**CHAPTER 6**

**CRYPTO-ARITHMETIC MAPPING PROBLEMS**

**Version July 28, 2010**

**Marek Perkowski, Alan Cheng, Yaswanath Naraharisetti and Gunawant Chaudhari**

**6.1. CRYPTO-ARITHMETIC MAPPING PROBLEMS AS EXAMPLES OF CONSTRAINT SATISFACTION PROBLEMS**

The general class of Constraints Satisfaction Problems (CSP) has many applications in computer science, physics, engineering, astronomy, biology and other areas. CSP’s are the baseline of intensive work in Artificial Intelligence and Operations Research. The problem is formulated by a set of constraints and a cost function. Some of the examples will be the cryptographic puzzles, graph coloring or image matching. The components of specialized processors to solve such problems are: the decision oracle, the optimization oracle, and the generator of binary vectors for exercising the oracle. The decision oracle decides whether the final outcome is TRUE if it is a logical ‘1’ or FALSE if it is a logical ‘0’. The optimization oracle finds a solution with the minimum value of cost function. The oracles use logical, arithmetical and relational blocks in order to determine the final decision. The final decision obtained will be the AND of all output bits of the sub oracles.

The Constraints Satisfaction Problem is formulated as a graph G = <NO, ED> with NO being a set of nodes and ED ⊂ NO × NO × … NO being a set of constraints. Constraints can be on any subset of nodes from NO. The nodes can have values, these values may be symbolic or numeric, represented as binary vectors for hardware realization. Any node Ni can have some set of values V(Ni). The simplest constraints are represented as edges, which are pairs, subsets of Cartesian Product NO × NO. The constraints can be of any type, for instance EQUAL (N1, N3), NOT-EQUAL (N2, N5), SMALLER-THAN (N3, N0). Each constraint is verified by certain logic block.

There are two formulations for oracles: ***Constraints\_Only*** and ***Constraints\_And\_Cost\_Function***.

**Problem 6.1.** Given is Graph G of constraints. Find such assignment of values to variables that all constraints are satisfied.

**Problem 6.2.** Given is Graph G of constraints. Given is cost function defined on G with integer of real values. Find such assignment of values to variables that all constraints are satisfied and the cost function is maximized (or minimized, depending on a problem).

Many Constraint Satisfaction problems can be reduced to a set of logic equations and next to a single equation. This idea of formally reducing logic problems to sets of simple problems, solving simple problems, and combining their results comes from Raymon Lullus (Ramon Llull), a Spanish monk who lived in Thirteen Century. The powerful idea of solving arithmetic equations was next generalized and formalized by famous philosopher Rene Descartes and finally applied to “Boolean data” by George Boole when the algebra of logic was finally completely formalized. Finally, the logic and arithmetic equations can be solved concurrently which leads to modern ideas of constraints satisfaction problems. The operators in these equations can be of many types such as: arithmetic (+,\*,/,-, etc), relational (predicates < , >, = , ≤ , ≠ , ≥ , etc) and logic (AND, OR, EXOR, etc). The cryptographic puzzles belong to this category of problems.

The particular class of Constraint satisfaction problems, which is also known as Verbal Arithmetic, is a type of mathematical game consisting of a mathematical equation among unknown numbers, whose digits are represented by letters. The first Constraint Satisfiability problem used in this chapter is the “SEND + MORE = MONEY” Problem, which is explained more elaborately in sections 6.2 – 6.4. Section 6.5 discusses another problem of this type, solved by a different type of oracle. Some other similar applications of Constraint Satisfaction problems are discussed in section 6.6. These types of problems are usually solved using Logic Matrices [Bizam Herczeg, Lauriere, other ref] and are excellent examples to explain the essence of using constraint satisfaction approaches to solve problems.

**6.2. CONSTRAINTS SATISFACTION PROBLEMS THAT ARE ALSO EQUATIONAL LOGIC PROBLEMS.**

**6.2.1. GENERAL ORACLE FOR “SEND+MORE+MONEY” PROBLEM THAT DOES NOT USE KNOWLEDGE IN ANY WAY**

In this section we will present a general approach to create oracles for cryptographic problems without using specific knowledge.

SEND + MORE = MONEY is the famous cryptographic puzzle—see Figure 6.2.1. The letters should be replaced with unique digits 0 ,…., 9 to make the equation valid. Directly from Figure 6.2.1 one can compile the Equation from Figure 6.2.2.

S E N D

+ M O R E

M O N E Y

*Figure 6.2.1: Cryptographic problem example. Substitute digits for letters to make the equation to be true.*

D + E = 10 C1 + Y C1  { 0, 1 }

N + R + C1 = 10 C2 + E C2  { 0, 1 }

E + O + C2 = 10 C3 + N C3 { 0, 1 }

S + M + C3 = 10 C4 + O C4 { 0, 1 }

C4 = M

*Figure 6.2.2: Equations compiled from the problem formulation from Figure 6.2.1.*

The specification of nodes is given in Figure 6.2.3. Observe that the carries Ci are binary single-bit signals but all letters require four bits in binary encoding, as shown in Figure 6.2.3. This Figure explains also that only some 4-bit strings are allowed for codes of letters, namely the strings 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001.

S { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

E{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

N { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

D { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

M { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

O { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

R { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Y { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

*Figure 6.2.3: Constraints for nodes in the graph. Each node is a 4-bit string. Because we need 4 bits to represent letters S, E, ..., Y, we need additional constraint to restrict the domain to digits.*

S ≠ E, S ≠ N, S ≠ D, S ≠ M etc.

