GALer

COLLABORATORS			
	TITLE : GALer		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

GALer

1.1 GALer

GALer V2.1

Table of Contents

1.2 What is GALer

GALer is a GAL programming device with relevant driver software.
So the project GALer consists of two components, a software and a hardware.

1.3 Shareware

GALer is SHAREWARE. If you use this program and/or the hardware, please send me 20 DM or 15 US $. The circuit diagram for the hardware and the component mounting diagram for the PCB will be sent to you on receipt of the shareware donation.

Send money by postal money order or cash, no checks please! If you send me a disk you will be sent the next update of GALer when it becomes available.

My address is: Christian Habermann
Asamstr. 17
85356 Freising
Germany
EMail: Christian.Habermann@t-online.de

The distribution of GALer on PD-disks or through the networks is permitted, as long as no profit is made from it, and that the included files remain unaltered, and are distributed in their entirety. If you build the GAL-prommer you are allowed to sell this one, to the price of the parts. It is not allowed to sell this product in commercial way. The circuit diagram must only be distributed privately and free of charge.

1.4 Important!!!

WARNING!!! The "GALer" and "GALerTest" programs send data over ↩ your Amiga's parallel port. This means: when you have a printer, digitizer, or something else attached, it should be switched off or disconnected, since it is possible that you may damage them.

I CANNOT BE HELD RESPONSIBLE FOR ANY DAMAGE TO YOUR COMPUTER OR PRERIPHERALS.

The "GALer" hardware and software has been running faultlessly on my A3000.
I have tested GALer on A1000, A3000 and A4000 machines. It should run on all other Amigas too.
The circuit diagram is 100% fault free. With careful construction there is no reason for anything going awry. Even so the construction should be carried out by electronics freak (you should at least have soldering experience, and be able to intelligently read circuit diagrams). More information about the construction can be found in chapter III.

```
Note: This manual sparely covers the operational theory of GAL programming.
It is not a substitute for
                further reading
                .

If you already have long experience with digital circuitry, and really don't
care how a GAL is made, or how it works internally, then this manual together
with the examples should suffice to intelligently implement your GALs.
```

1.5 History

```
  Versions:

    V1.0:    Test version

    V1.1:    Intuition interface added

    V1.2:    cured a few bugs

    V1.3:  - cured some bugs in the GAL assembler. /name.E is no longer allowed.
           - The pin names of the last assembled files can be assigned to the
             indicated GAL.

    V1.4:  - Kick 2.0
           - new Intuition environment
           - support of A- and B-type GALs
           - new format of JEDEC files
           - new functions: verify of programmed GALs
                            test whether security fuse is set or not
                            compare GALs
                            optimizer for Boolean equations
                            reassembler
           - IMPORTANT CHANGE!!!:
              '/' in pin declaration is now considered in the equations

    V1.41  - support of locale.library and gadtools.library
           - req.library replaced by reqtools.library, reqtools.library is
             copyright by Nico Francois
           - Help available

    V1.5   - support of GAL22V10 and GAL20RA10
           - needs WB 2.0 or later
           - detects A- and B-type 16V8 and 20V8 GALs automatically
           - use of #, &, ! for OR, AND and NOT is now possible
           - shortcuts changed
           - external editor can be called from GALer
           - comments in the source files can now be introduced by a ';'

    V1.5a  - this version is equal to version 1.5
             but now there is a layout included (see directory "Layout")

    V1.5b  - improved timing, therefore the access time is slowed down
             significantly
```

```
V1.5c  - bug removed, erasing of 22V10 GALs did not work

V2.0   - new GUI (Magic User Interfac, MUI)
       - bugs in reassembler removed
           translation of GAL22V10 JEDEC files in source codes often failed
           and the handling of negations in tristate enable equations was
           wrong
       - bug in assembler removed
           feedback of GAL22V10 registered outputs was wrong

V2.1   - Optimizer did not work at all in version 2.0 because of a
         typing error in the source code (I formated the source new
         in version 2.0 and by doing this a mistake occured - SORRY).
         Now it seems to be ok again.
```

1.6 Thanks

```
         Thanks to:   - all registerd GALer users

      - Thorsten Elle for the PCB

      - Frank Stange for beta-testing

      - Helmut Hohenwarter for reading the documentation files

      - Nico François for the reqtools.library
        reqtools.library is (c) by Nico François

      - Stefan Stuntz for the Magic User Interface (MUI)
        MUI is (c) by Stefan Stuntz (see
        System Requirements
        )

      - the firm Amiga for the great product Amiga
```

1.7 System Requirements

To use GALer you need an Amiga with at least 512 kByte RAM and
Workbench 2.0 or higher.

Furthermore you need the MagicUserInterface (MUI) which is NOT
included in the GALer package.

Since MUI is Shareware and not Freeware, I'm obliged to print the
following text:

This application uses

```
                          MUI - MagicUserInterface

                  (c) Copyright 1993/94 by Stefan Stuntz


  MUI is a system to generate and maintain graphical user interfaces. With
  the  aid  of  a  preferences program, the user of an application has the
  ability to customize the outfit according to his personal taste.

  MUI is distributed as shareware. To obtain a complete package containing
  lots of examples and more information about registration please look for
  a  file  called  "muiXXusr.lha"  (XX means the latest version number) on
  your local bulletin boards or on public domain disks.

            If you want to register directly, feel free to send


                          DM 30.-  or  US$ 20.-

                                   to

                              Stefan Stuntz
                         Eduard-Spranger-Straße 7
                              80935 München
                                 GERMANY

------------------------------------------------------------------------------
```

1.8 What are GALs?

```
I.1 What are GALs?

GALs (Generic Array Logic) are programmable logic devices. By appropriate
programming by the user, many standard gate functions can be resolved
into a single GAL chip.

Assuming you need the following logic functions for your circuit:

 - AND -Gate with 2 inputs
 - OR  -Gate with 2 inputs
 - NAND-Gate with 2 inputs
 - NOR -Gate with 2 inputs


Normally you would need FOUR standard TTL-ICs. These functions can be
replaced with ONE GAL. The main use of GALs is to make digital circuits as
simple as possible, by replacing many standard logic ICs by one or more
GALs.
A GAL is able, with appropriate programming, to replace all the logic
functions, such as for example: AND, OR, XOR, NAND, NOR, inverters,
FlipFlops, decoders (especially address decoders), mutiplexers, counters.
On top of all the GAL is reprogrammable (at least 100 times), so that the
desired logic functions may easily be altered.
```

1.9 The Intern Structure of GALs

```
I.2 The Intern Structure of GALs

Up to now there are many kinds of GALs. The most common GALs are the
GAL16V8, GAL20V8 and GAL22V10. These GALs are supported by GALer. Furthermore
GALer supports the GAL20RA10 too. The next sections shows you what’s inside
of such a GAL.
```

1.10 The GAL16V8 and GAL20V8

```
I.2.1 The GAL16V8 and GAL20V8

There are several types of GAL16V8 and GAL20V8 GALs, the standard-, A- and
B-type GAL16V8, GAL20V8. The A- and B-type GALs needs less power and are
faster. But since there is no greate difference between them I will only
talk about the standard GALs GAL16V8 and GAL20V8. When there are any
differences between A-, B- and standard-GALs I will mention this extra.

At first the pin designations:

                          GAL16V8
                         ---- ----
        Input or Clock  1|       |20  +5V
                Input   2|       |19  Configurable Output Cell
                Input   3|       |18  Configurable Output Cell
                Input   4|       |17  Configurable Output Cell
                Input   6|       |15  Configurable Output Cell
                Input   7|       |14  Configurable Output Cell
                Input   8|       |13  Configurable Output Cell
                Input   9|       |12  Configurable Output Cell
                  GND  10|       |11  Input or /OE
                         ---------


                          GAL20V8
                         ---- ----
        Input or Clock  1|       |24  +5V
                Input   2|       |23  Input
                Input   3|       |22  Configurable Output Cell
                Input   4|       |21  Configurable Output Cell
                Input   5|       |20  Configurable Output Cell
                Input   6|       |19  Configurable Output Cell
                Input   7|       |18  Configurable Output Cell
                Input   8|       |17  Configurable Output Cell
                Input   9|       |16  Configurable Output Cell
                Input  10|       |15  Configurable Output Cell
                Input  11|       |14  Input
                GND    12|       |13  Input or /OE
                         ---------


From the pin designations you can see that the only difference between the
GAL16V8 and GAL20V8 is the number of inputs. The choice of GAL then, is
```

dependant solely on the number of required inputs.

The essential part of every GAL is a logic-matrix. The input pins of the GAL are connected both inverted and uninverted to the columns of this matrix. If the GAL hasn't been programmed the rows and columns are connected to each other. Every connection between a row and a column represents an AND gate. If the GAL is programmed the particular connections are erased so that the wanted logic is programmed. A row is called a product term, because every column (input) that is still connected to a row represents an AND gate.

Eight of these rows (product terms) of a GAL16V8 or GAL20V8 are connected with the so called OLMC (Output Logic Macro Cell). In this OLMC the product terms are ORed together. The OLMC is a "configurable output cell".

What is a "Configurable Output Cell"?

