

CPE/EE 422/522 Advanced Logic Design L11

Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

- What we know
 - How to model Combinational Networks in VHDL
 - Structural, Dataflow, Behavioral
 - How to model Flip-flops in VHDL
 - Processes
 - Delays (delta, transport, inertial)
 - How to model FSM in VHDL
 - Wait statements
 - Variables, Signals, Arrays
- What we do not know
 - VHDL Operators
 - Procedures, Functions
 - Packages, Libraries
 - Additional Topics (if time)

01/07/2003

UAH-CPE/EE 422/522 ©AM

2

Review: VHDL Operators

1. Binary logical operators: **and or nand nor xor xnor**
 2. Relational: **= /= < <= > >=**
 3. Shift: **sll srl sla sra rol ror**
 4. Adding: **+ - &** (concatenation)
 5. Unary sign: **+ -**
 6. Multiplying: *** / mod rem**
 7. Miscellaneous: **not abs ****
- Class 7 has the highest precedence (applied first), followed by class 6, then class 5, etc

01/07/2003

UAH-CPE/EE 422/522 ©AM

3

Example of VHDL Operators

In the following expression, A, B, C, and D are bit_vectors:

```
(A & not B or C ror 2 and D) = "110010"
```

The operators would be applied in the order:

```
not, &, ror, or, and, =
```

If A = "110", B = "111", C = "01000", and D = "11011", the computation would proceed as follows:

```
not B = "000" (bit-by-bit complement)
```

```
A & not B = "110000" (concatenation)
```

```
C ror 2 = "000110" (rotate right 2 places)
```

```
(A & not B) or (C ror 2) = "110110" (bit-by-bit or)
```

```
(A & not B or C ror 2) and D = "110010" (bit-by-bit and)
```

```
[(A & not B or C ror 2) and D] = "110010"
```

```
[(A & not B or C ror 2) and D] = TRUE
```

```
(the parentheses force the equality test to be done last and the result is TRUE)
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

4

Example of Shift Operators

The shift operators can be applied to any bit_vector or boolean_vector. In the following examples, A is a bit_vector equal to "10010101":

```
A sll 2 is "01010100" (shift left logical, filled with '0')
```

```
A srl 3 is "00010010" (shift right logical, filled with '0')
```

```
A sla 3 is "10101111" (shift left arithmetic, filled with right bit)
```

```
A sra 2 is "11100101" (shift right arithmetic, filled with left bit)
```

```
A rol 3 is "10101100" (rotate left)
```

```
A ror 5 is "10101100" (rotate right)
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

5

VHDL Functions

- Functions execute a sequential algorithm and return a single value to calling program

```
function rotate_right (reg: bit_vector)  
return bit_vector is  
begin  
return reg ror 1;  
end rotate_right;
```

- A = "10010101"
B <= rotate_right(A);

- General form

```
function function-name (formal-parameter-list)  
return return-type is  
[declarations]  
begin  
sequential statements -- must include return return-value;  
end function-name;
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

6

For Loops

General form of a for loop:

```
[loop-label:] for loop-index in range loop
sequential statements
end loop [loop-label];
```

Exit statement has the form:

```
exit; -- or
exit when condition;
```

For Loop Example:

-- compare two 8-character strings and return TRUE if equal

```
function comp_string(string1, string2: string(1 to 8))
return boolean is
variable B: boolean;
begin
loopex: for j in 1 to 8 loop
B := string1(j) = string2(j);
exit when B=FALSE;
end loop loopex;
return B;
end comp_string;
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

7

Add Function

-- This function adds 2 4-bit vectors and a carry.
-- It returns a 5-bit sum

```
function add4 (A,B: bit_vector(3 downto 0); carry: bit)
return bit_vector is
variable cout: bit;
variable cin: bit := carry;
variable Sum: bit_vector(4 downto 0):="00000";
begin
loop1: for i in 0 to 3 loop
cout := (A(i) and B(i)) or (A(i) and cin) or (B(i) and cin);
Sum(i) := A(i) xor B(i) xor cin;
cin := cout;
end loop loop1;
Sum(4) := cout;
return Sum;
end add4;
```

Example function call:

```
Sum1 <= add4(A1, B1, cin);
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

8

VHDL Procedures

- Facilitate decomposition of VHDL code into modules
- Procedures can return any number of values using output parameters
- General form

```
procedure procedure_name (formal-parameter-list) is
[declarations]
begin
Sequential-statements
end procedure_name;
```

```
procedure_name (actual-parameter-list);
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

9

Procedure for Adding Bit_vectors

-- This procedure adds two n-bit bit_vectors and a carry and
-- returns an n-bit sum and a carry. Add1 and Add2 are assumed
-- to be of the same length and dimensioned n-1 downto 0.

```
procedure Addvec
(Add1,Add2: in bit_vector;
Cin: in bit;
signal Sum: out bit_vector;
signal Cout: out bit;
n: in positive) is
variable C: bit;
begin
C := Cin;
for i in 0 to n-1 loop
Sum(i) <= Add1(i) xor Add2(i) xor C;
C := (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
end loop;
Cout <= C;
end Addvec;
```

Example procedure call:

```
Addvec(A1, B1, Cin, Sum1, Cout, 4);
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

10

Parameters for Subprogram Calls

Mode	Class	Actual Parameter	
		Procedure Call	Function Call
in ¹	constant ²	expression	expression
	signal	signal	signal
	variable	variable	n/a
out/inout	signal	signal	n/a
	variable ³	variable	n/a

¹ default mode for functions ² default for in mode ³ default for out/inout mode

01/07/2003

UAH-CPE/EE 422/522 ©AM

11

Packages and Libraries

- Provide a convenient way of referencing frequently used functions and components
- Package declaration

```
package package-name is
package declarations
end [package][(package-name)];
```

- Package body [optional]

```
package body package-name is
package body declarations
end [package body][(package name)];
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