*Figure 6.2.4: Inequalities for unique encoding of nodes of the graph. One inequality is created for every pair of letters.*

The equations from Figure 6.2.2 correspond directly to the rules of arithmetic addition with carry. Carry signals are denoted by C1, C2 , C3  and C4 ; Ci{ 0 , 1 }. The equations in Figure 6.2.3 state that each symbol S, E,…..Y is a digit from 0 to 9. The equations in Figure 6.2.4 mean that all mappings of symbols S, E, ….Y to digits are unique, i.e. that they are one-to-one mappings. These are all typical equations that lead to typical arithmetic, predicate, logic circuits for a wide class of problems.

C4 = M C4 { 0, 1 }

M = 1

S + M + C3 = 10 C4 + O

S + M + C3 = 10 M + O

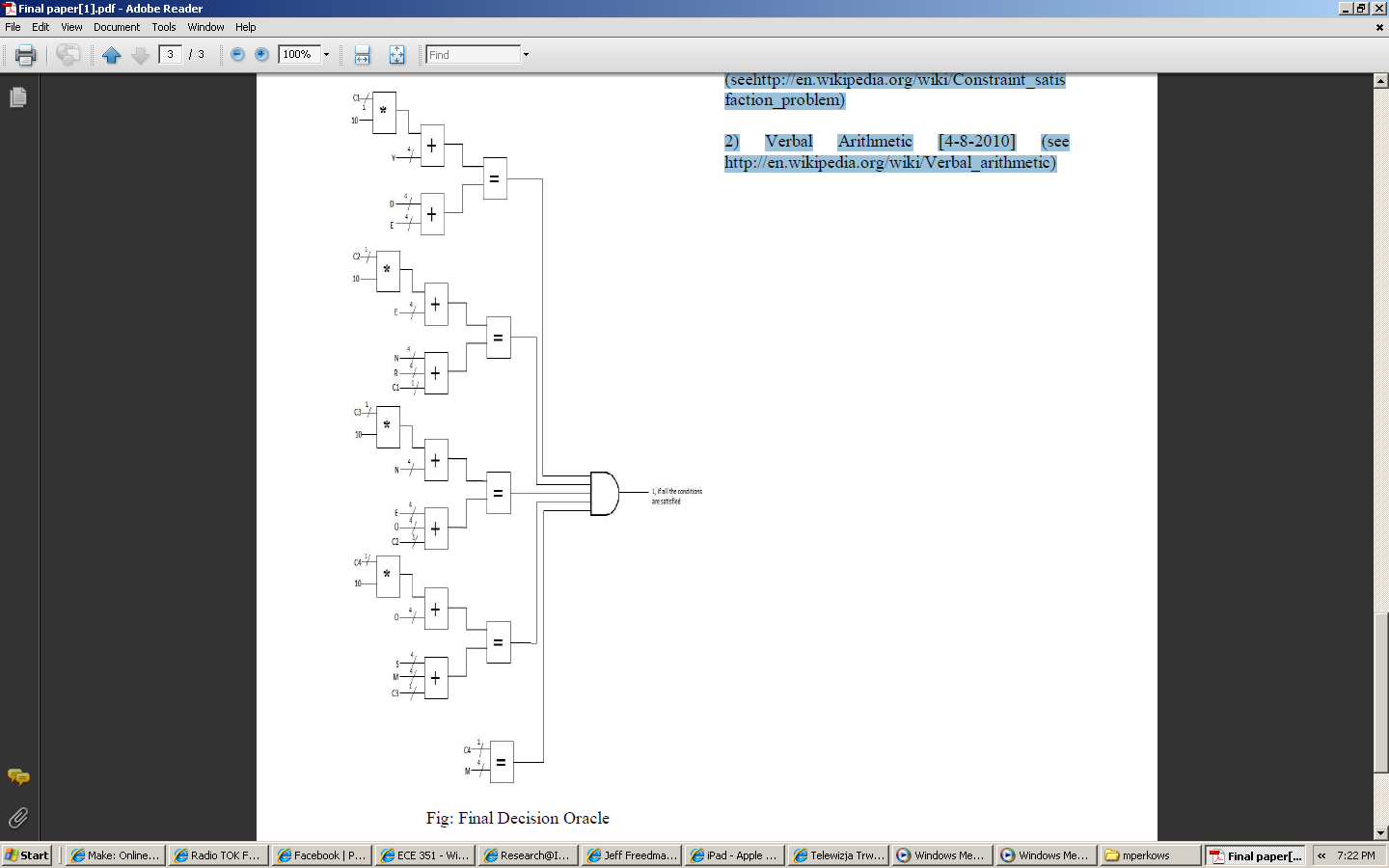
S + C3 = 9 M + O

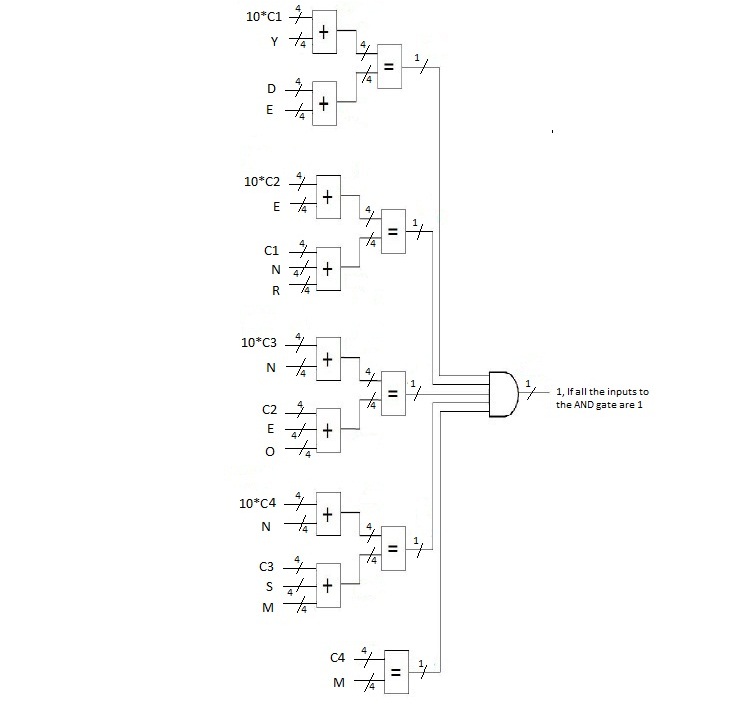
S + C3 = 9 + O

*Figure 6.2.5: Simplified Equations compiled from Figure 6.2.2.*

Equations from Figure 6.2.2 can be solved by exhaustive search that assigns binary values or strings of binary values to variables in all possible ways. With many binary variables this search can take too much time even with hardware accelerator, so we can limit the search by using some kind of knowledge. Remember that reducing the search from time complexity 2k+1 to time complexity 2k may mean reducingyour clock time from 8 hours to 4 hours, so every use of knowledge to reduce the number of binary problem-specifying variables is very important. This will be illustrated in this chapter on practical examples.

Using knowledge, equations from Figure 6.2.2 can be simplified to the form from Figure 6.2.5. This would simplify the oracle and speed-up the exhaustive search algorithm but we will not discuss this “intelligent preprocessing” variant in this section yet.





*Figure 6.2.6: Graph of constraints for the SEND+MORE=MONEY problem. We create signal* ***all\_equations\_OK*** *at output of the global AND gate.*

Figure 6.2.6 presents the part of the oracle to verify the equations from Figure 6.2.2. The operations of addition, multiplication and equality checking are replaced by logic blocks. Every symbolic variable S, E, ….Y is encoded using 4 bits, i.e. with 4 binary (decision) variables. The global AND has output 1 when all equations from Figure 6.2.2 are satisfied. The output of this AND gate is denoted by *all-equations-ok*. As one can see in Figure 6.2.6, only the following blocks are used: arithmetic adder with 2 inputs, arithmetic adder with 3 inputs, multiplier by 1 and 0, equality comparator and the AND gate.

