O. Bouketir
N. Mariun
I. Bin Aris
S. M. Bashi
S. Taib

J. Electrical Systems 1-2 (2005): 35-62

Journal of Electrical Systems

Regular paper

A Learning Aid Tool for Power Electronics Converters

It is known that power electronics and its related subjects are not easy to understand for students taking them for first time. This is due to nature of the subjects which involve many areas and disciplines. The introduction of general-purpose simulation package has helped the student a step further in understanding this subject. However, because of the generality of these tools and their drag-and-drop and ad-hoc features, the students still face problems in designing a converter circuit. In this paper, the problem above is addressed by introducing a learning aid tool that guides the student over prescribed steps to design a power electronics circuit. The tool is knowledge-based system where its knowledge base encompasses two types of knowledge; topologies and switching devices. The first step in the design procedure is the selection of the application of the desired circuit. Then few steps are to be followed to come out with the appropriate topology with the optimum switching devices and parameters. System structure, its different modules and the detailed design procedure are explained in this paper

Keywords: Class, converter, database, inference engine, knowledge base, OOP, switching device.

1. INTRODUCTION

Numerous research works have been conducted to come out with a CAD tool for power electronics converters. However, none of these works aimed to develop a fully-automated tool, nor claimed its complete success for the partial task assigned for (1-7). In the present tool; **p**ower **e**lectronic **d**esign **a**id **s**ystem (PEDAS) a different approach is introduced to overcome the deficiencies mentioned above in order to come out with a fully-automated tool specifically for designing power electronics converters. Though, the tool is interfaced with *Pspice* simulator, it establishes an interaction with the user starting from the selection of a specific application till arriving to the optimum topology with all parameter values and switches suggested. The topologies are stored in the knowledge base as schematic files, allowing the Schematic to be able to display the resulted circuit. This paper begins with the illustration of PEDAS general outlook and its graphical user interface (GUI). It states the various and attractive controls and tools used to build a smooth and flexible interaction

Corresponding author: O. Bouketir
Faculty Of Engineering (FOE)
Multimedia University
Malaysia

medium with the user. Then, the topologies knowledge base representation and implementation methods are detailed. This includes both types of this knowledge; type-based topologies and application-based topologies. The access paths to this knowledge and its manipulation procedures are explained when the inference engine module is elucidated. Instances of the explanation and help module are given. Lastly, the devices library module, its significance and its considerable features and functions are thoroughly described and demonstrated.

2. PEDAS LAYOUT (GUI)

The general layout or the system outlook or the graphical user interface (GUI) is of great importance. It gives the user the first impression about the tool. Hence, this outlook must be designed carefully and cautiously. Fortunately, the programming tool selected for the system development makes this task easy to accomplish. *Visual Basic* programming language, one of its famous features is the ability to provide pre-designed graphical controls (e.g. text boxes, command buttons, and list boxes), dialog boxes and flexible menu development tool. Each control has its own set of properties, methods, and events. This is to provide the user with a standard way to make selections, carry out commands, and perform input and output tasks. The programmer needs only to choose the appropriate controls and place them to the required position on his layout form and then write his own code of task the control to perform. Furthermore, *Visual Basic* –in its professional edition- is equipped by a mean that allows the programmer to create his own control to fit his specific needs. This mean is called *ActiveX technology* which is an extension to the *Visual Basic Toolbox*. The controls designed by using this technology can be added to the application even though they were developed by a different programmer in different locations. This is one of the features that make *Visual Basic* flexible and widely acceptable (Reference). Obviously, it's not possible to explore all the features and characteristics of *Visual Basic* that have been employed to build PEDAS GUI, but they will be imperceptibly revealed throughout this paper. Figure 1 shows some useful controls and menus that make up PEDAS layout.

3. SYSTEM'S KNOWLEDGE CODING

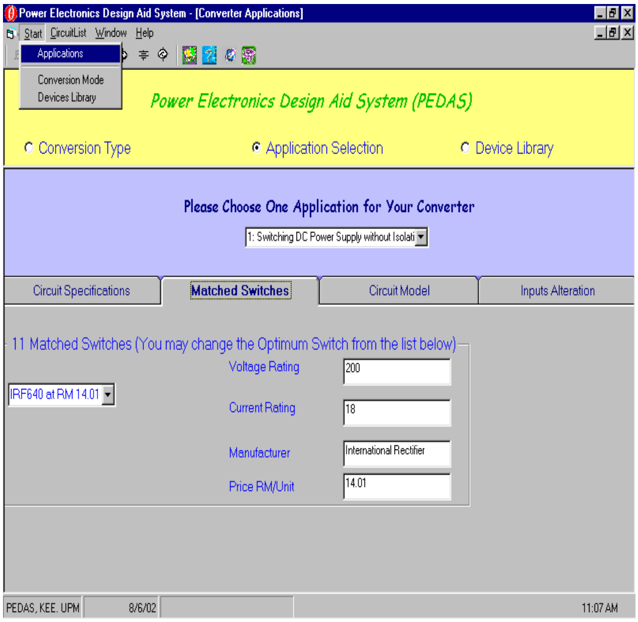
Using *Class Builder* utility offered in the *Visual Basic Add-ins* a total of sixteen (16) classes and subclasses (objects) were built. Fifteen (15) of them represent the topologies knowledge base. One class represent the interfacing module between *PEDAS* and *PSpice*. Among the first fifteen classes, fourteen (14) are application-based knowledge, while one class encompasses the type-based knowledge. The switching devices' knowledge is represented by a database object. The hierarchy of these classes is shown in Figure 2. It is worth to note that one can build his objects without the assistance of the *Class Builder* utility, but because this utility is meant to help build class and collection hierarchy it's better to exploit it for the sake of time saving. Furthermore, the *Class Builder* utility keeps track of the hierarchy of the built classes and collections and generates the framework code necessary to implement them including their interface (i.e. properties, methods and events).

In the figure only the interface (properties and methods) of the *Single_Phase* subclass is shown. The *Converters* class is the one that represents the type-based topologies. This class and the *InterfaceModule* class were created separately from the remaining ones. We didn't use the class builder for they don't have the hierarchy as the other class do. Nevertheless, the basic ideas are the same in terms of implementation and accessibility through their interfaces.

3.1 Type-Based Knowledge

This part of knowledge encompasses eleven types of converter organised under the four basic types of conversions (DC-to-DC, AC-to-DC, AC-to-AC and DC-to-AC). Figures below show how these different converters appear to the user. The basic schematic and brief information about the converter are provided within this illustration. These converters are as follows:

- Buck converter (as shown in figure 3)
- Boost converter
- Buck-Boost converter
- Cuk converter
- Single-phase full wave rectifier (Uncontrolled)
- Three-phase rectifier (Uncontrolled)
- Single-phase AC controller
- Three-phase AC controller (wye-connected)
- Three-phase AC controller (delta-connected)
- Square-wave inverter
- PWM Inverter



Please Choose One Application for Your Converter

1: Switching DC Power Supply without Isolati

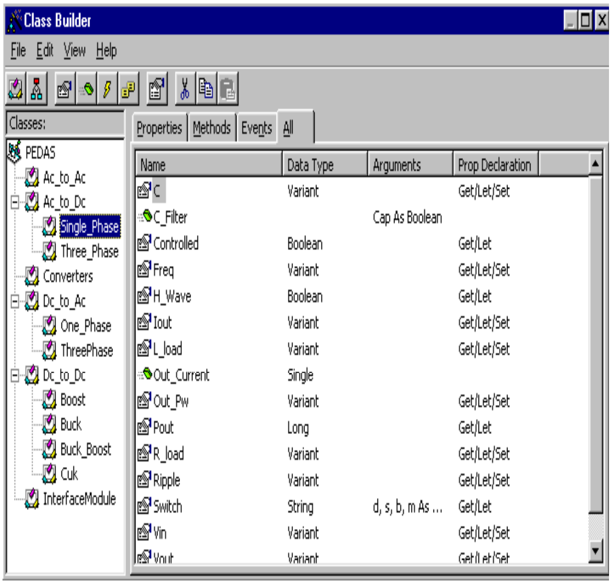
11 Matched Switches (You may change the Optimum Switch from the list below)