A GAL16V8 or GAL20V8 contains eight of these configurable output cells. These output cells may be configured as input, combinational output, tristate output, or register output.

```
Combinational Output:   This output can be HIGH or LOW.

Tristate Output:        This output can take one of three states:
                        HIGH, LOW, and HIGH IMPEDANCE
                        This is used if you want to tie two outputs together,
                        but only one may be active.

Register Output:        With this output the result of an equation is not
                        directly coupled to the output, but connected via
                        a D-FlipFlop. Only on receipt of a clock pulse, is
                        the signal passed to the output. When /OE is HIGH,
                        then the output goes HIGH-IMPEDANCE.
```

Besides the matrix, a GAL16V8, GAL20V8 contains extra bits:
((n) means that these bits are available for each output).

```
XOR (n) : The result of the digital connection can be negated with this bit.
        XOR (n) = 0 : Output is active LOW
        XOR (n) = 1 : Output is active HIGH


SYN, AC0, AC1(n):
    These bits determine in which mode the GAL works. Mode means, which type
    of outputs are used (register, tristate,...). There are three main
    operating modes in which the GAL works:

  Mode 1:  SYN = 1, AC0 = 0
        AC1(n) = 1 : OLMC as input
        AC1(n) = 0 : OLMC as combinational output

  Mode 2:  SYN = 1, AC0 = 1
        AC1(n) = 1 : tristate output
```

```
  Mode 3:  SYN = 0, AC0 = 1
        AC1(n) = 1 : OLMC as tristate output
        AC1(n) = 0 : OLMC as register output
```

```
PT0...63: (PT = product term)
      These bits indicate whether the rows (product terms) 0...63 in the
      GAL's matrix are valid or not.
      PTx = 1: AND-junction in the row x is valid.
      PTx = 0: AND-junction in the row x is not used (have no effect) on
               the output.
      (x = between 0 and 63; there are 64 rows in the matrix,
       so that each row can be individually activated or deactivated)
```

All these bits (82 bits) are tied together via the so called Architecture-Control-Word (ACW). The ACW is described in chapter III.

Signature:
Here are eight bytes for your own use. Normally a short comment or a version number of the GAL is placed here.

Security fuse: (Security-Bit)
By setting this bit the GAL can be protected from unauthorized copying. The reading of the logic matrix is no longer possible. Since the rest of the bits can still be read this protection is not very effective.

Bulk Erase:
By programming this row the whole GAL is erased. Now it is possible to program the GAL again. A GAL can be reprogrammed about 100 times.

The Operation Modes of a GAL16V8 and GAL20V8

As already explained the SYN, AC0 and AC1(n) bits determine the mode of the GAL. The pin designations of the GAL are determined by this mode.

GAL16V8:

```
  Mode 1 |  Mode 2  |  Mode 3                 Mode 1  | Mode 2  | Mode 3
  ---------------------------                 --------------------------
         |          |            --- ---              |         |
  In     |    In    |  Clock    1|     |20    +5V     |   +5V   |   +5V
  In     |    In    |    In     2|     |19    In/C    |   T*    |  In/T/R
  In     |    In    |    In     3|     |18    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     4|     |17    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     5|     |16     C      |   I/T   |  In/T/R
  In     |    In    |    In     6|     |15     C      |   In/T  |  In/T/R
  In     |    In    |    In     7|     |14    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     8|     |13    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     9|     |12    In/C    |   T*    |  In/T/R
```

```
  GND     |   GND     |   GND     10|     |11     In      |   In     |   /OE
                                    -------
```

GAL20V8:

```
  Mode 1 |  Mode 2  |  Mode 3                  Mode 1  | Mode 2  | Mode 3
  ---------------------------                  --------------------------
         |          |            --- ---               |         |
  In     |    In    |  Clock    1|     |24    +5V     |   +5V   |   +5V
  In     |    In    |    In     2|     |23        In      |    In   |    In
  In     |    In    |    In     3|     |22    In/C    |   T*    |  In/T/R
  In     |    In    |    In     4|     |21    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     5|     |20    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     6|     |19     C      |   In/T  |  In/T/R
  In     |    In    |    In     7|     |18     C      |   In/T  |  In/T/R
  In     |    In    |    In     8|     |17    In/C    |   In/T  |  In/T/R
  In     |    In    |    In     9|     |16    In/C    |   In/T  |  In/T/R
  In     |    In    |    In    10|     |15    In/C    |   T*    |  In/T/R
  In     |    In    |    In    11|     |14     In     |    In   |    In
  GND    |   GND    |   GND    12|     |13     In     |    In   |   /OE
                                 -------
```

Abbreviations:

```
        In   :  input

        C    :  combinational output without feedback

        T    :  tristate-output

        T*   :  tristate-output without freedback to the matrix, which means
                that this output cannot be configured as input

        R    :  register-output

      Clock  :  pulse for D-FlipFlops; only affects those pins which are
                configured as register-output

      /OE    :  output enable (low active): activate the register-outputs
               (see I.2)
```

From the pin designations you can see that pins 15 and 16 of the GAL16V8 and pins 18 and 19 of GAL20V8 cannot be programmed as inputs when the GAL is in mode 1. The same hold true for pins 12 and 19 and 15 and 22 for mode 2. In mode 1, pins 1 and 11 (GAL16V8) and pins 1 and 13 (GAL20V8) are reserved for Clock and /OE. These pins therefore cannot be used as inputs.

As I told, eight of the rows (product terms) of a GAL16V8 or GAL20V8 are connected via an OR gate to the OLMC. This means that you can use eight product terms to define your output. But when using a tristate output, one row of these eight rows is needed for the tristate control. So you use only seven product terms for tristate output definitions.

Here some sentences to the mode, which should make it easier to you to understand what mode is used:
If you need at least one register output in your GAL, then mode 3 is used. When you need at least one tristate output and no register output, then the GAL will be in mode 2. When you need neither a tristate nor a register output, mode 1 is used.

1.11 The GAL22V10

I.2.2 The GAL22V10

The GAL22V10 is the successor of the GAL16V8 and GAL20V8. This second generation of GALs are much more flexible and better to program.

At first the pin designations:

```
                      GAL22V10
                     ---- ----
      Clock and Input   1|       |24  +5V
                Input   2|       |23  Configurable Output Cell
                Input   3|       |22  Configurable Output Cell
                Input   4|       |21  Configurable Output Cell
                Input   5|       |20  Configurable Output Cell
                Input   6|       |19  Configurable Output Cell
                Input   7|       |18  Configurable Output Cell
                Input   8|       |17  Configurable Output Cell
                Input   9|       |16  Configurable Output Cell
                Input  10|       |15  Configurable Output Cell
                Input  11|       |14  Configurable Output Cell
                GND    12|       |13  Input
                     ---------
```

The GAL22V10 has ten OLMCs, whereas the GAL16V8 and GAL20V8 has "only" eight of them. GAL22V10's OLMCs are not as complex (and complicated to understand) as the OLMCs of the 16V8 and 20V8 GALs. But nevertheless there are less restrictions.

For each OLMC of a GAL22V10 are just two bits which can be programmed, the S0- and S1-bit. The S0-bit does the same as the XOR-bit of the GAL16V8 and GAL20V8. It determines whether an output is active high or active low.
Just for remembrance:

```
        XOR (S0) = 0 : Output is active LOW
        XOR (S0) = 1 : Output is active HIGH
```

The S1-bit does the same as the AC1-bit of the 16V8/20V8 GALs. It determins whether the OLMC is used as registered-output or as tristate output.
For each output and for each type of output you can define a tristate enable. So you can define a tristate enable for registered outputs too. This is another difference to the 16V8/20V8 GALs, because there you can switch just all registered outputs or non to high impetance by use of the /OE pin (operation mode 3).

Pin 1 of the GAL22V10 can be both at the same time a "normal" input and the clock-input for the registers.

There are two additional signals within this GAL, but they are not connected to any pin, they are internal signals. These are AR (asynchronous reset) and SP (synchronous preset). These signals are for the registered outputs which can be controlled by use of the AR and SP. AR resets the registered outputs when it becomes true, independent from the clock-signal at pin 1 of the GAL (asynchronous). SP sets the registered output when it becomes true and when a LOW-HIGH transition of the clock is detected (synchronous).

Another feature of the GAL22V10 is that the amount of rows which are connected to an OLMC is not constant. The number of rows connected to an OLMC is between 9 and 17(!).
Here is a table which shows the exactly number of OLMC's rows:

```
        OLMC at pin |  number of rows from
                    |   the logic-matrix
        ------------+----------------------
             23     |         9
             22     |        11
             21     |        13
             20     |        15
             19     |        17
             18     |        17
             17     |        15
             16     |        13
             15     |        11
             14     |         9
```

You have to consider that for each OLMC one row is needed for the tristate enable again. So you can use 8, 10, 12, 14 or 16 product terms for the output definition.

1.12 The GAL20RA10

I.2.3 The GAL20RA10

The next and last GAL which I want to indroduce is the GAL20RA10. This is also a second generation GAL, but this one is a special one. It is not as universal as a GAL16V8, GAL20V8 or GAL22V10.

Here the pin designations:

```
                         GAL20RA10
                         ---- ----
                  /PL   1|       |24  +5V
                Input   2|       |23  Configurable Output Cell
                Input   3|       |22  Configurable Output Cell
                Input   4|       |21  Configurable Output Cell
                Input   5|       |20  Configurable Output Cell
                Input   6|       |19  Configurable Output Cell
                Input   7|       |18  Configurable Output Cell
```

```
                Input   8|       |17  Configurable Output Cell
                Input   9|       |16  Configurable Output Cell
                Input  10|       |15  Configurable Output Cell
                Input  11|       |14  Configurable Output Cell
                GND    12|       |13  /OE
                         ---------
```

The GAL20RA10 provides ten OLMCs. To each OLMC eight rows of the logic matrix are connected. One row (product term) is needed for tristate control again. Furthermore there are three product terms needed for fully asynchronous control of the register set, reset and clock functions. This means that there are four product terms left for the output definition. The output enable product term is AND'ed with the input from pin 13 (/OE). This allowes either a hard wired external control or a product term control, or a combination of both.

Each OLMC has just one bit which can be programmed, the S0-bit (=XOR). This bit is for the active polarity control. This means that there is no other bit to define whether an output should be a registered or a tristated one. But there is no bit needed to do this: if both is true the product term of asynchronous reset and the product term of asynchronous preset, then the register is switched off and the output becomes a "normal" tristate output.

By use of pin 1, /PL (preload), all registered outputs can be preloaded. This enhances the functional testability of the programmed GAL. To preload a register do the following:

1. supply a high to /PL and to /OE (so the registered outputs become high impedance outputs)

2. impress the desired state on the register output pin

3. pulse low /PL for at least 35ns

After this the registers will be loaded.

1.13 The Software

Chapter II

The Software

In principle you can forget all of chapter I. In principle! What you should remember is the pin designations of the GAL in the different modes. Thereby avoiding many unnecessary failures. The determination of the function mode and the other parameters, which are to be taken into account during the programming of the GAL, are taken care of by the software.