*Figure 6.2.7: (a) Enumeration of cells in the M-map, (b) Groups of true minterms in the KMap for the circuit to check each equation from Figure 6.2.3. We generate here signal* ***good\_number*** *= GN = b1’+b2’ b3’ Extend Group b2’ b3’ ERROR!!. ALAN*

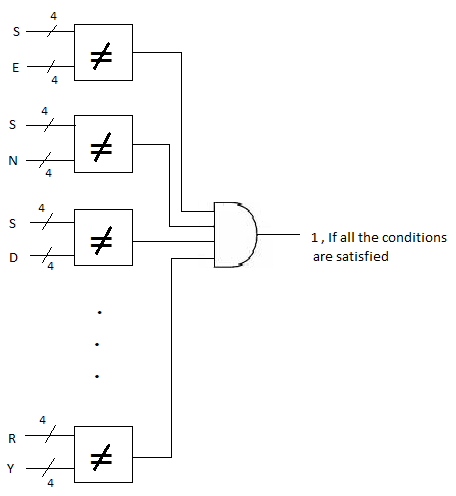
*Figure 6.2.8: Realization of circuit GN that checks if an argument is a binary-encoded digit, i.e. that checks if the binary argument is a Good Number, i.e., a digit 0,…, 9. DRAW CLASSIC LOGIC NOTATION. For b1’+b2’b3’ = NAND (b1, OR(b2,b3)). ALAN*

Figure 6.2.7a presents the method to calculate the circuit to verify that the argument b1b2b3b4 is a binary encoding of a digit 0 ,……, 9. Figure 6.2.7a presents the numbers of cells of KMap — all cells with values 0 ,…., 9 have output 1. This leads to the KMap will loops  and  from Figure 6.2.7b and finally to the circuit from Figure 6.2.8 being a part of the oracle GN. When the binary input combination b1b2b3b4 corresponds to a digit 0 ,., 9 then the output *good\_number* = 1.



*Figure 6.2.9: The remaining part of the oracle All-Good-Number for the SEND+MORE=MONEY problem. This checks the encoding of each symbol S, E, …, Y. It has 8 GN blocks from Figure 6.2.8 and the global AND gate.*

The part of the oracle that checks all numbers used in equations from Figure 6.2.2 is shown in Figure 6.2.9. GN is the block from Figure 6.2.8. Each such block uses only 3 qubits out of 4 qubits encoding every symbol S, E, ….., Y. This is marked with symbol “3” in vertical buses on inputs to each block GN in Figure 6.2.9. The output of this sub-oracle is denoted by all-good-numbers. All equations from Figure 6.2.3 are verified in the sub-oracle from Figure 6.2.10. The AND gate produces the signal all-different = 1 meaning that the mapping is a one-to-one mapping. The circuits from Figure 6.2.9 and Figure 6.2.10 are typical for many oracles for extended logic equations.

1. 



*Figure 6.2.10: (a) The part of an oracle All-Different for the SEND+MORE=MONEY problem that checks if the mapping is a one-to-one mapping, (b) systematic method to create all pairs of symbols for pair wise comparisons.*

Our assumption here is that the values (S, E, N, D, M, O, R, and Y) should be different one from another. Traditionally each letter represents a different digit. The values obtained for each of them should be different from each other, as presented in Figure 6.2.10c.

