的时候内容才会变化，但是锁存器的变化是 immediately

-Clocked D flip-flop(rising): $Q^+=D$

-Clocked J-K flip-flop(falling): $Q^+=JQ'+K'Q$

-Clocked T flip-flop(rising): $Q^+=QT'+Q'T=QxorT$

-S-R (set-reset) latch: $Q^+=S+R'Q$

-Gated D latch: $Q^+=GD+G'Q+DQ$

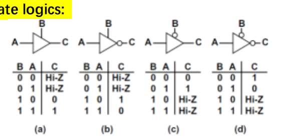
-Mealy Circuit: 输出与当前状态和当前输入有关

-Moore Circuit: 输出只与当前状态有关，但可以有输入

-最优状态分配: 1.States which have the same next state (NS) for a given input should be given adjacent assignments 2. States that are the next states of the same state should be given adjacent assignments 3. States that have the same output for a given input should be given adjacent assignments **to clump 1's together on K-map**

-状态等价: 两个状态等价的充要条件是对于每个输入 X 输出都一样并且下一个状态都一样。

-Tristate logics:



B A	(a) C	(b) C	(c) C	(d) C
0 0	Hi-Z	Hi-Z	0	1
0 1	Hi-Z	Hi-Z	1	0
1 0	0	1	Hi-Z	Hi-Z
1 1	1	0	Hi-Z	Hi-Z

CH2

-VHDL intro: The actual circuit will depend on the compiler/optimizer being used; Top-down design;
1.Behavioral level: 行为级，无需给出逻辑表达式; Data flow level: 数据流级，需给出逻辑方程; Structural level: 结构级，指定构成加电路的门和门之间的互连;

-Concurrent statements: 并列，语句右边的值变化就会马上执行；如果没有指定 delay, **delay=delta**

the time statement executes≠the time signal updated

-标识符命名规则: 字母开头，可由字母、数字、下划线组成，不能下划线结尾，不能使用保留字;

-初始值: 只对仿真有意义 :=

-Buffer mode: 表明是对外部世界的输出同时也能被结构里面读，inout 表示是双向但是不是电路的外部输入

-Process: 当 sensitive list 里的信号发生变化之后进程开始执行，结束之后返回开头等待敏感列表里面信号变化 Processes, functions, and procedures are the onlysections of code that are executed sequentially 但是这些代码块和外部的语句依然是并发的; **if 语句不能用作进程外的并发语句;** 函数中禁止进行信号声明和元件实例化

-wait: 对于进程的敏感列表的替代方式是使用 wait 语句

`wait on sensitivity-list;` 等待列表中信号发生变化

`wait for time-expression;` 等待一段时间经过

wait for 0 ns = wait for a delta time

`wait until Boolean-expression;` 当表达式中的信号变化时对表达式估值，为真就继续运行

如果多个语句对一个信号赋值，最后的赋值会 override

一个进程必须且只能包含敏感列表或者 wait 语句之一

-delay: 分为传输延时和惯性延时

-Transport delay: 单纯延时

`signa <= transport expression after delay-time;`

-Inertial delays(d): 不会将过短的脉冲从输入给输出，如果脉冲宽度小于 pulse-width 就消除 `signal_name <= reject pulse-width inertial expression after delay-time;`

实际上使用 reject 相当于用二者的组合: `Z3 <= reject 4 ns inertial X after 10 ns;` 等价于 `Zm <= X after 4 ns; Z3 <= transport Zm after 6 ns;`

-Simulation: 1.Analysis (compilation)分析（编译） 2. Elaboration 细化 3. Simulation 仿真

-Elaboration: ①Ports are created for each instance of a component ②Memory storage is allocated for the required signals ③The interconnections among the port signals are specified ④A mechanism is established for executing VHDL statementsin the proper sequence; ⑤The resulting data structure represents the digital system being simulated;

如果一个 model 里面有多个进程，他们是并行的，这些进程和其他的语句也是并行的。 A process **takes no time** to execute unless it has wait statements in it. Signals **take delta time** to update when no delay is specified;

-Data types: Real and time types are not synthesizable
VHDL 是强类型，如果类型不匹配那么需要显式的类型转换或者重载运算符

example: Ci 为 bit 类型，A, B, Sum 为 unsigned，则应 `Sum<='0' & A+B+unsigned'(0=>Ci)`, bit 不能直接与 unsigned 相加。

-Operators: 从下到上优先级降低

1	Binary logical	and or nand nor xor xnor
2	Relational	= /= < <= > >=
3	Shift	sll srl sla sra rol ror
4	Adding	+ - &
5	Unary sign	+ -
6	Multiplying	* / mod rem
7	Miscellaneous	not abs **

-shift: l = logical, a = arithmetic

sll: 左移补零 **srl**:右移补零 **rol:** 循环左移 **ror:** 循环右移

sla: 左移用最右边的位补 **sra:** 右移用最左边的位补

example: A = "10010101", A sll 2="01010100", A sla 3 ="10101111", A ror 5 ="10101100"

-条件信号赋值(simple when):

```
signal_name <= expression1 when condition1
       {else expression2 when condition2}
       [else expressionN];
```

-选择信号赋值(selected when):条件完全 others 可省略

```
with expression_s select
signal_s <= expression1 [after delay-time] when
choice1,
expression2 [after delay-time] when choice2,
...[expression_n [after delay-time] when others];
```

-case（这个在 process 里用，上面两个都是并发的）

```
case expression is
    when choice1 => sequential statement1
    when choice2 => sequential statement2
...[when others => sequential statements]
end case;
```

-IEEE: Institute of Electrical and Electronic Engineers.

