Computer Architecture. Week 4

Muhammad Fahim, Alexander Tormasov

Innopolis University

m.fahim@innopolis.ru
a.tormasov@innopolis.ru

September 24, 2020

Topic of the Lecture

- Hardware Building Blocks (Combinational Logic)

Topic of the Tutorial

- Hardware Description Languages and Combinational Logic Circuits

Topic of the Lab

- Combinational Logic on FPGA Board

Content of the Class

- Premises – Boolean Algebra
- Hardware Building Blocks
- AND, OR, NOT Gate
- Decoders
- Multiplexers
- Two-level Logic and PLAs
- Constructing Basics ALU
- Problem: Ripple carry adder is slow
- Solution: Carry Look Ahead
- Summary

Boolean Algebra – Wonderful World of 0 & 1

- Every time you use a computer you are relying on Boolean logic: a system of logic established long before computers were around, named after the English mathematician George Boole (1815 - 1864).
- In Boolean logic statements can either be $true/1$ or $false/0$
- Often a non-zero value is considered to "be evaluated" to true
- **General Theorem:** Any logical function can be made of two levels with a level of *and* and a level of *or*, plus *negation*

Boolean Algebra – Mathematically

- Two-valued Boolean algebra comprises:
  - the set of values $B = \{0, 1\}$
  - variables $a, b, c, ..., x, y, z$ which represent either element of B,
  - a pair of binary operators "$+$" and "." and a unary operator denoted "′"
  - a set of basic algebraic relations called axioms.

- A binary operator means a function f which operates on two variables, say a and b, to produce a dependent variable, say x

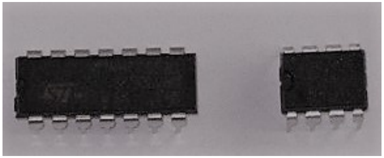
$$(a, b) \to f(a, b) = x = a + b \text{ (in case of +)}$$
$$or \quad (a.b) \text{ (in case of .)}$$

- Clearly a unary operator applies to a single variable.
- In Boolean algebra negative quantities are forbidden.

Note: More details in discrete math course

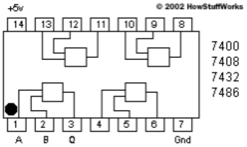
Integrated Circuit (IC)

- IC stands for “Integrated Circuit”.
- An IC is a tiny circuit made up of resistors, diodes and transistors and placed in a single package.
- Each IC is created for a specific purpose.
- The figure below shows from left to right – an IC that contains logic gates packaged in a 14 pin package, a timer IC in an 8 pin package.



Mapping of Hardware and Logic gates

- Example: 7408 AND chip (four gates per chip).



- If you look at the chip, there will normally be a dot or an indentation at the pin 1 end of the chip, or some other marking to indicate pin 1. Push the chip into the breadboard so it straddles the center channel.
- Pin 7 must connect to ground and pin 14 must connect to +5 volts.
- Connect those two pins appropriately. (If you connect them backward you will burn the chip out)

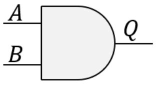
Hardware Building Blocks (1/5)

- Computer electronics are digital
  - Only two voltages of interest
    - high = 1 = asserted = active = true
    - low = 0 = deasserted = inactive = false
  - Really a single voltage level
    - All voltages above are high
    - All voltages below are low
  - Logic blocks
    - Combinational logic = no memory elements
    - Sequential logic = memory elements

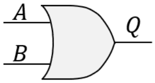
Hardware Building Blocks (2/5)

- CPUs and other blocks are built out of transistors and wires, and implemented as ICs
- Transistors are put together to make gates, and gates put together to make CPUs and other blocks
- **Gate**:
  - Hardware unit that receives boolean inputs and produces one output: implements a basic logic function
  - Can be represented as a truth table

