Implementation of A Rule-Based Expert System for Design Verification and Analysis

COM3410 - Artificial Intelligence Problem Solving
Project Report

Bulut F. Ersavas
bersavas@coe.neu.edu
March 5, 2003

Abstract
Today’s Computer-Aided Design (CAD) tools require sophisticated techniques to shorten the design cycle and ensure product quality and performance. As well as checking the errors in a design, a powerful CAD tool must be able to provide suggestions and warnings to the user. In this paper, I present a simplistic electronic design verification and analysis tool that employs a rule-based expert system.

1. Introduction

Most of the present electronic design automation (EDA) systems make use of artificial intelligence (AI) techniques including knowledge-based reasoning, neural networks, fuzzy logic, etc. These systems support design engineers at every stage of product development cycle to reduce the time and cost to develop the product with optimum performance. In this research, I will present one of the most powerful AI techniques, rule-based expert systems, and its application to the CAD & EDA field.

Modularity is a key feature for a CAD system that analyses and verifies a design using its expert knowledge. Addition of new information and modification of existing knowledge must be possible and be easily done by those who are not familiar with the implementation of the system. AI techniques for developing an expert system makes it possible to implement such a system.

In this project a ruled-based expert CAD program is developed using LISP programming language. The program has two stages, which separates the implementation of the knowledge base and rule checker or inference engine. The system is built using rules
2. Specifications

To simplify the development of the program and put the emphasis more on the application and implementation of rule-based expert systems, there are several restrictions and simplifications on the program. Most basically, the designs handled by the program are located on a fixed size design board. There are only metal wires (with a number of attributes) in the design. Actual devices and blocks (e.g. gates, transistors, resistors etc.) are not included. Detailed description of the design elements can be found in following subsections.

2.1. Design Board

Design board represents the medium where all electrical devices (e.g. transistors) and wires connecting them reside. The board is two-dimensional and limited to size 10 x 10. In other words, it includes 100 grid points. The wires will be placed on the grids of the board. A sample board configuration is given in Figure 1.
The wires will be located by their starting and ending points on the board. For example, the location of wire #1 on the board given in figure 1 is (1,1) (1,4) which corresponds to (x_{\text{start}} y_{\text{start}}) (x_{\text{end}} y_{\text{end}}).

### 2.2. Wires

The wires are the objects of consideration for the expert system. Each wire will be associated with several properties, which will be used by the rule checker and analyzer. The list of properties is given below.

Wires will have varying lengths but be limited to one direction. (i.e. zigzags are not allowed). Standard wire width is one grid point.

**Class**: Wires  
**Attributes**:  
- *Metal Level*: There are 4 levels of metal available. Wires on different metal levels, and passing thru the same grid point will not be shorted. (i.e. connected)  
- *Starting Point*: A lisp list that includes the starting grid point of the wire. (x_{1} y_{1})  
- *Ending Point*: A lisp list that includes the ending grid point of the wire. (x_{2} y_{2})  
- *Connectivity (Net Info)*: Electrical net name given to the wire. Wires with same net names should be connected.  
- *Type*: The type of the signal, which the wire carries. Examples: analog, digital, clock, power, ground  
- *Criticality*: Integer value in the range [1, 5] representing how critical the signal is.

Attribute table and their values for the board given in figure 1 is shown in table 1.
2.3. Operations

There is a set of operations, which will be used by the rule checker to test the rules in the knowledge base. Most obvious operations are those which return the values of attributes assigned to each wire. The major operations are briefly described below.

*(level wire1)*: This function returns the metal level of the given wire.

*(wtype wire1)*: This function returns the type of the given wire.

*(criticality wire1)*: This function returns the criticality value of the given wire.

*(net wire1)*: This function returns the connectivity (net name) of the given wire.

*(start_point / end_point wire1)*: This function returns the starting / ending point of the given wire.

*(wlength wire1)*: This function calculates the length of the given wire.