- IEEE.std_logic: has nine values, including '0', '1', 'X'(unknown), and 'Z' (high impedance) 没有运算

- IEEE.numeric_bit:有符号数用补码表示，有运算重载

"1011" + "110" = "1011" + "0110" = "0001"进位扔掉

-Variables: 信号在延迟后更新（new value generally only available at the conclusion of the Process, Function, or Procedure），变量立即更新；信号用<=赋值，变量用:=赋值；信号必须在进程外声明和初始化，变量必须在进程内；变量的更新是立即的，后续计算会用新值。

-数组: 声明类型

`type SHORT_WORD is array (3 downto 0) of bit;`
实例 `signal word1: short_word :="1101";`

-多维数组: 声明类型: `type matrix is array (1 to 4, 3 downto 1) of integer;`
实例 `variable mat : matrix := ( (1,2,3), (4,5,6), (7,8,9), (10,1,12) );`

矩阵索引: mat(2, 1)，第 2 行最后一个;

-无约束数组: 声明无约束数组: `type int_vec is array (natural range<>) of integer;` 实例必须指明 range: `signal int_vec5 : int_vec(1 to 5) := (3, 2, 6, 8, 1);` 声明字符串: `type string is array ( positive range <> ) of character;` 实例: `constant string1 : string(1 to 29) :="This string is 29 characters."`

-Subtype: 在声明类型之后，可以声明相关的子类型以包含该类型指定的值的子集。`subtype short_word is bit_vector(15 downto 0);`

-Loop: `[loop-label:] loop sequential statements end loop [loop-label];` exit: `exit;`或者 `exit when condition;` loop 是 sequential 的，只能在**进程函数过程**里面用，对于 for-loop: `[loop-label:] for loop-index in range loop sequential statements end loop [loop-label];`**loop-index 是进入循环之后自动定义的，不能被更改，但可以使用，loop 结束后不可用**
`[loop-label:] while condition loop sequential statements end loop [loop-label];`

IF, WAIT, CASE, LOOP are intended exclusively for sequential code，They can only be used inside a PROCESS（进程）, FUNCTION（函数） or PROCEDURE（过程）

CH3

-FPGA intro: FPGA is IC(集成电路) that contain an array of identical logic blocks with programmable interconnections. The user can program the functions realized by each logic block and the connections between the blocks. FPGAS are **less dense** than traditional gate arrays. In FPGAS, **a lot of resources** are spent merely to achieve the needed programmability. Programmable points have resistance and capacitance. They **slow down** signals, so FPGAS are **slower** than traditional gate arrays. **Interconnection delays are unpredictable** in FPGAS. "Field" programmability is achieved by reconfigurable elements(可被用户控制). FPGA 内部通常包含三个元素: **Programmable logic blocks、Programmable input/output blocks、Programmable routing resources.**

-MPGA: mask programmable gate arrays.

-Layout of FPGA: Arrays of programmable logic blocks are distributed within FPGA. Logic blocks are surrounded by input/output interface blocks. These I/O blocks can be considered to be on the periphery of the chip. They connect the logic signals to FPGA pins. The space between the logic blocks is used to route connections between the logic blocks.

-Programmable logic blocks: created by using multiplexers, look-up tables, and AND-OR or NAND-NAND arrays. Programming means: ①changing the input or control signals to the multiplexers. ②changing the look-up table contents. ③selecting or not selecting particular gates in AND-OR gate blocks.

-Programmable interconnect: be required to interconnect various blocks in the chip and to connect specific I/O pins to specific logic blocks. Programming means making or breaking specific connections.

-Programmable I/O blocks: can be programmed to be input, output, or bidirectional lines(双向线路).

The general-purpose interconnect gives FPGA a lot of **flexibility** But it has the disadvantage of being **slow**

-Architectures for FPGAs: Matrix-based(symmetrical array) architecture: 矩阵(对称阵列)型、Row-based architecture: 横向型、Hierarchical PIL architecture: 从属型、Sea-of-gate architecture: 门海型; classification is based on the layout of the general purpose logic region in the FPGA

-Matrix-based Architecture: The logic blocks are organized in a matrix-like fashion. The logic blocks in these architectures are typically of a large granularity(粒度) (capable of Implementing 4-variable functions or more). These architectures typically contain 8×8 arrays in the smaller chips and 100×100 or larger arrays in the bigger chips. 二维通道布线 Two-dimensional channeled routing: routing resources are generally available in horizontal and vertical directions.

-Row-based architecture: The logic blocks in this architecture are organized into rows. There are rows of logic blocks and routing resources. 一维通道布线 One-dimensional channeled routing: the routing resources are located as a channel in between rows of logic resources.

-Hierarchical architecture: Blocks of logic cells are grouped together by a local interconnect. Several such groups are interconnected by another level of interconnect. There is a hierarchy in the organization of these FPGAs: ①these FPGAs contain clusters of logic blocks with localized resources for interconnection. The global interconnect network is used for the interconnections between the clusters of logic blocks.

-Sea-of-gate architecture: The general FPGA fabric(结构) consists of a large number of gates. There is an interconnect **superimposed on the sea of gates.**

-reconfigurability: can be achieved by: ①changing the contents of static RAM cells; ②changing the contents of flash memory cells; ③fusing metal links. FPGAS use one of the following programming methods: ①StaticRAM programming technology; (key idea is to use **pass transistors** to create switches and then control them using SRAM content)
②EPROM/ EEPROM/ flash programming technology;
③AntiFuse programming technology **(irreversible, but faster)**

Only SRAM and EEPROM/Flash programming technologies allow in-circuit programmability 在线可重构编程

-Routing Matrix for General-Purpose Interconnection in an FPGA: Many FPGAS use switch matrices that provide interconnections between routing wires connected to the switch matrix. Each cross point in the switch matrix must support **six** possible interconnections.