IRF640 at RM 14.01

Voltage Rating	200
Current Rating	18
Manufacturer	International Rectifier
Price RM/Unit	14.01

PEDAS, KEE. UPM 8/6/02 11:07 AM

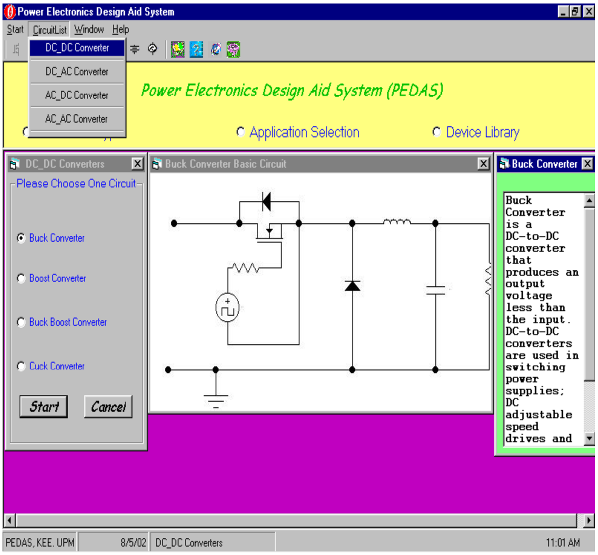
Figure 1: An instance of PEDAS general layout (GUI).



- PEDAS
  - Ac_to_Ac
  - Ac_to_Dc
    - Single_Phase
    - Three_Phase
  - Converters
  - Dc_to_Ac
    - One_Phase
    - ThreePhase
  - Dc_to_Dc
    - Boost
    - Buck
    - Buck_Boost
    - Cuk
  - InterfaceModule

Name	Data Type	Arguments	Prop Declaration
C	Variant		Get/Let/Set
C_Filter		Cap As Boolean	
Controlled	Boolean		Get/Let
Freq	Variant		Get/Let/Set
H_Wave	Boolean		Get/Let
Iout	Variant		Get/Let/Set
L_load	Variant		Get/Let/Set
Out_Current	Single		
Out_Pw	Variant		Get/Let/Set
Pout	Long		Get/Let
R_load	Variant		Get/Let/Set
Ripple	Variant		Get/Let/Set
Switch	String	d, s, b, m As ...	Get/Let
Vin	Variant		Get/Let/Set
Vout	Variant		Get/Let/Set

Figure 2. Class builder utility and classes' hierarchy



Buck Converter is a DC-to-DC converter that produces an output voltage less than the input. DC-to-DC converters are used in switching power supplies; DC adjustable speed drives and

Figure 3. Buck converter layout in PEDAS environment

This type of knowledge is coded under only one class (object) as mentioned earlier. This class is named *Converters*, where each of its methods represents one topology among the above. All operations concerning one topology are accomplished within this method. These operations vary from requesting the user's entries to circuit parameters calculation including the guidance of the user through the process and the suggestion further steps if the inference engine fails to come out with the required circuit. Below is a segment of listing code shows the implementation of the *SinglePhaseRectifier* method, which corresponds to the single-phase uncontrolled rectifier shown above.

```
Public Function SinglePhaseRectifier()
     On Error Resume Next
frst:     vinput = InputBox("Please Enter The Input Voltage
(Volt)")
            If Val(vinput) <= 0 Then
 res  =  MsgBox("The  Input  Voltage  Must  be  greater  than
zero", 1, "PEDAS")
               If res = vbCancel Then Exit Sub
               GoTo frst
            End If
            vinput = 1.41 * vinput
frth:     Freq   =   InputBox("Please   Enter   The   Operating
Frequency (Hz)")
            If Val(Freq) <= 0 Then
 res  =  MsgBox("The  frequency  Value  Must  be  greater  than
zero", 1, "PEDAS")
          If res = vbCancel Then Exit Sub
               GoTo frth
          End If
fith:    RLoad  =  InputBox("Please  Enter  The  Load  value
(Ohm)")
           If Val(RLoad) <= 0 Then
         res = MsgBox("The Load Value Must be greater than
zero", 1, "PEDAS")
         If res = vbCancel Then Exit Sub
               GoTo fith
            End If
capc:   Rf  =  InputBox("What'is  the  Maxminum  Ripple  Factor
for Vout (%)")
           If Val(Rf) <= 0 Then
       res = MsgBox("The Ripple Factor Must be greater than
zero", 1, "PEDAS")
    If res = vbCancel Then Exit Sub
               GoTo capc
            End If
        Rf = Rf / 100
        Cap = (1 / Rf)
        Cap = Cap / (2 * Freq * RLoad)
        Cap = Cap * 1000000
        Cap = Format(Cap, "##.##")
Open "c:\msim53\1phrect.sch" For Input As #10
Open "c:\outp.sch" For Output As #20
        Dim srg As String
        Dim Leng As Integer
        Input #10, srg
        Close #10
        choice = 1
       Leng = Len(srg)
       For i = 1 To Len(srg)
         wrote = False
         If Left(srg, 1) = "&" Then
         Select Case choice
         Case 1
              k = vinput
         Case 2
              k = Freq
         Case 3
              k = RLoad
         Case 4
            k = Cap
       End Select
       choice = choice + 1
       Print #20, k;
       wrote = True
     End If
     If wrote = False Then
         Print #20, Left(srg, 1);
    End If
    If i < Leng Then
     srg = Right(srg, Len(srg) - 1)
             Else
     srg = srg
           End If
      Next i
```

```
Close #20
SchInterface ("psched.exe c:\outp.sch")
SendKeys "{F11}", True
End Function
```

There are some remarks to be made about this type of knowledge. These remarks are about its consistency, authenticity and usefulness.

- The first remark is that this knowledge and its coding are meant to give a general idea about the four types of conversion in a simple and attractive way, without the complexity of using general-purpose simulation tool. Therefore, this part can serve as a demo or tutorial for students who deal with converters for the first time.
- The encompassed topologies are merely examples of the some basic topologies. One can easily observe the missing of many basic topologies such as controlled rectifiers in AC-to-DC conversion and dc choppers with isolation for the DC-to-DC conversion type.
- However, the missing topologies can be easily added and coded by adding methods to the *Converters* class in the same form of the existed ones.
- This knowledge is not dynamic knowledge in term of the switching devices. The switches are predefined, and the user is always provided by the same switch within the same topology, regardless of its ratings. Nevertheless, other parameters (e.g. filter values) are dynamically changed.
- The term “type-based” refers to the way that the user can access this knowledge. The user has no way to access a particular circuit only through its type of conversion. Therefore, this term doesn’t reflect the way the knowledge is represented.

3.2 Application-Based Knowledge

This knowledge constitutes the kernel of the knowledge base. It encompasses more than twenty-five (25) topologies organised under fourteen (14) objects (classes and subclasses) with their own interfaces. This is a very different knowledge from the above one. Although they may share some same topologies, the ways of representation are totally dissimilar.

This part of knowledge is arranged in such a way that the user has the access to a specific topology only through its application (see figure 4). Nevertheless, the user can choose the type of conversion of the circuit if he/she knows it in order to narrow the list of applications offered by PEDAS. Likewise, the internal implementation of this knowledge is based on its type of conversion following its way of representation. This implementation permits to exploit the usefulness of the inheritance feature of OOP between the class and subclass and hence reduce the development code and time drastically.

