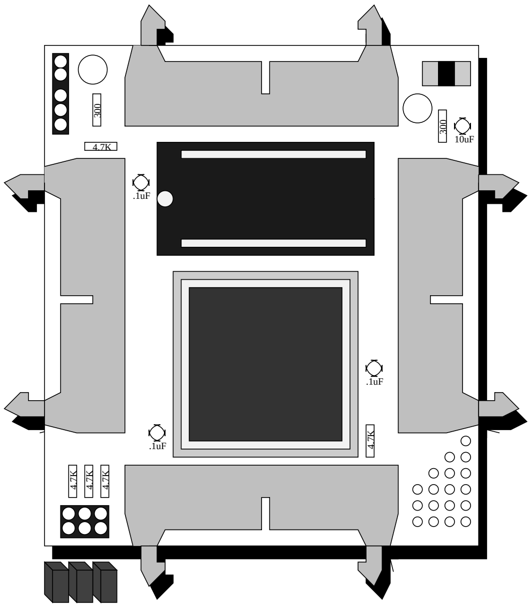
GEneric Reusable Module (GERM)



Low Cost Generic Reusable Module for the Rapid Prototyping of Subsystems

User's Guide

Version 2.0.2 *

J. D. Sterling Babcock, Apostolos Dollas and Brent Ward
Department of Electrical Engineering
Duke University
Durham, NC 27706

September 26, 1994

*Version 1.0 by Brent Ward

1 What the GERM is.

The GEneric Reusable Module (GERM) is a low cost educational prototyping tool. Each GERM has an FPGA and support for multiple cable connectors. GERMs singularly, or multiply, can be used to develop curriculum exposure to FPGA technology with extensions into Rapid System Prototyping and FPID technologies. The GERM provides users with the resources needed to quickly prototype both basic and somewhat complex subsystem circuit designs without the arduous task of wiring. More technical discussions of the GERM is given in [2] and in [3].

GERM Uses: State Machines, Combinational Circuitry, Glue Logic, Bitslice Compute Elements, Programmable Interconnect

Limitations: Manual Partitioning, Input/Output Limitations, No Protective Circuitry, Comparison to Other Boards

Capabilities: Easy-to-Use, Schematic or Behavioral Design, Stand-Alone or Tethered Loading, Resources Required: Personal Computer, Xilinx Tool Set, Download Cable or EEPROM

It is not expected that the GERM will replace other modes of teaching or prototyping, but merely supplement the development of projects and larger systems through the interchangeability and vast resources that reprogrammable components offer to students and designers. By building the GERM we have provided the "hooks" for students to quickly and easily access the FPGA and its capabilities through standardized software and hardware programming systems. We realistically do not expect ASIC level systems to be built out of the GERMs but the capability for large designs does exist if desired.

2 History of the GERM.

The GERM, originally GRM, was designed by Brent Ward as the result of a Master's Thesis at Duke University entitled "An Experimental Study in Subsystem Reusability Toward Rapid Prototyping of Microelectronic Systems", under the supervision of Dr. Apostolos Dollas [3].

The GERM was fabricated under a grant from MOSIS through the MULTEK fabrication facility in California. The population of the board has been supported by donations from Advanced Micro Devices (EPROMs) and Xilinx, Inc. (FPGAs). Sockets and other hardware materials have been obtained through the local Newark Electronics office.

3 Description of the GERM.

The GERM is basically a 3"x3.5" 2-layer printed circuit board, with a Xilinx FPGA surrounded by 4 quick connect headers (see Figure). The GERM has the capability to be programmed from on-board reprogrammable memory (EPROM) or through a standard download cable provided with Xilinx software. A small prototyping area has been provided in the lower right corner of the PCB for the addition of crystals or LEDs of any sort that might be desired in future applications.

In an introductory level class the GERM is expected to be used for basic circuits that introduce the student to basic concepts and also the relative usefulness of reprogrammable/programmable resources. Eventually larger designs can be attempted. The natural extensions in an Introductory level class would be outgrowth into external signal considerations (LED's, switches, timers, control, etc.). Thus, the GERM can be used to supplement basic and introductory level digital design courses where the emphasis is on creating circuits that behave in a specified manner. The student's time can then be shifted from spending time wiring discrete gates together, to designing more complex circuitry while the chip concerns itself with wiring the proper resources together.