12

Library BITLIB – bit_pack package

```

package bit_pack is
function s2b4 (sig: std_logic_vector(3 downto 0)) return bit;
function b2s4 (sig: bit_vector(3 downto 0)) return std_logic_vector;
function s2b8 (sig: std_logic_vector(7 downto 0)) return bit_vector;
function b2s8 (sig: bit_vector(7 downto 0)) return std_logic_vector;
function s2b16 (sig: std_logic_vector(15 downto 0)) return bit_vector;
function b2s16 (sig: bit_vector(15 downto 0)) return std_logic_vector;
function s2b32 (sig: std_logic_vector(31 downto 0)) return bit_vector;
function b2s32 (sig: bit_vector(31 downto 0)) return std_logic_vector;
function s2b64 (sig: std_logic_vector(63 downto 0)) return bit_vector;
function b2s64 (sig: bit_vector(63 downto 0)) return std_logic_vector;
function s2b128 (sig: std_logic_vector(127 downto 0)) return bit_vector;
function b2s128 (sig: bit_vector(127 downto 0)) return std_logic_vector;
function s2b256 (sig: std_logic_vector(255 downto 0)) return bit_vector;
function b2s256 (sig: bit_vector(255 downto 0)) return std_logic_vector;
function s2b512 (sig: std_logic_vector(511 downto 0)) return bit_vector;
function b2s512 (sig: bit_vector(511 downto 0)) return std_logic_vector;
function s2b1024 (sig: std_logic_vector(1023 downto 0)) return bit_vector;
function b2s1024 (sig: bit_vector(1023 downto 0)) return std_logic_vector;
function s2b2048 (sig: std_logic_vector(2047 downto 0)) return bit_vector;
function b2s2048 (sig: bit_vector(2047 downto 0)) return std_logic_vector;
function s2b4096 (sig: std_logic_vector(4095 downto 0)) return bit_vector;
function b2s4096 (sig: bit_vector(4095 downto 0)) return std_logic_vector;
function s2b8192 (sig: std_logic_vector(8191 downto 0)) return bit_vector;
function b2s8192 (sig: bit_vector(8191 downto 0)) return std_logic_vector;
function s2b16384 (sig: std_logic_vector(16383 downto 0)) return bit_vector;
function b2s16384 (sig: bit_vector(16383 downto 0)) return std_logic_vector;
function s2b32768 (sig: std_logic_vector(32767 downto 0)) return bit_vector;
function b2s32768 (sig: bit_vector(32767 downto 0)) return std_logic_vector;
function s2b65536 (sig: std_logic_vector(65535 downto 0)) return bit_vector;
function b2s65536 (sig: bit_vector(65535 downto 0)) return std_logic_vector;
function s2b131072 (sig: std_logic_vector(131071 downto 0)) return bit_vector;
function b2s131072 (sig: bit_vector(131071 downto 0)) return std_logic_vector;
function s2b262144 (sig: std_logic_vector(262143 downto 0)) return bit_vector;
function b2s262144 (sig: bit_vector(262143 downto 0)) return std_logic_vector;
function s2b524288 (sig: std_logic_vector(524287 downto 0)) return bit_vector;
function b2s524288 (sig: bit_vector(524287 downto 0)) return std_logic_vector;
function s2b1048576 (sig: std_logic_vector(1048575 downto 0)) return bit_vector;
function b2s1048576 (sig: bit_vector(1048575 downto 0)) return std_logic_vector;
function s2b2097152 (sig: std_logic_vector(2097151 downto 0)) return bit_vector;
function b2s2097152 (sig: bit_vector(2097151 downto 0)) return std_logic_vector;
function s2b4194304 (sig: std_logic_vector(4194303 downto 0)) return bit_vector;
function b2s4194304 (sig: bit_vector(4194303 downto 0)) return std_logic_vector;
function s2b8388608 (sig: std_logic_vector(8388607 downto 0)) return bit_vector;
function b2s8388608 (sig: bit_vector(8388607 downto 0)) return std_logic_vector;
function s2b16777216 (sig: std_logic_vector(16777215 downto 0)) return bit_vector;
function b2s16777216 (sig: bit_vector(16777215 downto 0)) return std_logic_vector;
function s2b33554432 (sig: std_logic_vector(33554431 downto 0)) return bit_vector;
function b2s33554432 (sig: bit_vector(33554431 downto 0)) return std_logic_vector;
function s2b67108864 (sig: std_logic_vector(67108863 downto 0)) return bit_vector;
function b2s67108864 (sig: bit_vector(67108863 downto 0)) return std_logic_vector;
function s2b134217728 (sig: std_logic_vector(134217727 downto 0)) return bit_vector;
function b2s134217728 (sig: bit_vector(134217727 downto 0)) return std_logic_vector;
function s2b268435456 (sig: std_logic_vector(268435455 downto 0)) return bit_vector;
function b2s268435456 (sig: bit_vector(268435455 downto 0)) return std_logic_vector;
function s2b536870912 (sig: std_logic_vector(536870911 downto 0)) return bit_vector;
function b2s536870912 (sig: bit_vector(536870911 downto 0)) return std_logic_vector;
function s2b1073741824 (sig: std_logic_vector(1073741823 downto 0)) return bit_vector;
function b2s1073741824 (sig: bit_vector(1073741823 downto 0)) return std_logic_vector;
function s2b2147483648 (sig: std_logic_vector(2147483647 downto 0)) return bit_vector;
function b2s2147483648 (sig: bit_vector(2147483647 downto 0)) return std_logic_vector;
function s2b4294967296 (sig: std_logic_vector(4294967295 downto 0)) return bit_vector;
function b2s4294967296 (sig: bit_vector(4294967295 downto 0)) return std_logic_vector;
function s2b8589934592 (sig: std_logic_vector(8589934591 downto 0)) return bit_vector;
function b2s8589934592 (sig: bit_vector(8589934591 downto 0)) return std_logic_vector;
function s2b17179869184 (sig: std_logic_vector(17179869183 downto 0)) return bit_vector;
function b2s17179869184 (sig: bit_vector(17179869183 downto 0)) return std_logic_vector;
function s2b34359738368 (sig: std_logic_vector(34359738367 downto 0)) return bit_vector;
function b2s34359738368 (sig: bit_vector(34359738367 downto 0)) return std_logic_vector;
function s2b68719476736 (sig: std_logic_vector(68719476735 downto 0)) return bit_vector;
function b2s68719476736 (sig: bit_vector(68719476735 downto 0)) return std_logic_vector;
function s2b137438953472 (sig: std_logic_vector(137438953471 downto 0)) return bit_vector;