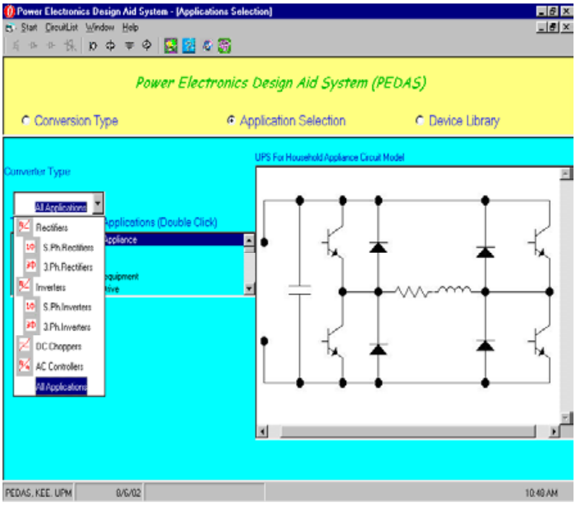


Figure 4. Application-based knowledge (all applications).

The following sections show samples of the Dc_to_Dc, Dc_to_Ac) classes and their interface implementation. Buck subclass also is dealt here with some details.

1. Dc to Dc Class

This class as from its name covers all dc chopper topologies. The main applications of this type of converters are in switching dc power supplies and dc drives. They are widely used for traction motor control in electric automobiles, trolley cars, marine hoists, forklift trucks and mine haulers. They provide smooth acceleration control, high efficiency, and fast dynamic response. They can be used also in conjunction with an inductor to generate a dc current source especially for the current source inverter.

To efficiently synthesize these converters, four basic topologies are encoded here. They are the buck, boost, buck-boost and Cuk topologies. However, other topologies such as flyback, forward, push-pull and full-bridge converters can be added to enlarge this knowledge. One should examine each of these topologies thoroughly in order to determine its necessary interface (properties and methods). Once the interface is completely determined, they can be implemented the same way as the first four were. Here only segments and passages from the source code listings are given to show the implementation of this class as it is impracticable to include the entire code.

```
Private Sub Class_Initialize()
  Set mvarBuck = New Buck
  Set mvarBoost = New Boost
  Set mvarBuck_Boost = New Buck_Boost
  Set mvarCuk = New Cuk
End Sub
Private Sub Class_Terminate()
  Set mvarCuk = Nothing
  Set mvarBuck_Boost = Nothing
  Set mvarBoost = Nothing
  Set mvarBuck = Nothing
End Sub
```

The first sub is to create the subclasses' instances whenever an instance of parent class is created. The second sub is to destroy the created objects and free the memory when the parent object is terminated. The following segments illustrate how to get the input voltage and output voltage values from the user. This is achieved by setting *Vin* and *Vout* properties in the parent class instead in the subclasses, this is because in all topologies these values must be set by the user. Hence these two properties are common between the four subclasses.

```
Public Property Get Vin() As Variant
        mvarVin = Val(Frm1.Text10)
        Vin = mvarVin
        mvarBoost.Vin = mvarVin
        mvarBuck.Vin = mvarVin
        mvarBuck_Boost.Vin= mvarVin
        mvarCuk.Vin= mvarVin
End Property
Public Property Get Vout() As Variant
        mvarVout = Val(Frm1.Text9)
        Vout = mvarVout
        mvarBoost.Vout = mvarVout
        mvarBuck.Vout = mvarVout
        mvarBuck_Boost.Vout= mvarVout
        mvarCuk.Vout= mvarVout
End Property
```

2. Ac to Dc Class

This is a parent class to cover rectifier topologies ranging from single phase uncontrolled rectifiers to three-phase controlled rectifiers. Rectifiers are used to change the ac input to a fixed (uncontrolled rectifiers) or to a controlled (thyristor rectifiers) dc output. They are used mainly for unregulated dc power supplies and variable-speed dc drives especially in high power applications benefiting from the high ratings of SCRs. This class has two subclasses *Single_Phase* and *Three_phase* based on the type of the input source available. The initialisation and termination procedures of this class are accomplished the same way for *Dc_to_Dc* class as the listings below show.

```
Private Sub Class_Initialize()
  'create the mThree_Phase object when the Ac_to_Dc class
is created
   Set mvarThree_Phase = New Three_Phase
  'create the mSingle_Phase object when the Ac_to_Dc class
is created
  Set mvarSingle_Phase = New Single_Phase
End Sub
Private Sub Class_Terminate()
    'destroy  the  mThree_Phase  object  when  the  Ac_to_Dc
class is destroyed
    Set mvarThree_Phase = Nothing
   'destroy  the  mSingle_Phase  object  when  the  Ac_to_Dc
class is destroyed
    Set mvarSingle_Phase = Nothing
End Sub
```

Samples of some properties and methods of this class are given in the listings below.

```
Public Sub C_Filter(Cap As Boolean)
 H = mvarRipple / 100
 If Cap Then
  mvarC = 1 / (2 * mvarFreq * mvarR_load * H)
  mvarC = 1000000 * mvarC
  mvarC = Round(mvarC)
 End If
End Sub
Public Property Get Ripple() As Variant
RippleVoltage:
    mvarRipple  =  Val(InputBox("Please  Enter  The  Maximum
Ripple Allowed In Your Output Voltage(%)"))
    If mvarRipple <= 0 Then
    MsgBox ("Please Enter Positive Numerical Value")
    GoTo RippleVoltage
    Exit Sub
    End If
        Ripple = mvarRipple
    End If
End Property
Public Property Get Vout() As Variant
If Not mvarControlled Then
    If mvarH_Wave Then
        mvarVout = (mvarVin * Sqr(2)) / (3.14159)
        Vout = mvarVout
    Else
        mvarVout = (2 * mvarVin * Sqr(2)) / (3.14159)
        Vout = mvarVout
    End If
 Else
OutVoltage:
   mvarVout  =  Val(InputBox("What's  your  Machine  Operating
Voltage, (Volt)"))
   If mvarVout <= 0 Then Exit Sub
    If mvarVout >= 0.9 * mvarVin Then
        Vin30  =  MsgBox("Your  Source  is  Unable  to  Provide
this Voltage, Please Change Either the Input or the Output
Volage", vbCritical + vbAbortRetryIgnore)
        If Vin30 = 3 Then Exit Sub
        GoTo OutVoltage
    Exit Sub
   End If
   Vout = mvarVout
 End If
End If
End Property
```

The first passage is to calculate the output filter value that corresponds to the desired output voltage ripple entered by the user. The second segment is just to show how to acquire the ripple value from the user and guide him if an invalid value is entered. The last segment is to obtain the output voltage (Vout property). Here two ways to get this value; if the application selected by the user needs a controlled rectifier then this value is obtained from him by soliciting procedure. In case of uncontrolled rectifier, the output voltage value has to be calculated for that the user has no control on it once the source voltage is specified. Note that in the controlled case, the user is prompted to change the desired output voltage if the source is unable to meet it.

3. Buck Subclass

This is only to give an example of how the subclasses are implemented as child classes of the parent classes. The Buck subclass is to cover all dc step-down converter instances. The basic topology and its parameter calculation were collected from various textbooks (8 and 9). Figure 5 shows an instance of a basic circuit of the step down converter with resistive load and square wave control scheme. Table 1 illustrates the properties and methods that were extracted by analysing this topology to build the interface of the subclass. It gives also brief description of each property and method.