-Direct Interconnects between Neighboring Logic Blocks: Many FPGAS provide direct interconnections to the four nearest neighbors(有时是 8 邻域)。直接互连不通过开关矩阵，而是通过专用 dedicated switches 开，延迟小。

-Global Lines: For purposes such as high fan-out and low-skew clock distribution, most FPGAs provide routing lines that span the entire width/height of device. A limited number (two or four) of such global lines are provided by many FPGAS in the horizontal and vertical directions.

-Typical Routing Resources in a Row-Based FPGA: The interconnects in row-based channeled architecture can be classified into two categories: non-segmented routing、segmented routing

-I/O Block: I/O blocks on modern FPGAs allow use of the pin as input and/or output, in direct (combinational) or latched forms, in tristate true or inverted forms, and with a variety of I/O standards. To use the cell as an **output**. The tristate buffer must be enabled. To use the cell as an **input**, the tristate control must be set to place the tristate buffer, which drives the output pin, in the high-impedance state.

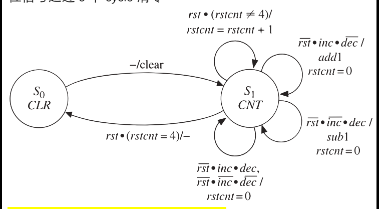
CH4

-Design: Design methodology splits a design into a "data path" and a "controller". ①**Controller**: sends control signals or commands to data path and obtain feedback(status signals) from data path. ②**Data path**: hardware that actually performs the data processing.

-CLA: 先行进位加法器 Carry look-ahead adders. Ci+1 = Ai·Bi + (Ai ⊕ Bi)·Ci = Gi + Pi, Gi = Ai·Bi, Pi = Ai ⊕ Bi. **Gi**= indicates whether a stage should generate a carry. **Pi**= indicates whether an adder should propagate the carry it receives from the lower stage. 优点: delays will be the same independent of the number of bits we need to add. 缺点: 4bit 以上逻辑复杂 CLA adders are usually implemented as 4-bit modules and are used in a hierachical structure to realize 4n-bit adders.

-State Graphs for Control Circuits: If an arc is labeled with XiXj /ZpZq , it means ①if inputs Xi and Xj are 1 (we don't care what the other inputs are), ②outputs Zp and Zq are 1 (and the other outputs are 0) ③and we traverse this arc to the next state. In general, if we label an arc with an input expression, I, we will traverse the arc when **I=1. Constraints on the input labels:** ①If Ii and Ij are any pair of input labels on arcs exiting state Sk , then Ii·Ij = 0 ②if i≠j If n arcs exit state Sk and the n arcs have input labels I1 , I2 , ... ,In , respectively, then I1+I2+...+In = 1

-Score Board: 2 位 BCD 计数 inc/dec 信号，数码管显示，复位信号超过 5 个 cycle 清零


$rst \bullet (rstcnt \neq 4)/$
$rstcnt = rstcnt + 1$
$\overline{rst} \bullet inc \bullet \overline{dec}$ /
$add1$
$rstcnt = 0$
–/clear
S_0
CLR
S_1
CNT
$rst \bullet (rstcnt = 4)/–$
$\overline{rst} \bullet \overline{inc} \bullet dec$ /
$sub1$
$rstcnt = 0$
$\overline{rst} \bullet inc \bullet dec$,
$\overline{rst} \bullet \overline{inc} \bullet \overline{dec}$ /
$rstcnt = 0$

-Synchronization and Debouncing: 去抖动:remove the transients in the switch output. **Three flip-flops can provide debouncing, synchronization, and single pulsing.**

-Add-and-Shift Multiplier: Multiplicand=被乘数 A, Multiplier=乘数 B, A*B=C; 串并乘法器: Multiplier bits are processed serially, but the addition takes place in parallel.

-Array Multiplier: Array of AND gates and adders to perform multiplication. This multiplier has no sequential logic or registers. 全加器 3 输入，半加器 2 输入. Carry must propagate along each row of cells. Sum must propagate from row to row. **n-bit-by-n-bit array multiplier requires: n^2 与门, n(n-1) 加法器. Array multiplier:** number of components required increases **quadratically**; **Serial-parallel multiplier:** the amount of hardware required in addition to the control circuit increases **linearly with n. worst-case multiply time: (3n-4)tad+tg, tag=加法器延时, tg=与门延时**

- Signed number representations: 有符号的 4-bit 二进制数: +8 没有原反补码; -8 没有原码反码，补码为 1000; +0 原反补码皆为 0000; -0 原码 1000，反码 1111，补码 0000; **反码 One's complement: 正数反码等于原码，负数的反码等于相反数的按位取反; 补码 Two's complement: 正数补码等于原码，负数补码等于求出反码之后+1(+0 例外)。**

-Fraction: The **leftmost bit** represents the sign of the fraction (sign bit). A signed binary fraction x0 .x-1x-2x-3 represents the quantity x0 (-2^0) + x-1 (2^-1) + x-2 (2^-2) + x-3 (2^-3)

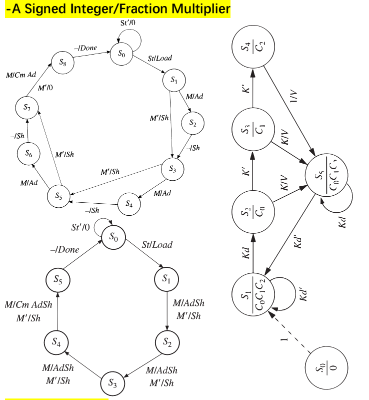
Example:
(0.101)two=0*(-2^0)+1*(2^-1)+0*(2^-2)+1*(2^-3)=5/8;
(1.011)two=1*(-2^0)+0*(2^-1)+1*(2^-2)+1*(2^-3)=-5/8

从右边开始找到第一个 1，对他左边的取反，比如:
(0.101)two=(5/8)dec，(1.011)two=(-5/8)dec

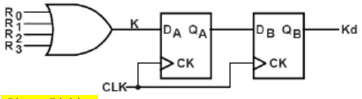
0.111... is the largest positive fraction,无法用补码表达 1.

