Lecture 1: Number Systems and Codes

Vlad-Cristian Miclea

Technical University of Cluj-Napoca

October 3, 2022

Contents

Course Details

Course general info

- Vlad.Miclea@cs.utcluj.ro;
- $users.utcluj.ro/\sim vmiclea \rightarrow$ Teaching
- Lectures in P03 or on Teams
- Quite complicated subject $\rightarrow$ Pay attention and do not miss lectures ("incremental"); Problems from the book!!
- Ask questions (anytime) either in English or Romanian

Grading

- Final exam: 70 points
  - Written exam
  - No midterm
- Lab evaluation: 30 points (last week of the semester)
  - Possible: tasks/quizzes during the semester

Lecture syllabus

1. Number systems and codes
2. Binary arithmetic
3. Boolean Algebra
4. Methods for minimizing Boolean functions
5. Combinational logic circuits analysis and design
6. Designing digital systems with SSI, MSI, LSI and VLSI circuits
7. Sequential logic circuits. Latches and Flip-Flops
8. Frequency dividers, counters
9. Data registers, converters, memories
10. Designing digital systems using Flip-Flops
11. Designing digital systems using memories, multiplexers, decoders, counters
12. Designing sequential synchronous systems
13. Designing digital systems using programmable devices

Lab Information

General Info

- The presence at the laboratory work is mandatory!!!
- Physical meetings: Rooms 204, 211, 213 - Observatorului 2, 2nd floor
- Requirement: Read lab before and draw main circuits!!
- Password for labs and circuits (on the website): **LD_2022_aut**

Simulator

- Generally, the lab work is done on real circuits and boards
- For some labs: we have to use simulators
- You will have to implement it using Logisim (you will receive details for installation)
- More details at the first lab!

Labs

1. Introduction
2. Fundamental Logic Circuits
3. Design and simulation of logic circutits using computer aided design software (I, II)
4. Combinational Logic Circuits CLC
5. MSI Combinational Logic Circuits
6. Implementing CLCs in ActiveHDL
7. Flip-Flops
8. Counters (I, II)
9. Registers and Shift Registers
10. Implementing CLSs in ActiveHDL
11. FPGA XILINX Circuits Family (1)
12. FPGA XILINX Circuits Family (2)

Number Systems

Numbers

- Most of the data is stored as numbers
- Information is **encoded** – need a form of representation

Number system: The set of rules that provide a representation for each number by using digits.

According to the type of representation

- Positional NS – value of a digit is determined by its position inside the number
- Un-positional NS – the position of the digit has a different relevance

Number Systems

Number **N**, in positional system, in number base **b** is represented:

$$N = a_{q-1}b^{q-1} + ... + a_0 b^0 + ...a_{-p}b^{-p} = \sum_{i=-p}^{q-1} a_i b^i \tag{1}$$

Details

- Base b is an integer number, generally $b > 1$
- Coefficient (digit) a_i is an integer, following $0 \leq a \leq b - 1$
- Notation $(N)_b$ means "number N in base b"
- If the base is not specified, it is generally 10
- Complement of a digit a (denoted $\bar{a}$ in base b) is defined in eq. (2):

$$\bar{a} = (b - 1) - a \tag{2}$$

Number Systems

Binary Systems

- Base b is 2
- The allowed digits are 1 and 0 ("bits")
- Complements: $\bar{0} = 1$ and $\bar{1} = 0$

Other useful Systems

- There are other common systems
- Octal, Hexadecimal - why?

Number Systems

Octal System

- Base b is 8
- The allowed digits are 0..7

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Number Systems

Octal System

- Base b is 8
- The allowed digits are 0..7

Hexadecimal System

- Base b is 16
- 16 digits: $0 - 9$ and $A - F$
- Really useful for representation (just 1 character)

Byte Representation

- **1 byte = 8 bits!!!**
- Which is the range?
- Mostly used for representation

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Number Base Conversion

- We often want to get from base b_1 to base b_2
- In positional systems: use a series of multiplications and divisions
- There are two main cases:
  - $b_1 < b_2$
  - $b_1 > b_2$
- https://www.rapidtables.com/convert/number/base-converter.html