Ultimately, the effect of the GERM is that of a LEGO™-like building block that can take on any digital shape the student or instructor wishes it to take on. And then multiple GERMs can be connected together to create yet a larger system. For instance, if a small CPU system were desired, different GERMs could act as ALU, Address Unit, Cache, and Instruction Register which could all be connected in a square structure as shown in Figure 1 or they could be connected through another GERM as depicted in 2.

The benefit of the "cross" architecture depicted in Figure 2, is that the connections between the units resides in the center GERM since it is being used as a router or FPID (Field Programmable Interconnect

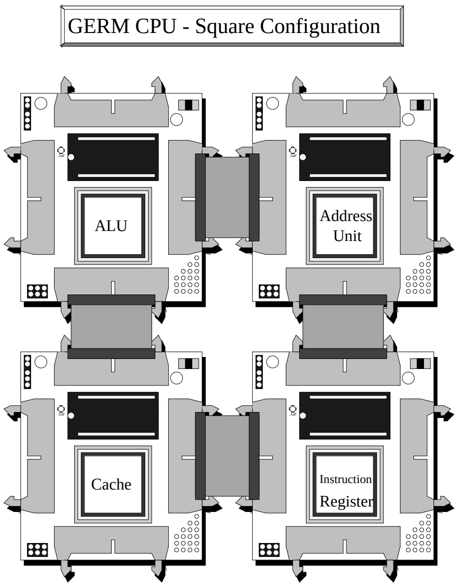


Figure 1: Interconnect for square topology CPU using GERMs

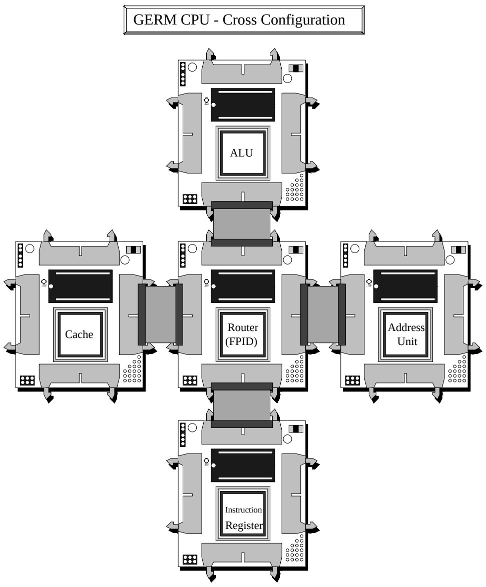


Figure 2: Interconnect for cross topology CPU using GERMs

Device) in a sense. Thus a natural introduction to FPIDs could be accommodated in the curriculum at the point in a design class where such a situation might arise.

4 Materials Needed to Assemble the GERM.

Depending on the level of development needed in the curriculum or research environment, the GERM may be configured to meet the exact needs of the user. The most basic, and still useable, configuration of the GERM is termed the “Yugo” model and is the least expensive, and least configurable, construction. It is used for tethered serial download of the FPGA configuration.

The fully functional, also most expensive, configuration of the GERM is termed the “Ferrari” model. It is used for either tethered serial download or untethered parallel download of the FPGA configuration, the latter from an EPROM.

Intermediaries and derivatives of between the Ferrari and the Yugo may also be configured for a given application but are left to the user to develop. Just the extreme ends of the configuration options will be discussed in detail below.

4.1 Ferrari Model

The full blown “Ferrari” version of the GERM requires the following components in the designated quantities (see Figure 3).

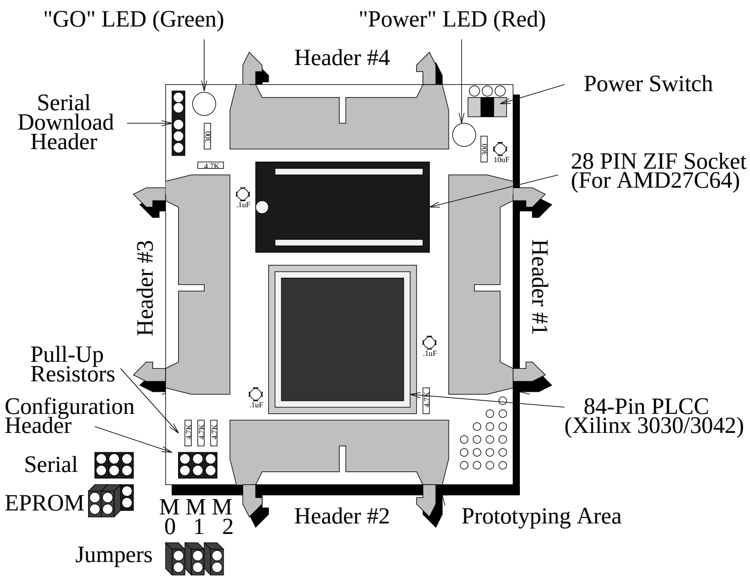


Figure 3: GERM - Ferrari Configuration

4.2 Yugo Model

The most basic, and still functional, “Yugo” version of the GERM requires the following reduced list of parts (see Figure 4).