特例: (1.000)two=(-1)dec (0.111)two=(7/8)dec

-A Signed Integer/Fraction Multiplier

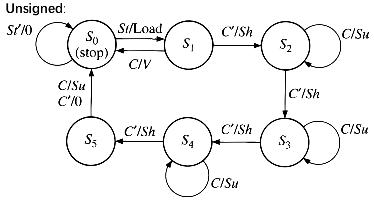


-Keypad Scanner

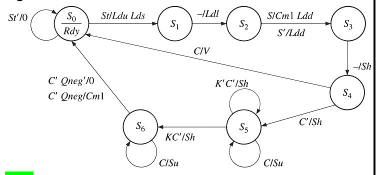


-Binary Dividers

Unsigned:



Signed:



CH5

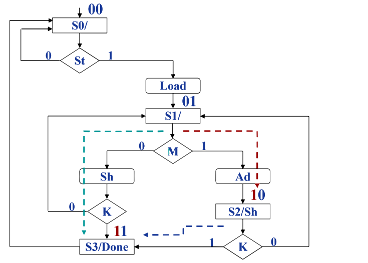
-SM Charts intro: SM is used to control a digital system carries out a **step-by-step** procedure or algorithm; **State graphs, state tables, and ASM charts** are different forms of FSM (Finite State Machine) representations

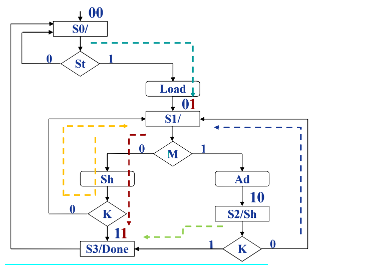
-State Graph: Proper state graph must obey some conditions: ①One and exactly one transition from a state must be true at any time; ②Next state must be uniquely defined for every input combination.**(Automatically satisfied for an SM chart)**

-SM Chart: state box, decision box, conditional output box.
SM chart is constructed from SM blocks: one state box、one entrance path、N exit path. No internal feedback within an SM block is allowed. SM block can have several parallel paths that lead to the same exit path. More than one of these paths can be active at the same time. All the tests take place within **one clock time** in both parallel and serial forms.

-Converting a State Graph to an SM Chart: Moore outputs: **in state boxes**. Mealy outputs: **in conditional output boxes**. Moore outputs change immediately **following a state change**, Mealy outputs change immediately **after a state change or an input change**. **All outputs will have their correct values at the time of the active edge.**

-Realization of SM Charts Example:





A+= A'BM'K + A'BM + AB'K =A'B(M+K)+AB'K
B+= A'B'St+A'BM'(K+K')+AB'(K+K')=A'B'St+A'BM'+AB'
每个 case 写一个 block 的代码

CH6

Functions 返回 single value, use **return** statement
Procedures 返回 any number of values, use **output** statement

-Functions: Functions, procedures: subprograms(子程序)

wait 语句、信号声明、component 实例化不能用在函数里

A function is a section of **sequential** code. The same statements that can be used in a process (**if, case, loop**) can also be used in a function, with the exception of **wait.**
Actual parameters are treated as input values and **cannot be changed** during the execution of the function 不能给输入赋值，可以函数内定义中间变量初始化为输入值然后去处理

For function, out or inout are not allowed

-Procedures: Procedure call can be a **sequential or concurrent** statement. Within procedure declaration: class (constant/signal/variable), mode, type are specified. **对 mode=in 则默认的 class=constant; 对 mode=out 和 inout 则默认的 class=variable; Parameters of modes out and inout can be changed in the procedure;**

		Actual Parameter	
Mode	**Class**	**Procedure Call**	**Function Call**
In¹	Constant²	Expression	Expression
	Signal	Signal	Signal
	Variable	Variable	n/a
Out/inout	Signal	Signal	n/a
	Variable³	Variable	n/a

1 Default mode for functions
2 Default for in mode
3 Default for out/inout mode
NOTE: n/a = "not applicable"

Function call: variable are not allowed

-Process, Function, Procedure:①The only pieces of sequential code②Employ same sequential statements (if, case, loop; **wait not allowed in function**)③进程: intended for immediate use in main code④函数和过程: mainly for library

-Package: package and package body must have the same name; 只有当包含一个或者多个 subprogram 的时候 Package body 才是必需的;

```
package package_name is
(declarations)
end package_name;
[package body package_name is
(function and procedure descriptions)
end package_name;]
```

-Component: component declaration 中的 port 和 entity declaration 里面 port 形式一样; component 的实例必须有 label;

-Attributes: Attributes can be associated with signals and arrays. VHDL 有两种 attributes: attributes that return a **value**、attributes that return a **signal; Event 事件=a change in the signal**; **Transaction 事务=the signal is evaluated, regardless of whether the signal changes or not**

-Attributes That Return a Value
①s'event: 如果在 Δ 时间内发生时间就返回 True;
②s'active: 如果 Δ 时间内发生 transaction，返回 True;
③s'last event: 返回距离上一个事件的时间;
④s'last value: 上一个值;

⑤s'last active：返回距离上一个 transaction 的时间；b. 返回信号：

-Attributes That Create a Signal [(Time)]可选默认 1 个 Δ

①s'delayed[(time)]：返回延时了 time 的信号 s，time 可选；
②s'stable[(time)]：返回 Boolean signal，如果指定时间(preceding)内 s 没有发生 events，信号值为 true；
③s'quiet[(time)]：返回 Boolean signal，如果指定时间(preceding)内 s 没有发生 transaction，信号值为 true(初始值为 false)；
④s'transaction：bit signal，s 发生 transaction 时，该信号发生翻转(初始值 1)；