Number Base Conversion

Case 1: $b_1 < b_2$

- $(N)_{b1}$ is expressed as a polynomial with coefficients in base b_1
- The result (with its operations: addition and multiplication) is evaluated in base b_2
- Example: $b_1 = 3$, $b_2 = 10$ and $(N)_3 = 2120.1$

$$\begin{aligned}(N)_{10} &= 2 \times 3^3 + 1 \times 3^2 + 2 \times 3^1 + 0 \times 3^0 + 1 \times 3^{-1} \\ &= 54 + 9 + 6 + 0 + 0.3 = 69.3\end{aligned} \tag{3}$$

Number Base Conversion

Case 2: $b_1 > b_2$

- The arithmetic is done in base b_1
- There are different algorithms for Integer part and Fractional Part

Case 2.1: Integer part

- Number is divided by the base b_2
- Results a Quotient and a Remainder
- The Remainder is stored
- The Quotient is further divided by the base b_2
- The algorithm continues until the **Quotient is** 0
- The Integer part of the transformed number is obtained by reading the resulting Remainders in **reverse** order (starting from the last division)

Number Base Conversion

Case 2: $b_1 > b_2$

- The arithmetic is done in base b_1
- There are different algorithms for Integer part and Fractional Part

Case 2.2: Fractional part

- Number is multiplied by the base b_2
- It results a new number, with a Integer part and a Fractional part
- The Integer part is stored
- The new Fractional part is further multiplied by the base b_2
- The algorithm continues until the **Desired precision is obtained**
- The Fractional part of the transformed number is obtained by reading the resulting Integer parts in **direct** order (starting from the first multiplication)

Base conversion

Example: $b_1 = 10$, $b_2 = 4$ and $(N)_{10} = 347.4$

Integer part

$$\begin{aligned} 374 \div 4 &= 86\,rem\,3 \\ 86 \div 4 &= 21\,rem\,2 \\ 21 \div 4 &= 5\,rem\,1 \\ 5 \div 4 &= 1\,rem\,1 \\ 1 \div 4 &= 0\,rem\,1 \end{aligned}$$

Result (integer part): $(11123)_4$

Fractional part

$$\begin{aligned} 0.4 \times 4 &= 1.6 \rightarrow 1 \\ 0.6 \times 4 &= 2.4 \rightarrow 2 \\ 0.4 \times 4 &= 1.6 \rightarrow 1 \\ 0.6 \times 4 &= 2.4 \rightarrow 2 \\ &\ldots \end{aligned}$$

Result (fractional part): $(1212..)_4$

Resulting Number: $(11213.1212...)_4$

Number Base Conversion

Special conversion cases

- Conversion from octal/hexadecimal into binary (and reverse)
- Just group/ungroup bits together (either 3 or 4)
- Octal/hexa to binary (be careful with 0's for formatting)

$$(123.4)_8 = (001\,010\,011)_2 = (1010011)_2$$
$$(2C5F)_{16} = (0010\,1100\,0101\,1111)_2 = (10110001011111)_2$$

- Binary to octal/hexa

$$(1010110.0101)_2 = (001\,010\,110.\,010\,100)_2 = (126.24)_8$$
$$(10111001101010.1)_2 = (0010\,1110\,0110\,1010.\,1000)_2 = (2E6A.8)_{16}$$

Binary Codes

Generalities

- In real-life we prefer decimal codes (also in human-machine interface)
- (Decimal) Numbers are represented in binary
- Each decimal digit (0-9) is represented on ? bits
- Binary codes:
  - Weighted
  - Un-weighted

Weighted binary codes

- Each binary digit has a specific weight
- The number is encoded by the sum of weights of 1 digits

$$N = \sum_{i=0}^{K-1} a_i b_i \tag{4}$$

- where K is the number of digits and $a_i \in \{0, 1\}$
- this is a particular case of equation (1) (see slide 9)

Weighted binary codes

BCD

- Natural binary code
- The weights correspond to the powers of 2

Decimal	b_3	b_2	b_1	b_0
D	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Weighted binary codes

BCD

- Natural binary code
- The weights correspond to the powers of 2

Auto-complementary codes

- Condition: sum of weights has to be 9
- Complement of N is $9 - N$
- Positive auto-complementary (2421)

Decimal	b_3	b_2	b_1	b_0
D	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Weighted binary codes

BCD

- Natural binary code
- The weights correspond to the powers of 2

Auto-complementary codes

- Condition: sum of weights has to be 9
- Complement of N is $9 - N$
- Positive auto-complementary (2421)
- Negative auto-complementary

Decimal	b_3	b_2	b_1	b_0
D	6	4	2	-3
0	0	0	0	0
1	0	1	0	1
2	0	0	1	0
3	1	0	0	1
4	0	1	0	0
5	1	0	1	1
6	0	1	1	0
7	1	1	0	1
8	1	0	1	0
9	1	1	1	1

Unweighted binary codes

Such codes have different building rules.

