IE1204 Digital Design



F11: Programmable Logic, VHDL for Sequential Circuits

KTH Informations- och kommunikationsteknik

Masoumeh (Azin) Ebrahimi (masebr@kth.se)

Elena Dubrova (dubrova@kth.se) KTH / ICT / ES

This lecture

• BV pp. 98-118, 418-426, 507-519

Programmable Logic Devices

- Programmable logic devices (PLDs) were introduced in the 1970s
- They are based on a structure with an AND-OR array that makes it easy to implement a sum-of-products expression

Structure of a PLD



Programmable Logic Array (PLA)

 Both AND and OR arrays are programmable

 $f_1 = x_1 x_2 + x_1 \overline{x}_3 + \overline{x}_1 \overline{x}_2 x_3$ $f_2 = x_1 x_2 + x_1 \overline{x}_2 x_3 + x_1 x_3$



Programmable Array Logic (PAL)

 Only the AND array is programmable

 $f_1 = x_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3$ $f_2 = \overline{x}_1 \overline{x}_2 + x_1 x_2 x_3$



Combinatorial and register outputs

- In earlier PLDs there were
 - combinatorial outputs
 - register outputs (outputs with a flip-flop)
- For each circuit the number of combinational and register outputs was fixed
- To increase flexibility, *macrocells* were introduced
 - one can choose if an output is combinatorial or has a flip-flop

Macrocells in a PLD



Programming of PLDs



Complex PLD's (CPLD)

- PLDs were quite small (PALCE 22V10 had 10 flip-flops)
- To program larger functions, structures consisting of several PLD-like blocks were developed called Complex PLD (CPLD)

CPLD Structure



Programming of CPLDs via the JTAG interface

- Modern CPLDs (and FPGAs) can be programmed by downloading circuit description (programming information) via a cable
- Download usually uses a standard port called *JTAG port* (Joint Test Action Group)

Programming via the JTAG port



(a) CPLD in a Quad Flat Pack (QFP) package



You can program the chips when they are soldered to the circuit board - using the programmer you can select which chip you want to program through the JTAG port

(b) JTAG programming

Field Programmable Gate Arrays

- CPLDs are based on the AND-OR array
- It is difficult to make really large functions using CPLDs
- FPGAs use a different concept based on *logic blocks*

Structure of an FPGA



Look-up-tables (LUT)



Logic Block in a FPGA

 A logic block in an FPGA often consists of a LUT, a flip-flop and a multiplexer to select register output



Programming the LUT's and the connection matrix in an FPGA

- Blue cross: switch is programmed
- Black cross: switch is not programmed

 $f = f_1 + f_2$ $f = x_1 x_2 + \overline{x_2} x_3$



DE2 University Board

- DE2 Board
 - Cyclone II EP2C35
 FPGA (Datorteknikcourse)
 - 4 Mbytes of flash memory
 - 512 Kbytes of static RAM
 - 8 Mbytes of SDRAM
 - Several I/O-Devices
 - 50 MHz oscillator



Cyclone II Logic Element



Cyclone II Family

Table 1–1. Cyclone II FPGA Family Features								
Feature	EP2C5	EP2C8 (2)	EP2C15 (1)	EP2C20 (2)	EP2C35	EP2C50	EP2C70	
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416	
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250	
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000	
Embedded multipliers <i>(3)</i>	13	18	26	26	35	86	150	
PLLs	2	2	4	4	4	4	4	
Maximum user I/O pins	158	182	315	315	475	450	622	

(3) Total Number of 18x18 Multipliers

DE2

Stratix III Family

Table 1–1. Stratix III FPGA Family Features												
	Device/ Feature	ALMs	LEs	M9K Blocks	M144K Blocks	MLAB Blocks	Total Embedded RAM Kbits	MLAB RAM Kbits(2)	Total RAM Kbits <i>(3)</i>	18×18-bit Multipliers (FIR Mode)	PLLs	
Stratix III	EP3SL50	19K	47.5K	108	6	950	1,836	297	2,133	216	4	
Logic	EP3SL70	27K	67.5K	150	6	1,350	2,214	422	2,636	288	4	
	EP3SL110	43K	107.5K	275	12	2,150	4,203	672	4,875	288	8	
	EP3SL150	57K	142.5K	355	16	2,850	5,499	891	6,390	384	8	ן
	EP3SL200	80K	200K	468	36	4,000	9,396	1,250	10,646	576	12	
	EP3SE260	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12	
	EP3SL340	135K	337.5K	1,040	48	6,750	16,272	2,109	18,381	576	12	
Stratix III Enhanced Family	EP3SE50	19K	47.5K	400	12	950	5,328	297	5,625	384	4	
	EP3SE80	32K	80K	495	12	1,600	6,183	500	6,683	672	8	
	EP3SE110	43K	107.5K	639	16	2,150	8,055	672	8,727	896	8	
	EP3SE260 (1)	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12	

DE3 Board

Multiple processors can be implemented on an FPGA

- Nine II is a so-called 'softprocessor' (32-bit) that can be implemented on Altera's FPGAs
- Today's FPGAs are so large that multiple processors can fit on a single FPGA chip



Very powerful multiprocessor systems can be created on an FPGA!

