## IE1204 Digital Design



KTH Informations- och kommunikationsteknik

## F11: Programmable Logic, VHDL for Sequential Circuits

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

$$
f_{1}=x_{1} x_{2}+x_{1} \bar{x}_{3}+\bar{x}_{1} \bar{x}_{2} x_{3}
$$

$$
f_{2}=x_{1} x_{2}+x_{1} \bar{x}_{2} x_{3}+x_{1} x_{3}
$$



## Programmable Array Logic (PAL)

- Only the AND array is programmable

$$
\begin{aligned}
& f_{1}=x_{1} x_{2} \bar{x}_{3}+\bar{x}_{1} x_{2} x_{3} \\
& f_{2}=\bar{x}_{1} \bar{x}_{2}+x_{1} x_{2} x_{3}
\end{aligned}
$$



## 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



Select
Enable

To AND plane
A programmable multiplexer can be used to select the type of output

## 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

(b) JTAG programming

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

## 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)



A LUT with $n$ inputs can realize all combinational functions with up to $n$ inputs.
The usual size of LUT in an FPGA is $n=4$

## 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

$$
\begin{aligned}
& f=f_{1}+f_{2} \\
& f=x_{1} x_{2}+\bar{x}_{2} x_{3}
\end{aligned}
$$



## 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 <br> Kbits plus <br> 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 <br> multipliers (3) | 13 | 18 | 26 | 26 | 35 | 86 | 150 |
| PLLs | 2 | 2 | 4 | 4 | 4 | 4 | 4 |
| Maximum user <br> I/O pins | 158 | 182 | 315 | 315 | 475 | 450 | 622 |

(3) Total Number of 18x18 Multipliers

## Stratix III Family

Table 1-1. Stratix III FPGA Family Features

|  | Device/ Feature | ALMs | LEs | M9K <br> Blocks | M144K <br> Blocks | MLAB <br> Blocks | Total Embedded RAM Kbits | MLAB <br> RAM <br> Kbits (2) | Total <br> RAM <br> Kbits (3) | 18×18-bit <br> Multipliers <br> (FIR Mode) | PLLs |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Stratix III Logic Family | EP3SL50 | 19K | 47.5K | 108 | 6 | 950 | 1,836 | 297 | 2,133 | 216 | 4 |
|  | EP3SL70 | 27K | 67.5K | 150 | 6 | 1,350 | 2,214 | 422 | 2,636 | 288 | 4 |
|  | FP3SL110 | 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 |
|  | 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 | EP3SE50 | 19K | 47.5K | 400 | 12 | 950 | 5,328 | 297 | 5,625 | 384 | 4 |
| Enhanced | EP3SE80 | 32 K | 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 <br> (1) | 102K | 255K | 864 | 48 | 5,100 | 14,688 | 1,594 | 16,282 | 768 | 12 |

## 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 <br> Time |
| :--- | :--- | :--- | :--- | :--- |
| FPGA | Low | High | Low | Short |
| Gate Array <br> (ASIC) |  | Low | High | Long |
| Standard Cell <br> (ASIC) | High |  |  |  |

## Design Trade-Offs



## 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?

$$
\mathrm{q}<=\mathrm{a} \text { and }(\text { not } \mathrm{b})
$$




Alt: B


Alt: C

## Quick question

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

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



## 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



## Vending Machine: Logic design



At next step, we develop the logic for the next state $\left(D_{A}, D_{B}, D_{C}\right)$ and outputs




## Decoder: Output signals



## Unused state?!



\[

\]

$$
\begin{aligned}
& A^{+}=\bar{A} \cdot B \cdot E Q+\bar{A} \cdot B \cdot G T+A \cdot \bar{C} \Rightarrow A^{+}(010)_{A B C}=1 \cdot 1 \cdot E Q+1 \cdot 1 \cdot G T+0 \cdot 1=E Q+G T \\
& B^{+}= \bar{A} \cdot B \cdot E Q+B \cdot \bar{C}+\bar{B} \cdot C \cdot \overline{C P}+A \cdot \bar{B} \cdot C \quad \Rightarrow \quad B^{+}(010)_{A B C}=1 \cdot 1 \cdot E Q+1 \cdot 1+\ldots=1 \\
& C^{+}=\bar{A} \cdot \bar{C} \cdot C P+B \cdot \bar{C} \cdot D R+A \cdot \bar{B} \cdot C R+\bar{B} \cdot C \\
& \Rightarrow \quad C^{+}(010)_{A B C}= 1 \cdot 1 \cdot C P+1 \cdot 1 \cdot D R+0 \cdot 0 \cdot C R+0 \cdot 0=C P+D R \\
& A^{+} B^{+} C^{+}=-1-=010,110,011,111 \rightarrow \Phi, d, c, e
\end{aligned}
$$

## 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 funtion 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 std_logic;
    gt_1_euro : IN std_logic;
    eq_1_euro : IN std_logic;
    lt_1_euro : IN std_logic;
    drop_ready : IN std_logic;
    changer_ready : IN std_logic;
    reset_n : IN std_logic;
    clk : IN std_logic;
    -- Outputs
    dec_acc : OUT std_logic;
    clr_acc : OUT std_logic;
    drop : OUT std_logic;
    return_10_cent : OUT std_logic);
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

```
CASE current_state IS
    WHEN a => IF coin_present = '1' THEN
            next_state <= b;
        ELSE
            next_state <= a;
        END IF;
    WHEN b => IF coin_present = '0' THEN
            next_state <= c;
        ELSE
            next_state <= b;
        END IF;
```



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

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

```
next_state <= current_state;
CASE current_state IS
WHEN a => IF coin_present = '1' THEN
    next_state <= b;
    END IF;
WHEN b => IF coin_present = '0' THEN
    next_state <= c;
    END IF;
```

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';
    CASE current_state IS
        WHEN d 
        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;
```


## Quick question

- Which state machine is represented by this VHDL code?

```
        case state is
```

            when \(0=>\)
                if ( \(k=11\) ') then
                nextstate <= 1;
                else
                nextstate <= 2;
                end if;
    when \(1 \Rightarrow\) nextstate <= 2;
    when others \(=>\) nextstate \(<=0\);
    end case;
    

## Mealy machine



- In a Mealy machine, output signals depend on both the current state and 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