S≠E, S≠N, S≠D, S≠M, S≠O, S≠R, S≠Y

E≠N, E≠D, E≠M, E≠O, E≠R, E≠Y

N≠D, N≠M, N≠O, N≠R, N≠Y

D≠M, D≠O, D≠R, D≠Y

M≠O, M≠R, M≠Y

O≠R, O≠Y

R≠Y

Figure 6.2.10c: Equations to be tested in Oracle for Inequality

Finally, Figure 6.2.11 shows the entire oracle for the SEND + MORE = MONEY problem that is composed of 3 oracles. The final global AND is the logic AND (conjunction) of answers from the partial oracles:



We just need a single AND gate to realize this final global decision signal, see at the bottom of Figure 6.2.11.



*Figure 6.2.11: The complete oracle for the SEND+MORE=MONEY problem. The output is the “solution” bit at the bottom. Rewrite to standard notation. CHANGE THE TOFFOLI GATE TO AND GATE. ALAN*

The Figure 6.2.11 is the block diagram for the complete decision oracle. Each block in the diagram shows the functioning of each of digits in the puzzle. The outputs of the adders are compared using a comparator which yields a single bit as the output. Thereafter the outputs of all the comparators are given to AND gate. This AND gate acts as the decision for the oracle. If the output of the AND gate is 1, then it shows that the input sequence is given correctly and the decision is satisfied. If the output is 0 it shows that input values are not given correctly and the decision is not satisfied.

* + 1. **USING KNOWLEDGE TO REDUCE SEARCH BY CALCULATING THE VALUES OF THE PARAMETERS FOR “SEND+MORE+MONEY” PROBLEM.**

In this section we will show how knowledge can be used to reduce search. Again, we discuss the same problem:

S E N D

+ M O R E

-----------------

M O N E Y

-----------------

From the above symbolic equation, the M value is obtained by adding the digits at the thousands place. According to our assumption, when two four-bit numbers are added the result will be either 0 or1. Here using standard notation for addition there is no possibility to have 0 as the output. So, the value of M will be 1.

S E N D

+ 1 O R E

-----------------

1 O N E Y

-----------------

The only possible value for S will be 9 since we are getting the carry as 1 and the one of the addends is 1. So, the value of S will automatically be considered as 9. In the same way the values of O will become 0 as 9 and 1 are added to form sum as 0 and the carry as 1. At this point the values at the thousands place are calculated.

9 E N D

+ 1 O R E

-----------------

1 0 N E Y

-----------------

Now look at the hundreds place. If there is no carry from the tens place, E and N would be the same because E+0 = N, but E and N can't be the same, so there must be a carry from the tens place. Now we have:

1 1 <-- carry

9 E N D

+ 1 0 R E

-----------------

1 0 N E Y

-----------------

Now the equation for the hundreds place is 1+E+0 = N or just 1+E = N. In the tens place we can have N+R = E+10 if there is no carry from the ones place, or we can have 1+N+R = E+10 if there is.

First test: no carry from the ones place:

N+R = E+10 and 1+E = N

(1+E)+R = E+10

1+R = 10

R = 10-1

R = 9

But S = 9, so R ≠ 9. That means there is a carry from the ones place and we get:

1+N+R = E+10 and 1+E = N

1+ (1+E) +R = E+10

2+R = 10

R = 10-2

R = 8

So now we have:

1 1 1 <-- carry

9 E N D

+ 1 0 8 E

--------------

1 0 N E Y

---------------

N cannot be 0 or 1 because 0 and 1 are taken. N cannot be 2 because 1+2+8 = 11 and then E would equal 1, but it cannot equal 1 because 1 is taken. N could equal 3,4,5,6 or 7 but it cannot equal 8 or 9 because 8 and 9 are taken (and E must be 2,3,4,5 or 6 because it is 1 smaller than N).

If E were 2, then for the ones place to carry D would have to be 8 or 9, and both 8 and 9 are taken, so E cannot be 2 (and N cannot be 3). If E were 3, then for the ones place to carry D would have to be 7, 8, or 9, but D cannot be 7 because then Y would be 0, which is taken, so E cannot be 3 (and N cannot be 4). If E were 4, then for the ones place to carry D would have to be 6, 7, 8, or 9. D cannot be 6 because Y would be 0, but D cannot be 7 because Y would be 1, and 0 and 1 are both taken, so E cannot be 4 (and N cannot be 5).

The only two possibilities for E now are 5 and 6. If E were 6, then N would be 7 and D would have to be 4 (which would make Y = 0), 5 (which would make Y = 1), 6 (which is taken by E), or 7 (which is taken by N). There are no solutions for E = 6, so E must be 5.

So now we have:

1 1 1 <-- carry

9 5 6 D

+ 1 0 8 5

---------------

1 0 6 5 Y

---------------

The only possible values for D will be 2,3,4,7. It should be more than 5 in order to yield a carry and the only possible value for D will be 7 and thereafter the value of Y becomes 2. The search has thus been reduced by using a reasoning. In this simple case the reasoning was based on knowledge of arithmetic and notation. In other CSP problems a much more sophisticated knowledge can be used, for instance knowledge of problem symmetries. An efficient search algorithm should use as much knowledge as possible.