1.14 Source File

II.1 Source File

First a source file has to be created with a text editor. This source file must contain the following information:

1. The GAL type (GAL16V8 or GAL20V8)

2. an 8 byte long comment, which will be written into the GAL as the signature (see I.2)

3. The pin names - here pin numbers are replaced with names, which is easier to oversee.

4. The Boolean equations

5. The keyword DESCRIPTION - after this you can place some desired text. This text generally describes the GAL's function. That way you can know years later what the GAL's intended use was.

Before it's getting boreing here is a first example (be happy).

Example 1:

Assuming you need the following gates in your circuit:

- one AND with 3 inputs
- one NAND with 2 Inputs
- one OR with 2 inputs
- one small digital circuit which feeds the outputs from 2 AND gates to the inputs of one OR gate.

Circuit diagram:
(sorry for the European symbols)

```
    AND:                        +----+                W = A * B * C
                         A ----|    |
                         B ----| &  |---- W
                         C ----|    |
                                +----+

    NAND:                       +----+                /X = D * E
                         D ----|    |
                               | &  |o--- X
                         E ----|    |
                                +----+
    OR:
                                +----+                Y = F + G
                         F ----|    |
                               | >1 |---- Y
                         G ----| =  |
                                +----+

    small  digital              +----+                Z = H * I + J * K
    circuit:             H ----|    |
```

```
             | &  |----+
       I ----|    |    |   +----+
             +----+    +---|    |
                           | >1 |---- Z
             +----+    +---|    |
       J ----|    |    |   +----+
             | &  |----+
       K ----|    |
             +----+
```

```
     Legend:
        * : AND  - connection
        + : OR - Connection
        / : low active
```

In order to create the source file we have to determine which type of GAL will suit our purpose.

For the implementation of the above logic functions, we need a total of 11 inputs and 4 outputs. From chapter I we know (or not?) that the type of GAL is dependant on the number of needed inputs.

The number of inputs in turn depends on the mode (see

pin designations of the various modes

). Since neither tristate nor register outputs are used, the GAL will be in mode 1 after programming.

It therefore follows, the GAL16V8 has 10 inputs and 8 configurable outputs. Since we only need 4 outputs, we can program the rest of the outputs as inputs, so that we obtain the total required 11 inputs and 4 outputs.

Therefore GAL16V8 is adequate for our purposes. GAL20V8 can also be used, but that leaves a lot of unused inputs (WASTE !).

The second thing we need is a signature for the GAL. Remember, it can be up to 8 characters long. For example "example".

Now we have to define the pins. The pins are named one after the other from 1 to 20. Pins that are not used should be named "NC" (not connected), ground with "GND" and +5V with "VCC".

```
here:    B  C   D   E   F  G  H  I  J  GND
         K  NC  NC  NC  Z  Y  X  W  A  VCC

that is:
        Pin  1 := B     Input
        Pin  2 := C     Input
        Pin  3 := D     Input
        Pin  4 := E     Input
        Pin  5 := F     Input
        Pin  6 := G     Input
        Pin  7 := H     Input
        Pin  8 := I     Input
```

```
Pin  9 := J      Input
Pin 10 := GND    Ground
Pin 11 := K      Input
Pin 12 := NC     Not Connected
Pin 13 := NC     Not Connected
Pin 14 := NC     Not Connected
Pin 15 := Z      Combinational Output
Pin 16 := Y      Combinational Output
Pin 17 := X      Combinational Output
Pin 18 := W      Combinational Output
Pin 19 := A      (Configurable Output defined as Input)
Pin 20 := VCC    Voltage Supply
```

```
              GAL16V8
             ---- ----
         B  1|       |20  VCC
         C  2|       |19  A
         D  3|       |18  W
         E  4|       |17  X
         F  5|       |16  Y
         G  6|       |15  Z
         H  7|       |14  NC
         I  8|       |13  NC
         J  9|       |12  NC
       GND 10|       |11  K
             ---------
```

Next come the Boolean equations:

```
W  = A * B * C
/X = D * E
Y  = F + G
Z  = H * I + J * K
```

Therewith we have all the parts required for the source file. Now the question arises, what format does such a file have?:

In line 1 must be the type of GAL. Here "GAL16V8"

In line 2 must be the signature. Here "Example"

Then follow the pin declaration:

```
 B  C   D   E   F  G  H  I  J  GND
 K  NC  NC  NC  Z  Y  X  W  A  VCC
```

Then the Boolean Equations:

```
 W = A * B * C
/X = D * E
 Y = F + G
 Z = H * I + J * K
```

and the keyword DESCRIPTION.

Comments are introduced by a ';'.

Now using a text editor you can create your source file, and save it with the title "example.pld". Don’t forget the extension ".pld".

This is how the file should look:

(**** These characters designate the start and end of the file, please don’t type it.)

```
******************************************************
GAL16V8         ; this is the GAL type
Example         ; this is the signature

B  C   D   E   F  G  H  I  J  GND       ; this is the pin declaration
K  NC  NC  NC  Z  Y  X  W  A  VCC

W  = A * B * C                  ; here are the pin definitions
/X = D * E
Y  = F + G
Z  = H * I + J * K

DESCRIPTION:
here could be a comment which describes the function
of this GAL
******************************************************
```

Negations (’/’) in the pin declaration are considered in the Boolean equations. This means, if you use a ’/’ in the pin declaration and if you use the related pin in a Boolean equation this pin will be negated.

Example:

```
        A  B  /C  D .....  GND
        K  L  M   N .....  VCC

        N = C
```

This is the same:

```
        A  B  C   D .....  GND
        K  L  M   N .....  VCC

        N = /C
```

How do you obtain from this source file a programmed GAL? For that you need the program "GALer" which is described in the following sections.

1.15 Installation

II.2.1 Installation

The installation of GALer is very easy, just start the installation script. Commodore's installer is used, so there should be no problems. Just do what your Amiga requests.

If you start GALer for the very first time, please select your hardware version of GALer by use of the menu "
Project - Hardware Version
"
and save this setting by use of the menu item "
Project - Save config.
".

You can start GALer either from Workbench or CLI.

For informations what GALer needs see
System Requirements
.

1.16 Allgemeines zur Bedienung

II.2.2 How to Use GALer

The usage is easy. For each action corresponding requesters will appear which informs you what's going on and what to do.

The most requesters can be confirmed/refused by pressing the corresponding gadgets or by using the Return/ESC key.

In addition GALer has a online help. So if you want to get help on a menu item, a gadget group or a window, just move the mouse pointer over it and press the HELP key.

GALer uses several files which have different extensions like ".pld", ".jed", ".chp", ".pin", ".fus". If you want to load or save such a file GALer will choose the right extensions automatically. So you don't have to care about it. If you don't enter the postfix, GALer will add it.

1.17 Menus

II.2.3 Menus

Project

```
About GALer

Hardware-Version

Save Config.

Quit
 GAL-Type

GAL16V8

GAL20V8

GAL22V10

GAL20RA10

Type-Requester
 GAL

Program

Copy

Erase

Compare

Blank test

Set security bit

Test security bit

Write access
 GAL-Assembler

Assemble file
 GAL-Disassembler

Read signature

Read ACW

GAL-Info

Generate JEDEC file

JEDEC file parameter

Reassembler
 Tools

Show pin names

Clear pin names
```

GAL-Checker

Select editor

Call editor

Optimizer

Help

1.18 Project

```
About GALer            Tells you who has done all this code. I can tell
                       you, I was it.
```

1.19 Project

```
Hardware-Version       Here you can select which hardware version of GALer
                       you have connected to your Amiga. This selection
                       must be conform with the version of your circuit
                       diagram. Otherwise GALer will not work correctly
                       in some cases.
```

1.20 Project

```
Save config.           Saves some settings to the file "S:GALer.config".
                       Starting GALer next time, GALer will read this file
                       and set your saved settings again.
```

1.21 Project

```
Quit                   Quit quits GALer, but ATTENTION! Handle this function
                       with care: If you use Quit too often, GALer will sell
                       your Amiga's CPU.

                       Are you thinking that I'm pulling your leg?
                       Ohhhh no..., really not!!! I never would do this.
```

1.22 GAL-Type

```
 GAL16V8               Here you can select the type of GAL which should be
 GAL20V8               read/programmed next time.
 GAL22V10
 GAL20RA10
```

1.23 GAL-Type

```
Type-Requester        Every time when GALer wants to read or program a GAL
                      GALer will bring up a requester in which you can
                      select the type of GAL you want to read or program.
                      This selection overrides the selection in the
                      GAL16V8, GAL20V8, ... menus.

                      If you don't want this behavior of GALer, deselect
                      the Type-Requester menu. GALer will then not bring up
                      this requester.
```

1.24 GAL

```
Program               Program a GAL. This is the most importent ←
   function of
      GALer. After selecting this menu, GALer will bring
      up a file requester in which you can select the

JEDEC file
 which should be programmed into the GAL.
```

1.25 GAL

```
Copy                  Copy a GAL. You can only copy a GAL if the
security bit
                      of the source GAL is not set and if the ←
                         destination GAL
      is not programmed.
```

1.26 GAL

```
Erase                 Erase a GAL. If you want to program a GAL the GAL
                      must be erased. Do this using this function.
```

1.27 GAL

```
Compare               There are three different types of ←
   comparison. You
      can compare a GAL with several GALs, a GAL with
      several
JEDEC files
```

```
or a JEDEC file
       with several GALs.
```

1.28 GAL

```
Blank test            Test whether the GAL is erased or not.
```

1.29 GAL

```
 Set security bit      Set the
security bit
 of a GAL. The logic matrix of
       such a protected GAL can't be read out after doing
       this.
```

1.30 GAL

```
 Test security bit     Test whether the
security bit
 of a GAL is set or not.
```

1.31 GAL

```
Write access          This function brings up a requester. In this
                      requester you can select what GALer should do before
                      or after programming, copying or erasing a GAL.

                      programming:
                        - with blank test: before programming a GAL test
                                           whether it is erased or not

                        - with verify    : verify GAL after programming


                      copying:
                        - with blank test: test destination GAL whether
                                           it is erased or not

                        - with verify    : verify programmed destination
                                           GAL

                      erasing:
                        - with blank test: test after erasing a GAL
```

whether it is really cleared
or not

1.32 GAL-Assembler

```
 Assemble file          Assemble a
source file
 (name.pld) and generate
        the
JEDEC file
 and some special files.

        For further information see:
Assembler
```

1.33 GAL-Disassembler

```
 Read signature         Read
signature
 of a programmed GAL and print it
        on the screen.
```

1.34 GAL-Disassembler

```
 Read ACW               Read the
architecture control word
 of a GAL and
        print it on the screen.
```

1.35 GAL-Disassembler

```
 GAL-Info               Gives you some informations about your GAL.
```

1.36 GAL-Disassembler

```
 Generate JEDEC file    Read a GAL and make a corresponding
JEDEC file
.
```

1.37 GAL-Disassembler

JEDEC file parameter   Selecting this menu puts up a requester in ↩
which
you can determine parameters concerning the writing
of
JEDEC files
.