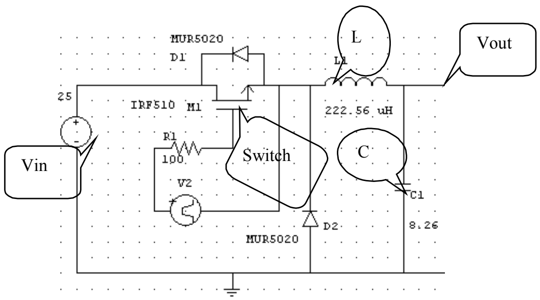


Figure 5: An instance of basic step-down Dc chopper with resistive load.

The following listing passages illustrate how some methods and properties were implemented.

```
Public Sub Filter(Cfilter As Boolean)
d = mvarVout / mvarVin
If Cfilter Then
mvarC = (1-d)/(8 * mvarL * (mvarRipple) * mvarSw_Freq ^ 2)
```

```
        If mvarC > 10000 Then
          res = MsgBox("The Capacitor is Too Large, " &
Str(mvarC) & " Please Increase the Input Voltage " &
mvarVin & " or Decrease the Output One" & mvarVout,
vbOKCancel)
          If res = vbCancel Then Exit Sub
          pass = False
          End If
Else
    mvarL = (1 - d) * mvarRLoad / (2 * mvarSw_Freq)
End If
End Sub
```

Table 1: Buck Subclass Properties and Methods

Name	Type	Description
C	Property	To hold the capacitor filter value (calculated from method)
Current	Method	To calculate the average output current
Filter	Method	To calculate the filters' values (L and C)
Iout	Property	To hold the output current value (calculated from method)
L	Property	To hold the inductor filter value (calculated from method)
Ripple	Property	To hold the desired ripple factor (from the user)
R_Load	Property	To hold the user's load value (from the user)
Sw_Freq	Property	To hold the suitable switching frequency (Suggested by the System)
Switch	Property	To hold the selected switching device (Selected by the System)
Vin	Property	To hold the input voltage value (from the user)
Vout	Property	To hold the desired output voltage value (from the user)

It is seen from the source code segment that this method serves to calculate both of the filter values (i.e. C and L). Which one is to be calculated depends on the setting of the argument (*Cfilter*) upon calling the method. If the argument is set to *True* then the capacitance will be calculated, otherwise the inductance is to be calculated and returned as a function value. The argument is set by the inference engine outside the interface, it set through the interaction module as it will be detailed later.

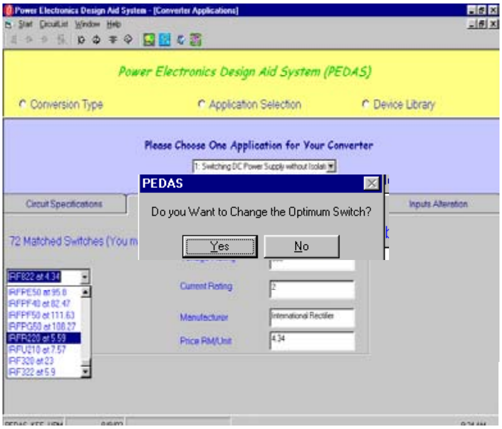
```
Public Property Get Switch() As Variant
        mvarSwitch    =    Libfrm3.SearchResult2("MOSFET",
mvarVin, mvarIout)
        Switch = mvarSwitch
End Property
```

The switch here is extracted from the database using *SerachResult2* function which is called from the devices library module where it is developed. Here the type switch selected "MOSFET" as a rule of thumb. Upon calling this function, the ratings (i.e. voltage and current ratings) are required. This means that it can not be called only after getting these two values which correspond to Vin and Iout properties respectively. This vital function was developed using structured query language (SQL) procedures and statements embedded in Visual Basic. The following listing is a portion taken from the source code of this function.

```
Set dbsPowerDevice = OpenDatabase(Filename,
dbDriverCompleteRequired, Fales, DatabaseConnect)
   Set rstSCR = dbsPowerDevice.OpenRecordset("SELECT * FROM
" & DeviceSearch)
     If VoltageSearch = "" Then
             VoltageSearch = "'*'"
     Else
        VoltageSearch = VoltageSearch
     End If
     If CurrentSearch = "" Then
        CurrentSearch = "'*'"
     Else
        CurrentSearch = CurrentSearch
     End If
Do While True
   strMain = "[Voltage] " & "> " & VoltageSearch & " and
[Current] " & "> " & _CurrentSearch
   With rstSCR
     .MoveLast
     .FindFirst strMain
      If .NoMatch = True Then
        searchNext = MsgBox("No" & DeviceSearch & " Meets
Your Requirements", vbOKOnly, "No Device found")
      Exit Do
   End If
   Do While True
     varBookmark = .Bookmark
     Data1 = rstSCR!Name
     Data2 = rstSCR!Device
     Data3 = rstSCR!Type
     Data4 = rstSCR!Voltage
     Data5 = rstSCR!Current
     Data6 = rstSCR!SFrequency
     Data7 = rstSCR!Package
     Data8 = rstSCR!manufacture
     If rstSCR!price = 0 Then
      Data9 = "-"
     Else
      Data9 = rstSCR!price
     End If
     Set ItmXY = Frm1.Combo2
     Set VoltCombo = Frm1.Combo1(0)
     Set currcombo = Frm1.Combo1(1)
     Set ManfCombo = Frm1.Combo1(2)
     Set PriceCombo = Frm1.Combo1(3)
     ItmXY.AddItem Data1 & " at " & Data9
     VoltCombo.AddItem Data4
     currcombo.AddItem Data5
     ManfCombo.AddItem Data8
     PriceCombo.AddItem Data9
     HowManyData = HowManyData + 1
     If Not rstSCR!price = 0 Then
       If rstSCR!price < CheapPrice Then
         CheapPrice = rstSCR!price
         CheapName = rstSCR!Name
```

```
          voltagerating = rstSCR!Voltage
          currentrating = rstSCR!Current
          manufac = rstSCR!manufacture
        End If
      End If
      rstSCR.FindNext strMain
      If IIf(rstSCR.NoMatch, False, True) = False Then
        .Bookmark = varBookmark
        SearchResult2 = CheapName
        Exit Function
      End If
     Loop
      End With
     Exit Do
Loop
rstSCR.Close
dbsPowerDevice.Close
```

The three argument of this function are required, if on argument or more is missing the searching process will not be launched and an error will be generated warning the user about the void research. Once the search is initiated and the ratings are within the stored values, the function doesn't only return one switch, but it returns all the devices that satisfy its arguments (i.e. the ratings). Yet it suggests and recommends the cheapest device among the resulted switches meanwhile the user is given the hand to change this device if his concern is more on other parameters (the manufacturer par example) than on the price as shown in figure 6.


Power Electronics Design Aid System - [Converter Applications]
Start CircuitList Window Help
Power Electronics Design Aid System (PEDAS)
Conversion Type
Application Selection
Device Library
Circuit Specifications
Inputs Alteration
Current Rating
Manufacturer
Price RM/Unit
Yes
No

Please Choose One Application for Your Converter

1: Switching DC Power Supply without Isolati

PEDAS

Do you Want to Change the Optimum Switch?

72 Matched Switches (You m

IRF822 at 4.34

IRFPE50 at 95.8
IRFPF40 at 82.47
IRFPF50 at 111.63
IRFPG50 at 108.27
IRFR220 at 5.59
IRFU210 at 7.57
IRF320 at 23
IRF322 at 5.9

Current Rating: 2
Manufacturer: International Rectifier
Price RM/Unit: 4.34

Fig.6. The returned results of the optimum switches.