* + 1. **USING VHDL FOR GENERAL ORACLE**
       1. **DESIGN CODE**

Structural Design flow is used to design the VHDL code for the functioning of the decision oracle. This model is perhaps the simplest of all the design flows because of declarations used and the level of complexity. Structural Design is widely used programming style to design a digital system. All the parts of the main decision oracle are declared individually and later subparts are called using the components

-----------------Oracle-------------------

------------------------------------------

-------Declaration for 2-input adder------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity add2 is

port(a,b : in std\_logic\_vector(3 downto 0);

y : out std\_logic\_vector(3 downto 0));

end add2;

architecture dataflow\_add2 of add2 is

begin

y <= a+b;

end dataflow\_add2;

-------------------------------------------

-------Declaration for 3-input adder-------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

entity add3 is

port(a,b,c : in std\_logic\_vector (3 downto 0);

y : out std\_logic\_vector (3 downto 0));

end add3;

architecture dataflow\_add3 of add3 is

begin

y <= a+b+c;

end dataflow\_add3;

------------------------------------------

--------Declaration for comparator--------

library ieee;

use ieee.std\_logic\_1164.all;

entity comparator is

port(a,b : in std\_logic\_vector (3 downto 0);

y : out std\_logic);

end comparator;

architecture dataflow\_comparator of comparator is

begin

process(a,b)

begin

if (a(0)=b(0) and a(1)=b(1) and a(2)=b(2) and a(3)=b(3)) then

y <= '1';

else

y<= '0';

end if;

end process;

end dataflow\_comparator;

------------------------------------------

--------Declaration for AND-gate----------

library ieee;

use ieee.std\_logic\_1164.all;

entity and1 is

port(a,b,c,d,e : in std\_logic;

y : out std\_logic);

end and1;

architecture dataflow\_and1 of and1 is

begin

process(a,b,c,d,e)

begin

--if ((a='1')and(b='1')and(c='1')and(d='1')and(e='1')) then

--y <= '1';

-- else

-- y <= '0';

-- end if;

y<=a and b and c and d and e;

end process;

end dataflow\_and1;

------------------------------------------

---------Declaration for Oracle-----------

library ieee;

use ieee.std\_logic\_1164.all;

entity main\_oracle is

port (S,E,N,D,M,O,R,Y: in std\_logic\_vector(3 downto 0);

C1,C2,C3,C4,C110,C210,C310,C410 : in std\_logic\_vector(3 downto 0);

Oracle: out std\_logic) ;

end main\_oracle;

architecture behav of main\_oracle is

signal A1,A2,A3,A4,A5,A6,A7,A8 : std\_logic\_vector(3 downto 0);

signal A9,A10,A11,A12,A13 : std\_logic;

component add2

port(a,b : in std\_logic\_vector (3 downto 0);

y : out std\_logic\_vector (3 downto 0));

end component;

component add3

port(a,b,c : in std\_logic\_vector (3 downto 0);

y : out std\_logic\_vector (3 downto 0));

end component;

component comparator

port(a,b : in std\_logic\_vector (3 downto 0);

y : out std\_logic);

end component;

component and1

port(a,b,c,d,e : in std\_logic;

y : out std\_logic);

end component;

begin

n1: add2

port map(C110, Y, A1);

n2: add2

port map(D, E, A2);

n3: add2

port map(C210, E, A3);

n4: add2

port map(C310, N, A4);

n5: add2

port map(C410, O, A5);

n6: add3

port map(N, R, C1, A6);

n7: add3

port map(E, O, C2, A7);

n8: add3

port map(S, M, C3, A8);

n9: comparator

port map(A1, A2, A9);

n10: comparator

port map(A3, A6, A10);

n11: comparator

port map(A4, A7, A11);

n12: comparator

port map(A5, A8, A12);

n13: comparator

port map(C4, M, A13);

n14: and1

port map(A9, A10, A11, A12, A13, Oracle);

end behav;

**6.4.2. Test Bench**

**What was the size of search space 2k what was k?**

This test bench is designed to check the output to be ‘1’, where all the set of input values satisfies the conditions given in the design code. The input values are incremented by using counter which increases for every change in clock pulse

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

entity testbench is

port(clk : in std\_logic;

Oracle : out std\_logic);

end testbench;

architecture tb of testbench is

component main\_oracle is

port (S,E,N,D,M,O,R,Y: in std\_logic\_vector(3 downto 0);

C1,C2,C3,C4,C110,C210,C310,C410 : in std\_logic\_vector(3 downto 0);

Oracle: out std\_logic) ;

end component;

signal S,E,N,D,M,O,R,Y: std\_logic\_vector(3 downto 0);

signal C1,C2,C3,C4,C110,C210,C310,C410 : std\_logic\_vector(3 downto 0);

signal x: std\_logic\_vector (0 to 63):="1001010101100111000100001000000000000000000000000000000000000000";

signal z: std\_logic\_vector (0 to 63):="1001010101100111000100001000001000010001000000011010101000001010";

begin

u\_oracle : main\_oracle port map (S,E,N,D,M,O,R,Y,C1,C2,C3,C4,C110,C210,C310,C410,Oracle);

process(clk)

begin

x <= x + "1";

S <= x(0 to 3);

E <= x(4 to 7);

N <= x(8 to 11);

D <= x(12 to 15);

M <= x(16 to 19);

O <= x(20 to 23);

R <= x(24 to 27);

Y <= x(28 to 31);

C1 <= x(32 to 35);

C2 <= x(36 to 39);

C3 <= x(40 to 43);

C4 <= x(44 to 47);

C110 <= x(48 to 51);

C210 <= x(52 to 55);

C310 <= x(56 to 59);

C410 <= x(60 to 63);

end process;

end;

**Initialization of signal x is not explained, checking correctness of signals M, D, etc is not explained and conditions like S NOT equal Y are not explained. Naraharisetti, improve it.**