*(distance wire1 wire2)*: This function calculates the minimum distance between two wires. Result is returned as a real number.

*(connected wire1 wire2)*: This function checks whether two wires are connected or not. It returns *t* if they are connected, *nil* otherwise.

*(direction wire1)*: This function returns the direction of the given wire. (vertical or horizontal)

*(parallel wire1 wire2)*: This function checks whether two wires are parallel to each other. If they are, it returns *t*; *nil* otherwise.

---

**Table 1.** Wire properties for the design given in Figure 1.
3. Knowledge Base / Rule Base

The inference engine uses the rule set provided in the knowledge base. The rules are applied to the wires with different orientations and attributes. They are specified in a separate file and assigned to four different global variables (*assertions_err_individual*, *assertions_err_pair*, *assertions_warn_individual*, *assertions_warn_pair*) according to their type. The reason for using separate global variables is to maintain a better organized knowledge base and to avoid calling individual checks over and over again. The knowledge base file can easily be edited and extended by a user who is not familiar with the implementation of the system. The Lisp format of the rules is as follows:

```
(rule if <premise> then <conclusion>)

<premise> :=
(\and <rule-1> <rule-2> ... <rule-n>)
(\or <rule-1> <rule-2> ... <rule-n>)
(\not <rule-1> <rule-2> ... <rule-n>)
```

There are two types of rules that will be checked in the knowledge base: errors and warnings. These are described in following subsections.

3.1. Errors:

These represent the rules that have to be obeyed. User will get error messages if the wires violate this type of errors. These are basically the physical restrictions of the process used to manufacture the integrated circuit. Therefore, they have to be fixed by the designer before the design is sent to the foundry. These rules vary depending on the technology used. Hence, it is crucial that the system allows easy modification to this type of rules.

Error checks are divided into two groups depending on whether they are a comparison between two wires or a check on an individual wire. Both types are defined in the same file, but assigned to two different global variables: *assertions_pair*, and *assertions_individual* respectively.

Example of an Individual Assertion:

A wire might have a minimum length depending on its level. This rule is implemented as follows in the knowledge base:

```
(rule if (\and (> 2 (\wlength wire1)) (equal (\level wire1) "METALL1"))
then (\format t "ERROR (#E.I.1): Minimum length for METALL1 wire has to be 2 grid points. ~%")
```
Example of a Pair Assertion:

Spacing of wire metals is one of the most basic process rules. Depending on the level of metal, a minimum space between two separate nets should be kept to ensure isolation between them.

\[
\text{\texttt{(rule if \ (and \ (equal \ (level \ wire1) \ (level \ wire2)) \ (equal \ (level \ wire1) \ "METAL1") \ (> \ 2 \ (distance \ wire1 \ wire2))) \ (not \ (connected \ wire1 \ wire2))) \ then \ (format \ t \ "ERROR \ (#E.P.1): \ Spacing \ between \ two \ METAL1 \ wires \ has \ to \ be \ at \ least \ 2 \ grid \ points. \ ~\%")}}
\]

3.2. Warnings:

Warnings represent the recommendations to the user. Users don’t have to change their design for these warning messages. These rules are to support designers to build an optimum design. They may be very different from each other in nature and does not necessarily depend on the process. These rules are formed by the experience and knowledge of the field experts.

Just like the error rules, for better manageability, these recommendation rules are kept in two global variables depending on whether they are individual wire or pair checks. (i.e. *assertions_warn_individual*, *assertions_warn_pair*).