The difference in the above configurations is noted to be less capability in the sense of access to I/O ports through connectors, reconfiguration, and stand-alone (or power-up) loading. The greatest difference is the difference in cost between the two versions of the tool. The Yugo version costs about $5 in mechanical components, while the Ferrari costs about $40 in mechanical components. Since we are currently receiving educational donations from Xilinx and AMD the cost of their chips is not factored into the cost of the GERM.

Ferrari GERM Components		
1	GERM	Printed Circuit Board
1	XC3042PC-100	Xilinx FPGA
1	AMD27C64	Advanced Microdevices EPROM
1	84 PIN PLCC	FPGA Socket
1	28 PIN ECONO-ZIP	EPROM Socket
4	26 Pin Male Header	Connector
1	1x6 Pin Header	Download
1	2x3 Pin Header	Configuration
1	3 PIN SPDT	Power Switch
1	Red LED	Power Indicator
1	Green LED	"GO" Indicator
3	4.7K Ohm Resistor	Pull-Up
2	300 Ohm Resistor	Current Limit
1	10uF Capacitor	Global Buffer
3	.1uF Capacitor	Local Buffer
3	2 Pin Jumper	Configuration

Table 1: Components for Ferrari Configuration

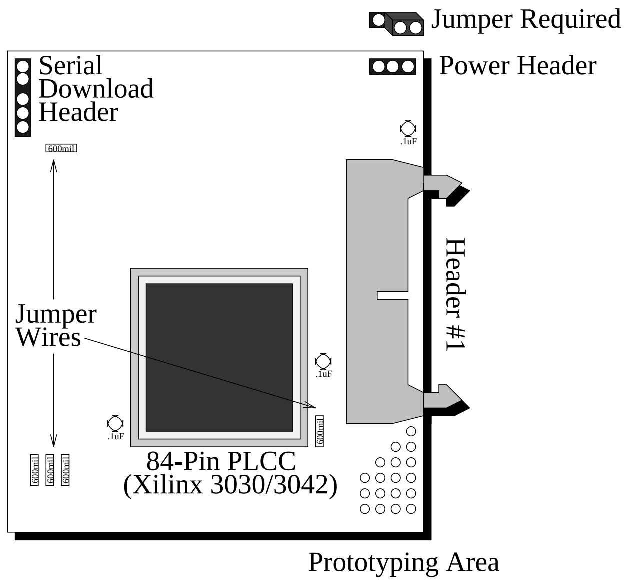


Figure 4: GERM - Yugo Configuration

Yugo GERM Components		
1	GERM	Printed Circuit Board
1	XC3042PC-100	Xilinx FPGA
1	84 PIN PLCC	FPGA Socket
1	26 Pin Male Header	Connector
1	1x6 Pin Header	Download
1	1x2 Pin Header	Power
1	2 Pin Jumper	Power
3	.1uF Capacitor	Local Buffer
5	600mil 22GA.	Jumper Wires

Table 2: Components for Yugo Configuration

5 GERM Design Creation

There are two primary avenues that may be taken in programming the GERM for use in projects and/or labs. The most basic path is an extension of current introductory level courses: schematic capture. The other, more advanced, path is through behavioral description languages: VHDL and Verilog and synthesis into the target technology (ie. XC3030 FPGA). Both paths will be described procedurally below. For further information, contact Dr. Overhauser for the most currently tutorials.

It should be noted here that as a "base" system, a site need only have the Xilinx design software and a download cable to use the GERM. Detailed steps using other vendors is primarily for use by universities working with large networks where large software packages have been purchased or acquired through donations. The Xilinx tool suite and download cable on the 486 is more than enough to accomplish designs and programming of the GERM for prototyping purposes.

5.1 Schematic

The schematic capture approach to programming the FPGA is perhaps the most intuitive to beginning designers. Basic logic gates and symbols are captured using a tool such as Mentor's Design Architect or Viewlogic's Powerview. The design is compiled into a file readable by the Xilinx translation tools. The translation tools then set the chip type and convert the design into a .bit file which can then be downloaded into the GERM or converted and programmed into an EPROM to be loaded into the GERM on power-up.

