BICEPS-8051

Emulator Manual

Version 7.9a

BRENDES DATENTECHNIK GmbH

Dresdener Str. 10 D-26160 Bad Zwischenahn Germany

Tel.: +49 4403 816838 Fax: +49 4403 816839 eMail: info@brendes.de

Internet: http://www.brendes.de

Contents

1 Introduction

The in-circuit-emulator **BICEPS** is a professional tool, which considerably eases the development of electronic circuits on the base of microcontrollers of the 80C51 family. The hardware components of the controller as well as the programs executed by it can be emulated in real time, i.e. at the original speed.

For in-circuit emulation the target board is connected to the **BICEPS** emulator via a Debug Connector. Optionally, the processor of the board can be replaced by a processor adapter (POD); several different PODs are available. Connected to a standard PC by an USB interface, the **BICEPS** emulator offers sophisticated real-time debugging features:

- Presentation and altering of all internal registers and memories of the CPU, but also of external memories and assemblies, located on the circuit under test and reachable by the CPU
- Alterations of the program memory contents
- Start of a program to be tested, execution of a program in real time and stop of the program run at different, free chooseable break conditions
- Tracing of states of buses and external signals in a real time trace memory.

The reader of this manual should be familiar with the architecture of processors of the 80C51 family as well as with the fundamental progression of a hardware and software development for microcontrollers. Suitable data books should be at hand, because an exact knowledge of the emulated processor is a supposition for a successful development work.

The performance characteristics of the **BICEPS** emulator are separated in BASIC features and PROFESSIONAL features. The BASIC features are in detail:

- Adaption to different processors of 80C51 family by adapter boards (PODs)
- No restrictions with respect to memory space, ports and interrupts
- Emulation of external bus application as well as single chip applications
- Support of single chip emulation with Brendes Softhooks™
- Debug connector for connection via pin header
- Emulation memory: 64k RAM program
- Breakpoints on program and extdata memory
- CPU clock: external from user board or internal (choosable frequency)
- **BicWin** source level debugging software for Windows
- Macros
- Supports file formats of several compilers
- USB interface

The PROFESSIONAL features include the BASIC features and additional:

- Emulation memory: 128k RAM
  - can be configured as: 64k program and 64k Extdata RAM
    128k program RAM (banked)
  - expandable to 512k program memory
  - emulator internal or external memory (mapping)
- Watchdog support
- 16 external inputs for break and trace
- 32 bit real time counter for time measurements in the range of 100 ns up to ca. 12 h
- 32k/128k x 96 bit real time trace memory with tracing of all states of buses, the external signals and the real time counter (time stamp)
- comfortable search and filter functions for the trace memory
- Performance analysis

The PROFESSIONAL features are activated by professional licences. For details see BicWin online help.

EMC

Hint: The **BICEPS**-Emulator is a device for testing of electronic circuits. It is designed for the cooperation with such circuits and therefore no device to operate standalone according to §5 point 5 EMVG (German EMC Law). It is restricted to exclusive use by development laboratories or other enterprises with know-how in the field of EMC.

We explicitly indicate that according to §5 point 6 EMVG at development of devices, testing and installation precautions have to be taken to avoid interferences of third parties. Since under certain circumstances dejam measures must be eliminated (e.g. remove the cover) to execute the test, suitable measures for the complete test arrangement have in case to be provided.

Attention !

The **BICEPS** hardware and the **BicWin**-software are protected by copyright. A copy of the software is only permitted for backup purposes and for own use; a transfer to others is explicitly prohibited. Software updates and support are given only to customers of **BRENDES DATENTECHNIK GmbH.**

H I N T !

You will find descriptions of current changes of software as well as additional user hints to this manual in the ”READ.ME” files on the enclosed system CD. Kindly notice the contents of these files absolutly.

2 Configurating and Starting of the BICEPS Emulator

In the following chapter the steps for starting the **BICEPS** emulator are described. If you want to get an overview over the features of the emulator, you should read the chapters 3-10. Detailed descriptions of the several functions (user interface ect.) you find in the **BICEPS** online help, which can be started without emulator.