function b2s137438953472 (sig: bit_vector(137438953471 downto 0)) return std_logic_vector;
function s2b274877907520 (sig: std_logic_vector(274877907519 downto 0)) return bit_vector;
function b2s274877907520 (sig: bit_vector(274877907519 downto 0)) return std_logic_vector;
function s2b549755815040 (sig: std_logic_vector(549755815039 downto 0)) return bit_vector;
function b2s549755815040 (sig: bit_vector(549755815039 downto 0)) return std_logic_vector;
function s2b1099511630080 (sig: std_logic_vector(1099511630079 downto 0)) return bit_vector;
function b2s1099511630080 (sig: bit_vector(1099511630079 downto 0)) return std_logic_vector;
function s2b2199023260160 (sig: std_logic_vector(2199023260159 downto 0)) return bit_vector;
function b2s2199023260160 (sig: bit_vector(2199023260159 downto 0)) return std_logic_vector;
function s2b4398046520320 (sig: std_logic_vector(4398046520319 downto 0)) return bit_vector;
function b2s4398046520320 (sig: bit_vector(4398046520319 downto 0)) return std_logic_vector;
function s2b8796093040640 (sig: std_logic_vector(8796093040639 downto 0)) return bit_vector;
function b2s8796093040640 (sig: bit_vector(8796093040639 downto 0)) return std_logic_vector;
function s2b17592186081280 (sig: std_logic_vector(17592186081279 downto 0)) return bit_vector;
function b2s17592186081280 (sig: bit_vector(17592186081279 downto 0)) return std_logic_vector;
function s2b35184372162560 (sig: std_logic_vector(35184372162559 downto 0)) return bit_vector;
function b2s35184372162560 (sig: bit_vector(35184372162559 downto 0)) return std_logic_vector;
function s2b70368744325120 (sig: std_logic_vector(70368744325119 downto 0)) return bit_vector;
function b2s70368744325120 (sig: bit_vector(70368744325119 downto 0)) return std_logic_vector;
function s2b140737488650240 (sig: std_logic_vector(140737488650239 downto 0)) return bit_vector;
function b2s140737488650240 (sig: bit_vector(140737488650239 downto 0)) return std_logic_vector;
function s2b281474977300480 (sig: std_logic_vector(281474977300479 downto 0)) return bit_vector;
function b2s281474977300480 (sig: bit_vector(281474977300479 downto 0)) return std_logic_vector;
function s2b562949954600960 (sig: std_logic_vector(562949954600959 downto 0)) return bit_vector;
function b2s562949954600960 (sig: bit_vector(562949954600959 downto 0)) return std_logic_vector;
function s2b1125899909201920 (sig: std_logic_vector(1125899909201919 downto 0)) return bit_vector;
function b2s1125899909201920 (sig: bit_vector(1125899909201919 downto 0)) return std_logic_vector;
function s2b2251799818403840 (sig: std_logic_vector(2251799818403839 downto 0)) return bit_vector;
function b2s2251799818403840 (sig: bit_vector(2251799818403839 downto 0)) return std_logic_vector;
function s2b4503599636807680 (sig: std_logic_vector(4503599636807679 downto 0)) return bit_vector;
function b2s4503599636807680 (sig: bit_vector(4503599636807679 downto 0)) return std_logic_vector;
function s2b9007199273615360 (sig: std_logic_vector(9007199273615359 downto 0)) return bit_vector;
function b2s9007199273615360 (sig: bit_vector(9007199273615359 downto 0)) return std_logic_vector;
function s2b18014398547230720 (sig: std_logic_vector(18014398547230719 downto 0)) return bit_vector;
function b2s18014398547230720 (sig: bit_vector(18014398547230719 downto 0)) return std_logic_vector;
function s2b36028797094461440 (sig: std_logic_vector(36028797094461439 downto 0)) return bit_vector;
function b2s36028797094461440 (sig: bit_vector(36028797094461439 downto 0)) return std_logic_vector;
function s2b72057594188922880 (sig: std_logic_vector(72057594188922879 downto 0)) return bit_vector;
function b2s72057594188922880 (sig: bit_vector(72057594188922879 downto 0)) return std_logic_vector;
function s2b144115188377845760 (sig: std_logic_vector(144115188377845759 downto 0)) return bit_vector;
function b2s144115188377845760 (sig: bit_vector(144115188377845759 downto 0)) return std_logic_vector;
function s2b288230376755691520 (sig: std_logic_vector(288230376755691519 downto 0)) return bit_vector;
function b2s288230376755691520 (sig: bit_vector(288230376755691519 downto 0)) return std_logic_vector;
function s2b576460753511383040 (sig: std_logic_vector(576460753511383039 downto 0)) return bit_vector;
function b2s576460753511383040 (sig: bit_vector(576460753511383039 downto 0)) return std_logic_vector;
function s2b1152921507022766080 (sig: std_logic_vector(1152921507022766079 downto 0)) return bit_vector;
function b2s1152921507022766080 (sig: bit_vector(1152921507022766079 downto 0)) return std_logic_vector;
function s2b2305843014045532160 (sig: std_logic_vector(2305843014045532159 downto 0)) return bit_vector;
function b2s2305843014045532160 (sig: bit_vector(2305843014045532159 downto 0)) return std_logic_vector;
function s2b4611686028091064320 (sig: std_logic_vector(4611686028091064319 downto 0)) return bit_vector;
function b2s4611686028091064320 (sig: bit_vector(4611686028091064319 downto 0)) return std_logic_vector;
function s2b9223372056182128640 (sig: std_logic_vector(9223372056182128639 downto 0)) return bit_vector;
function b2s9223372056182128640 (sig: bit_vector(9223372056182128639 downto 0)) return std_logic_vector;
function s2b18446744112364257280 (sig: std_logic_vector(18446744112364257279 downto 0)) return bit_vector;
function b2s18446744112364257280 (sig: bit_vector(18446744112364257279 downto 0)) return std_logic_vector;
function s2b36893488227328514560 (sig: std_logic_vector(36893488227328514559 downto 0)) return bit_vector;
function b2s36893488227328514560 (sig: bit_vector(36893488227328514559 downto 0)) return std_logic_vector;