For a schematic design in Powerview, make sure your `viewdraw.ini` includes the `X2000, X3000, X4000, MX2000, MX3000`, and `MX4000` libraries. Also make sure your `XACT` and `WDIR` variables are set and your path includes both the XACT program as well as the XACT workview libraries. Fig. 5 shows a simple example design. The steps involved in this procedure are detailed below.

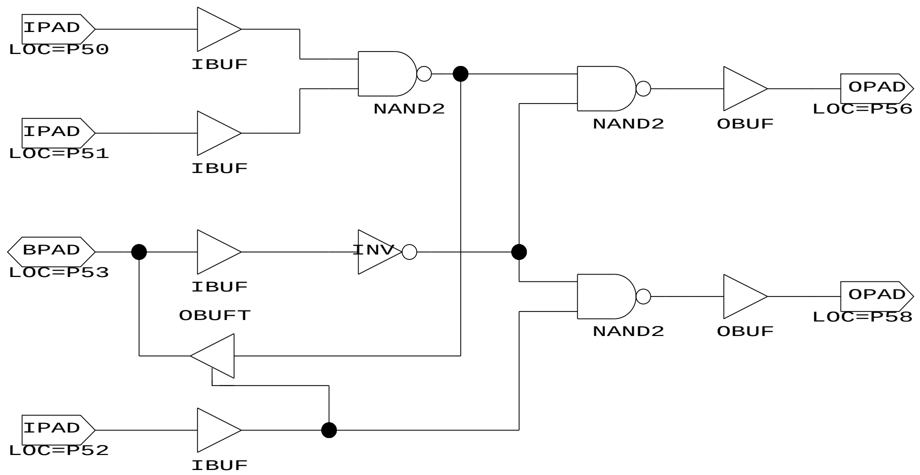


Figure 5: An Example Powerview Schematic

1. Schematic Capture or Logic Description of Circuit (ViewDraw). Using the `X?000` library (basic components) or the `MX?000` library (higher level components as macros)
2. Define Input and Output Pins (ViewDraw). In Xilinx FPGAs, external inputs must come via an `ipad` and `ibuf` combination. Any output goes to an `obuf` and then to an `opad`. An `bpad` provides a bidirectional signal.
3. Simulate and Verify Design (ViewSim)
4. Compile Design Using Xilinx Tools (XMAKE)

5. Transfer File to PC (ftp in binary)
6. Turn GERM Power On
7. Download Configuration into GRM (Xilinx)
8. Wait for Green LED "GO"
9. Exercise Circuit Through Headers
10. Verify Correct Operation

5.2 Behavioral

The more advanced, and more malleable, approach to programming the GERM is through the use of behavioral description languages such as VHDL and Verilog. Once described and simulated, the design can be synthesized using a tool such as Synopsis. The synthesized design is then converted using the Xilinx conversion tools and a `.bit` file is created. This file can then be downloaded serially into the GERM or converted to an EPROM format and programmed into the EPROM for loading the GERM on power-up. The detailed steps are listed below for following this design path.

1. Create a .VHD file that Synopsys will accept
2. Read into Design_Analyzer
3. Simulate Design
4. Insert pads with the command: Input Pads → Attributes → Optimization Directives → Input or Output
5. Optimize (Allow Boundary Optimization)
6. FPGA cells to Gates Using FPGA Compiler Option
7. Direct Design to Part (XC3030, XC3042, XC4005, etc.) with `set_attribute find(design "nand3") "part" -type string "4005pc 84"`
8. Set pad attributes:
   - `set_attribute "inputs<1>" "pad_location" -type string "P45"`
   - `set_attribute "inputs<0>" "pad_location" -type string "P46"`
   - `set_attribute out1 "pad_location" -type string "P47"`
9. Save as an XNF file and Exit Synopsys.
10. Merge Files with `xnfmerge -p 4005pc84 -d $SYNOPSYS/xilinx/xnf_files nand3.xnf nand3m.xnf`
11. Type: ppr nand3m
12. Run the Delay Tool on the Design with `xdelay nand3m`
13. Create a Bit File with `makebits nand3m`
14. Transfer to PC with ftp in binary
15. Download the Design or Program EPROM
16. Exercise the Design

5.3 Tutorials

Currently we have produced a tutorial as an introduction to using the GERM in our graduate and undergraduate level courses. The tutorial is available on request [1]. Other tutorials for using tools such as Mentor, VHDL, Powerview, LaTeX and others are also available.