Security bit: If this is enabled, GALer will write JEDEC files in which a special flag is set. Reading this JEDEC file to program a GAL will bring up a requester in which you can choose to set the security bit after programming or not.

Fuse-Checksum: If this is enabled, GALer will write JEDEC files with a checksum calculated over all fuses. Manually changed fuses (with text editor) will be detected by GALer and GALer will warn you that this JEDEC file has been changed when reading this file next time. You are allowed to change comments etc. but you are not allowed to change fuses ('0' and '1').

File-Checksum: If this is enabled, GALer will write JEDEC files with a checksum calculated over all characters in this file. This means that you are not allowed to change anything in this file by using a text editor.

1.38 GAL-Disassembler

Reassembler   This function reads a
JEDEC file
and generates then
the original
source file
. So you can read a unknown
GAL and get back a source file with the Boolean
equations.

1.39 Tools

Show pin names   Prints the pin names of the last assembled source file on the screen.

1.40 Tools

```
Clear pin names        Clears pin names from screen.
```

1.41 Tools

```
 GAL-Checker            There you can check a programmed GAL  ←
    whether it does
        this what you want to do it or not.

        Further information see:
How to Test Programmed GALs
```

1.42 Tools

```
 Select editor          Selecting this will open a window in which ←
     you can
        choose your favorit editor which is called when you
        select the menu item
Call editor
. In a string
        gadget of this window you can enter the editor and
        it's parameter. %s is thereby replaced by the name
        of the last assembled
source file
.
```

1.43 Tools

```
 Call editor            This function starts the
selected editor
 and tries to
        load the last assembled source file in it.
```

1.44 Tools

Optimizer Optimize Boolean equations.

For further information see:
Optimizer

1.45 Tools

Help Explains how to get help on a specific menu item.

1.46 Assembler

II.2.4 Assembler

In order to program a GAL the
source file
(".pld") must be transposed into a
so called
JEDEC file
. This task is assumed by the GAL-Assembler.

The JEDEC file (extension ".jed") is a ASCII file in which all the bits which can be set in a GAL are listed. The state of the fuses (0 or 1) is mediated by the GAL-Assembler from the source file.

Besides the JEDEC file, the GAL-Assembler can generate three other files. These files are for documentation only. GALer do not need them really:

- The Fuse-File (extension ".fus") shows the state of the bits in the logic matrix.

- The chip diagram (extension ".chp") shows the connection diagram of the GAL and the

- pin diagram file (extension ".pin") lists all the pins and shows, whether these are programmed as inputs or outputs.

The files can be read with a text-editor and possibly post-processed.

Selecting the menu
Assemble file
pops up a requester, called
assembler requester. In the assembler requester you can select which files should be generated by the assembler. Just click on the corresponding gadget.

Furthermore you can select two other gadgets:

Autosave: This means that all selected files are generated automatically without bringing up an extra file requester. The name of the generated files are taken from the source file name.

Adjust type of GAL: This means that the type of GAL for which the source file is, is taken over from GALer. For example: You have set a GAL20V8 in GALer's menu. Now you are assembling a source file for a GAL16V8. If the assembly is successful, GAL16V8 will be set in GALer's menu.

Selecting the 'Continue' gadget of the assembler requester will pop up a file requester. Now you have to choose your source file. After this the GAL assembler will start assembling. If GALer detects no errors, further file requesters will pop up in which you have to select where to store the generated files.

1.47 How to Program GALs

II.2.5 How to Program GALs

After the GAL-Assembler has created the JEDEC file from the source file, the GAL can be programmed using this JEDEC file. To initiate the programming of the GAL, simply select Program and give the JEDEC file name. As soon as the GAL is programmed, a requester pops up, and tells you the GAL is programmed, and it is OK to remove the GAL from the programmer's socket.

The steps in programming a GAL:

1. With a text editor create the source file and save this file as "name.pld" (add the extension .pld!)

2. Assemble the source file. As a result of this you'll get a JEDEC file ("name.jed").

3. Select the GAL type

4. Insert the GAL in the programmer's socket

5. Perform the Blank test to verify that the GAL is empty. When the GAL is not empty, then you must first erase the GAL before it can be programmed, use therefore the function Erase

.

6. Initiate programming by selecting
   Program
   .

7. Take the GAL out of the programmer's socket, - DONE !

1.48 How to Test Programmed GALs

II.2.6 How to Test Programmed GALs

Once the GAL has been programmed, the question remains, "does it work the way you envisaged it?". Answering this question is the purpose of the GAL-Checker.

In order to verify the GAL's functions, you must of course first plug the GAL into the programmer's socket, and select the correct GAL-type. Now you can select the menu item GAL-Checker.

In the middle of the screen you'll see a symbolic GAL displayed. In this GAL, you'll see a number of 'I's and 'O's. The 'I' stands for Input and the 'O' for Output.
The 'O' is a gadget. By clicking on the 'O' it turns into an 'I' and clicking on it again it becomes an 'O' again. In other words, you can determine if this pin is to be used as an input or an output.

If a pin is an input, then you can select from another gadget if the pin is to be in a "High" ('H') or in a "Low" ('L') state. An output can assume three states: 'H' (High), 'L' (Low) and~'Z' (high impedance).
The current state is displayed near the corresponding pin.

If you're using the GAL from the
above example
, pin 19
must be defined as an input (="A") by clicking on the 'O' (the one at pin 19), since this pin was defined as an input during programming in the above example.

The inputs of the AND gate are: pin 19 (="A"), pin 1 (="B"), pin 2 (="C").
The output is pin 18 (="W").

If you now set the inputs of the AND gate HIGH (by clicking on the gadgets), the output (=pin 18) should also go HIGH. If it doesn't work or if the output also goes high with other combinations of input levels, then the fault is probably in the source file. The error should be corrected in the source file. The GAL must then be erased and reprogrammed (a GAL can be erased and reprogrammed at least a hundred times). In this manner the whole GAL can be fully tested, and if no errors are detected, it can be used in your circuit.

1.49 Optimizer

II.2.7 Optimizer

Boolean equations can be simplified very often, but for human beings it is a hard way to do. A computer can do this much faster (in most times).

The Optimizer of GALer tries to optimize Boolean equations by use of the Quine-McCluskey algorithm. How this algorithm works can be read in many books which deal with Boolean mathamatics, so it's not described here. I just will described the usage of the Optimizer.

Hm, well, the usage of the Optimizer:

Just select the menu
Optimizer
to start GALer's Optimizer.
After this a file requester pops up. Now you have to select the

source file
which equations you want to be optimized.
After successfully loading this source file GALer starts to optimize the equations.

GALer displays the original equation and the optimized one. If you are happy with the result of the optimization you should select the gadget 'use it'. Then the original equation is replaced by the optimized equation in your loaded source file.

If you don't like the result of the optimization, you should select the gadget 'reject'. Then the original equation is not replaced.

After trying to optimize all equations GALer will pop up a file requester again. Now you have to select a file name of your optimized source file. Please don't use the file name of the original source file for the optimized source file. Give the new file a new name, just like name_opt.pld instead of name.pld.

Example of optimization:

Original Boolean equation:

```
X = /A*/C + A*/C + C*/D + /B*/C + /A*C*D + B*/D
```

By GALer optimized Boolean equation:

```
X = /C + /D + /A
```

Both equations are equal, but the second one is much easier to read.

Not all equations can be simplified. It could be that a "optimized" equation is more complicate than the original one. Just try it.

1.50 JEDEC File

```
                II.3 JEDEC File

JEDEC means (J)oint (E)lectron (D)evice (E)nineering (C)ouncil.
This file is a ASCII file in which every bit which can be set in a
GAL is listed. The JEDEC file has the extension ".jed" and it's
generated by the GAL-Assembler.


The JEDEC file can start with any text until there is a asterisk (*).
The first '*' introduces the command field. The command field starts
with the first '*' and ends at the file end.
Within the command field are... (now be astonished) commands! A command
is introduced by one character and it ends with a '*' character.

All commands are optional. Not every command must be in a JEDEC file.
The
                GAL-Assembler
                 normaly uses: L, F and G commands (see below)


Possible commands are:


 N: This introduces a comment.
    Example:   N this is a comment *

               ^        ^          ^
               |        |          |
          command    any text    end of command



 F: You don't have to list all states of the fuses in the GAL. If you don't
    list all fuses GALer must know what the state of the missed fuses is.

    F0 *: not listed fuses are set to 0
    F1 *: not listed fuses are set to 1



 G: Security Fuse

    G0 *: don't set the security fuse after programming the GAL
    G1 *: ask user (you) whether to set the security fuse after
          programming the GAL or not


 L: L defines the address of a fuse and what the state of the fuse
    should be.

    Example:  L0000  10110111111111111111111111011111 *
```

```
    this means:  set fuse at address 0 to 1
                 set fuse at address 1 to 0
                 set fuse at address 2 to 1
                              .
                              .
                              .

possible addresses are:

    GAL16V8, GA16V8A, GAL16V8B:
      0000-2047: matrix of fuses (AND-array)
      2048-2055: XOR bits
      2056-2119: signature
      2120-2127: AC1 bits
      2128-2191: product term disable bits
      2192     : SYN bit
      2193     : AC0 bit

    GAL20V8, GAL20V8A, GAL20V8B:
      0000-2559: matrix of fuses (AND-array)
      2560-2567: XOR bits
      2568-2631: signature
      2632-2639: AC1 bits
      2640-2703: product term disable bits
      2704     : SYN bit
      2705     : AC0 bit

    GAL22V10
      0000-5807: matrix of fuses (AND-array)
      5808-5827: S0/S1-bits of the OLMCs
      5828-5891: signature

    GAL20RA10
      0000-3199: matrix of fuses (AND-array)
      3200-3209: S0-bits of the OLMCs
      3210-3273: signature


QF: Defines how many fuses in the JEDEC file are. A GAL16V8 has
    2194 fuses and a GAL20V8 has 2706 fuses. Now GALer can
    identify for which type of GAL this JEDEC file is.

    Example: QF2194 *



C: C is followed by a 16 bit hex number which is the fuse checksum of
   the JEDEC file (see description of menu 'JEDEC file parameter').

   Example: C6402 *



<STX>, <ETX>: These are control characters.
   <STX>: 0x02 = CTRL-B
   <ETX>: 0x03 = CTRL-C
```