Examples:

It is usually not a good practice to run digital and analog signal nets parallel and very close to each other. This rule can be checked with the following rule:

\[
\text{\texttt{(rule if \ (or \ (and \ (parallel \ wire1 \ wire2) \ (equal \ (wtype \ wire1) \ "analog") \ (equal \ (wtype \ wire2) \ "digital") \ (> \ 4 \ (distance \ wire1 \ wire2))) \ (and \ (parallel \ wire1 \ wire2) \ (equal \ (wtype \ wire1) \ "digital") \ (equal \ (wtype \ wire2) \ "analog") \ (> \ 4 \ (distance \ wire1 \ wire2)))) \ then \ (format \ t \ "WARNING \ (#W.2): \ Analog \ and \ digital \ signals \ should \ not \ run \ parallel \ to \ each \ other \ with \ less \ than \ 4 \ grid \ point \ spacing.\ ~\%")}}
\]

Another example is as follows: Unless it is designer’s intention, two wires with different connectivity (i.e. different net names) should not be connected:

\[
\text{\texttt{(rule if \ (and \ (connected \ wire1 \ wire2) \ (not \ (equal \ (net \ wire1) \ (net \ wire2)))) \ then \ (format \ t \ "WARNING \ (#W.1): \ Wires \ with \ different \ connectivity \ are \ shorted. \ Disconnecting \ them \ will \ solve \ the \ problem.\ ~\%")}}
\]
4. Program Execution

The Lisp code is developed, compiled, and tested using Allegro\(^1\) Common Lisp environment, which is also used by the NEU Computer Science Department. The program has 3 files, which are listed as follows:

- expert_system.lisp
- knowledge_base.lisp
- configure_board.lisp

The main program is in “expert_system.lisp” file. For program to be able to run, there has to be a knowledge base, or a set of rules, which is provided in “knowledge_base.lisp” file. (This file is the one to be extended to provide more checks and recommendations.) Also, the design board should be represented in a way that the expert system can understand. Two different board configurations are generated in “configure_board.lisp” file.

The first configuration, which is given in figure 1, violates many rules. This board can be loaded and fed to the rule checker by running:

\[(\text{configureBoard})\]

The second board configuration does not violate any rule, therefore, the rule checker returns \textit{nil} which indicates that the design is clean. To use this board run:

\[(\text{configureBoard}_\text{Clean})\]

After running one of the above board configuration functions, the rule checker can be started by typing:

\[(\text{runcad n}i\text{l} ~*\text{wirelist}*\text{)}\]

Please note that when the knowledge base file is compiled, the rules are directly loaded into the corresponding \texttt{*assertions}_xxx* global variable. Therefore, there is no need to run any function for enabling the knowledge base.

The output corresponding to the design of figure 1 is given below.

\[> (\text{runcad n}i\text{l} ~*\text{wirelist}*\text{)}\]
Rule Check Between: 9 & 9 -->
WARNING (#W.I.2): It is recommended to route "power" and "ground" signals on METAL3 or METAL4.
Rule Check Between: 9 & 2 -->
WARNING (#W.P.1): Wires with different connectivity are shorted. Disconnecting them will solve the problem.
Rule Check Between: 8 & 5 -->

\(^1\) The trial version of the tool can be downloaded from \url{http://www.franz.com}
WARNING (#W.P.3): Analog and digital signals should not run parallel to each other with less than 4 grid point spacing.
Rule Check Between: 8 & 4 -->
WARNING (#W.P.1): Wires with different connectivity are shorted. Disconnecting them will solve the problem.
Rule Check Between: 8 & 2 -->
WARNING (#W.P.4): Clock signals should run perpendicular to critical signals.
Rule Check Between: 7 & 7 -->
ERROR (#E.I.4): Minimum length for METAL4 wire has to be 4 grid points.
Rule Check Between: 7 & 6 -->
WARNING (#W.P.2): Two wires with same connectivity are not connected. Connect them to get a more complete design.
Rule Check Between: 7 & 5 -->
ERROR (#E.P.4): Spacing between two METAL4 wires has to be at least 4 grid points.
Rule Check Between: 7 & 5 -->
WARNING (#W.P.3): Analog and digital signals should not run parallel to each other with less than 4 grid point spacing.
Rule Check Between: 7 & 2 -->
WARNING (#W.P.4): Clock signals should run perpendicular to critical signals.
Rule Check Between: 6 & 6 -->
ERROR (#E.I.1): Minimum length for METAL1 wire has to be 2 grid points.
Rule Check Between: 6 & 4 -->
WARNING (#W.P.3): Analog and digital signals should not run parallel to each other with less than 4 grid point spacing.
Rule Check Between: 6 & 1 -->
WARNING (#W.P.4): Clock signals should run perpendicular to critical signals.
Rule Check Between: 4 & 3 -->
ERROR (#E.P.2): Spacing between two METAL2 wires has to be at least 3 grid points.
Rule Check Between: 4 & 3 -->
WARNING (#W.P.2): Two wires with same connectivity are not connected. Connect them to get a more complete design.
Rule Check Between: 2 & 1 -->
WARNING (#W.P.2): Two wires with same connectivity are not connected. Connect them to get a more complete design.
Rule Check Between: 1 & 1 -->
WARNING (#W.I.1): It is recommended to route "clock" signals on METAL1 or METAL2.
NIL