To adapt the **BICEPS** emulator to the operation mode needed, several configuration options can be used. The configuration is done on the hardware side by jumpers and on the software side by the „Options“ menue of the BicWin software. This configurating procedure has to be done before the emulator is started the first time.

Basically, there are three options for connecting the BICEPS emulator to the target board

1. Debug connector
   The BICEPS emulator is connected to a pin header, which is provided by the target board and should be located near the CPU. The debug connector enables the emulator to control the CPU. The user software is executed by the microcontroller on the target board.

2. Processor adapter (POD)
   The CPU on the board is replaced by an adapter (POD). The user software is executed by the microcontroller on the POD.

3. External Program Memory connector (universal adapter)
   Special form of the Debug connector, which has the pinout of a standard memory circuit.

Pinouts and jumper configuration can be found in chapters 11-13.

2.1 Start of the BICEPS Emulator

For the start of the **BICEPS** emulator, it is recommended to proceed in the here given sequence:

1.) Start the installation program on the **BICEPS** disc (setup.exe). It installs the **BicWin** software on your PC.

2.) Connect emulator and PC by means of the enclosed USB cable.

3.) Switch on the power supply of the **BICEPS** emulator (indicated by the green LED). The USB driver software installation is started automatically when the emulator is switched on the first time. The BICEPS USB driver is found on the BicWin-CD.

4.) Switch the emulator off and start the **BicWin** software. After a short delay, you will be asked for configuring the emulator. Confirm this questions with “Yes”. You will get the **BicWin** “Options” dialog. You must set the following options (in the register cards "Interface", "Processor" and “Hardware”). Further options settings as directories, compiler etc. can be done now, but also later after the start of the emulator.
Interface: Select emulator type BICEPS-6 and USB interface
Processor: defines the controller to be emulated and the clock frequency
Hardware: selects the kind of adapter (debug connector, processor adapter) and the mode of ports P0/P2. The emulator hardware initialization depends on this setting.

5.) Click on the “Ok” button of the **BicWin** options dialog now. The emulator is started and **BicWin** presents the default desktop with Register-, Intdata- and Assembler window. The contents of the Special function registers must contain the reset values (e.g. PC=0000, SP=07).

6.) Much success for your work with the **BICEPS** emulator.

2.2 Banking

To emulate banking applications a program memory with more than 64k must be addressed. Therefore additional address lines are necessary (A16..A18). If a POD is used, A16..A18 are connected to a special pins on the POD named E0..E3. If universal adapter or debug connector are used, banking signals are connected via this adapters.

In banking applications with bank sizes of 32k or 16k, the address of the common bank is generated by setting the bank nummer to 0 if A15=0 resp. A15=A14=0 (AND function). The **BICEPS** emulator supports this banking variants; it is assumed, that the universal adapter or the debug connector is used and that A15 and A14 are generated correctly by the test board.

2.3 Keil µVision integration

As alternative to the BicWin debugging software, the **BICEPS** emulator can be integrated in the Keil µVision IDE. Brendes offers a special DLL which expands the µVision software.

The debugger is used as known by the simlutor, but the user programs are loaded in the **BICEPS** emulator and executed in real-time. There is no need for learning a second debugging environment if µVision is already in use. Please refer to the Brendes software package for more information on installing this feature.

3 Basics of the BICEPS-Emulator

3.1 Structure of the BICEPS-Emulator

The **BICEPS** emulator consists of eight essential components:

- The **Control Processor** takes over the control of the emulation process as well as the communication with the host computer (PC) via an USB interface.

- For the **Emulation Memory** 128k bytes are available, which can be configured as
  - 64k RAM program memory and 64k RAM external data memory or
  - 128k RAM banked program memory (Professional version)

  The emulation memory can be expanded to 512k RAM.

- A **32-bit-Real Time Counter** allows exact time measurements of the program executing duration with a solution of 100 ns. A measurement range of up to 12 h is achieved by a switchable time base. (Professional version)

- The bus signals, the external signals, which can be fed-in from the circuit under test by means of test clipses (Logic Probes) as well as the state of the real time counter can be traced in the **Real Time Trace Memory**. In this way the temporal progression of the program run or the access to the external data memory and the execution time of program parts can be represented (also during a running program). (Professional version)

- The **Address Range Selection** is used for marking either addresses or address ranges. For each address of the address area it can be defined whether a memory access of these address resp. this address range
  - shall lead to a program break
  - shall be traced in the real time trace memory

- Besides the possibility of defining address ranges for program breaks, using the **Break Logic** one can also determine conditions for the data bus and for external logic probes. Up to two break events can be defined, which are counted (break counter) and combined in real time. The break events can:
  - interrupt the real time program execution (break)
  - start or stop the real time trace
  - trigger an external device.