Your text editor displays this characters in this way:
  <STX> ?
  <ETX> ?
<STX> defines the start of the JEDEC file and <ETX> the end of the JEDEC file. <ETX> is followed by the file checksum (see description of menu 'JEDEC file-parameter'). The file checksum is a 16 bit hex number.

V: V introduces a test vector. GALer 1.4 does not support this. GALer interprets this command as a N command (comment).

1.51 Examples

II.4 Examples

In the next few sections I will show you some examples about GALs, the source file format etc.

1.52 GAL16V8

II.4.1 GAL16V8

At first I want to show an example with a tristate output.
(sorry again for the European symbols)

```
      /A ---------+
                  |\|
                  | \
       B -------|  +------------------------- Y1
                  | /
                  |/
                                    VCC
                                     |
                +----+             |\|
       C -------|    |             | \
                | >1 |-----------|  +-------- Y2
       D -------| =  |             | /
                +----+             |/


                +----+
       E -------|    |
```

```
             | &  |------+
F -------|    |      |
         +----+      |
                    |\|
                    | \
G ------------------|  +o------------- Y3
                    | /
                    |/
```

Y1 should only be in the "B" state, when "A" = LOW. Y2 should always be active (either HIGH or LOW - depending on "B" and "C"). This corresponds to a combinational output. Y3 should only be active if "D" and "E" = HIGH.

```
                     GAL16V8
                    ---- ----
                A  1|       |20  VCC
                B  2|       |19  Y1
                C  3|       |18  Y2
                D  4|       |17  Y3
                E  5|       |16  NC
                F  6|       |15  NC
                G  7|       |14  NC
               NC  8|       |13  NC
               NC  9|       |12  NC
              GND 10|       |11  NC
                    ---------
```

In the source file, tristate outputs are designated with a ".T". The tristate control is followed with an ".E". If the tristate control is absent then the normal free switching is assumed (=VCC). Tristate control = GND means high impedance.

NOTE: With tristate outputs you can only have seven product terms in your equation (all other output formats have a maximum of eight). In the tristate control you can only have ONE product term (no OR) in your equation.

The source file looks like this:

```
******************************************************
GAL16V8
ex.2

A  B  C  D  E  F  G  NC NC GND
NC NC NC NC NC NC Y3 Y2 Y1 VCC

Y1.T = B

Y2.T = C + D

Y3.T = /G

Y1.E = /A
```

```
Y3.E = E * F


DESCRIPTION
*******************************************************



In the last GAL16V8-example we will deal with register outputs. We will
program a 4-bit-counter.

First the pin layout:


                            GAL16V8

                           ---- ----
        (Input)    Clock  1|       |20  VCC
        (Input)       D0  2|       |19  Q0              (Output)
        (Input)       D1  3|       |18  Q1              (Output)
        (Input)       D2  4|       |17  Q2              (Output)
        (Input)       D3  5|       |16  Q3              (Output)
        (Input)      Set  6|       |15  NC              (not used)
        (Input)    Clear  7|       |14  NC              (not used)
        (Input)       NC  8|       |13  NC              (not used)
        (Input)       NC  9|       |12  NC              (not used)
                     GND 10|       |11  /OE             (Input)
                           ---------


Since register output sets the GAL in mode 3, this means that pins 1
and 11 are reserved for Clock and /OE. When /OE is HIGH, all register
outputs (Q0-Q3) go to "high impedance" (=Z).

When LOW-HIGH transition pulse is presented at the clock input, then
the counter will be incremented.

When Clear = HIGH and a (LOW-HIGH) clock transition occurs,
the outputs are cleared.

The inputs D0-D3 can be used to preset the counter. While Set = HIGH
and a Clock pulse the values in D0-D3 are transferred to Q0-Q3.


In the source file register outputs are designated with a ".R".

*******************************************************
GAL16V8                 4-Bit-Counter
Counter


Clock D0    D1    D2    D3    Set   Clear NC    NC   GND
/OE   NC    NC    NC    NC    Q3    Q2    Q1    Q0   VCC
```

```
Q0.R =   /Clear *  Set *  D0
       + /Clear * /Set * /Q0

Q1.R =   /Clear *  Set *  D1
       + /Clear * /Set * /Q1 *  Q0
       + /Clear * /Set *  Q1 * /Q0

Q2.R =   /Clear *  Set *  D2
       + /Clear * /Set *  Q2 * /Q1
       + /Clear * /Set *  Q2 * /Q0
       + /Clear * /Set * /Q2 *  Q1 *  Q0

Q3.R =   /Clear *  Set *  D3
       + /Clear * /Set *  Q3 * /Q2
       + /Clear * /Set *  Q3 * /Q1
       + /Clear * /Set *  Q3 * /Q0
       + /Clear * /Set * /Q3 *  Q2 *  Q1 *  Q0


DESCRIPTION
******************************************************
```

1.53 GAL22V10

II.4.2 GAL22V10

The next circuit can’t be realized with a GAL16V8 or GAL20V8. The reason for this is that there are different tristate controls for the register outputs. A GAL16V8 and GAL20V8 has only one tristate control for all register outputs, this is the /OE (output enable) pin in mode 3.

```
 Enable1 ---------------------------------+
                                          |
                                        |\|
                        +----+          | \
   Clock ---o---------------|>  Q|------|  +------- Y1
            |  +----+       |    |      | /
       A ---+--|    |   +---|D   |      |/
            |  | &  |---+   +----+
       B ---+--|    |
            |  +----+                   |\
            |               +----+      | \
            +---------------|>  Q|------|  +------- Y2
                            |    |      | /
       C -------------------|D   |      |/|
                            +----+        |
                                          |
                            +----+        |
 Enable2 -------------------|    |        |
                            | &  |--------o-------- Y3
                        +---|    |
                        |   +----+
```

```
             +----+      |
      D ----|    |      |
            |    |      |
      E ----| >1 |-----o-------------------------- Y4
            | =  |
      F ----|    |
             +----+
```

Futhermore all register outputs should be reseted asynchronously
when the inputs F and AsyncRe are high and they should be preseted
synchronously when the input SyncPre is high.
(To keep the circuit diagram as simple as possible the inputs
AsyncRe and SyncPre are not drawn in it.)

Don’t think about what the function of this circuit is - there is non.
It is just an example.

Well, and this is one of many possible pin designations:

```
                           GAL22V10

                           ---- ----
                    Clock  1|       |24  +5V
                        A  2|       |23  Y1
                        B  3|       |22  Y2
                        C  4|       |21  Y3
                        D  5|       |20  Y4
                        E  6|       |19  Enable1
                        F  7|       |18  Enable2
                       NC  8|       |17  NC
                       NC  9|       |16  NC
                       NC 10|       |15  NC
                       NC 11|       |14  AsyncRe
                     GND  12|       |13  SyncPre
                           ---------
```

The only thing which you have to keep in mind here is that the clock
signal for the register outputs must be at pin 1 and that the outputs
must be at OLMC pins (pin 14-23).

To define the asynchronous reset and the synchronous preset for the register
outputs the GAL assembler offers you the keywords AR (asynchronous reset) and
SP (synchronous preset). Since AR and SP are keywords it is not allowed to
use them as pin names when a GAL22V10 is used.

The source file of this example looks like this:

```
******************************************************
GAL22V10
22V10

Clock   A       B  C  D  E       F       NC NC NC NC GND
SyncPre AsyncRe NC NC NC Enable2 Enable1 Y4 Y3 Y2 Y1 VCC
```

```
Y1.R  = A * B                  ; Y1 is a registered output => .R
Y1.E  = Enable1

Y2.R  = C
Y2.E  = Enable2 * Y4           ; Attention: there is a feedback of Y4
                               ; (Y4 is defined as output but it’s
                               ; used as input again)

Y3    = Enable2 * Y4           ; there is a feedback again

Y4    = D + E + F


AR    = F * AsyncRe            ; define asynchronous reset

SP    = SyncPre                ; define synchronous preset


DESCRIPTION
******************************************************

Instead of Y3 = Enable2 * Y4 you could also write:

  Y3 = Enable2*D + Enable2*E + Enable2*F.

But using a feedback is much more comfortable than using this long
equation. Another reason for using a feedback is for example the
tristate control of Y2. In tristate controls there is just one
product term allowed, this means no ORs.
```

1.54 GAL20RA10

```
                II.4.3 GAL20RA10

The next example can be realized only with a GAL20RA10. The reason for this
is that each register output needs it’s own clock input and tristate
control. But GAL16V8, GAL20V8 and GAL22V10 offers just one clock input
for all registers and GAL16V8 and GAL20V8 offers just one tristate enable
for all registers.


 Enable1 ----------------------------------+
                  +----+                   |
 ResetA1 ---------|    |                   |
                  | &  |-------+           |
 ResetB1 ---------|    |       |         |\|
                  +----+    +-----+      | \
  Clock1 -------------------|> R Q|------|  +------- Y1
                            |     |      | /
      D1 ---o---------------|D S  |      |/
            |               +-----+
            |                  |
```

```
     Set ---+--o---------------+
            |  |  +----+
            |  +--|    |
            |     | >1 |---------------------------- Y2
            o-----| =  |
            |     +----+                 |\
            |               +------+     | \
  Clock2 ---+---------------|>    Q|------|  +o------ Y3
            |               |      |     | /
      D2 ---+--o------------|D     |     |/|
            |  |            +------+       |
 Enable2 ---+--+---------------------------o
            |  |                           |
            |  |                           |
            |  |  +----+                 |\o
            |  +--|    |                 | \
            |     | &  |o----------------|  +-------- Y4
            +-----|    |                 | /
                  +----+                 |/
```

Don't think about what the function of this circuit is - there is non again.
It is just an example again.