**6.4.3. Output waveform obtained on Modelsim**

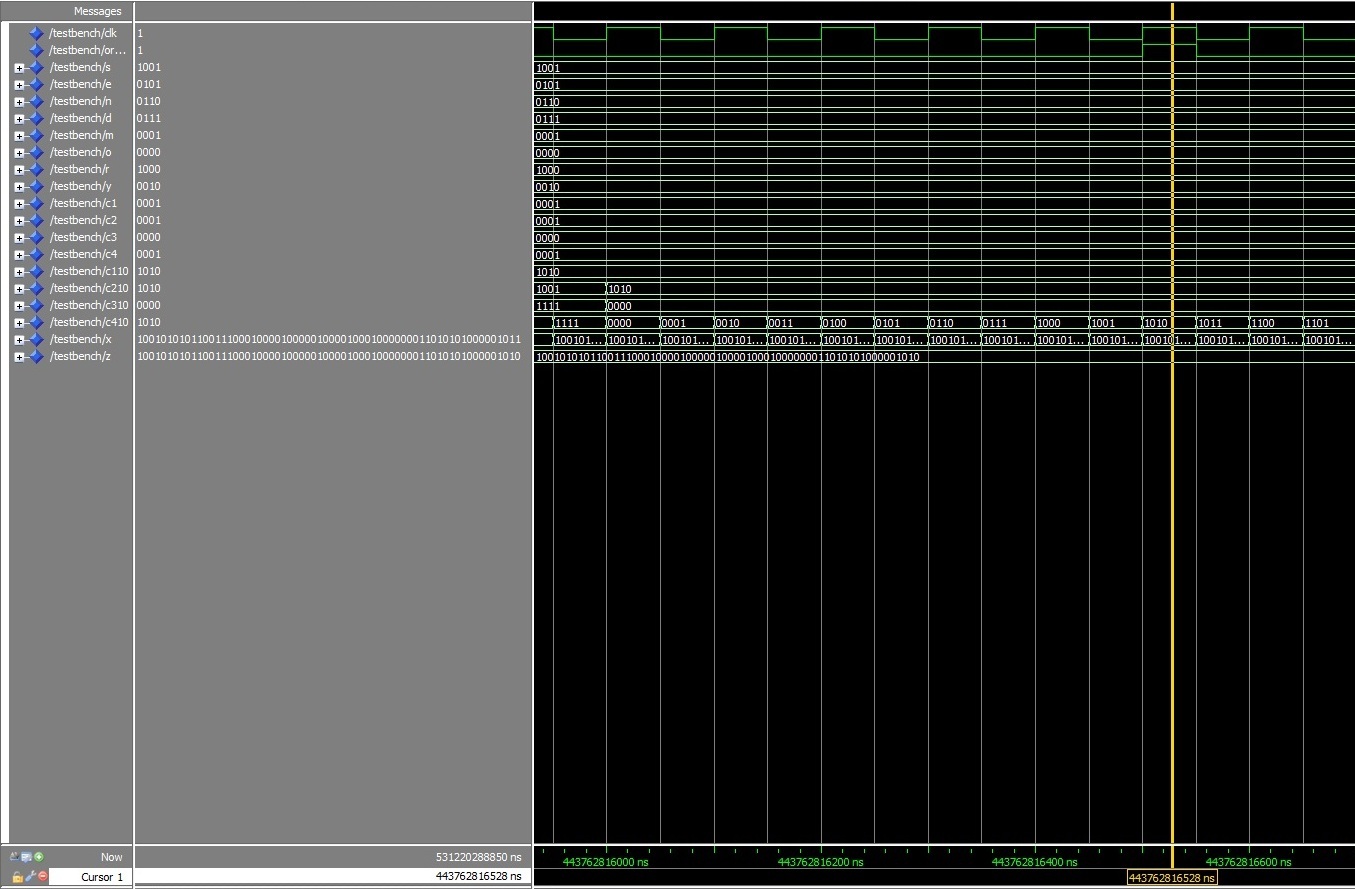
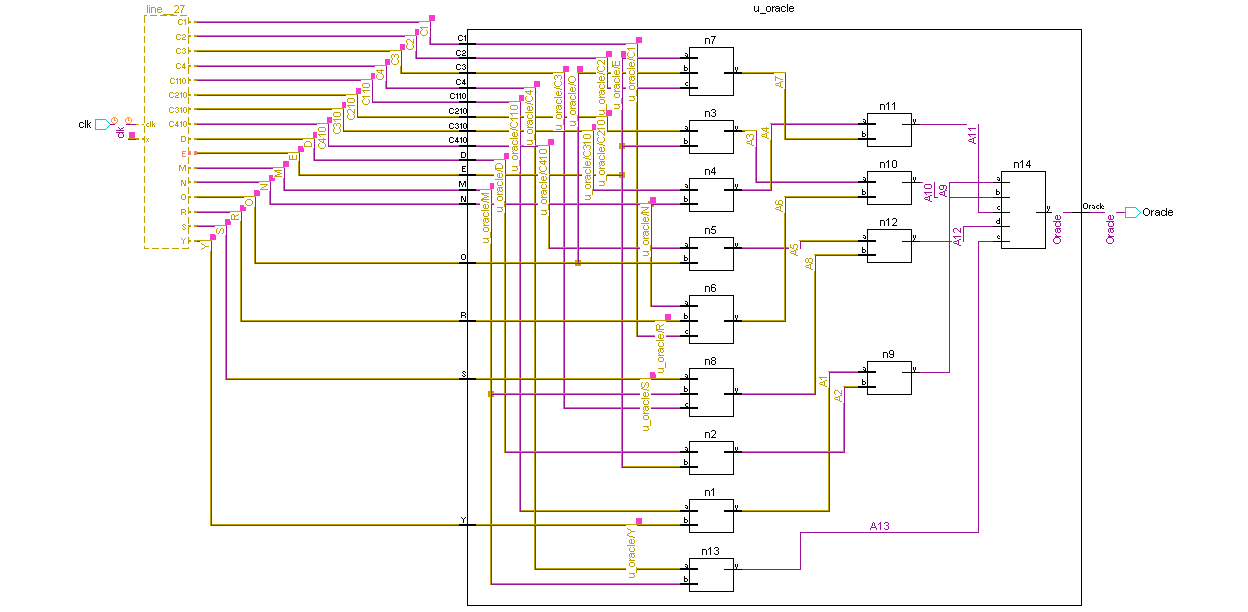


Fig: Output waveform when Oracle is ‘1’

**6.4.4. Schematic obtained in Veloce emulator**



*Fig: Schematic for the Oracle NOT EXPLAINED*

NARAHARISETTI:

Where is some file created as a low level logic gates synthesized by Veloce?  
Where are the results of simulation of this circuit?