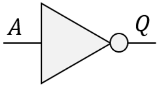
AND, OR, NOT Gate



AND Gate



OR Gate

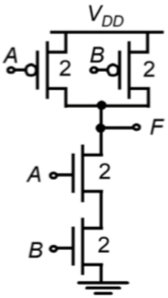


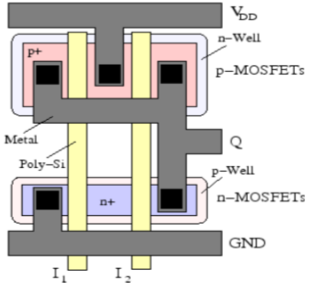
NOT Gate

- Note:
  - The inverter is sometimes drawn as just a bubble, without the triangle.
  - Physical implementations of complex tasks can be minimized using the Boolean laws while the functionality will remain the same.

Example – The Physical and Circuit Layout

- CMOS Two-input NAND Gate





De Morgan's Laws

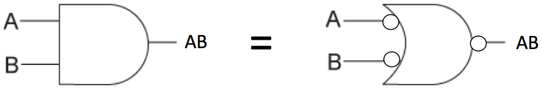
- A mathematician named De Morgan developed a pair of important rules regarding group complementation in Boolean algebra.
- De Morgan's law
  - An OR gate with all inputs inverted (a Negative-OR gate) behaves the same as a NAND gate, and an AND gate with all inputs inverted (a Negative-AND gate) behaves the same as a NOR gate.

$$\overline{A+B} = \overline{A}\,\overline{B}$$
$$\overline{AB} = \overline{A} + \overline{B}$$

Application of De Morgan's Laws

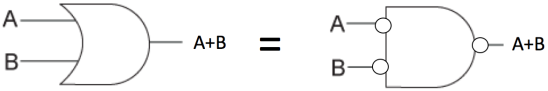
- Construction of AND gate with an OR and NOT gates

$$\overline{\overline{A}+\overline{B}} = AB$$



- Construction of OR gate with an AND and a NOT gate

$$\overline{\overline{A}\ \overline{B}} = A+B$$

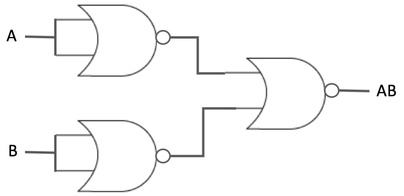


Hardware Building Blocks (3/5)

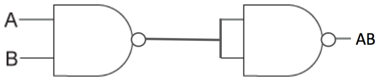
- Any logical function can be constructed:
  - using AND gates and inverter
  - using OR gates and inverter
- There are two "inverting" gates NOR and NAND and correspond to inverted OR and inverted AND gates, respectively.
- NOR and NAND gates are called universal gate, since any logic function can be built using this one gate type

Hardware Building Blocks (4/5)

- Construction of an AND with a NOR and NAND gate



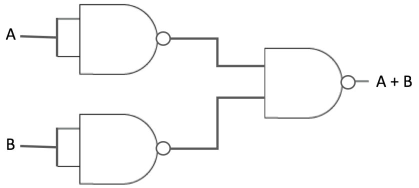
AND gate with NOR gate



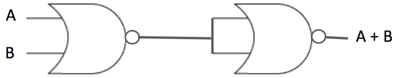
AND gate with NAND gate

Hardware Building Blocks (5/5)

- Construction of an OR with a NAND and NOR gate



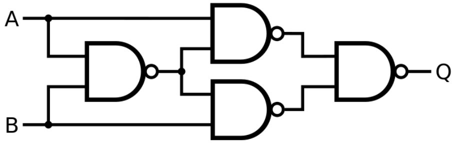
OR gate with NAND gate



OR gate with NOR gate

The XOR Gate

- Building of XOR gate with universal NAND gate

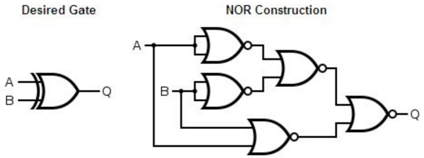


Truth Table

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

The XOR Gate

- Building of XOR gate with universal NOR gate



Truth Table

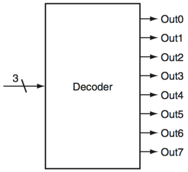
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

Combinational Logic

- Decoders
- Multiplexors
- Two-Level Logic and PLAs
- Constructing the ALU
- Tailoring the ALU to the MIPS
- Problem: Ripple carry adder is slow
- Carry Look Ahead

Decoders

- **Definition:** A decoder is a logic block that has an n-bit input and 2^n outputs where only one output is true for each input combination.