  Real time program execution can be interrupted automatically at certain events or manually by user inputs.

- The **Port replacement unit** (PRU) generates the P0 and P2 signals for the test board, because these signals of the controller are needed as bus signals. Depending on the operation mode the external signals can be bus or static I/O-port signals. More information about P0/P2 emulation can be found in chap. 10.

3.2 An Outline of the Emulation Process

After the initialization phase (the RESET command) the emulation processor transfers the contents of its internal registers to the control processor and is stopped by the latter. Now one can access the contents of the recovered register as well as those of the emulation memory from the host computer by the help of the control processor. Moreover, one can display and alter memory contents, which are accessible only from the emulation CPU (externally mapped memory ranges). Should the emulation processor now execute a user's program, the (eventually changed) register contents are read and a jump into this program follows. After termination of the program's execution the new register contents are again transferred to the control processor.

The simplest form of program processing is execution in single steps. In this case the internal state of the emulation processor is presented after each program step. However, the normal procedure is the execution of a program in real time. Then the emulation processor is started at a certain position of the user's program. The program execution can be stopped either manually by the user or automatically by previously defined break conditions and the state of the emulation CPU is displayed. The break options are presented in chapt. 6.

If the Universal adapter or the Debug connector of the **BICEPS** emulator is used, the emulator must be able to control the processor of the circuit under test. This is possible by a sophisticated emulator control logic which surveys the program execution in real time and switches between user program and hidden emulator-internal routines. The emulator maintains in this way the control about the processor also during program execution. An additional communiction logic permits the transfer of the internal register contents to the PC. No accesses to the external data memory are used for that purpose to avoid restrictions related to the functionallity of the ports.

4 The BicWin User Interface

4.1 Survey of the Operation Facilities

In order to facilitate a comfortable operation of the **BICEPS** emulator, the user interface **BicWin** offers multiple functions. The representation is given in a clear window technique with pull-down menus and status line. All windows can be opened and closed arbitraryly and changed in size and position. The operation is carried out with mouse or keyboard.

General information to the possibilities of the **BicWin** user interface are presented in the following chapters; more detailed descriptions, especially of dialogs and menus are offered by the Online help of the **BicWin** software.

Following a short survey is given .

1.) All the memory cells accessible from the emulation processor can be represented and altered. For this serve the windows Extdata, Intdata and Program and the "Modify" functions. "Watch" variables can be defined and represented in special watch windows. With "Copy" memory ranges can be moved and with "Initialize" set to a certain value. With the "Map" function it is defined which memory ranges internally in the emulator and which ones externally on the user board are addressed. There is the possibility to copy the contents of an external EPROM directly into the emulation memory (see chapter 4).
2.) The special function registers of the CPU are displayed in the Register window. They can also be retrieved and changed directly by means of the "Modify" function, respectively (chapter 4).
3.) Execution of a program is initiated by function keys or by the "Debug" menue. In the single step mode there is a possibility of executing the subroutines in real time ("jump over calls").

4.) The program memory contents can be represented in disassembled form. The input of mnemonic instructions will be done with the **BicWin** line assembler.
5.) There is the possibility to set or clear breakpoints in the Assembler or Sourcetext window directly. More complex break conditions (e.g. combination of break events or break counters) are defined via the "Break" dialog.
6.) By means of the "Trace" functions one can either display the contents of the real time trace memory of the **BICEPS** emulator or select various options of tracing data in the trace memory (see chapter 7). Further functions are e.g. searching of certain contents or calculating of time differences.
7.) The memory contents of the emulator as well as the contents of the real time trace memory can be saved on or loaded from a disc. Several standard file formats are supported.
8.) Besides absolute addresses symbolic addresses can be used, which were created by an assembler or a compiler. If the use of symbols can be helpful, this is offered by BicWin with the "?" button.
9.) Sourcetext Debugging allows the test of a program on source text level. The high level language program can be executed in single steps, i.e. line by line. Breakpoints are set directly in the source text. (chapter 9).
10.) All debugging commands are recorded in the "Command" window and can be altered and repeated easily.
11.) User-own macros provide the summarization of **BicWin** commands under a free chooseable name. These can be executed by entering the macro name but also automatically at the start of emulator and in case of a break.
12.) To handle several projects, the actual configuration of desktop, break and trace settings etc. can be loaded and saved. A context sensitive Help Function offers additional support at the work with the **BicWin** user interface