function s2b73786976454657029120 (sig: std_logic_vector(73786976454657029119 downto 0)) return bit_vector;
function b2s73786976454657029120 (sig: bit_vector(73786976454657029119 downto 0)) return std_logic_vector;
function s2b147573952909314058240 (sig: std_logic_vector(147573952909314058239 downto 0)) return bit_vector;
function b2s147573952909314058240 (sig: bit_vector(147573952909314058239 downto 0)) return std_logic_vector;
function s2b2951479058186281164480 (sig: std_logic_vector(2951479058186281164479 downto 0)) return bit_vector;
function b2s2951479058186281164480 (sig: bit_vector(2951479058186281164479 downto 0)) return std_logic_vector;
function s2b5902958116372562328960 (sig: std_logic_vector(5902958116372562328959 downto 0)) return bit_vector;
function b2s5902958116372562328960 (sig: bit_vector(5902958116372562328959 downto 0)) return std_logic_vector;
function s2b11805916232745124657920 (sig: std_logic_vector(11805916232745124657919 downto 0)) return bit_vector;
function b2s11805916232745124657920 (sig: bit_vector(11805916232745124657919 downto 0)) return std_logic_vector;
function s2b23611832465490253315840 (sig: std_logic_vector(23611832465490253315839 downto 0)) return bit_vector;
function b2s23611832465490253315840 (sig: bit_vector(23611832465490253315839 downto 0)) return std_logic_vector;
function s2b47223664930980506631680 (sig: std_logic_vector(47223664930980506631679 downto 0)) return bit_vector;
function b2s47223664930980506631680 (sig: bit_vector(47223664930980506631679 downto 0)) return std_logic_vector;
function s2b94447329861961013263360 (sig: std_logic_vector(94447329861961013263359 downto 0)) return bit_vector;
function b2s94447329861961013263360 (sig: bit_vector(94447329861961013263359 downto 0)) return std_logic_vector;
function s2b188894659723922026526720 (sig: std_logic_vector(188894659723922026526719 downto 0)) return bit_vector;
function b2s188894659723922026526720 (sig: bit_vector(188894659723922026526719 downto 0)) return std_logic_vector;
function s2b377789319447844053053440 (sig: std_logic_vector(377789319447844053053439 downto 0)) return bit_vector;
function b2s377789319447844053053440 (sig: bit_vector(377789319447844053053439 downto 0)) return std_logic_vector;
function s2b755578638895688106106880 (sig: std_logic_vector(755578638895688106106879 downto 0)) return bit_vector;
function b2s755578638895688106106880 (sig: bit_vector(755578638895688106106879 downto 0)) return std_logic_vector;
function s2b1511157277791376212213760 (sig: std_logic_vector(1511157277791376212213759 downto 0)) return bit_vector;
function b2s1511157277791376212213760 (sig: bit_vector(1511157277791376212213759 downto 0)) return std_logic_vector;
function s2b3022314555582752424427520 (sig: std_logic_vector(3022314555582752424427519 downto 0)) return bit_vector;
function b2s3022314555582752424427520 (sig: bit_vector(3022314555582752424427519 downto 0)) return std_logic_vector;
function s2b6044629111165404848855040 (sig: std_logic_vector(6044629111165404848855039 downto 0)) return bit_vector;
function b2s6044629111165404848855040 (sig: bit_vector(6044629111165404848855039 downto 0)) return std_logic_vector;
function s2b12089258222330808697710080 (sig: std_logic_vector(12089258222330808697710079 downto 0)) return bit_vector;
function b2s12089258222330808697710080 (sig: bit_vector(12089258222330808697710079 downto 0)) return std_logic_vector;
function s2b24178516444661637394420480 (sig: std_logic_vector(24178516444661637394420479 downto 0)) return bit_vector;
function b2s24178516444661637394420480 (sig: bit_vector(24178516444661637394420479 downto 0)) return std_logic_vector;
function s2b4835703288932327478884960 (sig: std_logic_vector(4835703288932327478884959 downto 0)) return bit_vector;
function b2s4835703288932327478884960 (sig: bit_vector(4835703288932327478884959 downto 0)) return std_logic_vector;
function s2b9671406577864654957773920 (sig: std_logic_vector(9671406577864654957773919 downto 0)) return bit_vector;
function b2s9671406577864654957773920 (sig: bit_vector(9671406577864654957773919 downto 0)) return std_logic_vector;
function s2b19342813155729309155547840 (sig: std_logic_vector(19342813155729309155547839 downto 0)) return bit_vector;
function b2s19342813155729309155547840 (sig: bit_vector(19342813155729309155547839 downto 0)) return std_logic_vector;
function s2b3868562631145860183113760 (sig: std_logic_vector(3868562631145860183113759 downto 0)) return bit_vector;
function b2s3868562631145860183113760 (sig: bit_vector(3868562631145860183113759 downto 0)) return std_logic_vector;
function s2b7737125262291720366227520 (sig: std_logic_vector(7737125262291720366227519 downto 0)) return bit_vector;
function b2s7737125262291720366227520 (sig: bit_vector(7737125262291720366227519 downto 0)) return std_logic_vector;
function s2b1547425052458344073245040 (sig: std_logic_vector(1547425052458344073245039 downto 0)) return bit_vector;
function b2s1547425052458344073245040 (sig: bit_vector(1547425052458344073245039 downto 0)) return std_logic_vector;
function s2b30948501049166881464890880 (sig: std_logic_vector(30948501049166881464890879 downto 0)) return bit_vector;
function b2s30948501049166881464890880 (sig: bit_vector(30948501049166881464890879 downto 0)) return std_logic_vector;
function s2b6189700209833376292979360 (sig: std_logic_vector(6189700209833376292979359 downto 0)) return bit_vector;
function b2s6189700209833376292979360 (sig: bit_vector(6189700209833376292979359 downto 0)) return std_logic_vector;
function s2b12379400419666751785958720 (sig: std_logic_vector(12379400419666751785958719 downto 0)) return bit_vector;