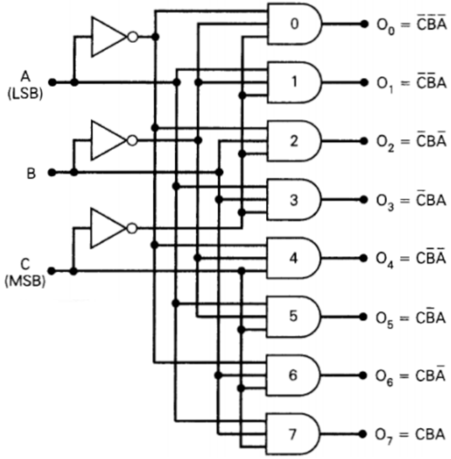


a. A 3-bit decoder

Truth Table

Input			Output							
I2	I1	I0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

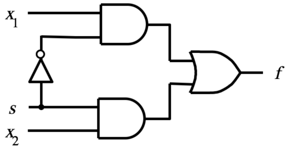
Decoder Example with Gate



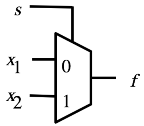
- Note: LSB: Least Significant Bit
- MSB: Most Significant Bit

Multiplexors

- Multiplexor is a combinational logic device
- A multiplexor (shortly, MUX) could be called a selector
- **Definition**: A selector value (or control value) is the control signal that is used to select one of the input values of a multiplexor as the output of the multiplexor.
- **Example:** A two-input multiplexor



Circuit



Graphical symbol

$$f(s, x1, x2) = \overline{s}x1 + sx2$$

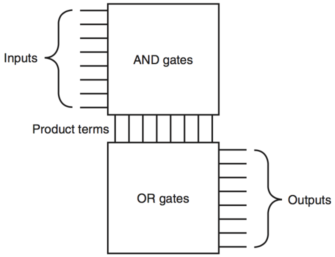
- **Note**: You can do verification using truth table

Multiplexors

- General Muxes:
  - It contains control bits and data bits
  - Control bits select which data bit will pass through and all others are blocked
- In general:
  - 1 control bit selects between 2 data bits
  - 2 control bits select between 4 data bits
  - 3 control bits select between 8 data bits
  - n control bits select between 2^n data bits
  - We can build a mux of any size to serve our purpose

Programmable Logic Array (PLA) (1/2)

- A Programmable Logic Array (PLA) is structured-logic element composed of a set of inputs and corresponding input complements and two stages of logic:
  - The first generates product terms of the inputs
  - The second generates sum terms of the product terms.
- Hence, PLAs implement logic functions as a sum of products representation.

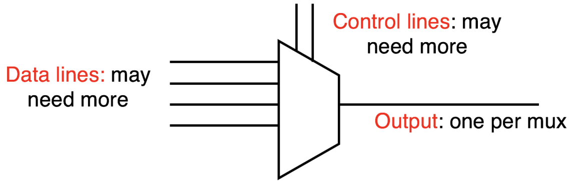


Programmable Logic Array (PLA) (2/2)

- PLAs were popular in late 70s.
- Worth mentioning, but the approach used to design PLAs is not currently used.
- **Reasons**
  - PLA is not fast enough and consumes more voltage.
  - Not practical for modern circuits due to its low speed and high gate count.
- Note: The logic synthesis tools are used to in modern circuits design.

Constructing a Basic ALU

- **Strategy:** implement all functions and select desired result.
- Therefore the ALU consists of 32 muxes (one for each necessary output bit).