This is one of many possible pin designations:

```
                           GAL20RA10

                           ---- ----
                     /PL  1|       |24  +5V
                     Set  2|       |23  Y1
                 Enable1  3|       |22  Y2
                 Enable2  4|       |21  Y3
                  Clock1  5|       |20  Y4
                  Clock2  6|       |19  D1
                      NC  7|       |18  D2
                      NC  8|       |17  ResetA1
                      NC  9|       |16  ResetB1
                      NC 10|       |15  NC
                      NC 11|       |14  NC
                     GND  12|       |13  /OE
                           ---------
```

Pin 1 and 13 are reserved for /PL (preload) and /OE (output enable), they can't be used for your own. Since Y1 to Y4 are outputs they must be at OLMC pins.

When using a GAL20RA10 the GAL assembler provides three additonal suffixes: .CLK, .ARST and .APRST.

.CLK defines the clock signal for the corresponding register output.
You have to define such a clock signal for EACH register output!

.ARST (asynchronous reset) and .APRST (asynchronous preset) are optinal, you need not to define them for each register output.

For .CLK, .ARST and .APRST is only one product term allowed. For the definition of the output function are four product terms allowed.

Please keep this in mind: when both is true the asynchnonous reset and the synchronous preset the register is swichted off and the output becomes to a "normal" tristate output (see section
The GAL20RA10
).
So the type of the output can be changed "on the fly".

This is source file for this example:

```
******************************************************
GAL20RA10
20RA10

/PL Set Enable1 Enable2 Clock1  Clock2 NC NC NC NC NC GND
/OE NC  NC      ResetB1 ResetA1 D2     D1 Y4 Y3 Y2 Y1 VCC


Y1.R     =  D1                        ; define register output
Y1.E     =  Enable1                   ; define tristate control
Y1.CLK   =  Clock1                    ; define clock for the register
Y1.ARST  =  ResetA1 * ResetB1         ; define async. reset
Y1.APRST =  Set                       ; define async. preset

Y2  =  Set + D1                       ; Y2 is a "normal" output

/Y3.R  =  D2                          ; Y3 is active low and a reg. output
Y3.E   =  Enable2
Y3.CLK =  Clock2

Y4.T   =  /D1 + /D2                   ; Y4 is a tristate output
Y4.E   =  /Enable2


DESCRIPTION
******************************************************
```

1.55 Error Messages

II.5 Error Messages

In this section I want to describe all possible error messages which GALer can create.

1.56 Error Messages

```
II.5.1 Assembler

"Line  1: type of GAL expected"
   The first line of your source file must define for what type of GAL
   this source file is. So the first line must contain one of the
   following keywords: GAL16V8, GAL20V8, GAL16V8A, GAL20V8A


"unexpected end of file"
   Normaly this error occurs when there is no DESCRIPTION keyword
   at the end of your Boolean equations.


"pin name expected after '/'"
   A '/' must be followed by a pin name. If there is a '/' but no pin name
   this error will occur.


"max. length of pin name is 8 characters"
   Pin names are not allowed to be longer than 8 characters.


"illegal character in pin declaration"
   In a pin name is a character which is not allowed to use. Possible
   characters are: a..z, A..Z, 0..9, /


"illegal VCC/GND assignment"
   VCC and GND are keywords. It's not allowed to use these words for
   other pins. Use it only for the pins VCC and GND.


"pn declaration: expected VCC at VCC pin"
   The pin VCC must have the name VCC.


"pin declaration: expected GND at GND pin"
   The pin GND must have the name GND.


"pin name defined twice"
   In the pin declaration a pin name is used multiple.


"illegal use of '/'"
   Negations ('/') must be followed by a pin name.


"unknown pin name"
   Within a Boolean equation is a undefined pin name.


"NC (Not Connected) is not allowed in logic equations"
   NC is a keyword for unused pins. So don't use this in your
```

Boolean equations.

"unknown suffix found"
A '.' must be followed by a T, E, R, CLK, ARST or APRST. This defines

"'=' expected"
A '=' is expected but not found (what else should I say).

"this pin can't be used as output"
You have tried to define a pin as output which can't be used as output.

"same pin is defined multible as output"
It's easier to show an example:

```
X = ...
X = ...
```

This brings up this error message.

"before using .E, the output must be defined"
You have defined a Boolean equation for tristate enable but there was no Boolean equation for the trisate output.
The order must be:

```
name.T = ...
name.E = ...
```

Possibly you have done:

```
name.E = ...
name.T = ...
```

this is wrong!

"GAL22V10: AR and SP is not allowed as pinname"
When using a GAL22V10 AR and SP are keywords for the asynchronous reset and synchronous preset. So AR and SP is not allowed to be used as pin names.

".E, .CLK, .ARST and .APRST is not allowed to be negated"
The definitions for tristate control,... can't be negated.

"mode 2: pins 12, 19 can't be used as input"
The GAL would be in mode 2. In this mode you can't define the pins 12 and 19 as input pins. These pins do not have a feedback too. This means that the following equation is not allowed.

```
a := pin 19
b := pin 4
y := pin 17

a = b                a is output, b is input
y = a * b            y is output
                     a is used as input, this is not allowed in mode 2
```

```
                        because there is on feedback

"mode 2: pins 15, 22 can't be used as input"
   The GAL would be in mode 2. In this mode you can't define the pins
   15 and 22 as input pins. These pins do not have a feedback too. This
   means that the following eqauation is not allowed.

     a := pin 22
     b := pin 4
     y := pin 17

     a = b              a is output, b is input
     y = a * b          y is output
                        a is used as input, this is not allowed in mode 2


"tristate control is defined twice"
   Example:    name.E = A * B
               name.E = C
   this is not allowed!


"GAL16V8/20V8: tri. control for reg. output is not allowed"
   When using a GAL16V8/20V8 it is not possible to define a tristate
   control for registered outputs.


"tristate control without previous '.T'"
   There is a tristate control for a combinational output.
   wrong:       name   = ...
                name.E = ...

   right:       name.T = ...
                name.E = ...


"use GND, VCC instead of /VCC, /GND"
   I think there is nothing to explain.


"mode 3: pins 1,11 are reservated for 'Clock' and '/OE'"
   Using register outputs causes mode 3 for the GAL. In this mode the
   pins 1 and 11 of a GAL16V8 can't be used by your own. These pins are
   reserved for Clock and /OE.


"mode 3: pins 1,13 are reservated for 'Clock' and '/OE'"
   Using register outputs causes mode 3 for the GAL. In this mode the
   pins 1 and 13 of a GAL20V8 can't be used by your own. These pins are
   reserved for Clock and /OE.


"use of VCC and GND is not allowed in equations"
   Expressions like  "X = A * VCC" are not allowed (and not necassary).


"tristate control: only one product term allowed (no OR)"
```

In Boolean equations for tristate controls only one product term can be used. This means no ORs in your name.E=... equation.

"too many product terms"
In this definition are too many product terms.

"use of AR and SP is not allowed in equations"
AR and SP are keywords which can’t be used in output definitions.

"negation of AR and SP is not allowed"
AR and SP definitions can’t be negated.

"no equations found"
Sorry, but there are no Boolean equations in your source file. So GALer does not know what to do with your source file.

".CLK is not allowed when this type of GAL is used"
A clock definition is only allowed when a GAL20RA10 is used.

".ARST is not allowed when this type of GAL is used"
A .ARST definition is only allowed when a GAL20RA10 is used.

.APRST is not allowed when this type of GAL is used
A .ARPST definition is only allowed when a GAL20RA10 is used.

"GAL20RA10: pin 1 can’t be used in equations"
Pin 1 of the GAL20RA10 is reserved for the preloading /PL.

"GAL20RA10: pin 13 can’t be used in equations"
Pin 13 of the GAL20RA10 is reserved for the output enable /OE.

"AR, SP: no suffix allowed"
It’s not allowed to add a suffix to AR, SP definitions.

"AR or SP is defined twice"
A AR or SP definition is defined twice.

"missing clock definition (.CLK) of registered output"
When using a GAL20RA10 all registered outputs must get a clock definition.

"before using .CLK, the output must be defined"
At first you have to define the registered output by using .R before you can define the clock for this output.

```
"before using .ARST, the output must be defined"
   At first you have to define the registered output by using .R before
   you can define the asynchronous reset.


"before using .APRST the output must be defined"
   At first you have to define the registered output by using .R before
   you can define the asynchronous preset.


"several .CLK definitions for the same output found"
   You have defined more than one clock definition for the same output.


"several .ARST definitions for the same output found"
   You have defined more than one asynchronous reset definition for the
   same output.


"several .APRST definitions for the same output found"
   You have defined more than one asynchronous preset definition for the
   same output.


"use of .CLK, .ARST, .APRST only allowed for registered outputs"
   Well, use of .CLK, .ARST and .APRST is only allowed for registered
   outputs :-)
```

1.57 Error Messages

```
II.5.2 JEDEC File

"unexpected end of file"
   The last thing in a JEDEC file should be either the file checksum or
   a '*'.


"unknown command found"
   There is a unknown command in your JEDEC file (see chapter JEDEC File for
   possible commands). In most cases the reason for this error message
   is a missing '*'.


"bad format of number"
   A dez. or hex. number is expected and found, but there are illegal
   characters in it.
   Example:  C1a#3


"number expected after command"
   After this command a dez. or hex. number is expected but not found.
```

"0 or 1 expected"
Fuses can be set to 0 or 1. Using another digit will cause this error.