5. Conclusions

This project showed how to implement a rule-based expert system for design verification and analysis. Like many areas, which require valuable expert knowledge, electronic design automation field extensively utilizes such systems for faster and better product design.

By nature of the project, the development has many limitations and restrictions. The scope of the program must be extended for more realistic usage. For example, together with the wires, the design should include the actual devices/blocks and rules corresponding to the relations between them. Also, the reporting must be improved to provide the coordinates of the violations and recommendations for easier corrections.
References


Appendix A
Lisp Source Code

; Structure for Representing Wires
(defstruct Wire
  (no 0) ; Unique wire identification number
  (level "METAL1") ; Metal Level of the wire
  (startPt '(0 0)) ; Starting Point of the wire (x1, y1)
  (endPt '(0 0)) ; Starting Point of the wire (x2, y2)
  (net nil) ; Connectivity Info
  (type nil) ; Type of the wire: analog, digital, clock, power, ground
  (criticality 1) ; Criticality of the signal: 1 least, 5 most critical
)

; Function to return the id number of wire
(defun no (wire)
  (Wire-no wire))

; Function to return metal level of the wire
(defun level (wire)
  (Wire-level wire))

; Function to return the starting point of the wire
(defun start_point (wire)
  (Wire-startPt wire))

; Function to return the ending point of the wire
(defun end_point (wire)
  (Wire-endPt wire))

; Function to return the net/connectivity of the wire
(defun net (wire)
  (Wire-net wire))

; Function to return the type of the wire
(defun wtype (wire)
  (Wire-type wire))

; Function to return the criticality integer of the wire
(defun criticality (wire)
  (Wire-criticality wire))

; Function to check whether two wires are parallel
; Returns t if they are, nil otherwise
(defun parallel (wire1 wire2)
  (cond
    ; First check whether they are both vertical
((and (equal (- (car (start_point wire1)) (car (end_point wire1))) 0) 
  (equal (- (car (start_point wire2)) (car (end_point wire2))) 0)) t) 
; Second check whether they are both horizontal 
((and (equal (- (cdr (start_point wire1)) (cdr (end_point wire1))) 0) 
  (equal (- (cdr (start_point wire2)) (cdr (end_point wire2))) 0)) t) 
; Otherwise, they are not parallel 
(t nil))

; Function that returns the direction of the given wire 
(defun direction (wire)
  (cond 
    ((equal (car (start_point wire)) (car (end_point wire))) 'vertical) 
    ((equal (cdr (start_point wire)) (cdr (end_point wire))) 'horizontal) 
    (t 'unknown) 
  )
)

; Function that calculated the length of the given wire 
(defun wlength (wire)
  ; Since they are either vertical or horizontal, no need for square root operation. 
  (+ (- (car (end_point wire)) (car (start_point wire))) 
      (- (cdr (end_point wire)) (cdr (start_point wire))))
)

; This function checks whether two wires are connected. 
(defun connected (wire1 wire2)
  (if (equal (level wire1) (level wire2)) 
      (if (equal (distance wire1 wire2) 0.0) t nil) 
      nil 
  )
)