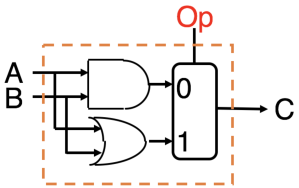


- Now we go through instruction set and add data lines to implement all necessary functionality.

NOTE: ALU: Arithmetic and Logic Unit

Logical Instructions

- AND Instruction
  - A single data output should be a simple AND function
- OR Instruction
  - Output should be a simple OR gate



Definition

Op	C
0	A and B
1	A or B

Constructing a basic ALU – Addition

- Binary Addition Review

	1	1	1	0	0	Carries
		1	0	1	1	Augend
+		1	1	1	0	Addend
	1	1	0	0	1	Sum

- Adds two input bits to produce a sum and carry out (Half-Adder).

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$C = XY$$

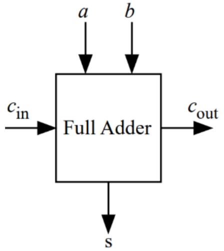
$$S = X'Y + XY' = X \oplus Y$$

XOR

- Adding 3 bits together to get a two bit number (Full-Adder)

Constructing a basic ALU – Addition

- Addition is not going to be so easy.
- The component which will perform a 1-bit addition is called a full adder.
- A 1-bit full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs.



Definition

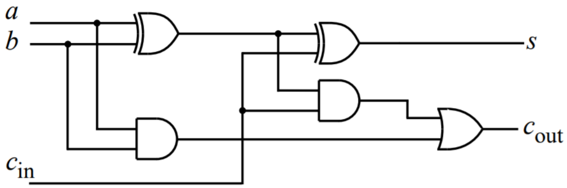
a	b	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

One-bit Full Adder (2/2)

To create one-bit full adder:

- Implement gates for Sum
- Implement gates for CarryOut
- Connect all inputs with same name
- The symbol for one-bit adder now represents this collection of gates and wires (simplifies description)

The Gate Implementation of 1-bit Full Adder

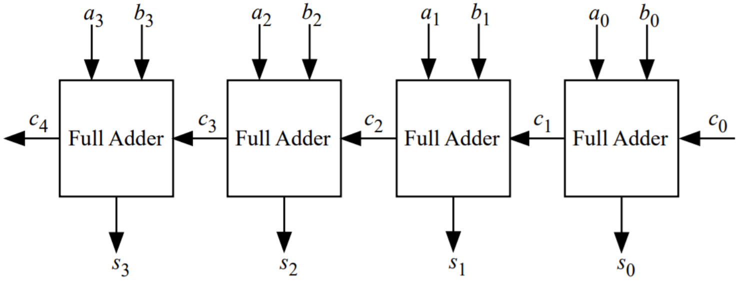


$$s = (a \oplus b) \oplus c_{in}$$
$$c_{out} = ab + (a \oplus b)c_{in}$$

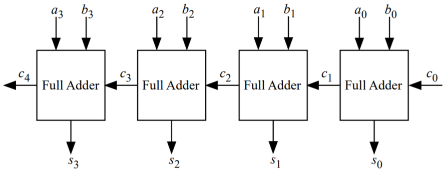
- To make a 32-bit adder we could simply connect 32 1-bit adder together
- It can be constructed with full adders connected in cascaded way (known as Ripple Carry Adder).

Ripple Carry Adder

- A ripple carry adder is a circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the carry output from each full adder connected to the carry input of the next full adder in the chain.



Ripple Carry Adder Delays



- In the ripple carry adder, the output is known after the carry generated by the previous stage is produced.
- Thus, the sum of the most significant bit is only available after the carry signal has rippled through the adder from the least significant stage to the most significant stage.
- As a result, the final sum and carry bits will be valid after a considerable delay (Get very slow – What is the solution?)