-Array Attributes:

A can either be an array name or an array type

-4-Valued Logic System:

Symbols	represents ...	Examples
'X'	Unknown	➢ The initial values of a signal is unknown ➢ A signal is simultaneously driven to two conflicting values
'0'	0	
'1'	1	
'Z'	High impedance	Tristate buffers and buses

-Signal Resolution Functions: necessary when different wires in a system are driving a common signal path; VHDL with multivalued logic can be used to create resolutions when signals are connected. **Resolved signals have an associated resolution function;** With unresolved signals, if we drive a bit signal to two different values in two concurrent statements, the compiler will flag an error;

-IEEE 9-Valued Logic System:

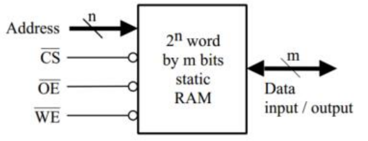
①'U'：未初始化； ②'0'：强 0； ③'1'：强 1；
④'X'：强未知； ⑤'Z'：高阻态； ⑥'W'：弱未知；
⑦'L'：弱 0； ⑧'H'：弱 1； ⑨'-'：无所谓；
大多数只限于仿真，'0', '1' and 'Z' 可综合，无限制。
区别：①'1'：as perfect as the power supply voltage；②'H'：logically high, but has a voltage drop；③'0'：perfect ground；④'L'：logically low；**If a forcing signal and a weak signal are tied together, the forcing signal dominates.**

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

std_logic is a subtype of std_ulogic
IEEE.numeric_std package uses std_logic

-SRAM Model

n address lines, **m** data lines, **3** control lines: **CS_b, OE_b, WE_b**
This memory can store **2^n** words, each **m** bits wide



$\overline{CS}$	$\overline{OE}$	$\overline{WE}$	Mode	I/O pins
H	X	X	not selected	high-Z
L	H	H	output disabled	high-Z
L	L	H	read	data out
L	X	L	write	data in

CS_b: chip select selects memory chip so that memory read and write operation are possible; OE_b: output enables memory output onto an external bus; WE_b: write enable data to be written to RAM; Writing to the latches in the memory cells is not completed until: ①**WE_b goes high** ②**CS_b goes high.**

-Generics: generics is a way of specifying a generic parameter; Generic parameter is a static parameter; It can be easily modified and adapted to different applications

`generic (n : integer := 8);` Whenever n is found in the entity itself or in the architecture that follows, its value will be assumed to be 8

-Named association: 按位置或者显式地指定

-Generate Statements:

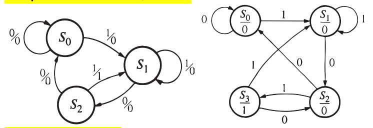
for 生成语句：编译的时候会按照规则生成一系列语句

```
generate_label: for identifier in range generate
[begin]
      concurrent statement(s)
end generate [generate_label];
```

-Conditional Generate: 条件为真的时候生成，不允许 else

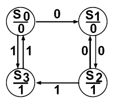
```
generate_label: if condition generate
[begin]
      concurrent statement(s)
end generate [generate_label];
```

-Sequence Detector：检测 101



-NRZ to Mancherst

Manchester code: 0=前半 0 后半 1，1=前半 1 后半 0



(b) State Graph

Present State	Next State X = 0	Next State X = 1	Present Output (Z)
S0	S1	S3	0
S1	S2	–	0
S2	S1	S3	1
S3	–	S0	1

(c) State table

-BCD to Seven-Segment Display Decoder

```
entity bcd_seven is
  port(bcd: in bit_vector(3 downto 0); seven: out bit_vector(7 downto 1));
end bcd_seven;

architecture behavioral of bcd_seven is
begin
  process(bcd)
  begin
    case bcd is
      when "0000" => seven <= "0111111";
      when "0001" => seven <= "0000110";
      when "0010" => seven <= "1011011";
      when "0011" => seven <= "1001111";
      when "0100" => seven <= "1100110";
      when "0101" => seven <= "1101101";
      when "0110" => seven <= "1111101";
      when "0111" => seven <= "0000111";
      when "1000" => seven <= "1111111";
      when "1001" => seven <= "1101111";
      when others => null;
    end case;
  end process;
end behavioral;
```

-A BCD Adder

```
library IEEE;
use IEEE.numeric_bit.all;

entity BCD_Adder is
  port(X, Y: in unsigned(7 downto 0); Z : out unsigned(11 downto 0));
end BCD_Adder;

architecture BCDadd of BCD_Adder is
  alias Xdig1: unsigned(3 downto 0) is X(7 downto 4);
  alias Xdig0: unsigned(3 downto 0) is X(3 downto 0);
  alias Ydig1: unsigned(3 downto 0) is Y(7 downto 4);
  alias Ydig0: unsigned(3 downto 0) is Y(3 downto 0);
  alias Zdig2: unsigned(3 downto 0) is Z(11 downto 8);
  alias Zdig1: unsigned(3 downto 0) is Z(7 downto 4);
  alias Zdig0: unsigned(3 downto 0) is Z(3 downto 0);
  signal S0, S1: unsigned(4 downto 0);
  signal C: bit;
  begin
    S0 <= '0'&Xdig0 + Ydig0;
    Zdig0 <= S0(3 downto 0) + 6 when S0 > 9
      else S0(3 downto 0);       -- add 6 if needed
    C <= '1' when S0 > 9 else '0';
    S1 <= '0' & Xdig1 + Ydig1 + unsigned'(0=>C);
    Zdig1 <= S1(3 downto 0) + 6 when S1 > 9
      else S1(3 downto 0);
    Zdig2 <= "0001" when S1 > 9 else "0000";
  end BCDadd;
```