function b2s12379400419666751785958720 (sig: bit_vector(12379400419666751785958719 downto 0)) return std_logic_vector;
function s2b24758800839333503711917440 (sig: std_logic_vector(24758800839333503711917439 downto 0)) return bit_vector;
function b2s24758800839333503711917440 (sig: bit_vector(24758800839333503711917439 downto 0)) return std_logic_vector;
function s2b4951760167866700742383680 (sig: std_logic_vector(4951760167866700742383679 downto 0)) return bit_vector;
function b2s4951760167866700742383680 (sig: bit_vector(4951760167866700742383679 downto 0)) return std_logic_vector;
function s2b9903520335733401484767360 (sig: std_logic_vector(9903520335733401484767359 downto 0)) return bit_vector;
function b2s9903520335733401484767360 (sig: bit_vector(9903520335733401484767359 downto 0)) return std_logic_vector;
function s2b1980704067146680296954720 (sig: std_logic_vector(1980704067146680296954719 downto 0)) return bit_vector;
function b2s1980704067146680296954720 (sig: bit_vector(1980704067146680296954719 downto 0)) return std_logic_vector;
function s2b3961408134293360593909440 (sig: std_logic_vector(3961408134293360593909439 downto 0)) return bit_vector;
function b2s3961408134293360593909440 (sig: bit_vector(3961408134293360593909439 downto 0)) return std_logic_vector;
function s2b7922816268586721187818880 (sig: std_logic_vector(7922816268586721187818879 downto 0)) return bit_vector;
function b2s7922816268586721187818880 (sig: bit_vector(7922816268586721187818879 downto 0)) return std_logic_vector;
function s2b1584563253717344375663760 (sig: std_logic_vector(1584563253717344375663759 downto 0)) return bit_vector;
function b2s1584563253717344375663760 (sig: bit_vector(1584563253717344375663759 downto 0)) return std_logic_vector;
function s2b3169126507434688751327520 (sig: std_logic_vector(3169126507434688751327519 downto 0)) return bit_vector;
function b2s3169126507434688751327520 (sig: bit_vector(3169126507434688751327519 downto 0)) return std_logic_vector;
function s2b6338253014869377502655040 (sig: std_logic_vector(6338253014869377502655039 downto 0)) return bit_vector;
function b2s6338253014869377502655040 (sig: bit_vector(6338253014869377502655039 downto 0)) return std_logic_vector;
function s2b12676506029738755005310080 (sig: std_logic_vector(12676506029738755005310079 downto 0)) return bit_vector;
function b2s12676506029738755005310080 (sig: bit_vector(12676506029738755005310079 downto 0)) return std_logic_vector;
function s2b25353012059477510010620160 (sig: std_logic_vector(25353012059477510010620159 downto 0)) return bit_vector;
function b2s25353012059477510010620160 (sig: bit_vector(25353012059477510010620159 downto 0)) return std_logic_vector;
function s2b50706024118955020021240320 (sig: std_logic_vector(50706024118955020021240319 downto 0)) return bit_vector;
function b2s50706024118955020021240320 (sig: bit_vector(50706024118955020021240319 downto 0)) return std_logic_vector;
function s2b101412048237910040042480640 (sig: std_logic_vector(101412048237910040042480639 downto 0)) return bit_vector;
function b2s101412048237910040042480640 (sig: bit_vector(101412048237910040042480639 downto 0)) return std_logic_vector;
function s2b202824096475820080084961280 (sig: std_logic_vector(202824096475820080084961279 downto 0)) return bit_vector;
function b2s202824096475820080084961280 (sig: bit_vector(202824096475820080084961279 downto 0)) return std_logic_vector;
function s2b405648192951640160169922560 (sig: std_logic_vector(405648192951640160169922559 downto 0)) return bit_vector;
function b2s405648192951640160169922560 (sig: bit_vector(405648192951640160169922559 downto 0)) return std_logic_vector;
function s2b811296385903280320339845120 (sig: std_logic_vector(811296385903280320339845119 downto 0)) return bit_vector;
function b2s811296385903280320339845120 (sig: bit_vector(811296385903280320339845119 downto 0)) return std_logic_vector;
function s2b1622592771806560640679690240 (sig: std_logic_vector(1622592771806560640679690239 downto 0)) return bit_vector;
function b2s1622592771806560640679690240 (sig: bit_vector(1622592771806560640679690239 downto 0)) return std_logic_vector;
function s2b3245185543613121281359380480 (sig: std_logic_vector(3245185543613121281359380479 downto 0)) return bit_vector;
function b2s3245185543613121281359380480 (sig: bit_vector(3245185543613121281359380479 downto 0)) return std_logic_vector;
function s2b649037108722624456271876160 (sig: std_logic_vector(649037108722624456271876159 downto 0)) return bit_vector;
function b2s649037108722624456271876160 (sig: bit_vector(649037108722624456271876159 downto 0)) return std_logic_vector;
function s2b129807421445248911245435322240 (sig: std_logic_vector(129807421445248911245435322239 downto 0)) return bit_vector;
function b2s129807421445248911245435322240 (sig: bit_vector(129807421445248911245435322239 downto 0)) return std_logic_vector;
function s2b25961484289049782490886444480 (sig: std_logic_vector(25961484289049782490886444479 downto 0)) return bit_vector;
function b2s25961484289049782490886444480 (sig: bit_vector(25961484289049782490886444479 downto 0)) return std_logic_vector;
function s2b51922968578099564981772888960 (sig: std_logic_vector(51922968578099564981772888959 downto 0)) return bit_vector;
function b2s51922968578099564981772888960 (sig: bit_vector(51922968578099564981772888959 downto 0)) return std_logic_vector;
function s2b103845937157199129963557777920 (sig: std_logic_vector(103845937157199129963557777919 downto 0)) return bit_vector;
function b2s103845937157199129963557777920 (sig: bit_vector(103845937157199129963557777919 downto 0)) return std_logic_vector;
function s2b20769187431
```