"can’t find out type of GAL"
GALer can’t identify for which type of GAL (GAL16V8 or GAL20V8) this JEDEC file is. But GALer must know this in order to program a GAL.

"QF multible found"
In the JEDEC file the command QF is found multiple. This is not allowed.

"QP multible found"
In the JEDEC file the command QP is found multiple. This is not allowed.

"ending ’*’ expected"
GALer expects a ’*’ character.

"after ’C’ command no ’L’ command allowed"
After the fuse checksum no change of fuses (L command) is allowed.

"bad fuse checksum"
The fuse checksum is bad. The reason for this can be that you have changed some state of fuses with a text editor.

"too many <STX> (= CTRL-B, 0x02) found"
The <STX> control character should be once at the beginning of the JEDEC file.

"too many <ETX> (= CTRL-C, 0x03) found"
The <ETX> control character should be once at the end of the JEDEC file.

"bad sequence of <STX>, <ETX>"
The first control character must be a <STX> then a <ETX> not vice versa.

"after file checksum end of file expected"
There is a character after a file checksum which is not a Space, TAB or Carriage Return. This is not allowed.

"bad fuse address (L... too short)"
It’s easier to show an example:

```
L0010 1011*
L0013 0111*
```

Address 13 is defined twice.

```
"addresses skiped but no default value (F0/1*) defined"
   It's easier to show an example:
          L0010 11*
          L0015 01*
    The fuses of the addresses 12, 13 and 14 are not defined and there
    is no F command which would define the values of missing fuses.


"'*' expected"
   '*' expected but not found (what else should I say here).


"QF... and last fuse address (L...) are not equal"
   QF defines the number of fuses in this JEDEC file (GAL16V8: 2194,
   GAL20V8: 2706). If the last fuse of a L command does not match
   to the QF command, this error will occur.


"no values for the fuses found (no F0/1, L...)"
   In your JEDEC file are no fuses defined. Such a file is useless
   and therefore rejected by GALer.


"only QF2194 *, QF2706 *, QF3274 *, QF5892 * allowed"
   There is a QF... command which fits to no GAL which is supported by
   GALer.

"too many fuses found"
   In your JEDEC file are too many fuses defined.


"found several fuse checksumms"
   In your JEDEC file are several fuse checksumms. This is not allowed.


"selected type of GAL fits not to JEDEC file"
   You have selected a to-be-programmed-GAL which does not fit to
   the JEDEC file.
```

1.58 Error Messages

```
II.5.3 Reassembler

"mode AC0 = SYN = 0 is not supported"
   In the JEDEC file the bits AC0 and SYN are set to 0. This mode
   is not supported by GALer.


"Pin xx: pin name defined twice"
   A pin name is used for more than one pin.
```

```
"Pin xx: illegal character"
   Legal characters are  : digits, letters and the '/'
   Illegal characters are: Space, #, *, ...


"Pin xx: no pin name found"
   There is no name for the pin xx defined.


"Pin xx: VCC/GND at wrong pin"
   VCC and GND must be the pin names for the VCC and GND pin of the GAL.


"Pin xx: illegal use of '/'"
   Usage of the negation character: /pinname
   Illegal:  pinname/, /, //pinname etc.


"Pin xx: GND expected"
   Pin 10 of GAL16V8 respectively pin 12 of GAL20V8 must be defined as GND.


"Pin xx: VCC expected"
   Pin 20 of GAL16V8 respectively pin 24 of GAL20V8 must be defined as VCC.


"Pin xx: AR is not allowed as pin name"
   When using a GAL22V10, AR is a keyword which is not allowed as pin name.

"Pin xx: SP is not allowed as pin name"
   When using a GAL22V10, SP is a keyword which is not allowed as pin name.
```

1.59 Programming GALs

```
                III.1 Programming GALs

The first question that arises is how can you program a GAL when all the
pins are already defined and no pins are free for the programming?

If you apply a voltage of 12.00 Volt up to 16.5 Volt (depends on GAL type
and on that whether the GAL should be read or programmed) to pin 2, then
the pin description changes, the GAL is then in the Edit mode.

In this section I'll explain how a GAL16V8 and GAL20V8 is read and
programmed. The reading/programming algorithm of the GAL22V10 and GAL20RA10
is similar to that of the GAL16V8 and GAL20V8 so I'll explain only the
GAL16V8 and GAL20V8.

Here the pins in the Edit mode:

                                GAL16V8
                                ---- ----
                         VIL   1|       |20  +5V
                        EDIT   2|       |19  P,/V
```

```
                RAG1   3|       |18  RAG0
                RAG2   4|       |17  VIL
                RAG3   5|       |16  VIL
                RAG4   6|       |15  VIL
                RAG5   7|       |14  VIL
                SCLK   8|       |13  VIL
                SDIN   9|       |12  SDOUT
                GND   10|       |11  /STR
                        ---------


                        GAL20V8
                        ---- ----
                 VIL   1|       |24  +5V
                EDIT   2|       |23  VIL
                RAG1   3|       |22  P,/V
                RAG2   4|       |21  RAG0
                RAG3   5|       |20  VIL
                 VIL   6|       |19  VIL
                 VIL   7|       |18  VIL
                RAG4   8|       |17  VIL
                RAG5   9|       |16  VIL
                SCLK  10|       |15  SDOUT
                SDIN  11|       |14  VIL
                 GND  12|       |13  /STR
                        ---------
```

Whether the GAL is to be read from or written to is determined by the level of P,/V. A HIGH means write, a LOW read.

The to be read/written addresses are presented to pins RAG0-RAG5. The programming occurs as follows:

After giving the addresses to RAG0-RAG5, the to be written bits have to be presented to SDIN (serially) and by clocking SCLK with a LOW-HIGH-transition the data is transferred to an internal shift register. A LOW-pulse on the /STR pin programs the addressed row. This repeats until the whole GAL is programmed.

The reading of a GAL proceeds similarly: After presenting the addresses to RAG0-RAG5 the bits of the corresponding address are put into the internal shift register by clocking /STR with a LOW-pulse. By clocking SCLK with a LOW-HIGH-clock transition all the various bits are sent out the SDOUT pin. The bit width of an address determines the number of SCLK pulses required to complete the programming or reading the address.

VIL means Input Voltage Low. These pins must be connected to ground or LOW when the GAL is in the Edit mode.

GALs do have different algorithm codes. These codes determine the parameters Edit mode voltage and STR pulse width. GALer supports the algorithm codes 0 to 4. The function

GAL-Info

of GALer returns the algorithm code of the inserted GAL. This code is not very importent for you, but GALer needs this code for reading and programming GALs.

	READ		PROGRAM	
Algorithm	Edit mode read voltage	STR pulse	Edit mode prog. voltage	STR pulse
0	12 \ensuremath{\pm} 0,25 V	5 us	15,75 \ensuremath{\pm} 0,25 V	80 \ensuremath{\pm} 5 ms
1	12 \ensuremath{\pm} 0,25 V	5 us	15,75 \ensuremath{\pm} 0,25 V	80 \ensuremath{\pm} 5 ms
2	12 \ensuremath{\pm} 0,25 V	5 us	16,50 \ensuremath{\pm} 0,25 V	10 \ensuremath{\pm} 1 ms
3	12 \ensuremath{\pm} 0,25 V	5 us	14,50 \ensuremath{\pm} 0,25 V	40 \ensuremath{\pm} 5 ms
4	12 \ensuremath{\pm} 0,25 V	5 us	14,00 \ensuremath{\pm} 0,25 V	100 \ensuremath{\pm} 5 ms

To erase a GAL you have to apply HIGH to P/V then pulse STR low for 100 ms and then apply LOW to P/V. After this the GAL is erased and ready to be programmed again.

The internal organization of the GAL (addresses of the parts) looks as follows:

GAL16V8, GAL16V8A,B:

Address		Width
0-31	Fuse-Matrix	64 Bit
32	Signature	64 Bit
33-59	reserved space	64 Bit
60	Architecture-Control-Word ACW	82 Bit
61	Security bit	
62	reserved	
63	Bulk Erase	

GAL20V8, GAL20V8A,B:

Address		Width
0-39	Fuse-Matrix	64 Bit
40	Signature	64 Bit
41-59	reserved space	64 Bit
60	Architecture-Control-Word ACW	82 Bit
61	Security bit	
62	reserved	
63	Bulk Erase	

```
The Architecture-Control-Word has the following structure (82 Bit wide):


GAL16V8:
Bits  0-31: 32 bit product term enable 0-31
Bits 32-35: 4 Bit XOR(n) for OLMC pins 19-16
Bit     36: AC0-Bit
Bits 37-44: 8 Bit AC1(n) for OLMC pins 19-12
Bit     45: SYN-Bit
Bits 46-49: 4 Bit XOR(n) for OLMC pins 15-12
Bits 50-81: 32 Bit product term enable 32-63

GAL16V8A,B:
Bits  0-3: 4 Bit XOR(n) for OLMC pins 19-16
Bit     4: AC0
Bit   5-8: 4 Bit AC1(n) for OLMC pins 19-16
Bit  9-72: 64 Bit product term enable PT0 - PT63
Bit 73-76: 4 Bit AC1(n) for OLMC pins 15-12
Bit    77: SYN
Bit 78-81: 4 Bit XOR(n) for OLMC pins 15-12


GAL20V8:
Bits  0-31: 32 Bit product term enable 0-31
Bits 32-35: 4 Bit XOR(n) for OLMC pins 22-19
Bit     36: AC0-Bit
Bits 37-44: 8 Bit AC1(n) für OLMC pins 22-15
Bit     45: SYN-Bit
Bits 46-49: 4 Bit XOR(n) für OLMC pins 18-15
Bits 50-81: 32 Bit product term enable 32-63

GAL20V8A,B:
Bits  0-3: 4 Bit XOR(n) for OLMC pins 22-19
Bit     4: AC0
Bit   5-8: 4 Bit AC1(n) for OLMC pins 22-19
Bit  9-72: 64 Bit product term enable PT0 - PT63
Bit 73-76: 4 Bit AC1(n) for OLMC pins 18-15
Bit    77: SYN
Bit 78-81: 4 Bit XOR(n) for OLMC pins 18-15
```

1.60 Circuit Description

III.2 Circuit Description

In the following section I'll describe the functioning of my GAL-programming device. I'll refer to my circuit diagram, so if you haven't ordered that, you can skip this section.