; This function calculates and returns the minimum distance between two wires 
(defun distance (wire1 wire2)
  (let ((min 10) (x1 0) (y1 0) (x2 0) (y2 0) (dist 0) 
        (x1w1 (car (start_point wire1))) 
        (x2w1 (car (end_point wire1))) 
        (y1w1 (cdr (start_point wire1))) 
        (y2w1 (cdr (end_point wire1))) 
        (x1w2 (car (start_point wire2))) 
        (x2w2 (car (end_point wire2))) 
        (y1w2 (cdr (start_point wire2))) 
        (y2w2 (cdr (end_point wire2)))
      )
    (cond ; #1 
      ((equal (direction wire1) 'vertical) 
       (progn 
         (setf x1 x1w1) 
         (do ((y1 y1w1 (+ 1 y1))) 
             (> y1 y2w1)) 
           (cond ; #2 
             ((equal (direction wire2) 'vertical) 
              (progn 
                (setf x2 x1w2) 
                (do ((y2 y1w2 (+ 1 y2))) 
                    (> y2 y2w2)) 
                  (setf dist (sqrt (+ (\* (- x1 x2) (- x1 x2)) 
                                   (\* (- y1 y2) (- y1 y2)))))) 
           ) 
         ) 
       ) 
    )
)
((equal (direction wire2) 'horizontal)
(progn
  (setf y2 ylw2)
  (do ((x2 xlw2 (+ 1 x2)))
    (>(x2 x2w2))
    (setf dist (sqrt (+ (* (- x1 x2) (- x1 x2))
                        (* (- y1 y2) (- y1 y2))))))
  (if (< dist min) (setf min dist))
  )
  )
  )
)
; end progn
) ; end cond #2
) ; end do
) ; end progn
) ; end wire1 vertical

((equal (direction wire1) 'horizontal)
(progn
  (setf y1 ylw1)
  (do ((x1 xlw1 (+ 1 x1)))
    (> x1 x2w1))
  (cond
    ((equal (direction wire2) 'vertical)
      (progn
        (setf x2 xlw2)
        (do ((y2 ylw2 (+ 1 y2)))
          (> y2 y2w2))
        (setf dist (sqrt (+ (* (- x1 x2) (- x1 x2))
                            (* (- y1 y2) (- y1 y2))))))
        (if (< dist min) (setf min dist))
      )
    )
  )
  )
  )
  )
  )
  )
  )
  )
  )
  ; end do
  )
) ; end progn
) ; end wire1 horizontal
) ; end cond #1
) ; end let
) ; end defun distance

; Main function that starts the rule checker
(defun runCad (wire wirelist)
  (if (null wire)
    (if (not (null wirelist))
      (while (not (null wirelist))
        (setf wire (car wirelist))
        (setf wirelist (cdr wirelist))
        (checkRule wire wire "assertions_err_individual")
        (checkRule wire wire "assertions_warn_individual")
      )
    )
  )
(runCad wire wirelist)
)
)
(if (null wirelist) '()
(progn
  (format t "~-A ~A & ~A is ~A ~%" "Minimum Distance Between" (no wire) (no
(car wirelist)) (distance wire (car wirelist)))
  (checkRule wire (car wirelist) *assertions_err_pair*)
  (checkRule wire (car wirelist) *assertions_warn_pair*)
  (runCad wire (cdr wirelist))
  (format t "~-A ~A ~%" "Direction: " (direction (car wirelist)))
)
)

;Same rule format as the book uses.