Cascaded Counters (cont'd)

```

library IEEE;
use IEEE.std_logic_arith;

entity c74163test is
    port (Clk: in bit;
          Din1, Din2: in bit_vector(3 downto 0);
          Qout1, Qout2: inout bit_vector(3 downto 0);
          Carry2: out bit);
end c74163test;

architecture test of c74163test is
    component c74163
    port (LDN, Clk, P, T, DC: in bit; D: in bit_vector(3 downto 0);
          Cout: out bit; Q: inout bit_vector(3 downto 0));
    end component;
    signal Carry1: bit;
    signal Count: integer;
    signal temp: bit_vector(3 downto 0);
begin
    c1: c74163 port map (LDN, Clk, P, T, DC, Din1, Carry1, Qout1);
    c2: c74163 port map (LDN, Clk, P, Carry1, Clk, Din2, Carry2, Qout2);
    Count <= acc2Hz(temp);
end test;
    
```

01/07/2003

UAH-CPE/EE 422/522@AM

19

Additional Topics in VHDL

- Attributes
- Transport and Inertial Delays
- Operator Overloading
- Multivalued Logic and Signal Resolution
- IEEE 1164 Standard Logic
- Generics
- Generate Statements
- Synthesis of VHDL Code
- Synthesis Examples
- Files and Text IO

01/07/2003

UAH-CPE/EE 422/522@AM

20

Signal Attributes

Attributes associated with signals that return a value

Attribute	Returns
S'EVENT	True if an event occurred during the current delta, else false
S'ACTIVE	True if a transaction occurred during the current delta, else false
S'LAST_EVENT	Time elapsed since the previous event on S
S'LAST_VALUE	Value of S before the previous event on S
S'LAST_ACTIVE	Time elapsed since previous transaction on S

A'event – true if a change in S has just occurred

A'active – true if A has just been reevaluated, even if A does not change

01/07/2003

UAH-CPE/EE 422/522@AM

21

Signal Attributes (cont'd)

- Event
 - occurs on a signal every time it is changed
- Transaction
 - occurs on a signal every time it is evaluated
- Example:

A <= B -- B changes at time T

	Event	Event
T		
T + dt		

01/07/2003

UAH-CPE/EE 422/522@AM

22

Signal Attributes (cont'd)

```

entity test is
begin
    if (A'event) then Aev := '1';
    else Aev := '0';
    end if;
    if (A'active) then Aac := '1';
    else Aac := '0';
    end if;
    if (B'event) then Bev := '1';
    else Bev := '0';
    end if;
    if (B'active) then Bac := '1';
    else Bac := '0';
    end if;
    if (C'event) then Cev := '1';
    else Cev := '0';
    end if;
    if (C'active) then Cac := '1';
    else Cac := '0';
    end if;
end process;
end test;
    
```

01/07/2003

UAH-CPE/EE 422/522@AM

23

Signal Attributes (cont'd)

```

ns
/test/a /test/line_15/bev
delta /test/b /test/line_15/bac
/test/c /test/line_15/cev
/test/line_15/aev /test/line_15/cac
/test/line_15/aac
0 +0 0 0 0 0 0 0 0 0
0 +1 0 1 0 0 0 1 1 0 1
20 +0 1 1 0 1 1 0 0 0 0
20 +1 1 1 1 0 0 0 0 1 1
40 +0 0 1 1 1 1 0 0 0 0
40 +1 0 1 0 0 0 0 0 1 1
    
```

01/07/2003

UAH-CPE/EE 422/522@AM

24

Signal Attributes (cont'd)

Attributes that create a signal

Attribute	Creates
S'DELAYED [(time)]*	signal same as S delayed by specified time
S'STABLE [(time)]*	Boolean signal that is true if S had no events for the specified time
S'QUIET [(time)]*	Boolean signal that is true if S had no transactions for the specified time
S'TRANSACTION	signal of type BIT that changes for every transaction on S

* Delta is used if no time is specified.

01/07/2003

UAH-CPE/EE 422/522 ©AM

25

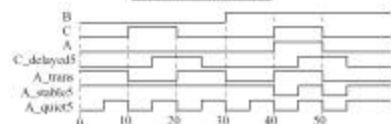
Examples of Signal Attributes

VHDL Code for Attribute Test

```
entity attr_ex is
  port (B,C : in bit);
end attr_ex;

architecture test of attr_ex is
  signal A, C_delayed5, A_trans : bit;
  signal A_stable5, A_quiet5 : boolean;
begin
  A <= B and C;
  C_delayed5 <= C_delayed(5 ns);
  A_trans <= A'transaction;
  A_stable5 <= A'stable(5 ns);
  A_quiet5 <= A'quiet(5 ns);
end test;
```

Waveforms for Attribute Test



01/07/2003

UAH-CPE/EE 422/522 ©AM

26

Using Attributes for Error Checking

```
check: process
begin
  wait until rising_edge(Clk);
  assert (D'stable(setup_time))
  report ("Setup time violation")
  severity error;
  wait for hold_time;
  assert (D'stable(hold_time))
  report ("Hold time violation")
  severity error;
end process check;
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

27

Array Attributes

Type ROM is array [0 to 15, 7 downto 0] of bit;
Signal ROM1 : ROM;

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

A can be either an array name or an array type.

Array attributes work with signals, variables, and constants.

01/07/2003

UAH-CPE/EE 422/522 ©AM

28

Recap: Adding Vectors

-- This procedure adds two n-bit bit_vectors and a carry and returns an n-bit sum and a carry. Add1 and Add2 are assumed -- to be of the same length and dimensioned n-1 downto 0.