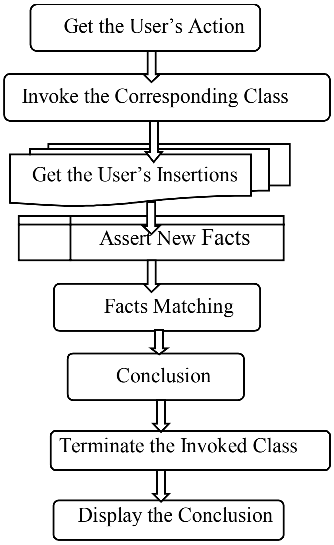


Fig.7. Ordinary steps of the inference engine flow.

3.3 Inference Engine

The inference engine operates on the knowledge base in its search of solutions. It matches the facts asserted from the user inputs against the stored facts in the system's knowledge base in order to come out with results or new facts. The inference engine accesses the knowledge base through the class interfaces. Once the user selects his application or type of converter through the interaction module, the inference engine invokes the class methods and properties whenever needed to infer conclusions or assert facts. A sketch of the inference engine process flow is illustrated in figure 7.

The steps shown above are general steps to be followed in order to reach the final result. The first step is to know what type of knowledge the user wants to access in order to invoke the right class module as a response to the user action. Then, through the interaction module the user is required to insert his inputs and specification which will be used by the inference engine to assert new facts and finalise the results through the invoked class interface. Once the results are finalised, they are to be sent for displaying through the interaction and interface modules. Sample of the source code to show how to access the Boost subclass is shown below.

```
Private Sub BoostCon()
Set con = New Dc_to_Dc
Vout = con.Vout
        If Vout <= 0 Then
```

```
           res = MsgBox("The Output Voltage Must be Greater
Than Zero", vbOKCancel)
           Exit Sub
       End If
RLoad = con.RLoad
       If RLoad <= 0 Then
              res = MsgBox("The Load Must be Greater Than
Zero", vbOKCancel)
              Exit Sub
       End If
       Rload1 = RLoad / 1000 ‘ convert to kOhm
Ripple = con.Boost.Ripple
       If Ripple <= 0 Then
          ms = "The Ripple Value Must be Greater than Zero"
          res = MsgBox(ms, vbOKCancel)
          Exit Sub
        End If
Vin = con.Vin
      If Vin >= Vout Or Vin <= 0 Then
      res = MsgBox("The Input Voltage must be Less than the
Output One" & Str(Vout), vbOKCancel)
       Exit Sub
      End If
d = (1 - (Vin / Vout))
f = con.Boost.Sw_Freq '
Lmin = con.Boost.L
Imax = con.Boost.Iout * Lmin * f)))
C = con.Boost.C
C = Format(C, ".00")
Lmin = Format(Lmin, ".000")
Imax = Format(Imax, ".000")
swtch = con.Boost.Switch
k = 0

For i = 0 To Combo1(0).ListCount - 1
ReDim Preserve swtype(k), price(k), VoltR(k), CurrR(k),
IDa(k), manfact(k)
price(k) = Combo1(3).List(i)
VoltR(k) = Combo1(0).List(i)
CurrR(k) = Combo1(1).List(i)
manfact(k) = Combo1(2).List(i)
IDa(K) = Combo1(5).List(i)
k = k + 1
Next i
If swtch <> Empty Then
 minp = Libfrm3.CheapPrice
 VoltageR = Libfrm3.voltagerating
 CurrRating = Libfrm3.currentrating
 manufac = Libfrm3.manufac
 Combo2.Text = swtch + "    at RM " + Str(minp)
 Frame3.Visible = False
 If Combo2.ListCount > 1 Then
  Frame1.Caption = Str(k) + " Matched Switches (You may
change the Optimum Switch
  from   the list below)"
Else
```

```
  Frame1.Caption = "Only One Switch Matches Your Specifications"
End If
 Output ‘ call the output subroutine to display the output results
End If
End Sub
```

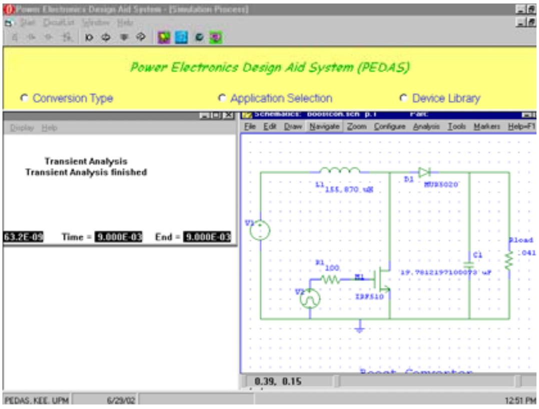
The first step to access this segment of the knowledge base is by creating a new instance of the Dc_to_Dc class, using the *Set* and *New* keywords. This automatically will create an instance of the Boost subclass in the *Class_Initialize sub* as was shown in the precedent section. After gaining access to the Boost subclass, the inference engine starts asserting the facts and concluding results through the subclass interface before it finalises the results and send them to be displayed.

3.4 Interface Module

This module is the intermediate channel between PEDAS and the simulation package (PSpice). It serves in displaying the resulted circuit, its simulation process and results in PEDAS environment. After the inference engine decision is made, it sends relevant data and information (files *.sch, *.net and *.cir) to this module in order to call the simulation package to display and simulate the circuit. The connection process was achieved by using *Application Programming Interface* (API) technique. This technique is a complicated set of functions, messages, and structures allowing programmers in all types of programming languages to build applications that run on the Windows and Windows NT operating systems (Noel and Erik, 1996). PEDAS uses this set to request and carry out lower-level services performed by a computer's operating system. The following listing shows how this technique is exploited in the *InterfaceModule* class. All functions preceded by the keywords *Private Declare Function* (e.g. *GetParent* function) are API functions called by PEDAS in incoming procedures. An instance of displaying the resulted circuit within PEDAS environment is shown figure 8.

```
Private mvarPid As Long 'local copy
Private mvarInterfProg As String 'local copy
Private Declare Function FindWindow Lib "user32" Alias "FindWindowA" (ByVal lpClassName As Long, ByVal lpWindowName As Long) As Long
Private Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, ByVal hWndNewParent As Long) As Long
Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As Long, lpdwProcessId As Long) As Long
Private Declare Function GetWindow Lib "user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function DestroyWindow Lib "user32" (ByVal
hwnd As Long) As Long
Private  Declare  Function  Putfocus  Lib  "user32"  Alias
"SetFocus" (ByVal hwnd As Long) As Long
Public Function ProgrInstance(ByVal Target_pid As Long) As
Long
Dim test_hwnd As Long, test_pid As Long, test_thread_id As
Long
    'Find the first window
    test_hwnd = FindWindow(ByVal 0&, ByVal 0&)
    Do While test_hwnd <> 0
        'Check if the window isn't a child
        If GetParent(test_hwnd) = 0 Then
            'Get the window's thread
            test_thread_id                                  =
GetWindowThreadProcessId(test_hwnd, test_pid)
            If test_pid = Target_pid Then
                ProgrInstance = test_hwnd
                Exit Do
            End If
        End If
        'retrieve the next window
        test_hwnd = GetWindow(test_hwnd, 2)
    Loop
End Function
Public Property Let InterfProg(ByVal vData As String)
'used when assigning a value to the property, on the left
side of an assignment.
'Syntax: X.InterfProg = 5
    mvarInterfProg = vData
End Property
Public Property Get InterfProg() As String
'used when retrieving value of a property, on the right
side of an assignment.
'Syntax: Debug.Print X.InterfProg
    InterfProg = mvarInterfProg
End Property
Public Property Let Pid(ByVal vData As Long)
'used when assigning a value to the property, on the left
side of an assignment.
'Syntax: X.Pid = 5
    mvarPid = vData
End Property

Public Property Get Pid() As Long
'used when retrieving value of a property, on the right
side of an assignment.
'Syntax: Debug.Print X.Pid
    Pid = mvarPid
End Property
```