;;; rule format is
;;; (rule if <premise> then <conclusion>)
(defun premise (rule) (nth 2 rule))
(defun conclusion (rule) (nth 4 rule))
(defun rulep (pattern)
  (and (listp pattern)
    (equal (nth 0 pattern) 'rule)))

;;; Rules of the form:
;;; (and <rule-1> <rule-2> ... <rule-n>)
;;; (or <rule-1> <rule-2> ... <rule-n>)
;;; (not <rule-1> <rule-2> ... <rule-n>)
(defun conjunctive-rule-p (rule)
  (and (listp rule)
    (equal (car rule) 'and)))
(defun disjunctive-rule-p (rule)
  (and (listp rule)
    (equal (car rule) 'or)))
(defun negation-rule-p (rule)
  (and (listp rule)
    (equal (car rule) 'not)))

;Function that returns the body of the premise of a rule.
(defun body (rule_premise) (cdr rule_premise))

;;; checkRule will take two wires check all three rules exist in the knowledgerbase
(defun checkRule (wire1 wire2 ruleList)
  (let ((rule (car ruleList)) (result nil))
    (while (not (null ruleList))
      (cond
        ((conjunctive-rule-p (premise rule))
          (setf result (check-conj-rules (body (premise rule)) wire1 wire2)))
        ((disjunctive-rule-p (premise rule))
          (setf result (check-disj-rules (body (premise rule)) wire1 wire2)))
        ((negation-rule-p (premise rule))
          (setf result (check-neg-rules (body (premise rule)) wire1 wire2)))
        (t (setf result (solve (premise rule) wire1 wire2)))))
   result))
(if result
  (progn
    (format t "Rule Check Between: ~A & ~A --> ~%" (no wire1) (no wire2))
    (solve (conclusion rule) wire1 wire2)))
  (setf ruleList (cdr ruleList))
  (setf rule (car ruleList))
)
)
);end let
);end defun checkRule

;This function returns the result of the conjunctive rule check.
(defun check-conj-rules (rules wire1 wire2)
  (if (null (cdr rules))
    (solve (car rules) wire1 wire2)
    (and (solve (car rules) wire1 wire2) (check-conj-rules (cdr rules) wire1 wire2)))
)

;This function returns the result of the disjunctive rule check.
(defun check-disj-rules (rules wire1 wire2)
  (if (null (cdr rules))
    (progn
      (solve (car rules) wire1 wire2)
    )
    (or (solve (car rules) wire1 wire2) (check-disj-rules (cdr rules) wire1 wire2)))
)

;This function returns the result of the conjunctive rule check.
(defun check-neg-rules (rules wire1 wire2)
  (not (solve (car rules) wire1 wire2))
)