```
procedure Addvec
  (Add1,Add2 : in bit_vector;
   Cin : in bit;
   signal Sum : out bit_vector;
   signal Cout : out bit;
   n : in positive) is
  variable C : bit;
begin
  C := Cin;
  for i in 0 to n-1 loop
    Sum(i) <= Add1(i) xor Add2(i) xor C;
    C := (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
  end loop;
  Cout <= C;
end Addvec;
```

Note: Add1 and Add2 vectors must be dimensioned as N-1 downto 0.

Use attributes to write more general procedure that places no restrictions on the range of vectors other than the lengths n must be same.

01/07/2003

UAH-CPE/EE 422/522 ©AM

29

Procedure for Adding Bit Vectors

-- This procedure adds two bit_vectors and a carry and returns a sum and a carry. Both bit_vectors should be of the same length.

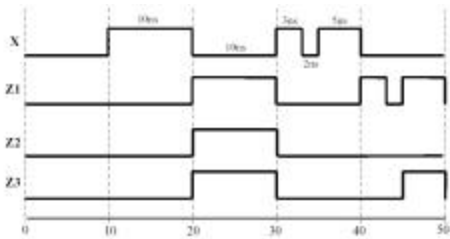
```
procedure Addvec2
  (Add1,Add2 : in bit_vector;
   Cin : in bit;
   signal Sum : out bit_vector;
   signal Cout : out bit) is
  variable C : bit := Cin;
  alias n1 : bit_vector(Add1'length-1 downto 0) is Add1;
  alias n2 : bit_vector(Add2'length-1 downto 0) is Add2;
  alias S : bit_vector(Sum'length-1 downto 0) is Sum;
begin
  assert ((n1'length = n2'length) and (n1'length = S'length))
  report "Vector lengths must be equal"
  severity error;
  for i in S'reverse_range loop
    S(i) <= n1(i) xor n2(i) xor C;
    C := (n1(i) and n2(i)) or (n1(i) and C) or (n2(i) and C);
  end loop;
  Cout <= C;
end Addvec2;
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

30

Transport and Inertial Delay



Z1 <= transport X after 10 ns; -- transport delay
 Z2 <= X after 10 ns; -- inertial delay
 Z3 <= reject 4 ns X after 10 ns; -- delay with specified rejection pulse width

01/07/2003

UAH-CPE/EE 422/522 ©AM

31

Transport and Inertial Delay (cont'd)

Z3 <= reject 4 ns X after 10 ns;

Reject is equivalent to a combination of inertial and transport delay:

Zm <= X after 4 ns;

Z3 <= transport Zm after 6 ns;

Statements executed at time T

- B at T+1, C at T+2

A <= transport B after 1 ns;

A <= transport C after 2 ns;

Statements executed at time T

- C at T+2:

Statements executed at time T

- C at T+1:

A <= B after 1 ns;

A <= C after 2 ns;

A <= transport B after 2 ns;

A <= transport C after 1 ns;

01/07/2003

UAH-CPE/EE 422/522 ©AM

32

Operator Overloading

- Operators +, - operate on integers
- Write procedures for bit vector addition/subtraction
 - addvec, subvec
- Operator overloading allows using + operator to implicitly call an appropriate addition function
- How does it work?
 - When compiler encounters a function declaration in which the function name is an operator enclosed in double quotes, the compiler treats the function as an operator overloading ("+")
 - when a "+" operator is encountered, the compiler automatically checks the types of operands and calls appropriate functions

01/07/2003

UAH-CPE/EE 422/522 ©AM

33

VHDL Package with Overloaded Operators

```
-- This package provides two overloaded functions for the plus operator
package bit_overload is
function "+" (A00: integer, A01: bit_vector) return bit_vector;
function "+" (A00: bit_vector, A01: integer) return bit_vector;
end bit_overload;

library STD;
use STD.StdLogicPackage;
package body bit_overload is
-- This function returns a bit_vector sum of two bit_vector operands.
-- The add is performed bit by bit with an internal carry.
function "+" (A00: integer, A01: bit_vector) return bit_vector is
variable sum: bit_vector(A01'length-1 downto 0); -- no carry in
variable c: bit := '0';
alias n1: bit_vector(A00'length-1 downto 0) is A00;
alias n2: bit_vector(A01'length-1 downto 0) is A01;
begin
for i in sum'range loop
sum(i) := n1(i) xor n2(i) xor c;
c := (n1(i) and n2(i) or (n1(i) and c) or (n2(i) and c);
end loop;
return sum;
end "+";
-- This function returns a bit_vector sum of a bit_vector and an integer
-- using the previous function after the integer is converted.
function "+" (A00: bit_vector, A01: integer) return bit_vector is
begin
return (A00 + int2vec(A01, A01'length));
end "+";
end bit_overload;
```

01/07/2003

UAH-CPE/EE 422/522 ©AM

34

Overloaded Operators

- A, B, C – bit vectors
- A <= B + C + 3 ?
- A <= 3 + B + C ?
- Overloading can also be applied to procedures and functions
 - procedures have the same name – type of the actual parameters in the procedure call determines which version of the procedure is called

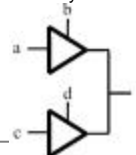
01/07/2003

UAH-CPE/EE 422/522 ©AM

35

Multivalued Logic

- Bit (0, 1)
- Tristate buffers and buses => high impedance state 'Z'
- Unknown state 'X'
 - e. g., a gate is driven by 'Z', output is unknown
 - a signal is simultaneously driven by '0' and '1'



01/07/2003

UAH-CPE/EE 422/522 ©AM

36

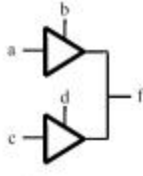
Tristate Buffers

```

use WORK_fourpack.all;
entity t_buf_comp is
    port (A,B,C,D : in X01Z; -- signals are
          F : out X01Z); -- 4 values
end t_buf_comp;

architecture t_buf_comp of t_buf_comp is
begin
    F <= a when b = '1' else Z';
    F <= c when d = '1' else Z';
end t_buf_comp;

architecture t_buf_drv of t_buf_comp is
begin
    buf1: process (A,B)
    begin
        if (b='1') then f<=a;
        else
            f<=Z'; --"drive" the output high Z when not enabled
        end if;
    end process buf1;
    buf2: process (C,D)
    begin
        if (d='1') then f<=c;
        else
            f<=Z'; --"drive" the output high Z when not enabled
        end if;
    end process buf2;
end t_buf_drv;
    
```