6 Using the GERM

Due to the variations of the different configuration options described above, the actual use in the development environment is discussed below.

6.1 GERM Setup

The GERM requires power to be provided either at the test points above the power switch or via the power (pin 25) and ground (pin 1) signals from one of the headers. The microswitch on board connects the GERM to the power source that is provided. TP1 is ground. TP2 provides power for the download cable, EPROM, and the FPGA. TP3 connects to the four header connector power pins (pin 25). Note that the micro switch in the center position makes no connection (i.e. power on TP2 and headers unpowered). The right position shorts TP2 and TP3 so that power for the GERM is also the header power (this is the most common position when the headers are used). The left position may be used as an "OFF" position provided that power is applied from the headers (not on TP2) and the right position is used as "ON". The switch should never be used in the left position when power is applied to TP2 as this shorts TP1 (GND) to TP2 (Vcc).

Note that the connectors are keyed to remove the possibility of inserting the connector incorrectly. The choices of pin 1 and 25 remove the possibility of shorting the device if a non-keyed connector is inserted upside down. Also note the outputs and inputs of the GERM are taken directly from the chip to the pins on the connector. The signals are not buffered by '245's to protect the FPGA.

6.2 Basic Yugo Configuration

The Yugo GERM is typically used in Introductory Level Courses. The jumper wires on the Yugo connect the M0, M1, and M2 configuration inputs of the FPGA to a logic '1". This places the GERM in Serial Slave Mode and the FPGA is expecting to take data in through the DIN pin of the Download Header [4].

Using either the download or Xchecker cable, a `.bit` file can be loaded directly into the FPGA using the executable supplied by Xilinx. For example:

`download -port LPT1 design.bit`

or `xchecker design.bit`

After the program loads into the GERM, the FPGA will assume the behavior of the program loaded into it's configuration memory. The download program will also indicate that the DONE signal went high - indicating that the GERM successfully received the program.

From this point, the GERM can be exercised through Header #1. Although header connectors may not be soldered into the other connector slots, they may still be used and should be considered "live" at all times. To reload the GERM, the user need merely invoke the download program again and reprogram the GERM.

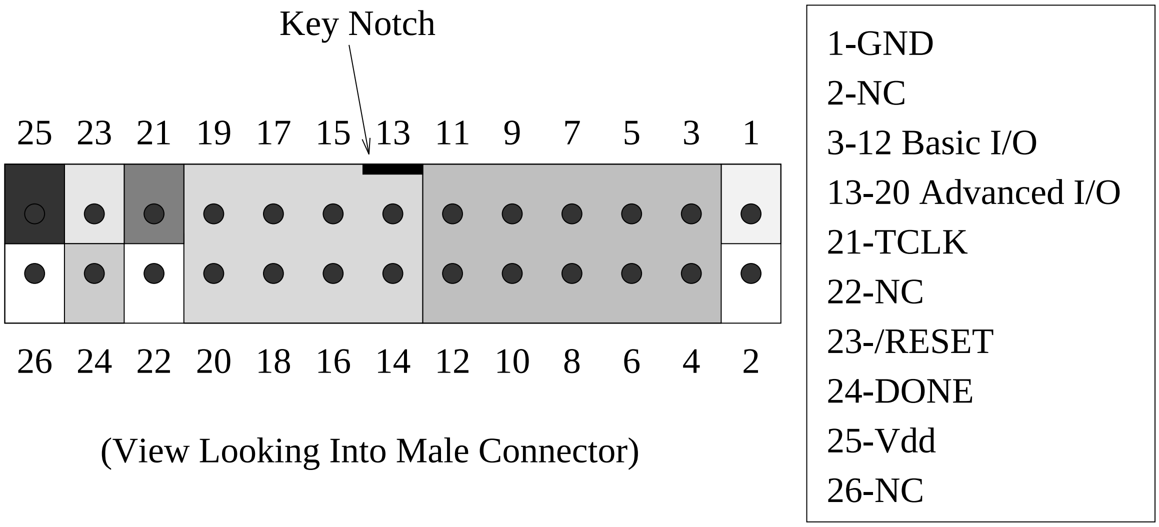
Note: Make a mental note that when the power is turned off to the GERM, its memory is lost. The Xilinx FPGAs are volatile and thus must have a constant power supply for correct behavior.

6.3 Advanced Ferrari Configuration

The Ferrari GERM may be used for more advanced designs. The Ferrari GERM can be placed into serial slave mode by setting the three configuration inputs to a logic level '1'. This requires that the jumpers not be used since the 4.7K Ohm resistors act to actively pull-up the voltage on the inputs to a logic level '1'. The instructions above may be used to load the FPGA serially.