What can we tell about timing?

**6.5. TWO + TWO = FOUR PROBLEM.**

In this section, we will take a different example, the TWO + TWO = FOUR cryptographic puzzle and create an oracle for it. Like the SEND + MORE = MONEY problem, each letter can only represent an unique value 0-9.

TWO

+ TWO

FOUR

*Figure 6.5.1: The cryptographic puzzle TWO + TWO = FOUR.*

C3C2C1

TWO

+ TWO

FOUR

*Figure 6.5.2: The problem formulated to include the carries.*

The problem can then be formulated to include the carries C1,C2, and C3. Thus, the equations for the oracle can be written.

O + O = 10C1 + R C1  { 0, 1 }

W + W = 10C2 + U C2  { 0, 1 }

T + T = 10C3 + O C3  { 0, 1 }

C3 = F

*Figure 6.5.3: The equation representation of Figure 6.5.2.*

As with the SEND + MORE = MONEY problem, each Ci is only one bit and each letter can only be 4-bit values 0000 through 1001. Since F has to be a non-zero value, 1 = C3 = F. The equation then could be rewritten to a simpler form. Also, O + O = 2(O), W + W=2(W), and T + T = 2(T).

2O = 10C1 + R C1  { 0, 1 }

2W = 10C2 + U C2  { 0, 1 }

2T = 10 + O C3 = 1

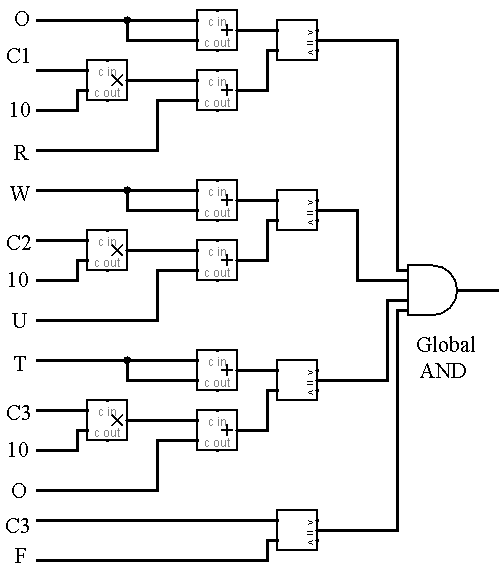
F = 1

*Figure 6.5.4: The simplified, rewritten equation from Figure 6.5.3.*

1

4

4



1

1

1

1

4

4

4

4

4

4

4

4

4

1

4

4

1

4

4

4

1

4

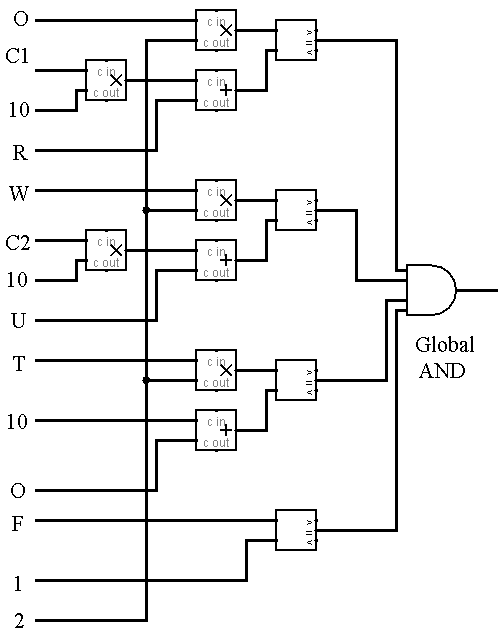
4

4

1

*Figure 6.5.5: The oracle circuit derived from Figure 6.5.3*

Figure 6.5.5 shows the part of the oracle that checks the input combinations. This circuit is derived from the equations of Figure 6.5.2. Multiplication is replaced by multipliers, addition is replaced by adders, and equality is replaced by equality comparators. All letters are replaced by 4-bit strings and all carries all replaced by 1-bit strings. The outputs of the comparators are 1-bit. The global AND gate becomes 1 if all equations are satisfied.



4

4

2

1

4

4

4

4

4

4

4

4

4

4

4

4

1

4

4

4

1

4

1

1

1

1

1

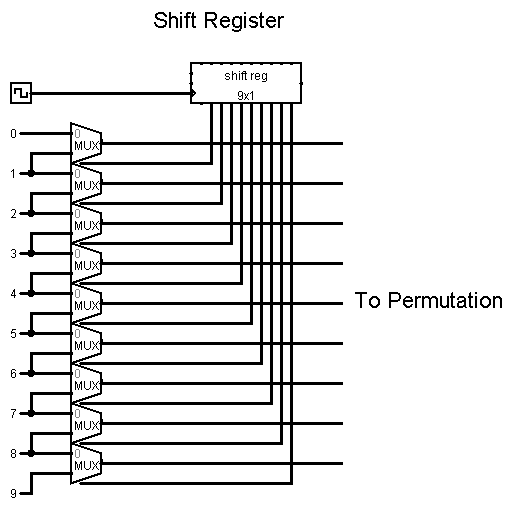
*Figure 6.5.6: The oracle circuit derived from Figure 6.5.3.*

The oracle circuit made from Figure 6.5.3 is very similar to the non-simplified equations. The only difference is that the only 2-bit value of constant 2 is presented. F is directly compared with constant value 1.