-Keyboard Scanner

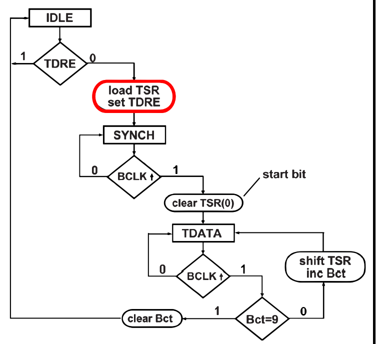
S1: wait with output C0=C1=C2=1 until a key is pressed
S2: C0=1, so if the pressed key is in column 0, K = 1, and the circuit outpus V=1 and goes to S5
S3, **S4**: If no key is found in column 0, column 1 is checked, and if necessary, column 2 is checked
S5: The circuit waits until all key are released and Kd goes to 0 before resetting

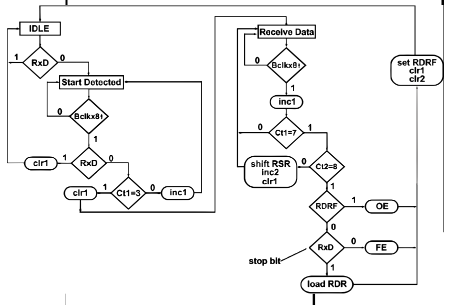
```
entity scanner is
  port(R0, R1, R2, R3, CLK: in bit;   C0, C1, C2: inout bit;
      N0, N1, N2, N3, V: out bit);
end scanner;
architecture behavior of scanner is
  signal QA, K ,Kd: bit;
  signal state, nextstate: integer range 0 to 5;
  begin
    K <= R0 or R1 or R2 or R3;
    N3 <= (R2 and not C0) or (R3 and not C1);
    N2 <= R1 or (R2 and C0);
    N1 <= (R0 and not C0) or (not R2 and C2) or (not R1 and not R0 and
C0);
    N0 <= (R1 and C1) or (not R1 and C2) or (not R3 and not R1 and not
C1);
  process(state, R0, R1, R2, R3, C0, C1, C2, K, Kd, QA)
    begin
    C0 <= '0'; C1 <= '0'; C2 <= '0'; V <= '0';
    case state is
      when 0 => nextstate <= 1;
      when 1 => C0 <= '1'; C1 <= '1'; C2 <= '1';
        if (Kd and K) = '1' then nextstate <= 2;
        else nextstate <= 1;
        end if;
      when 2 => C0 <= '1';
        if (Kd and K) = '1' then V <= '1'; nextstate <= 5;
        elsif K = '0' then nextstate <= 3;
        else nextstate <= 2;
        end if;
      when 3 => C1 <= '1';
        if (Kd and K) = '1' then V <= '1'; nextstate <= 5;
        elsif K = '0' then nextstate <= 4;
        else nextstate <= 3;
        end if;
      when 4 => C2 <= '1';
        if (Kd and K) = '1' then V <= '1'; nextstate <= 5;
        else nextstate <= 4;
        end if;
      when 5 => C0 <= '1'; C1 <= '1'; C2 <= '1';
        if Kd = '0' then nextstate <= 1;
        else nextstate <= 5;
        end if;
    end case;
  end process;
  process(CLK)
  begin
    if CLK = '1' and CLK'EVENT then
      state <= nextstate;
      QA <= K;
      Kd <= QA;
    end if;
  end process;
 end behavior;
```