(defun solve (rule wire1 wire2)
  ;; (format t "SOLVE: ~A ~%" rule)
  (let ((arg1 nil) (arg2 nil))
    (cond
      ((conjunctive-rule-p rule)
        (check-conj-rules (body rule) wire1 wire2))
      ((disjunctive-rule-p rule)
        (check-disj-rules (body rule) wire1 wire2))
      ((negation-rule-p rule)
        (check-neg-rules (body rule) wire1 wire2))
    )
    ;; length = 1
    (if (equal 1 (length rule))
      (funcall (car rule))
      ;; (format t "- ~A ~%" (car rule))
    )
    ;; length = 2
    (if (listp (cadr rule))
      (funcall (cadr rule) wire1 wire2)
    )
    (setf arg1 (solve (cadr rule) wire1 wire2))
    (setq arg1 (cadr rule)))
    (funcall (car rule) arg1 wire1 wire2)
    (funcall (car rule) arg1)
    (funcall (car rule) arg1))
    (format t "--- ~A ~A ~%" (car rule) arg1)
    (funcall (car rule) arg1))
  )
)
; length = 3
  (equal 3 (length rule))
  (if (listp (cadr rule))
    (setf arg1 (solve (cadr rule) wire1 wire2))
    (setf arg1 (cadr rule)))
  (if (equal arg1 'wire1)
    (setf arg1 wire1))
  (if (equal arg1 'wire2)
    (setf arg1 wire2))
  (if (listp (caddr rule))
    (setf arg2 (solve (caddr rule) wire1 wire2))
    (setf arg2 (caddr rule)))
  (if (equal arg2 'wire1)
    (setf arg2 wire1))
  (if (equal arg2 'wire2)
    (setf arg2 wire2))
  ; (format t "--- ~A ~A ~A-%" (car rule) arg1 arg2)
  (funcall (car rule) arg1 arg2)
)
(defun configureBoard ()
  (setq *wireList* '())
  (push (make-Wire :no 1 :level "METAL3" :startPt ' (1 1) :endPt ' (1 4) :net 'net2 :type "clock" :criticality 1) *wireList*)
  (push (make-Wire :no 2 :level "METAL1" :startPt ' (4 1) :endPt ' (8 1) :net 'net2 :type "clock" :criticality 1) *wireList*)
  (push (make-Wire :no 3 :level "METAL2" :startPt ' (3 3) :endPt ' (5 3) :net 'net1 :type "digital" :criticality 3) *wireList*)
  (push (make-Wire :no 4 :level "METAL2" :startPt ' (7 4) :endPt ' (7 8) :net 'net1 :type "digital" :criticality 3) *wireList*)
  (push (make-Wire :no 5 :level "METAL4" :startPt ' (1 6) :endPt ' (5 6) :net 'net3 :type "digital" :criticality 2) *wireList*)
  (push (make-Wire :no 6 :level "METAL1" :startPt ' (2 5) :endPt ' (6 5) :net 'net4 :type "digital" :criticality 1) *wireList*)
  (push (make-Wire :no 7 :level "METAL4" :startPt ' (0 9) :endPt ' (3 9) :net 'net4 :type "analog" :criticality 4) *wireList*)
  (push (make-Wire :no 8 :level "METAL2" :startPt ' (6 8) :endPt ' (9 8) :net 'net0 :type "analog" :criticality 5) *wireList*)
  (push (make-Wire :no 9 :level "METAL1" :startPt ' (8 1) :endPt ' (8 3) :net 'net5 :type "power" :criticality 1) *wireList*)
)

(defun configureBoard_Clean ()
  (setq *wireList* '())
  (push (make-Wire :no 1 :level "METAL1" :startPt ' (1 1) :endPt ' (1 5) :net 'net2 :type "clock" :criticality 1) *wireList*)
  (push (make-Wire :no 2 :level "METAL2" :startPt ' (8 1) :endPt ' (8 3) :net 'net1 :type "digital" :criticality 2) *wireList*)
  (push (make-Wire :no 3 :level "METAL4" :startPt ' (1 9) :endPt ' (5 9) :net 'net3 :type "analog" :criticality 1) *wireList*)
  (push (make-Wire :no 4 :level "METAL3" :startPt ' (2 5) :endPt ' (6 5) :net 'net4 :type "digital" :criticality 1) *wireList*)
  (push (make-Wire :no 5 :level "METAL1" :startPt ' (1 2) :endPt ' (4 2) :net 'net2 :type "clock" :criticality 1) *wireList*)
)
;; COM3410 - Artificial Intelligence ;;
;; Problem Solving Course Project ;;
;; Rule Based Expert System for ;;
;; Design Verification and Analysis ;;
;; Developed by Bulut F. Ersavas ;;
;; March 5, 2003 ;;
;; Knowledge Base ;;

; Knowledge Base - Individual Error Rules
(setq *assertions_err_individual* '(

 ;RULE #E.I.1
 (rule
   if (and (> 2 (wlength wire1)) (equal (level wire1) "METAL1") )
   then (format t "ERROR (#E.I.1): Minimum length for METAL1 wire has to be 2 grid points. ~\%")
 )

 ;RULE #E.I.2
 (rule
   if (and (> 2 (wlength wire1)) (equal (level wire1) "METAL2") )
   then (format t "ERROR (#E.I.2): Minimum length for METAL2 wire has to be 2 grid points. ~\%")
 )

 ;RULE #E.I.3
 (rule
   if (and (> 3 (wlength wire1)) (equal (level wire1) "METAL3") )
   then (format t "ERROR (#E.I.3): Minimum length for METAL3 wire has to be 3 grid points. ~\%")
 )