To program the Ferrari GERM using the EPROM requires modifying the `.bit` file output from XMAKE. The `.bit` file must have each byte reversed (bit 7 becomes bit 0, etc) and the first 118 bytes must be set to 0 to erase the file information. This is easily done with the `bit2prom` program in appendix 1. Just type `bit2prom filename` without the `.bit` extension. Use the EPROM programmer to load the resulting `filename.prm` to the EPROM. The EPROM used in the GERM is the Advanced Microdevices AM27C64 UV-Eraseable PROM. These EPROMs take approximately 20 minutes to erase under UV erasers and approximately 1 minute to program using the MCT programmer (attached to a 486 machine).

To use the EPROM in the Ferrari GERM, first make sure the serial cable is disconnected. Next, it is necessary that the user place the jumpers on the M0 and M1 configuration bits (these are the two sets of



jumper pins on the left side of the configuration block). This sets the MO and M1 bits to logic level '0' inputs and the M2 inputs at logic level '1'. Once these jumpers are put in place, the FPGA is receiving signals to put it in the Master Parallel mode, and on power up will attempt to address the EPROM using an internal clock and counter [4]. After the FPGA is programmed, the GREEN "GO" LED will light to indicate that the GERM may now be used for operation.

NOTE: The power switch must be placed in the rightmost position to connect the local power bus to the external power bus. If this is not done, then the GERM will not be able to program. When the switch is placed in the far right position, the RED "POWER" LED with glow brightly indicating that the GERM is "ON".

6.4 Prototyping Area

A small prototyping area has been provided for the addition of crystals to be used in conjunction with the FPGA's ability to generate a clock internally. Please refer to the FPGA manual from Xilinx for further discussion on this application.

Other expected uses for the prototyping area include Status LEDS, pushbuttons, capacitors, resistors, op-amps, opto-switches, etc. It was not added as an extensive prototyping area, but more as a support circuitry design area.

6.5 Pinouts

The pinouts for the connectors on the edges of the GERM as shown in Figure 6.5 and the pinout for each connector with reference to the FPGA pins is given in Table 6.5. When assigning pin numbers for user defined input and output, it will be necessary to reference the profile and the pinout table.

Pins labeled as Basic I/O are dedicated I/O pins for the FPGA and are recommended for inputs. Advanced I/O pins have a primary function during the FPGA configuration and should not be driven for an input unless they are tristated until DONE is high. For this reason, Advanced I/O pins are recommended for outputs.

Header Pinout and Information for 84 pin PLCC					
PIN No.	Func	H1	H2	H3	H4
1	GND	GND	GND	GND	GND
2	N.C.				
3	I/O	P48	P47	P30	P14
4	I/O	P49	P46	P29	P15
5	I/O	P50	P45	P28	P16
6	I/O	P51	P44	P27	P17
7	I/O	P52	P41	P26	P18
8	I/O	P53/XTAL	P40	P25	P19
9	I/O	P59	P39	P24	P7
10	I/O	P63	P38	P23	P6
11	I/O	P68	P37	P21	P79
12	I/O	P69	P35	P20	P80
13	I/O	P56-D8	P57XTL1	P9- A8	P5-A7
14	I/O	P58-D7	P61/CS0	P11-A9	P3-A6
15	I/O	P60-D6	P42/INIT	P10-A10	P84-A5
16	I/O	P62-D5	P66/CS1	P8- A11	P82-A4
17	I/O	P65-D4	P71RDY	P4- A12	P78-A3
18	I/O	P67-D3	P73DOUT	P2- A13	P77-A2
19	I/O	P70-D2	P36/LDC	P83-A14	P76-A1
20	I/O	P72-D1	P34HDC	P81-A15	P75-A0
21	CLK	TCLKIN	TCLKIN	TCLKIN	TCLKIN
22	N.C.				
23	RES	/RESET	/RESET	/RESET	/RESET
24	DN	DONE	DONE	DONE	DONE
25	PWR	PWR	PWR	PWR	PWR
26	N.C.				

7 Technical Information

The following technical information is provided for more experienced users.

TP1 is ground. TP2 provides power for the download cable, EPROM, and the FPGA. TP3 connects to the four header connector power pins (pin 25). Note that the micro switch in the center position makes no connection. The right position shorts TP2 and TP3 and the left position shorts TP1 and TP2 (Vcc and GND shorted!). The rest of the test points are described in the table.