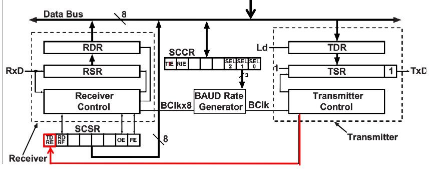
-UART

When no data is being transmitted, D remains high.

```
library IEEE;use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;-use this if unsigned type is used.
entity UART_Transmitter is port(Bclk,sysclk,rst_b,TDRE,loadTDR:in std_logic;
    DBUS:in unsigned(7 downto 0);
    setTDRE,TxD:out std_logic);
end UART_Transmitter;

architecture xmit of UART_Transmitter is
  type stateType is (IDLE,SYNCH,TDATA);
  signal state,nextstate:stateType;
  signal TSR:unsigned(8 downto 0);- Shift Register
  signal TDR:unsigned(7 downto 0);-- Data Register
  signal Bct:integer range 0 to 9;
  signal inc,clr,loadTSR,shftTSR,start:std_logic;
  signal Bclk_rising,Bclk_Dlayed:std_logic;
begin
  TxD <=TSR(0);
  setTDRE <= loadTSR;
  Bclk_rising <= Bclk and (not Bclk_Dlayed); -indicates the rising edge of bit
clock
Xmit_Control: process(state,TDRE,Bct,Bclk_rising)
  Begin
    inc <='0';clr <='0';loadTSR <='0';shftTSR <='0';start <='0'; -reset
    case state is
      when IDLE =>
        if (TDRE ='0')then
          loadTSR <='1';nextstate <=SYNCH;
        else nextstate <=IDLE;end if;
      when SYNCH => --synchronize with the bit clock
        if (Bclk_rising ='1')then
          start <='1';nextstate <=TDATA;
        else nextstate <=SYNCH;end if;
      when TDATA =>
        if (Bclk_rising '0')then nextstate <TDATA;
        elsif (Bct /=9)then
          shftTSR <='1';inc <='1';nextstate <=TDATA;
           else clr <='1';nextstate <=IDLE;end if;
    end case;end process;
  Xmit_update:process(sysclk,rst_b)
  Begin
    if (rst_b='0')then
      TSR <="111111111";
      state<=IDLE;Bct <=0;Bclk_Dlayed <='0';
    elsif (sysclk'event and sysclk='1')then
      state <=nextstate;
      if (clr ='1')then Bct <=0;
      elsif (inc='1')then Bct <=Bct+1;end if;
      if (loadTDR ='1')then TDR <=DBUS;end if;
      if (loadTSR='1')then TSR <=TDR&'1';end if;
      if (start ='1')then TSR(0)<='0';end if;
      if (shftTSR ='1')then TSR <'1'&TSR(8 downto 1);
        end if; --shift out one bit
      Bclk_Dlayed <=Bc1k;--Bclk delayed by 1 sysclk
    end if; end process; end xmit;

entity UART_Receiver is
  port(RxD,BclkX8,sysclk,rst_b,RDRF:in std_logic;
    RDR:out unsigned(7 downto 0);
    setRDRF,setOE,setFE:out std_logic);
end UART_Receiver;

architecture rcvr of UART_Receiver is
  type stateType is (IDLE,START_DETECTED,RECV_DATA);
  signal state,nextstate:stateType;
  signal RSR:unsigned(7 downto 0);-receive shift register
  signal ctl integer range 0 to 7;--indicates when to read the RxD input
  signal ct2 integer range 0 to 8;--counts number of bits read
  signal inc1,inc2,clr1,clr2,shftRSR,loadRDR:std_logic;
  signal BclkX8_Dlayed,BclkX8_rising:std_logic;
begin
  BclkX8_rising <=BclkX8 and (not BclkX8_Dlayed);-indicates the rising edge
of bitX8 clock
Rcvr_Control: process(state,RxD,RDRF,ct1,ct2,BclkX8_rising)
begin
  -reset control signals
  inc1<='0';inc2<='0';c1r1<='0';c1r2<='0';
  shftRSR <='0';loadRDR<='0';setRDRF<='0';setOE <='0';setFE <='0';
  case state is
    when IDLE =>
      if (RxD ='0')then nextstate <=START_DETECTED;
      else nextstate <=IDLE;end if;
    when START_DETECTED =>
      if (BclkX8_rising '0')
      then nextstate <=START_DETECTED;
      elsif (RxD='1')then clr1 <='1';nextstate <=IDLE;
      elsif(ct1=3)thenclr1 <='1';nextstate<=RECV_DATA;
      else incl <='1';nextstate <=START_DETECTED;
      end if;
    when RECV_DATA =>
      if (BclkX8_rising='0')then nextstate <=RECV_DATA;
      else incl <='1';
        if (ct1 /=7)then nextstate <=RECV_DATA;
          --wait for 8 clock cycles
        elsif (ct2 /=8)then
          shftRSR <'1';inc2 <'1';clr1 <='1';
          --read next data bit
          nextstate <=RECV_DATA;
        else
          nextstate <IDLE;
          setRDRF<='1';c1r1<='1';c1r2<='1';
          if (RDRF '1')then setOE <='1';
          elsif (RxD '0')then setFE <='1';
          else loadRDR <='1';end if;
        end if;end if;end case;end process;
Rcvr_update:process(sysclk,rst_b)
begin
  if (rst_b='0')then state<=IDLE;BclkX8_Dlayed <=0';
    ct1<=0;ct2<=0;
  elsif (sysclk'event and sysclk='1')then
    state<=nextstate;
    if (clr1='1')then ct1 <=0;elsif (inc1='1')then
    ct1<=ct1+1;end if;
    if (clr2='1')then ct2 <=0;elsif (inc2='1')then
    ct2<=ct2+1;end if;
    if (shftRSR='1')then RSR<=RxD&RSR(7 downto 1);
    end if;
      --update shift reg.
    if (loadRDR='1')then RDR<=RSR;end if;
    BclkX8_Dlayed<=BclkX8;--BclkX8 delayed by 1 sysclk
  end if;
end process;
end rcvr;
```







-Dice Game

```
entity DiceGame is
port (Rb, Reset, CLK: in bit;
    Sum: in integer range 2 to 12 ;
    Roll, Win, Lose: out bit) ;
end DiceGame ;

architecture DiceBehave of DiceGame is
  signal State, Nextstate: integer range 0 to 5;
  signal Point: integer range 2 to 12;
  signal Sp: bit;
begin
  process (Rb, Reset,   Sum, State)
  begin
    Sp<='0';Roll<='0';Win<='O';Lose<='0';
    case State is
      when 0 =>
        if Rb='1' then Nextstate<=1;
        end if;
      when 1 =>
        if Rb='1' then Roll<='1';
        elsif Sum = 7 or Sum = 11 then Nextstate <= 2;
        elsif Sum = 2 or Sum = 3 or Sum =12 then Nextstate <= 3;
        else Sp <= '1'; Nextstate <= 4;
        end if;
      when 2 =>Win<='1';
        if Reset = '1' then Nextstate <= 0; end if;
      when 3 =>Lose<='1';
        if Reset = '1' then Nextstate <= 0; end if;
      when 4 => if Rb = '1' then Nextstate <= 5; end if;
      when 5 =>
        if Rb='1' then Roll <= '1';
        elsif Sum = Point then Nextstate <= 2;
        elsif Sum = 7 then Nextstate <= 3;
        else Nextstate <= 4;
        end if;
     end case ;
  end process;

  process (CLK)
    begin
      if rising_ edge (CLK) then
        State <= Nextstate;
        if Sp = '1' then Point <= Sum;
        end if;
      end if;
  end process;
end DiceBehave;
```