In this problem, there all three parts of the oracle, one that generates the values of each letters and permutes the values such that all of the possible combinations are cycled through, one that checks the inequality of the values, and one that checks the values and checks if they satisfy the equations (explained above).

In the part of the oracle that assigns the values, it has the pick 7 out 10 values, one for each letter. In order to generate the values of c­­i, we have to check if the value of 2(Letter value) > 9. Therefore, if there is a value 10+, the output is 1. It is 0 if the value of the product is 9 or less.

To design the circuit which will assign 7 of the 10 values to each letter, multiplexers are necessary in order to reduce three of the ten values. It is easier to first start out with a selector for 9 out of 10 values. Since nine values need to be selected, nine multiplexers are used.



*Figure: 6.5.7: 9 out of 10 selector.*

In order to make the 9 out of 10 selector work, the shift register must be preset to 111111111. Then zeros must be cycled through it.

111111111

011111111

001111111

000111111

000011111

000001111

000000111

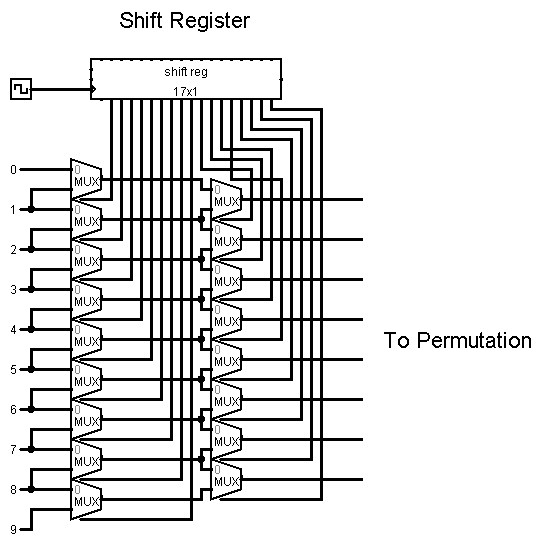
000000011

000000001

000000000

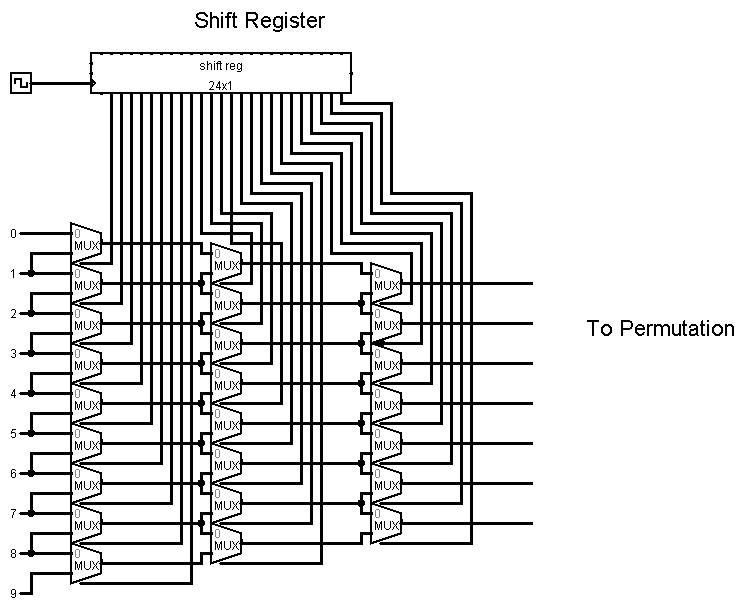
*Figure 6.5.8: Order that the selector is cycled through.*

Using the 9 out of 10 selector, the 8 out of 10 selector then can be built by adding multiplexers on top of the previous multiplexers. Thus adding 8 multiplexers to the original 9 would create an 8 out of 9 selector. However, since 9 is selecting out of 10, then it is an 8 out of 10 selector.



*Figure 6.5.9: 8 out of 10 selector.*

The 7 out of 10 selector can be made using the same process of adding to the previous selector.



*Figure 6.5.10: 7 out 10 selector.*

The final selector works in the same manner as the others and should not repeat any values unless of a glitch in the system.

After the selector, then the numbers have to be permuted in order to go through all possible combinations of the numbers selected. There are many different way in order to permute all possible combinations.

Show the whole system to solve this problem, exercising part together with oracle.

Calculate what is the gain by using permutations instead of all binary values. Write some comments about this approach which can often give a big advantage.

VERILOG CODE AND ITS DISCUSSION> ALAN

* 1. **VERBAL PUZZLES THAT CAN BE REDUCED TO LOGIC MATRICES**

**Alan, you can work on this when I will be on vacation and consult with my students about Verilog or VHDL and access to Veloce. Verilog!**

**6.7. CONCLUSIONS**

The decision oracle to implement the Constraint Satisfiability Problem is designed and implemented in Modelsim and Velocesolo emulator. The main CSP used in this project is SEND+MORE=MONEY. The VHDL code for decision oracle is designed using the structural flow and the final results obtained are accurate. The same was done for other Cryptographic Problems. The conclusion is that …. TIMING! COMPLEXITY?

**Literature**

1. Pseudo-Boolean Problems
2. Integer programming approaches
3. Book Bizam and Herczeg
4. Logic Matrices and their solving
5. Constraint Satisfiability Problem [4-8-2010] (seehttp://en.wikipedia.org/wiki/Constraint\_satisfaction\_problem)
6. Verbal Arithmetic [4-8-2010] (see <http://en.wikipedia.org/wiki/Verbal_arithmetic>)
7. Add more from what I gave you Alan.
8. Add new literature when you find something. I am also working on literature.