Transient Analysis
Transient Analysis finished

63.2E-09 Time = 9.000E-03 End = 9.000E-03

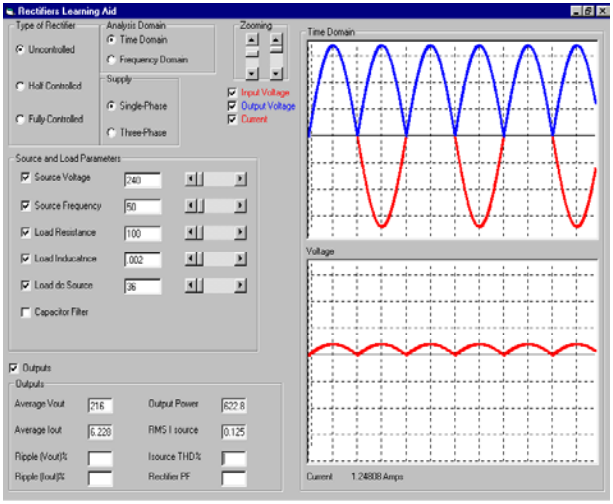
Figure 8. Displaying the Resulted Circuit within PEDAS Environment

3.5 Explanation Module

The explanation facility allows the system to explain its reasoning to the user in language that he/she can understand through a user interface. These explanations include justification for the system's conclusions (the know how), and the explanation why the system needs a particular data (the why query). In some systems, a tutorial explanations or deeper theoretical justifications of program's action are granted. For instance, in PEDAS this module is designed to guide and help the user:

- to find out how the results have been achieved and
- to provide general information about the circuit topologies and switching devices and
- to enhance the understanding of the converter operation by providing a tutorial-like facility.

In fact, the first point is inclusive in the interaction module. From the communication process, the user can know how certain results have been reached. A help module was developed to accomplish the second purpose. This module gathers basic information, formulas and topologies for each type of conversion. Relevant information has to be gathered and collected from various books and articles and then coded into this module. An auxiliary component to this module is developed to provide a simulation-like demo of basic converters based on their ideal parameters (see figure 9). This component can be considered as a learning aid system for students to get a comprehensive understanding of power electronics converters operations. Various parameters are made available for the user to change and see their impacts on the converter performance through its waveforms.



Type of Rectifier: [✓ Uncontrolled] Half Controlled Fully Controlled

Analysis Domain: [✓ Time Domain] Frequency Domain

Supply: [✓ Single-Phase] Three-Phase

☑ Input Voltage
☑ Output Voltage
☑ Current

Source and Load Parameters

☑ Source Voltage	240
☑ Source Frequency	50
☑ Load Resistance	100
☑ Load Inducalnce	.002
☑ Load dc Source	36
☐ Capacitor Filter	

☑ Outputs

Outputs

Average Vout	216	Output Power	622.8
Average Iout	6.228	RMS I source	0.125
Ripple (Vout)%		Isource THD%	
Ripple (Iout)%		Rectifier PF	

Current 1.24808 Amps

Figure 9: An instance of PEDAS simulation-like Demo

3.6 Devices Library Module

This module is a part of the knowledge base module described above. It contains power electronics switching devices along with their relevant necessary information such as type, current and voltage ratings. Figure10 gives a general view of this module. Such module encompasses huge data and allows the user to add more data during the lifetime of the system. This requires the use of a database to provide efficient handling and management of the data. Visual Basic offers a special engine called JET (Joint Engine Technology) to ease the dealing with databases. JET allows the programmer to use the methods and properties of *Data Access Objects* (DAOs) to access and manipulate database information. These methods and properties allow the user to retrieve data, modify the data and change the presentation order of the data. The programmer could even modify the structure of the database by creating, modifying, and deleting fields, tables and indexes.

Visual Basic 6 provides two controls to work with database files; the data control (with its associated bound controls) and data access objects. The two controls are not mutually exclusive, they can be used together to take advantage of each.

The first step in gaining access to information in a database is to open the database itself. "*OpenDatabase*" is a method of the "*Workspace*" object used to open the database. How to use this method is illustrated below.

```
 'Using   the   OpenDatabase   method   to   access   the
database.
Dim dbsPowerDevice As Database
Set dbsPowerDevice = OpenDatabase (“PowerDevice.mdb”,
False, False, DataBaseConnect)
```

As it is seen from the above listing, the *OpenDatabase* method has four arguments. The first argument specifies the name of the database to be opened. This is a required argument (i.e. it must be set), while the second, third and fourth arguments are optional. The second argument specifies whether the database will be opened for exclusive use (that's no one else can access the database while the user works with it). The third argument indicates whether the database is to be opened for read-only, meaning that the user cannot modify its contents. The fourth argument specifies various connections information, including passwords. It is worth to note that the database “PowerDevice.mdb” is already created using any Database Management System (DBMS) such as *Microsoft Access* for instance.

This module is enhanced by several functions that can be seen clearly from the main menu (or toolbar) Here only few of them are discussed in details.

1. Search Function

Its icon is in most left side of the toolbar. It can be accessed also form the *Search* submenu under the *File* menu (Ctrl + S). As shown in figure 11 it provides several criteria for searching a specific record. These criteria can be set all together as well as, the user has the advantage to set only some (at least one) of them for fast searching.

Note that for this function, the *Recordset* is opened as snapshot since there is no need to change the data inside the database. The following listing lists the steps to find a record.

```
Set dbsPowerDevice = OpenDatabase(Filename, False, Fales,
DatabaseConnect)
Set rstSCR = dbsPowerDevice.OpenRecordset("SELECT * FROM "
& ImageCombo1.Text, dbOpenSnapshot)
strMain = "[Name] LIKE " & "'" & NameSearch & "'" & _
       " and [Device] LIKE " & "'" & DeviceSearch & "'" & _
        " and [Type] LIKE " & "'" & TypeSearch & "'" & _
     " and [Voltage] LIKE " & "'" & VoltageSearch & "'" & _
     " and [Current] LIKE " & "'" & CurrentSearch & "'" & _
     "   and   [SFrequency]   LIKE   "   &   "'"   &
SwitchingFrequencySearch & "'"
         With rstSCR
         .MoveLast
         .FindFirst strMain
If .NoMatch = True Then
         searchNext  =  MsgBox("Device  not  found.  Find
another device with ratings  similar to it?", vbYesNo, "No
records found")
```

```
    If searchNext = vbYes Then
       VoltagePercent   =   "<   "   &   VoltageSearch   +
((VoltageSearch * 25) / 100)
       VoltageSearch = "> " & VoltageSearch
       CurrentPercent   =   "<   "   &   CurrentSearch   +
((CurrentSearch*25) / 100)
       CurrentSearch = "> " & CurrentSearch
    End If
strMain  =  "[Name]  LIKE  "  &  "'"  &  NameSearch  &  "'"  &
" and [Device] LIKE " & "'" & DeviceSearch & "'" & " and
[Type] LIKE " & "'" & TypeSearch & "'" &" and [Voltage] " &
VoltageSearch & " and [Voltage] " & VoltagePercent & " and
[Current]  "  &  CurrentSearch  &  "  and  [Current]  "  &
CurrentPercent   &"   and   [SFrequency]   LIKE   "   &   "'"
&SwitchingFrequencySearch
     .MoveLast
     .FindFirst strMain
If .NoMatch = True Then
    MsgBox "No device with similar ratings.", , "Not Found"
    Exit Do
End If
End If
```

It can be seen from the above listing, that in case of searching a device by its voltage or current rating and if the *Nomatch* property is true, the user is provided by an optional search of a similar device which has (±25%) tolerance of rating values. Even though some lines of the above listing are similar to that in *SearchResult2* function discussed above both function are different in that:

- The *Search* Function deals directly with the user, while *SearchResult*2 function deals internally within the system in terms of arguments and return values.
- *SearchResult*2 function needs all its arguments to be set, while the arguments of *Search* function are optional but at least one of them has to be set.