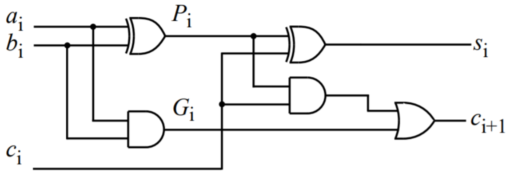
Carry Lookahead Adder (CLA)

- CLA solves the carry delay problem by calculating the carry signals in advance based on the input signals.
- It is based on the fact that a carry signal will be generated in two cases:

  (1) when both bits a_i and b_i are 1, or
  (2) when one of the two bits is 1 and the carry-in is 1.
  Thus, one can write:

$$\begin{aligned} c_{i+1} &= a_i.b_i + (a_i \oplus b_i).c_i \\ s_i &= (a_i \oplus b_i) \oplus c_i \end{aligned}$$

Carry Lookahead Adder (CLA)



$$
\begin{aligned}
c_{i+1} &= G_i + P_i.c_i \\
s_i &= P_i \oplus c_i
\end{aligned}
$$

where

$$
\begin{aligned}
G_i &= a_i.b_i \\
P_i &= a_i \oplus b_i
\end{aligned}
$$

- G_i and P_i are called the carry generate and carry propagate terms, respectively.

Carry Lookahead Adder (CLA)

- Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively.
- If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value.

Carry Lookahead Adder (CLA) – Example

- Let's apply this to a 4-bit adder to make it clear.

$$
\begin{aligned}
c_{i+1} &= G_i + P_i.c_i \\
s_i &= P_i \oplus c_i
\end{aligned}
$$

where

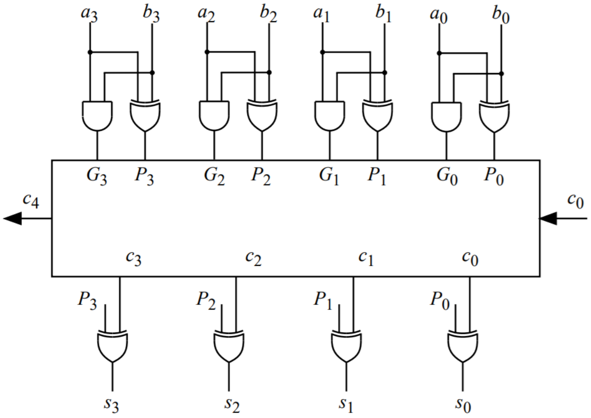
$$
\begin{aligned}
G_i &= a_i.b_i \\
P_i &= a_i \oplus b_i
\end{aligned}
$$

- Putting i=0,1,2,3 in carry equation

$$
\begin{aligned}
c_1 &= G_0 + P_0.c_0 \\
c_2 &= G_1 + P_1.G_0 + P_1.P_0.c_0 \\
c_3 &= G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0 \\
c_4 &= G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0
\end{aligned}
$$

  - Notice that the carry-out bit, c_{i+1}, of the last stage will be available after four delays: two gate delays to calculate the propagate signals and two delays as a result of the gates required to implement c_4

Carry Lookahead Adder (CLA)



Carry Lookahead Adder (CLA)

- The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits.
- For this reason, CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bit

Constructing Hardware to Match Definition

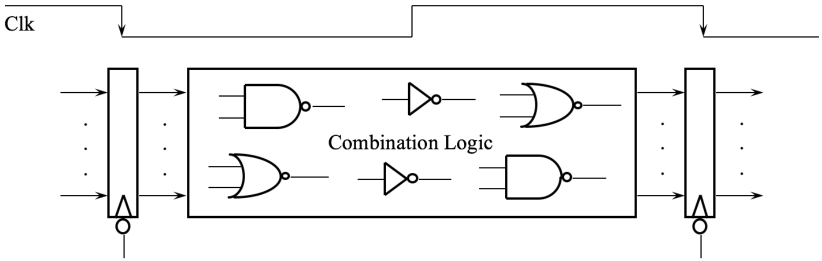
- Given any table of binary inputs for a binary output, programs can automatically connect a minimal number of gates to produce the desired function
- Such programs called “logic synthesis” tools, part of a general class of tools called “Computer Aided Design”, or CAD

Clock Signal

- A clock signal is a particular type of signal that oscillates between a high and a low state.
- The signal acts like a metronome, which the digital circuit follows in time to coordinate its sequence of actions.
- Digital circuits rely on clock signals to know when and how to execute the functions that are programmed.
- The function of clock in a design is like the heart and clock signals are the heartbeats that keep the system in motion.
- The clock signal is produced by a clock generator or Crystal Oscillator.