 ;RULE #E.I.4
 (rule
   if (and (> 4 (wlength wire1)) (equal (level wire1) "METAL4") )
   then (format t "ERROR (#E.I.4): Minimum length for METAL4 wire has to be 4 grid points. ~\%")
 )
))

; Knowledge Base - Pair Error Rules
(setq *assertions_err_pair* '(

 ;RULE #E.P.1
 (rule
   if (and (equal (level wire1) (level wire2)) (equal (level wire1) "METAL1") (> 2 (distance wire1 wire2)) (not (connected wire1 wire2)))
   then (format t "ERROR (#E.P.1): Spacing between two METAL1 wires has to be at least 2 grid points. ~\%")
 )

 ;RULE #E.P.2
 (rule
   if (and (equal (level wire1) (level wire2)) (equal (level wire1) "METAL2") (> 3 (distance wire1 wire2)) (not (connected wire1 wire2)))
   then (format t "ERROR (#E.P.2): Spacing between two METAL2 wires has to be at least 3 grid points. ~\%")
 )

 ;RULE #E.P.3
 (rule

))
if (and (equal (level wire1) (level wire2)) (equal (level wire1) "METAL3") (> 4 (distance wire1 wire2)) (not (connected wire1 wire2))) then (format t "ERROR (#E.P.3): Spacing between two METAL3 wires has to be at least 4 grid points. ~%")

;RULE #E.P.4
(rule
  if (and (equal (level wire1) (level wire2)) (equal (level wire1) "METAL4") (> 4 (distance wire1 wire2)) (not (connected wire1 wire2))) then (format t "ERROR (#E.P.4): Spacing between two METAL4 wires has to be at least 4 grid points. ~%")
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Knowledge Base - Individual Warning Rules
(setq *assertions_warn_individual* '(
  ;RULE #W.I.1
  (rule
    if (and (equal (wtype wire1) "clock") (or (equal (level wire1) "METAL3") (equal (level wire1) "METAL4"))) then (format t "WARNING (#W.I.1): It is recommended to route "clock" signals on METAL1 or METAL2. ~%")
    
  ;RULE #W.I.2
  (rule
    if (and (or (equal (wtype wire1) "power") (equal (wtype wire1) "ground")) (or (equal (level wire1) "METAL1") (equal (level wire1) "METAL2"))) then (format t "WARNING (#W.I.2): It is recommended to route "power" and "ground" signals on METAL3 or METAL4. ~%")
    
  ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Knowledge Base - Pair Warning Rules
(setq *assertions_warn_pair* '(
  ;RULE #W.P.1
  (rule
    if (and (connected wire1 wire2) (not (equal (net wire1) (net wire2)))) then (format t "WARNING (#W.P.1): Wires with different connectivity are shorted. Disconnecting them will solve the problem.-%")
    
  ;RULE #W.P.2
  (rule
    if (and (not (connected wire1 wire2)) (equal (net wire1) (net wire2))) then (format t "WARNING (#W.P.2): Two wires with same connectivity are not connected. Connect them to get a more complete design.-%")
    
  ;RULE #W.P.3
  (rule
    if (or
      (and (parallel wire1 wire2) (equal (wtype wire1) "analog") (equal (wtype wire2) "digital") (> 4 (distance wire1 wire2)))
  ))
(and (parallel wire1 wire2) (equal (wtype wire1) "digital") (equal (wtype wire2) "analog") (> 4 (distance wire1 wire2)))
then (format t "WARNING (#W.P.3): Analog and digital signals should not run parallel to each other with less than 4 grid point spacing.~")

;RULE #W.P.4
(rule
  if (or
    (and (parallel wire1 wire2) (equal (wtype wire1) "clock") (< 3 (criticality wire2)))
    (and (parallel wire1 wire2) (equal (wtype wire2) "clock") (< 3 (criticality wire1)))
  )
then (format t "WARNING (#W.P.4): Clock signals should run perpendicular to critical signals.~")
)