Excess3

- Formed by adding 0011 (binary representation of digit 3) to each word in BCD representation
- It results an auto-complementary code
- Does not contain the combination 0000

Decimal	b_3	b_2	b_1	b_0
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Unweighted binary codes

Gray

- Cyclic encoding: successive digits will differ by only 1 digit
- Reflective encoding: n-bit code will be formed by reflecting the $n-1$-bit codes
- Ex: 2-bit encoding formed by reflecting 2 1-bit codes

0	0
0	1
1	1
1	0

Decimal	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1

Error detecting and correcting Codes

Generalities

- Encoding is really useful when transferring information
- Due to the transfer, the information can alter
- We need the correct information
  - Detect if the information is altered
  - Correct the code

Error detecting codes – EDC

The occurrence of a single bit-flip will change a valid word to an invalid one.

Parity check method

- Add a single bit to each word
- The bit will tell if the word has an even or an odd number of 1 values
- Example: send the word (or bitstring) 1011
  - For odd parity: will add a 1, resulting the new word 1 1011
  - For even parity: will add a 0, resulting the new word 0 1011

Error detecting codes

"2 out of 5" code

- It has the weights 1, 2, 4, 7
- Exception: encoding for digit 0
- The weight associated to the bit corresponding to 0 – will tell if the number of 1's is odd or even

Decimal	0	1	2	4	7
0	0	0	0	1	1
1	1	1	0	0	0
2	1	0	1	0	0
3	0	1	1	0	0
4	1	0	0	1	0
5	0	1	0	1	0
6	0	0	1	1	0
7	1	0	0	0	1
8	0	1	0	0	1
9	0	0	1	0	1

Error correcting codes – ECC

A code is ECC if the correct bitstring can be inferred from the erroneous one.

Hamming codes

- Singular error correcting codes: allow for a single error!
- The minimum distance (nr of differences) between two different bitstrings has to be 3
- Number of "control bits" (additional bits for correction) is given by:

$$2^k \geq m + k + 1$$

- m is the number of useful bits
- k is the number of control bits

Error correcting codes – ECC

Hamming code - example

- Hamming code for a message of size $m = 4$ in BCD code
- if we apply Hamming relation $\rightarrow k = 3$
- Control bits will be inserted on the positions = powers of 2

1	2	3	4	5	6	7
c_1	c_2	b_1	c_3	b_2	b_3	b_4

- Control bits are computed using the relations:

$$c_1 = b_1 \oplus b_2 \oplus b_4$$
$$c_2 = b_1 \oplus b_3 \oplus b_4$$
$$c_3 = b_2 \oplus b_3 \oplus b_4$$

- where $\oplus$ is the xor operator – what does this compute?

Error correcting codes – ECC

Full table for generating the Hamming code on multiple bits'

p are the control bits;

d are the message bits

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		**p1**	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**	**p8**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**p16**	**d12**	**d13**	**d14**	**d15**	
	p1	✓		✓		✓		✓		✓		✓		✓		✓		✓		✓		
	p2		✓	✓			✓	✓			✓	✓			✓	✓			✓	✓		…
Parity bit coverage	**p4**				✓	✓	✓	✓					✓	✓	✓	✓					✓	
	p8								✓	✓	✓	✓	✓	✓	✓	✓						
	p16																✓	✓	✓	✓	✓	

Error correcting codes – ECC

Hamming code - example

- $b_1 b_2 b_3 b_4 = 0100$ – (send 4, in BCD)
- compute control bits: $c_1 = 1$; $c_2 = 0$; $c_3 = 1$.
- Resulting message will be: 1001100

Conclusions

Summary

- Number Systems
  - Binary, Octal, Hexadecimal
- Number base conversion
  - Two cases depending on the relation between bases
- Binary codes
  - Weighted: BCD, Auto-complementary
  - Unweighted: Excess3, Gray
- Error detecting codes
- Error correcting codes

Next week

- Number representation
- Binary arithmetic

Thank you for your attention!