ASICs

- An ASIC (Application Specific Integrated Circuit) is a circuit which is manufactured at a semiconductor factory
- In a *full custom* integrated circuit, the entire circuit is customized
- In an ASIC, some design steps have already been made to reduce design time and cost
- There are several types of ASICs:
 - Gate array ASICs
 - Standard cell ASIC

ASICs: Gate Array

 In a gate array ASIC, gates (or transistors) are already on silicon



ASICs: Gate Array

 We only need to create the links between the inputs and outputs of gates





ASICs: Standard Cells

 A standard cell can for example be AND, OR, Invert, XOR, XNOR, buffer, or a storage function as flipflop or latch.



Comparison FPGA, Gate Array, Standard Cell

	Initial Cost	Cost per part	Performance	Fabrication Time
FPGA	Low	High	Low	Short
Gate Array (ASIC)				
Standard Cell (ASIC)	High	Low	High	Long

Design Trade-Offs



Performance

VHDL: Sequential circuits

Moore machine



 In a Moore-type machine output signals depend only on the current state

How to model a state machine in VHDL?

- In a Moore machine, we have three blocks
 - Next state decoder
 - Output decoder
 - State register
- These blocks are executed in parallel

Quick question

Which logic gate is represented by the following VHDL code?

 $q \leq a$ and (not b);



Quick question

• Which logic gate is represented by the following VHDL code?

```
if (a /= b) then
  q <= '1';
else
  q <= '0';
end if;</pre>
```



Processes in VHDL

- A architecture in VHDL can contain multiple processes
- Processes are executed in parallel
- A process is written as a sequential program

Moore-machine processes

- For a Moore machine, we create three processes
 - Next state decoder
 - Output decoder
 - State register

Internal signals

- Moore machine contains internal signals for
 - Current state
 - Next state
- These signals are declared in the *architecture* description

Bottle dispenser vending machine in VHDL

- We use bottle dispenser vending machine as an example
- We describe its system controller in VHDL



Vending Machine: Flow diagram





Vending Machine: State diagram



- The state diagram contains all information required to generate an implementation
- <u>Assumption</u>: D flip-flops are used as state register
- 7 states: 3 flip-flops are needed

The state variable order is ABC, i.e. state (c) is A = 0, B = 1, C = 1

Vending Machine: Logic design



At next step, we develop the logic for the next state (D_A, D_B, D_C) and outputs









Unused state?!



		AB								
		00 01 11 10								
	0	а	-	d	f					
С	1	b	С	е	g					
$\Phi = (010)_{ABC}$										

$$\begin{split} A^{+} &= \overline{A} \cdot B \cdot EQ + \overline{A} \cdot B \cdot GT + A \cdot \overline{C} \implies A^{+}(010)_{ABC} = 1 \cdot 1 \cdot EQ + 1 \cdot 1 \cdot GT + 0 \cdot 1 = EQ + GT \\ B^{+} &= \overline{A} \cdot B \cdot EQ + B \cdot \overline{C} + \overline{B} \cdot C \cdot \overline{CP} + A \cdot \overline{B} \cdot C \implies B^{+}(010)_{ABC} = 1 \cdot 1 \cdot EQ + 1 \cdot 1 + \dots = 1 \\ C^{+} &= \overline{A} \cdot \overline{C} \cdot CP + B \cdot \overline{C} \cdot DR + A \cdot \overline{B} \cdot CR + \overline{B} \cdot C \\ \implies C^{+}(010)_{ABC} = 1 \cdot 1 \cdot CP + 1 \cdot 1 \cdot DR + 0 \cdot 0 \cdot CR + 0 \cdot 0 = CP + DR \\ A^{+}B^{+}C^{+} = -1 - = 010, 110, 011, 111 \rightarrow \Phi, d, c, e \end{split}$$

Vending Machine: Logic Design



Now you can design "Next State Decoder" and "Output Decoder" by knowing the logic function of D_a , D_b , D_c , and logic function of outputs "Drop", "Return_10_Cent", "CLR_ACC", and "DEC_ACC".