Resolution function to determine the actual value of f since it is driven from two different sources

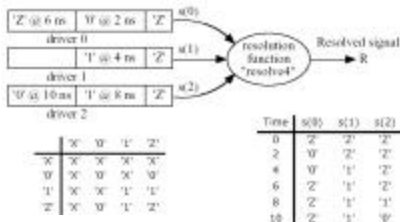
Signal Resolution

- VHDL signals may either be resolved or unresolved
- Resolved signals have an associated resolution function
- Bit type is unresolved –
 - there is no resolution function
 - if you drive a bit signal to two different values in two concurrent statements, the compiler will generate an error

Signal Resolution (cont'd)

```

signal R : X01Z := 'Z'; ...
R <= transport '0' after 2 ns, 'Z' after 6 ns;
R <= transport '1' after 4 ns;
R <= transport '1' after 8 ns, '0' after 10 ns;
    
```



Resolution Function for X01Z

```

package fourpack is
    type t_x01z is ('X','0','1','Z'); -- t_x01z is unresolved
    type t_x01z_vector is array (natural range <->) of t_x01z;
    function resolve4 (t_x01z_vector) return t_x01z;
    subtype t_x01z is resolved t_x01z;
    -- t_x01z is a resolved subtype which uses the resolution function resolve4
    type t_x01z_vector is array (natural range <->) of t_x01z;
end fourpack;

package body fourpack is
    type t_x01z_table is array (t_x01z,t_x01z) of t_x01z;
    constant resolution_table : t_x01z_table := (
        ('X','X','X'),
        ('X','0','0'),
        ('X','1','1'),
        ('X','Z','Z'));
    function resolve4 (t_x01z_vector) return t_x01z is
        variable result : t_x01z := 'Z';
    begin
        if (t_x01z_vector'length = 1) then
            return (t_x01z_vector)(1);
        else
            for i in t_x01z_vector'range loop
                result := resolution_table(result,t_x01z_vector(i));
            end loop;
        end if;
        return result;
    end resolve4;
end fourpack;
    
```

Define AND and OR for 4-valued inputs?

AND and OR Functions Using X01Z

AND	'X'	'0'	'1'	'Z'
'X'	'X'	'0'	'1'	'X'
'0'	'0'	'0'	'0'	'0'
'1'	'1'	'0'	'1'	'X'
'Z'	'Z'	'Z'	'Z'	'Z'

OR	'X'	'0'	'1'	'Z'
'X'	'X'	'X'	'1'	'X'
'0'	'0'	'0'	'1'	'X'
'1'	'1'	'1'	'1'	'1'
'Z'	'Z'	'Z'	'Z'	'Z'

IEEE 1164 Standard Logic

- 9-valued logic system
 - 'U' – Uninitialized
 - 'X' – Forcing Unknown
 - '0' – Forcing 0
 - '1' – Forcing 1
 - 'Z' – High impedance
 - 'W' – Weak unknown
 - 'L' – Weak 0
 - 'H' – Weak 1
 - '-' – Don't care
- If forcing and weak signal are tied together, the forcing signal dominates.
- Useful in modeling the internal operation of certain types of ICs.
- In this course we use a subset of the IEEE values: X10Z

Resolution Function for IEEE 9-valued

```

CONSTANT resolution_table : stdlogic_table := (
-- | U X 0 1 Z W L H -
-- | U X 0 1 Z W L H -
| 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', -- U
| 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', -- X
| 'U', '0', '0', '0', '0', '0', '0', '0', '0', '0', -- 0
| 'U', '1', '1', '1', '1', '1', '1', '1', '1', '1', -- 1
| 'U', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', -- Z
| 'U', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', 'W', -- W
| 'U', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', -- L
| 'U', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', 'H', -- H
| 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', -- -
);

```

14

01/07/2003

UAH-CPE/EE 422/522 ©AM

43

AND Table for IEEE 9-valued

```

CONSTANT and_table : stdlogic_table := (
-- | U X 0 1 Z W L H -
-- | U X 0 1 Z W L H -
| 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', -- U
| 'U', 'X', 'U', 'X', 'U', 'U', 'X', 'U', 'U', 'U', -- X
| 'U', '0', 'U', '0', 'U', '0', 'U', '0', 'U', '0', -- 0
| 'U', '1', 'U', '1', 'U', '1', 'U', '1', 'U', '1', -- 1
| 'U', 'Z', 'U', 'Z', 'U', 'Z', 'U', 'Z', 'U', 'Z', -- Z
| 'U', 'W', 'U', 'W', 'U', 'W', 'U', 'W', 'U', 'W', -- W
| 'U', 'L', 'U', 'L', 'U', 'L', 'U', 'L', 'U', 'L', -- L
| 'U', 'H', 'U', 'H', 'U', 'H', 'U', 'H', 'U', 'H', -- H
| 'U', 'X', 'U', 'X', 'U', 'X', 'U', 'X', 'U', 'X', -- -
);

```

3r

01/07/2003

UAH-CPE/EE 422/522 ©AM

44

AND Function for std_logic_vectors

```

function 'and' (l : std_logic; r : std_logic) return LOGIC is
begin
    return (and_3840, r);
end 'and'.

function 'and' (l, r : std_logic_vector) return std_logic_vector is
alias lv : std_logic_vector ( 1 to r'LENGTH ) is l;
alias rv : std_logic_vector ( 1 to r'LENGTH ) is r;
variable result : std_logic_vector ( 1 to r'LENGTH );
begin
    if ( r'LENGTH /= l'LENGTH ) then
        assert FALSE
            report "operands of overloaded 'and' operator are not of the same length"
            severity FAILURE;
    else
        for i in result'RANGE loop
            result(i) := and_table (lv(i), rv(i));
        end loop;
    end if;
    return result;
end 'and'.

```

01/07/2003

UAH-CPE/EE 422/522 ©AM

45