2. *Update Function*

This function is to update the data in the opened *Recordset* firstly and then in the database. The *Edit* method is used to copy the new information entered by the user to the data buffer. Then the new data are assigned to fields of the *Recordset* and the *Update* method (not function) is called to write (physically) the record's changes and store them. This function is accessible from the *Update* submenu under the *File* main menu. It's also accessible from the toolbar or shortcut key "Ctrl U".

A listing of code source below shows a segment of Update function and its dialogue box is given in figure 12.

```
Set      dbsPowerDevice      =      OpenDatabase(Filename,
dbDriverCompleteRequired, False, DatabaseConnect)
```

```
   Set rstSCR = dbsPowerDevice.OpenRecordset("SELECT * FROM
" & Devicetxt, dbOpenDynaset)
With rstSCR
!Manufacture = Manftxt
rstSCR!price = Pricetxt
.Update
.Bookmark = .LastModified
.Edit
!Name = strName
!Type = strType
End With
```

Only two fields can be modified and hence updated, these are the manufacturer and the price of the device. The *Enable* property of other fields is set to "*False*" so the user has no access to them.

3. Add Function

The Add function is to enable user to add in new components when they are introduced into the market. By this way, the user will be able to keep the program updated as and when new products are released by the manufacturers. The code below shows how the addition operation is performed.

```
Set      dbsPowerDevice      =      OpenDatabase(Filename,
dbDriverCompleteRequired, False, DatabaseConnect)
    Set  rstSCR  =  dbsPowerDevice.OpenRecordset("SELECT  *
FROM " & ImageCombo1.Text, dbOpenDynaset)
   strName = NameAdd
   strDevice = ImageCombo1.Text
   strType = TypeAdd
   ……….
  If strName <> "" And strDevice <> "" And strType <> "
Then
   rstSCR.AddNew
   rstSCR!Name = strName
   rstSCR!Device = strDevice
   rstSCR!Type = strType
   rstSCR.Update
   rstSCR.Bookmark = rstSCR.LastModified
  Set AItem = Form1.ListView1.ListItems.Add()
  'To visualize the new added record
  AItem.Text = strName
  AItem.SubItems(1) = strDevice
  AItem.SubItems(2) = strType
  MsgBox ("The component is stored into Database."), ,
"Add"
End if
```

There are three fields that must be specified by the user before a component can be added into the database. The compulsory fields are *'Name'*, *'Voltage'* and *'Current.'* After these ratings are entered, the component will be added to the database. User will be prompted that the component has been added.

This function is accessible by *'Add Data'* Submenu under *'File'* menu or by the shortcut key is 'Ctrl + A'. It can also be called upon by clicking on its corresponding icon on the toolbar menu. An instance of this function is shown in figure 13.

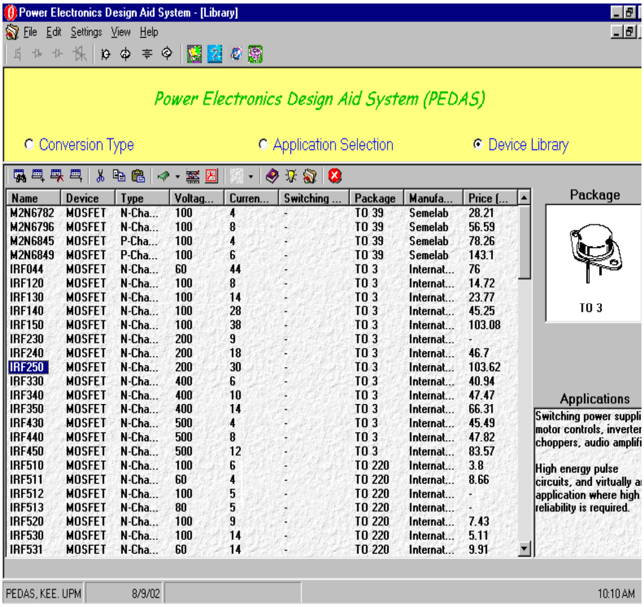
4. Delete Function

The *'DELETE'* function is to enable the user to delete a component when it is no longer exists in the market. This keeps the database alive and managed efficiently. The following segment of source code gives an idea how this process is accomplished.

```
Sub Delete()
Dim dbsPowerDevice As Database
   Dim rstSCR As Recordset
   Set   dbsPowerDevice   =   OpenDatabase(Filename,   False,
False, DatabaseConnect)
   Set                    rstSCR                         =
dbsPowerDevice.OpenRecordset(ListView1.SelectedItem.SubItem
s(1), dbOpenDynaset)
Picture1.Picture = LoadPicture
Libfrm1.Label1.Caption = ""
YNDelete = MsgBox("Are you sure you want to delete '" &
Libfrm1.ListView1.SelectedItem.Text & "' ?", 4, "Warning!")
If YNDelete = vbNo Then
    Exit Sub
End If
With rstSCR
.MoveFirst
 Do While True
    If (rstSCR!Name) = ListView1.SelectedItem.Text Then
         Exit Do
    Else
        .MoveNext
    End If
 Loop
End With
rstSCR.Delete
Dselect = ListView1.SelectedItem.Index
ListView1.ListItems.Remove (Dselect)
MsgBox ("The component has been removed from Database."), ,
"Delete"
rstSCR.Close
dbsPowerDevice.Close
End Sub
```

To delete a particular component, first select the component by highlighting it. Then, click the 'DELETE' icon on the toolbar menu a 'Delete' dialog box will appear and prompt for confirmation whether the component is to be deleted. Click 'Yes' to delete the component from the database. To cancel the process, click 'No'. The deleting process will be stopped.

Note that the component will be permanently deleted from the database by this process. If the user wishes to have the component back into the database, he needs to use the '*ADD*' function. There are other functions provided in this module, such as the characteristic curves of certain devices, their data-sheets and package numbers. This is shown in figure 14.


Power Electronics Design Aid System - [Library]
File Edit Settings View Help
Power Electronics Design Aid System (PEDAS)
Package
TO 3
Applications

( ) Conversion Type ( ) Application Selection [✓ Device Library]