Vending Machine in VHDL: Entity

- Entity describes the system as a 'black box '
- Entity describes the interface to the outside world
- All inputs and outputs are described
- Apart from the input and output signals, block diagram needs signals for
 - Clock
 - Reset (active low)

```
ENTITY Vending_Machine IS
PORT (
-- Inputs
```

	coin_present	:	IN	st	d_logic;
	gt_1_euro	:	IN	st	d_logic;
	eq_1_euro	:	IN	st	d_logic;
	lt_1_euro	:	IN	st	d_logic;
	drop_ready	:	IN	st	d_logic;
	changer_ready	:	IN	st	d_logic;
	reset_n	:	IN	st	d_logic;
	clk	:	IN	st	d_logic;
	Outputs				
	dec_acc		: 01	JT	std_logic;
	clr_acc		: 01	JT	std_logic;
	drop		: 01	JT	std_logic;
	return_10_cent		: 01	JT	<pre>std_logic);</pre>
END	Vending_Machine;				

Vending Machine in VHDL: Architecture

- The architecture describes the function of the machine
- We define
 - internal signals for the current and next states
 - three processes for next-state decoder, output decoder and state register

Vending Machine in VHDL: Internal Signals

- We need to create a type for internal signals
- Since we describe the states, we use an enumerated type with the values a, b, c, d, e, f, g
- We declare one variable for the current state (current_state) and one for the next state (next_state)

```
ARCHITECTURE Moore_FSM OF Vending_Machine IS
   TYPE state_type IS (a, b, c, d, e, f, g);
   SIGNAL current_state, next_state: state_type;
BEGIN -- Moore_FSM
```

•••

Vending Machine in VHDL: Internal Signals

- If we do not specify a state assignment, synthesis tool will select it
- We can force a certain encoding using attributes (NOTE: Attributes are dependent on synthesis tool and thus are not portable!)

ARCHITECTURE Moore_FSM OF Vending_Machine IS

TYPE state_type IS (a, b, c, d, e, f, g);

- -- We can use state encoding according to BV 8.4.6
- -- to enforce a particular encoding (**for Quartus**)

ATTRIBUTE enum_encoding : string;

ATTRIBUTE enum_encoding OF state_type : TYPE IS "000 001 011 110 111 100 101";

SIGNAL current_state, next_state : state_type; BEGIN -- Moore_FSM

•••

Vending Machine in VHDL: Process for Next-State Decoder



- Next-State-Decoder is described as a process
- Sensitivity list contains all the inputs that 'activate' the process

NEXTSTATE : PROCESS (current_state, coin_present, gt_1_euro, eq_1_euro, lt_1_euro, drop_ready, changer_ready) -- Sensitivity List BEGIN -- PROCESS NEXT_STATE

Vending Machine in VHDL: Process for Next-State-Decoder



• We now use a CASE statement to describe the transitions to the next state from each state conditions



Vending Machine in VHDL: Process for Next-State-Decoder

• We can simplify the description by specifying a default value for the next state

It is important to we specify all options for next_state signal. Otherwise we may implicitly set next_state <= next_state which generates a loop.

...

...

Vending Machine in VHDL: Process for Next-State-Decoder

 We terminate the CASE statement with a WHEN OTHERS statement. Here we specify that we should go to the state a if we end up in an unspecified state

WHEN g => next_state <= c; WHEN OTHERS => next_state <= a; END CASE; END PROCESS NEXTSTATE;

...

Vending Machine in VHDL: Process for Output-Decoder



- Output decoder is described as a separate process
- Sensitivity list contains only the current state because the outputs are directly dependent on it

Vending Machine in VHDL: Process for Output-Decoder

OUTPUT : PROCESS (current state) BEGIN -- PROCESS OUTPUT drop <= '0'; clr_acc <= '0'; dec_acc <= '0'; return 10 cent <= '0';</pre> CASE current state IS WHEN d => drop <= '1'; WHEN e => clr_acc <= '1'; WHEN f => return_10_cent <= '1'; WHEN g => dec_acc <= '1'; WHEN OTHERS => NULL; END CASE; END PROCESS OUTPUT;

Vending Machine in VHDL: Process for State register



 State register is modeled as a synchronous process with asynchronous reset (active low)

```
CLOCK : PROCESS (clk, reset_n)
BEGIN -- PROCESS CLOCK
IF reset_n = '0' THEN -- asynchronous reset (active low)
        current_state <= a;
ELSIF clk'EVENT AND clk = '1' THEN -- rising clock edge
        current_state <= next_state;
END IF;
END PROCESS CLOCK;</pre>
```

Quick question

• Which state machine is represented by this VHDL code?



Mealy machine



 In a Mealy machine, output signals depend on both the current state <u>and</u> inputs

Mealy machine in VHDL

- A Mealy machine can be modeled in the same way as the Moore machine
- The difference is that output decoder is also dependent on the input signals
- Process which models outputs needs to have input signals in the sensitivity list as well!

More on VHDL

- The sample code for bottle dispenser available on the course website
- Look at the study of "VHDL synthesis" on the course website
- Both Brown/Vranesic- and Hemert-book includes code samples

Summary

- PLD, PAL, CPLD
- FPGA
- ASIC gate array and standard cell
- Modeling sequential circuits with VHDL
- Next lecture: BV pp. 584-640