Example Configurations	
Yugo Serial	No jumpers at J1 Jumper between TP2 and TP3 so that power for FPGA and headers is connected together Download or Xchecker cable on DWNLD header Wire jumper in place of R3, R4, R5, R6, R7 Capacitors at C1, C2, C4 Components: FPGA, Header1
Ferrari Serial	No jumpers at J1 Switch in right position so that power for FPGA and headers is connected together Download or Xchecker cable on DWNLD header Resistor R9 is needed in the prototyping area if Xchecker cable is being used Resistors at R2, R3, R4, R5, R6, R7, R8, R9 Capacitors at C1, C2, C3, C4 Components: FPGA, EPROM, Headers, 2 LEDs
Ferrari EPROM	Jumpers at J1: left and center present, right missing Switch in right position so that power for FPGA and headers is connected together No Download or Xchecker cable connected! Resistor R9 is optional Resistors at R2, R3, R4, R5, R6, R7, R8, R9 Capacitors at C1, C2, C3, C4 Components: FPGA, EPROM, Headers, 2 LEDs

Table 3: Example Configurations

8 Construction

When constructing the GERM in whatever configuration desired, it is only necessary that the user solder in the desired components in the proper locations. A couple points to remember however:

- Remember to solder the components in without the chips in place! Chips act as good heat sinks and typically don't appreciate soldering iron heat.
- The download cable is keyed so as not to apply power to a digital signal input on the FPGA. Thus, it will be necessary to clip out the 3^{rd} pin down on the download header.
- The use of the 4.7K resistors is primarily to allow users desiring more inputs and outputs to utilize these configuration pins. If however, reconfiguration or I/O options are not desired, then wire jumpers may be soldered in as in the Yugo configuration without any negative results.
- When using the EPROM it will be important to remember that the EPROM should be programmed using the MAKEPROM facility. If the `.bit` file is loaded directly into the EPROM, the data bits will be 180deg out of phase (reverse order).
- Should brighter LED response be desired, a lower valued resistor or a different LED may be used.
- The LEDs are a bit difficult to get into the allocated holes, but with a little applied pressure they will fit nice and snuggly. (Designer goof!)

GERM Technical Information		
Test Point		Signal
TP1		GND
TP2		Vcc for GERM
TP3		Vcc for Headers
		See Section 5 about TP2 and TP3
TP4		RST
TP5		PwrDn
Resistor	Value	Purpose
R1	1K	Vcc (TP2) to GND
		R1 should not be present on board
R2	300	PWR LED pulldown
R3	4.7K	M2 Pullup
R4	4.7K	M1 Pullup
R5	4.7K	M0 Pullup
R6	4.7K	RST pullup
R7	4.7K	PwrDn pullup
R8	300	DONE LED pulldown
R9	4.7K	DONE LED pullup
		Placed in prototyping area

Table 4: GERM Technical Information

- When making cables, it is important to remember the orientation of the cable in the keyed connector. Just make sure that ground goes to pin 1 and power to pin 25. Remember: "Measure twice, cut once."
- Invest in a PLCC extractor such as the 16766-ESM PLCC Surface Mount IC Extractor for $3.49 from Jameco™ at 1-800-831-4242. This will save you a lot of frustration and PLCC sockets!
- I have provided a table of the parts needed and stock numbers if ordering from Newark. Lisa Bunting at 1-800-992-9587 is familiar with the project and can help you place an order.
- You will need to have Xilinx software to make full use of the design. And if the EPROM programming is desired, you will also need an EPROM programmer, such as the Modular Circuit Technology "High-Quality Programmer" Model MOD-EMUP available from JDR Microdevices at 1-800-538-5000. The price was $699.00
- The jumper wires shown on the Yugo Configuration are 22 ga. 600 mil long wires, and are used to short the M0, M1, and M2 inputs to Vcc. Thus, they won't be usable as user programmable I/O. Only with the use of the 4.7K pull-up resistors can then be designated as outputs by the user.

```
Newark Electronics
3825 West Market St.
Suite 106
Greensboro, NC  27407
1-800-992-9587
```

```
Ferrari Components

QNTY  UNIT DESCRIPTION                    COST
 1    98F3911  84-pin PLCC socket (FPGA)   2.78
 1    50F2909  ECONO ZIF 28-pin .600"      8.60
 1    46F1725  ON-OFF-ON Switch            5.97
 4    89F4797  Quickie III Right (26-pin)  1.98
 4    89F4738  Quickie III Female (26-pin) 1.51
 1    90F9257  1x36 Post Headers           1.49
 1    81F198   L311DR Red LED              0.77
 1    81F198   L311DG Green LED            0.77
 1    08F1150  300 Ohm Resistor (50)       0.35
```