-Divider

```
library IEEE;
use IEEE. numeric_bit.all ;
entity sdiv is
  port(CLK, St: in bit; Dbus: in unsigned(15 downto 0) ;
     Quotient: out unsigned(15 downto 0);
     V,  Rdy: out bit) ;
  end sdiv;
architecture Signdiv of sdiv is
  signal State: integer range 0 to 6;
  signal Count: unsigned(3 downto 0);
  signal Sign,  C,  Cm2: bit;
  signal Divisor, Sum,  Compout: unsigned(15 downto 0);
  signal Dividend: unsigned(31 downto 0) ;
  alias Acc: unsigned(15 downto 0) is Dividend(31
downto 16);
begin
  Cm2 <= not divisor(15);
  compout <= divisor when Cm2 = '0' else not divisor ;
  Sum <= Acc + compout + uns igned' (0=>Cm2) ;
  C <= not Sum(15) ;
  Quotient <= Dividend(15 downto 0)
  Rdy <='1' when State = 0 else '0';
  process (CLK)
  begin
    if CLK' event and CLK =' 1' then
      case State is
        when 0 =>
            if St = '1' then
              Acc <= Dbus ; -- load upper dividend
              Sign <= Dbus (15);
              State <= 1 ;
              V <= '0'; -- initialize overf1ow
              Count <=0000"; -- initialize counter
            end if;
          when1=>Dividend(15 downto 0) <=Dbus; -- load lower dividend
            State <= 2;
          when 2 =>
            Divisor <= Dbus;
            if Sign = '1' then    -- two's complement Dividend if necessary
              Dividend <= not Dividend + 1;
            end if;
   State <= 3;
            when 3 =>
              Dividend <= Dividend(30 downto 0) & '0'; -- left shift
              Count <= Count + 1;
              State <= 4;
            when_ 4 => if C = '1' then
                V <='1';
                State <= 0;
              else
                Dividend <= Dividend(30 downto 0) & '0';
                Count <= Count + 1;
                State <= 5;
              end if;
            when 5 => if c ='1' then
                Acc <= Sum;   --subtract
                Dividend(0) <=' 1 ' ;
              else
                Dividend <= Dividend(30 downto 0) & '0'; -- left shift
                if Count = 15 then State <= 6; end if;     -- KC'
                Count <= Count + 1 ;
              end if;
          when 6 => State <= 0;
            if C = '1'' then
              Acc <= Sum;   --subtract
              Dividend(0) <='1';
              State <= 6 ;
            elsif (Sign xor Divisor(15)) ='1' then
              Dividend <= not Dividend + 1;
            end if;
          end case;
        end if;
      end process ;
      end Signdiv;
```

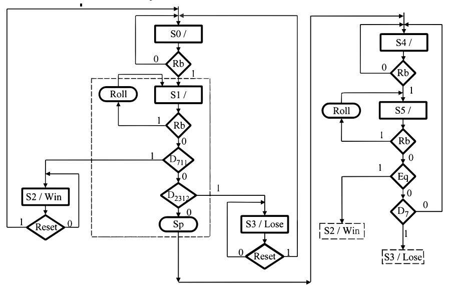
In the following VHDL process A, B, C, and D are all integers that have a value of 0 at time = 10 ns. If E changes from '0' to '1' at time = 20 ns, specify the time(s) at which each signal will change and the value to which it will change. List these changes in chronological order (20, 20+Δ, 20+2Δ, etc.)

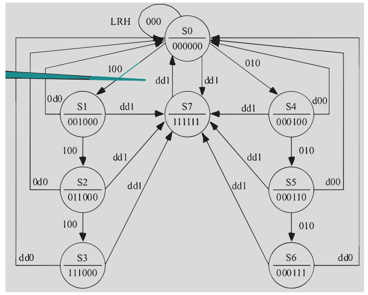
Time	A	B	C	D	E
10 ns	0	0	0	0	0
20 ns	0	0	0	0	1
20 + Δns	0	1	0	0	1
20 + 2Δns	0	8	0	0	1
23 ns	0	8	0	0	1
30 ns	0	8	1	0	1
35 ns	5	8	1	0	1

Packages	Type defined	Conversion functions	Logic operations	Arithmetic operations
std_logic_1161	std_logic std_logic_vector		√	x
numeric_bit	(un)signed (using bit_vector)	to_integer(A) to_unsigned(B,N) unsigned(A) bit_vector(B)	√	√
numeric_std	(un)signed (using std_logic_vector)		√	√
std_logic_unsigned			x	√

```
Type ROM Is array (0 to 15, 7 downto 0) of bit;
Signal ROM1 : ROM;
```

Attribute	Returns	Examples
A'LEFT(N)	left bound of Nth index range	ROM1'LEFT(1)=0 ROM1'LEFT(2) = 7
A'RIGHT(N)	right bound of Nth index range	ROM1'RIGHT(1) = 15 ROM1'RIGHT(2) = 0
A'HIGH(N)	largest bound of Nth index range	ROM1'HIGH(1) = 15 ROM1'HIGH(2) = 7
A'LOW(N)	smallest bound of Nth index range	ROM1'LOW(1) = 0 ROM1'LOW(2) = 0
A'RANGE(N)	Nth index range	ROM1'RANGE(1) = 0 to 15 ROM1'RANGE(2) = 7 downto 0
A'REVERSE_RANGE(N)	Nth index range reversed	ROM1'REVERSE_RANGE(1) = 15 downto 0 ROM1'REVERSE_RANGE(2) = 0 to 7
A'LENGTH(N)	size of Nth index range	ROM1'LENGTH(1) = 16 ROM1'LENGTH(2) = 8


S0 /
Rb
Roll
S1 /
Rb
D711
D2312
Sp
S2 / Win
Reset
S3 / Lose
Reset
S4 /
Rb
Roll
S5 /
Rb
Eq
D7
S2 / Win
S3 / Lose


LRH
000
S0 000000
S1 001000
S2 011000
S3 111000
S7 111111
S4 000100
S5 000110
S6 000111
100
010
dd1
0d0
dd0
d00