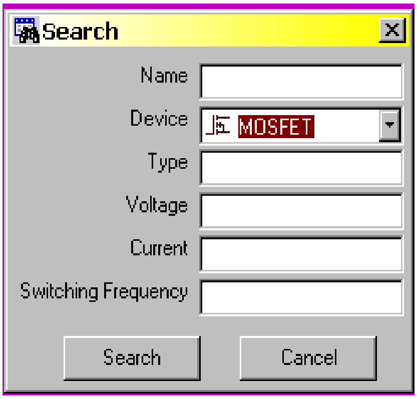
Name	Device	Type	Voltag...	Curren...	Switching ...	Package	Manufa...	Price (...
M2N6782	MOSFET	N-Cha...	100	4	-	TO 39	Semelab	28.21
M2N6796	MOSFET	N-Cha...	100	8	-	TO 39	Semelab	56.59
M2N6845	MOSFET	P-Cha...	100	4	-	TO 39	Semelab	78.26
M2N6849	MOSFET	P-Cha...	100	6	-	TO 39	Semelab	143.1
IRF044	MOSFET	N-Cha...	60	44	-	TO 3	Internat...	76
IRF120	MOSFET	N-Cha...	100	8	-	TO 3	Internat...	14.72
IRF130	MOSFET	N-Cha...	100	14	-	TO 3	Internat...	23.77
IRF140	MOSFET	N-Cha...	100	28	-	TO 3	Internat...	45.25
IRF150	MOSFET	N-Cha...	100	38	-	TO 3	Internat...	103.08
IRF230	MOSFET	N-Cha...	200	9	-	TO 3	Internat...	-
IRF240	MOSFET	N-Cha...	200	18	-	TO 3	Internat...	46.7
IRF250	MOSFET	N-Cha...	200	30	-	TO 3	Internat...	103.62
IRF330	MOSFET	N-Cha...	400	6	-	TO 3	Internat...	40.94
IRF340	MOSFET	N-Cha...	400	10	-	TO 3	Internat...	47.47
IRF350	MOSFET	N-Cha...	400	14	-	TO 3	Internat...	66.31
IRF430	MOSFET	N-Cha...	500	4	-	TO 3	Internat...	45.49
IRF440	MOSFET	N-Cha...	500	8	-	TO 3	Internat...	47.82
IRF450	MOSFET	N-Cha...	500	12	-	TO 3	Internat...	83.57
IRF510	MOSFET	N-Cha...	100	6	-	TO 220	Internat...	3.8
IRF511	MOSFET	N-Cha...	60	4	-	TO 220	Internat...	8.66
IRF512	MOSFET	N-Cha...	100	5	-	TO 220	Internat...	-
IRF513	MOSFET	N-Cha...	80	5	-	TO 220	Internat...	-
IRF520	MOSFET	N-Cha...	100	9	-	TO 220	Internat...	7.43
IRF530	MOSFET	N-Cha...	100	14	-	TO 220	Internat...	5.11
IRF531	MOSFET	N-Cha...	60	14	-	TO 220	Internat...	9.91

Switching power suppli
motor controls, inverter
choppers, audio amplifi

High energy pulse
circuits, and virtually a
application where high
reliability is required.

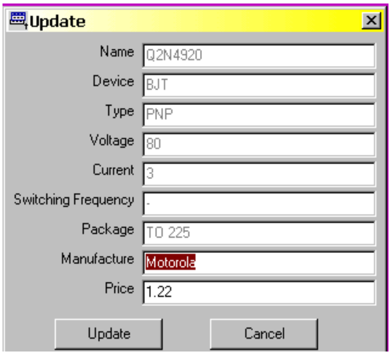
PEDAS, KEE. UPM 8/9/02 10:10 AM

Figure 10. PEDAS switching devices' library



Name	
Device	MOSFET
Type	
Voltage	
Current	
Switching Frequency	

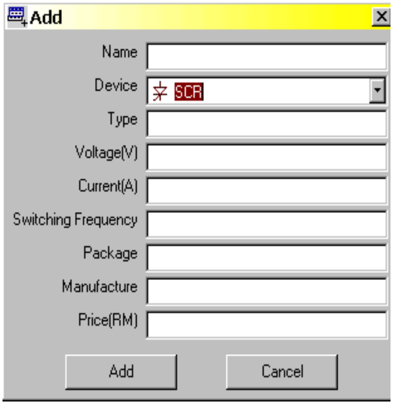
Figure 11. Search function dialog box and its search fields



Update

Name	Q2N4920
Device	BJT
Type	PNP
Voltage	80
Current	3
Switching Frequency	-
Package	TO 225
Manufacture	Motorola
Price	1.22

Figure 12: Update function dialog box



Add

Name	
Device	SCR
Type	
Voltage(V)	
Current(A)	
Switching Frequency	
Package	
Manufacture	
Price(RM)	

Figure 13: Add Function Layout

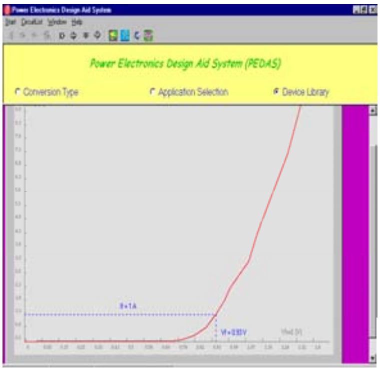


Figure 14. Output characteristics of D1N4001 in PEDAS

4. CONCLUSION

PEDAS is academic software was developed to push the design of power electronics converters one step forward. It aimed to improve the design procedure in terms of simplicity, flexibility and smoothness as well as time-consumption. This is successfully achieved by introducing knowledge-based techniques. The proposed system architecture assures a flexible interaction between the user and the tool as well as among the various modules constituting the system itself. The knowledge base containing topologies and switching devices is represented using object-oriented paradigm. It is the function of the inference engine to find the suitable topology that matches the user entries. Benefiting from the object oriented programming and the design visibility features of Visual Basic programming language; an attractive user interface is developed to assure the interaction of the user with the system. Furthermore, a library module enhanced by several vital functions is developed. This library makes the developed tool independent in its devices from that of the simulation package. A help module that explains how to use the developed tool and gathers information about power electronics converters is built and linked to the tool. This is enhanced by a flexible demo for further strengthen the understanding of converters operation. PEDAS still need some improvement in terms of expansion of the knowledge base and explanation modules.

5. REFERENCES

[1] M. J. Cumbi, D.W. Shepherd, and L. N. Hulley, Development of an Object-Oriented Knowledge-Based system for Power Electronic Circuit Design, IEEE Trans. Power Electronics 11(3): 393-404, 1996.

[2] D. Fezzani, H. Piquet and H. Foch, Expert System for the CAD in Power Electronics – Application to UPS, IEEE Trans. Power Electronics 12(3):578-587, 1997.

[3] S. J. Wang and Y. S. Lee, Development of an Expert System for Designing, Analysing and Optimising Power Converters, IEEE Trans. Power Electronics, 1996.

[4] L. E. Amaya, Computer Synthesis of Switching Power Converters, Ph.D. diss., Dept. of Electrical and Computer Engineering Illinois Univ. at Urbana-Champaign, 1998.

[5] N. Masatoshi, A Fast Computer Algorithm for Switching Converters, IEEE Tran. Power Electronics 12(1):180-186, 1997.

[6] K. Debebe and V. Rajagopalan, A Learning Aid for Power Electronics with Knowledge-Based Components, IEEE Tarns.Education, 38(2):171-176, 1995.

[7] D. Fezzani, H. Piquet, H. Foch and Ph. Nogaret, A Few Discussions on Expert System Development for Electrical Power Systems-Optimization of Inverter-Motor of a Railway Traction Chain, Eur. Phys. J. AP 4: 53-64, 1998.

[8] M. H. Rashid, Power Electronics Circuits Devices and Applications, 2nd ed. Prentice-Hall International, Inc, 1993.

[9] W. H. Daniel, Introduction to Power Electronics. Prentice-Hall International, Inc 1997.

[10] O. Bouketir, M. Norman, A. Ishak, M. B. Senan and T. Soib, Expert System-Based Approach to Automate the Design Process of Power Electronics Converters. Proceedings of the International Conference "IEEE/PES T&D Asian Pacific". Oct 2002, Yokohama, Japan, pp: 1943-1946.

[11] O. Bouketir, M. Norman, A. Ishak, M. B. Senan, and T. Soib, Computer Aided Design Tool for Power Electronic Converters, Proceedings of the International Conference ROVISP, Jan 2003, Penang Malaysia, pp 709-716.

[12] B. Omrane, N. Mariun, I. Aris, S. Mahmoud and T. Soib, Knowledge-Based Design Aid Tool for Power Electronic Converters, to appear in *Engineering Computations Journal*, UK.