The hardware is connected to the Amiga's parallel port. The connected data signals are D0-D4 and the BUSY-Line.

IC1, IC3, IC4 and IC5 are eight way "serial in/parallel out" shift registers. The outputs of the shift-register from IC3, IC4 and IC5 are connected to the Textool-Zero insertion force socket for the GAL. Therefore it is possible to (besides VCC and GND) define each of the GAL's pins with a level (HIGH or LOW).

The possible outputs of the GAL (pin 14 to 23) can be read through IC7 and IC6. IC7 is an eight way "parallel in/serial out" shift register.

IC1 is so to speak the switch centre, this IC selects IC3, 4 and 5 (OE). Furthermore, this IC switches the programming voltages for the GAL (VCC, Edit-voltage) on or off.

Since the ICs 1, 3, 4, 5, 7 can be individually accessed, a separate clock line is provided for each IC. These clock lines are selected via IC2, a 1 out of 4-decoder, by the parallel port's data lines D0 and D1.

D3 determines whether a read (low) or write (high) operation is to occur. It must be ensured that D3 does not go low until the IC to be accessed is selected through D0 and D1. Otherwise an IC gets an unwanted clock-pulse and at the next Strobe-pulse (D2) the wrong (once left shifted) data is presented at the outputs.

The Strobe-pulse for the shift-register is derived from D2 . When D2 goes high, the data in the shift registers is transferred to the output registers of ICs 1, 3, 4, 5.

Through IC7, D2 can be made high (D2 = high), so that the data on Pins P1-P8 are transferred to the internal shift register and may be read through the BUSY line by clocking the relevant clock-line.

D4 transfers the individual bits from the Amiga to the GAL-Burner.

Since Pins 2 and 4 of the Textool-socket may be supplied with the programming voltage of up to 16.5 Volt, we have to protect IC4 with the diodes D2 and D3 against over voltage.

The programming voltage is derived from IC9, a switch mode voltage regulator. This voltage can be precisely adjusted with the trimpots R40-R44.

The relay K1 connects the supply voltage of the GAL to pin 24 of the Textool-Socket, Relay K2 connects the output Q7 from IC3 or +5V supply voltage (according to the GAL type) to Pin 22 of the Textool-Socket. Both relays are driven by IC1.

The LED shows whether voltage is supplied to the Textool-Socket or not (controlled by software). When the LED is on, a GAL may not be inserted or removed from the socket.

Parallel-Port:

```
 D0-D1:  Selection of individual ICs by Clk
 D2:     Strobe-pulse for IC 1, 3, 4, 5
 D3:     write = low, read = high
```

```
 D4:     Data line for "Write Bits"
 BUSY:   Data line for "Read Bits"

IC1:
 Q1   make 16.5V (but don't switch it!)
 Q2   switch edit-current on pin 2 for GAL20V8
 Q3   switch edit-current on pin 4 for GAL16V8
 Q5   switch Vcc
 Q6   OE for IC3, 4, 5
 Q7   controlles LED on : high, low = off
 Q8   not used

IC3:
 Q1-Q8  pin 16-23 of the Textool-socket over R3-R8

IC4:
 Q1-Q8  pin 1-8 of the Textool-socket

IC5:
 Q1-Q3  pin 9-11 of the Textool-socket
 Q4     pin 13 of the Textool-socket
 Q5-Q6  pins 14, 15 of the Textool-socket over R9, R10

IC6:
 a      read level at pin 13 of the Textool-socket
 b      read buffer of IC7
 c      read level at pin 23 of the Textool-socket

IC7:
 P1-P8  read level at pin 14-21 of the Textool-socket
```

1.61 Construction

III.3 Construction

The parts list can be found in the appendix. If you want to save the cost of the Textool-socket, you can also use a normal 24 pin socket, but since the two pin rows are too far apart, you will have to carefully cut the socket along it's length, and set them to the correct distance. Or you might like to use "MOLEX" pins instead. For IC-sockets you should only use precision sockets. The 25 Pin Sub-D-socket (A1000) or the Sub-D-plug (other Amiga models) is connected as follows:

```
  D0   = pin 2
  D1   = pin 3
  D2   = pin 4
  D3   = pin 5
  D4   = pin 6
  BUSY = pin 11
  GND  : A1000  pin 14
         A500, A2000, A3000, ...  pin 17
```

The supply voltage (+5V and ground (GND) you will have to get from the

Amiga’s Expansion-Port or from a regulated +5V power supply. The GAL-Burner has a maximum current consumption of about 220 mA.

The main difficulty in the construction of the GAL-burner is probably the printed circuit board (PCB). Most likely a double sided through plated PCB is required, so probably the best solution is to use a piece of veroboard, and connect the relevant pins with fine hookup wire, or wire wrap wire. If you want the whole thing to look impressive, you can use wiring guides. The wires can be conducted via these guides, and wont go here there an everywhere. I have constructed my GAL-Burner and endless other projects in this way.

If you are able to make a PCB, you can find the PostScript- or Gerber-Files in the directory "Layout". This PCB-layout is copyright by Thorsten Elle. Thank you Thorsten for giving me the permission to distribute your layout together with GALer.
If you want to use the PCB-layout, please not this:
On the PCB the resistors R28 to R32 of the circuit diagram are replaced by a resistor-array (8 * 1.8kOhm). Futhermore there is for each IC a capacitor for anti-interferencing. These capacitors are C5 to C14 (each 100nF).

When you have built the circuit and you have connected it to the Amiga, and the Amiga DIDN’T blow up, you can run the test program "GALerTest". The program set various voltage levels on the Textool-socket, which you can check with a VOM. The programming voltages are adjusted with R40-R44. For this you can use "GALerTest".

```
DON'T FORGET: adjust the programming voltage with the trimpots, otherwise
              the GAL-burner will NOT FUNCTION !!!!
              (start the "GALerTest" and click through to the
               relevant test points).
```

If the GAL-Burner works just as the test program demands, then you can try burning a test GAL and test it with the GAL-Checker function of GALer. If this all works then there are no faults in the hardware. GALs of the type GAL16V8 must be inserted in the Textool-socket, so that pin 1 of the GAL lines up with pin 3 of the socket. With GALs of the type GAL20V8 pin 1 of the GAL must line up with pin 1 of the socket.

Here is the pin layout of the used transistors (bottom view):

```
                       BC 237 and BC 327
                             _
                            -   -
                           -     -
                           -o o o-
                            -----

                            E B C
```

Well that’s about it. Chau and have fun with the cute GALs.

1.62 Keywords of the Source File

```
Keywords of the Source File

GAL16V8                         designates the GAL type
GAL20V8
GAL22V10
GAL20RA10

NC                              not connected (unused) pin
GND                             GROUND (=LOW)
VCC                             +5V    (=HIGH)
.T                              output pin is tristate output
.E                              tristate enable through product term
.R                              output pin is register output
=                               output pin is given an equivalence
+                               OR
*                               AND
/                               NOT
DESCRIPTION                     indicates the end of the Boolean equations
```

1.63 List of Parts

```
List of Parts:


        ICs:

         IC1, IC3, IC4, IC5   :   4 x   4094
         IC2                  :   1 x   4555
         IC6                  :   1 x   4503
         IC7                  :   1 x   4021
         IC8                  :   1 x   74LS06
         IC9                  :   1 x   TL 497
         IC10                 :   1 x   74LS145

        Diodes:

         D1-D4                :   4 x   1N4148
         LED                  :   1 x   rot, 3 mm

        Transistors:

         T2, T4, T5           :   3 x   BC237B
         T1, T3               :   2 x   BC327
```

```
Resistors (5%, 1/4 Watt):

 R1, R2, R35-39       :   7 x   1   KOhm
 R3-13, R19-26        :  19 x   10  KOhm
 R28-32               :   5 x   1,8 KOhm
 R14                  :   1 x   1   Ohm
 R15                  :   1 x   27  KOhm
 R34                  :   1 x   220 Ohm
 R18                  :   1 x   4,7 Ohm
 R27                  :   1 x   47  Ohm
 R33                  :   1 x   22  KOhm
 R40-44               :   5 x   2   KOhm trimpots
 (there are no R16, R17)

Relays:

 K1, K2                :   2 x  relays, e.g. reed-relays,
                                1 x single pole double throw

Coil:

 L1                   :   1 x   100 uH, miniature fixed


Capacitors:

 C1                   :   1 x   100 pF
 C2                   :   1 x   4,7 uF, tantalum
 C3                   :   1 x   100 nF
 C4                   :   1 x   220 uF, 25V


Sundries:

 IC-Sockets
   7 x 16 pin
   3 x 14 pin
   1 x Textool-Socket 24 pin, narrow! (.3 row spacing) or use
       a universal type

   1 x 25 pin female Sub-D connector for Amiga 1000
       or 25 pin male Sub-D connector for all other Amiga types
```

u = mikro

Please note this: If you use the PCB-layout, replace R28 to R32 by a resistor-array 8 * 1.8kOhm. Furthermore you can add ten anti-interference capacitors C5 to C14 (each 100nF). These capacitors are not drawn in the circuit diagram, but you can see them in the component mounting diagram.

1.64 Further Reading

Further Reading

1) GALs - Programmierbare Logikbausteine in Theorie und Praxis
   Bitterle
   Franzis-Verlag

2) Programmable Logic Manual - GAL Products
   SGS-Thomson

3) GAL Handbook
   Lattice

1.65 Index

VIL
W

Write Access
X

XOR