1	08F1150	4.7K Ohm Resistor (50)	0.35
3	17F2165	.1uF Cap .200" lead space	0.19
1	87F5118	10uF Cap Case E	0.80
3	90F9280	Shorting Jumpers Open	0.13

Yugo Components

QNTY	UNIT	DESCRIPTION	COST
1	98F3911	84-pin PLCC socket (FPGA)	2.78
1	89F4797	Quickie III Right (26-pin)	1.98
1	89F4738	Quickie III Female (26-pin)	1.51
1	90F9257	1x36 Post Headers	1.49
3	17F2165	.1uF Cap .200" lead space	0.19
1	90F9280	Shorting Jumpers Open	0.13

The GERM printed circuit boards can be obtained from Duke Electrical Engineering Department or from the Technical Univ. of Crete at the addresses below.

Dr. David Overhauser
Assistant Professor
Dept. of Electrical Engineering
Duke University
179 Hudson Hall
Box 90291
Durham, NC 27708-0291
email: ovee@ee.duke.edu

Dr. Apostolos Dollas
Associate Professor
Dept. of Electronics and Comp. Eng.
Technical Univ. of Crete
73100 Chania
Greece
email: dollas@ced.tuc.gr
email: dollas@ics.forth.gr

9 Questions

Questions may be mailed to ovee@ee.duke.edu. Dr. Overhauser will then forward the messages or questions on to those individuals in the department that can best help you determine the proper course of action for obtaining chips, sample designs, curriculum usage documentation and labs. A video has also been made of a talk given on the GERMs with instructional information on the process of using the GERM.

References

[1] Kai-Ti Huang and Yucheng Wang. Make Xilinx FPGAs from Powerview and XDM. Technical report, Department of Electrical Engineering, Duke University, 1994.

[2] B. Ward, A. Dollas, and J.D.S. Babcock. Low Cost Generic Reusable Module for the Rapid Prototyping of Subsystems. Technical report, Duke University - Electrical Engineering Dept., 1994.

[3] B. T. Ward. An Experimental Study in Subsystem Reusability Toward Rapid Prototyping of Microelectronic Systems. Master's thesis, Department of Electrical Engineering, Duke University, 1993.

[4] Xilinx, Inc. *The Programmable Logic Data Book*, 1994.

10 Appendix - bit2prom.c

```
/* A simple program to reverse the bit order in a file */
/* #define DEBUG */
#include <stdio.h>
main(argc, argv)
int argc;
char **argv;
{
unsigned char ch, ch2; /* Unsigned so that printf 02x does not extend */
int  i, count=0;
char filename[80];
FILE *in,*out;

   if (argc <= 1) {  /* Check for filename */
      printf("usage: %s design_name (without .bit extension)\n",argv[0]); exit(-1); }

   /* open the file to convert and its corresponding output file */
   strcpy(filename, argv[1]);
   strcat(filename,".bit");
   in = fopen(filename, "rb");
   if (in == NULL) {
      printf("Could not open %s for conversion. Do not add .bit extension.\n",
      filename); exit (-1); }

   strcpy(filename, argv[1]);
   strcat(filename,".prm");
   #ifdef DEBUG
      printf(" input   -=> reversed (in -=> rv hex) Note: first 118 bytes stripped\n",filename);
   #endif
   out = fopen(filename, "wb");
   if (out == NULL) {
      printf("Could not open %s for output.\n",filename); fclose(in); exit(-1); }

   count = 0;
   while (!feof(in)){
      ch = fgetc(in);
      count ++;
      if (feof(in)) break;

      if (count <= 118) ch2 = 0; /* Strip out header of first 118 bytes */
      else { /* Reverse the bits in each byte thereafter */
         ch2 = 0;
         for (i=0;i<8;i++)
            if (ch & (1<<i)) ch2 = ch2^ (1<<(7-i));
         }

   #ifdef DEBUG  /* Prints out binary forward and reverse to show it works */
      for (i=0;i<8;i++) if (ch  & (1<<i)) printf("1"); else printf("0");
      printf(" -=> ");
      for (i=0;i<8;i++) if (ch2 & (1<<i)) printf("1"); else printf("0");
      printf(" (%02x -=> %02x)\n", ch, ch2);
   #endif

      fputc(ch2,out);
   }
   fclose(in);
   fclose(out);
}
```