To Finalize the ALU (and of any circuit)...

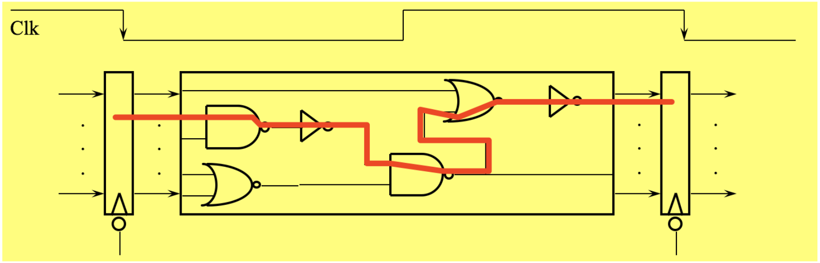
- We need to consider timing effects:



- All storage elements are clocked by the same clock edge
- The combination logic block's:
  - Inputs are updated at each clock tick
  - All outputs MUST be stable before the next clock tick

Pay Attention...

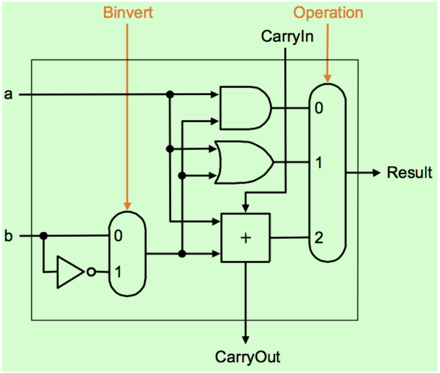
- We need to consider timing effects:



- Critical path: Each logic gate has a propagation delay. This is the delay between when the inputs are applied and the correct logical output is available.

What About Subtraction (a–b)?

- Two's complement approach: just negate b and add.
- How do we negate?
- A very clever solution:



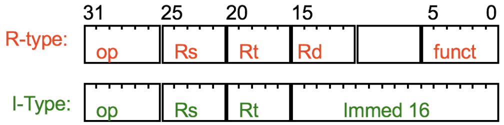
Tailoring the ALU to the MIPS

- Need to support the set-on-less-than instruction (slt)
  - Remember: slt is an arithmetic instruction
  - produces a 1 if rs $<$ rt and 0 otherwise
  - use subtraction: (a-b) $<$ 0 implies a $<$ b
- Need to support test for equality (beq \$t5, \$t6, \$t7)
  - use subtraction: (a-b) $=$ 0 implies a $=$ b

Requirements for the ALU

- Add, AddU, Sub, SubU, AddI, AddIU
  - 2's complement adder/sub with overflow detection
- And, Or, AndI, OrI, Xor, Xori, Nor
  - Logical AND, logical OR, XOR, nor
- SLTI, SLTIU (set less than)
  - 2's complement adder with inverter, check sign bit of result

Format of the Instructions



Type	op	funct
ADD	00	40
ADDU	00	41
SUB	00	42
SUBU	00	43
AND	00	44
OR	00	45
XOR	00	46
NOR	00	47

Type	op	funct
	00	50
	00	51
SLT	00	52
SLTU	00	53

Type	op	funct
ADDI	10	xx
ADDIU	11	xx
SLTI	12	xx
SLTIU	13	xx
ANDI	14	xx
ORI	15	xx
XORI	16	xx
LUI	17	xx

Summary

- Today we discussed about decoders and multiplexors and basic construction of ALU. We also talked about addition and subtraction and logical operations. More details are required to tailor it with MIPS. It will be discussed in the coming lectures.

Acknowledgements

- This lecture was created and maintained by Muhammad Fahim, Giancarlo Succi and Alexander Tormasov