4.2 General Information to BicWin

The **BicWin** user interface consists of a screen divided into four parts:

- in the middle the working plane with windows and dialogs
- above that the selection menu
- below the status line
- the mouse panel with the most important actual functions.

In addition to that every window posseses a local context menu.

All functions can be invoked either with the keyboard or with the mouse.

Opened windows can be moved arbitrarily on the working area resp. enlarged or reduced. Only one window is always selected; this is recognizeable by the highlighted bar.

For all functions and windows short keys (function keys and ALT-key-combinations) are available. The most important are the ALT-keys, with which menu items and windows are selected and the function keys for releasing **BicWin** actions.

The context sensitive **BicWin** help system offers detailed descriptions of every function or windows. In the help window you find key words that can be selected by the mouse and lead you to more help information.

4.3 Windows and context menues

A window is a part of the screen, which you can shift, enlarge and reduce, cover with other windows , open and close. Several windows can be opened, but always only one window can be active (recognizeable at the highlighted bar at the head of the window).

Wih the ALT-key combinations a window can aimed to be opened or activated (e.g. ALT+A = assembler window, ALT+1 = watch window 1). With CTRL+F4 a window is closed, with F6 and Shift+F6 the next or the previous window is activated (see also overview short keys).

An own context menu is assigned to every window, which can be activated with the right mouse key or the context menu key. The context menu key is also designated as PopUp key. For this reason the offered key combinations are called "Pop+letter" so e.g. **Pop+B.** In comparison with the CTRL key the PopUp key has the advantage that it need not be activated synchronously with the letter key and it can therefore better be operated with one hand.

The following windows are available:

Label	Function	ALT key
Assembler	Disassembled presentation of the program memory	ALT+A
Intdata	Hex presentation of the internal data memory	ALT+I
Extdata	Hex presentation of the external data memory	ALT+E
Program	Hex presentation of the program memory	ALT+P
Register	Special function registers of the CPU	ALT+R
Sourcetext	Presentation of source text or other text files	ALT+S
Trace	Trace memory content	ALT+T
Watch1..4	Output of watched variables	ALT+1 ... ALT+4
Command	Records BicWin commands (with possibility to repeat)	ALT+C

4.4 Mouse palette and Status line

In the mouse palette the most important functions, which are just available, are offered. These can either be executed directly by mouse click or by input of the highlighted short key. The mouse palette can be oriented horizontally or vertically.

The mouse palette consists of a general part, which stays unchanged and a locally assigned part, where the offered functions change, depending on the just selected window. For instance the function **F7** (single step) is always available at the same position, while **Pop+B** (break point set/reset) is only offered in the assembler window and in the source text window.

The status line at the lower margin of the **BicWin** window represents the following information:

1. The runtime of the program executed in real-time or in single steps since the last reset of the real-time counter.
2. The status of the external digital inputs (logic probes)
3. A message when the program to be tested is executed.
4. A message about the just executed macro

4.5 Dialog boxes

Menu options with dots (...) open a dialog box, in which settings can be shown and altered. A dialog box must at first be closed, before another window can be activated.

In a dialogue window there are different operating elements:

- Action switches: release an action by activating with mouse or keyboard
- Marking and switching fields: switching options on and off
- Input fields: for input of text
- List window: Selection of elements with the mouse or arrow keys

5 Accessing the Memory

5.1 Basic Information

The **BICEPS** emulator is equipped with various emulation memories according to the achitecture of the emulated processor. The memory contents can be displayed (output) and changed in order to create specific test conditions. Besides this, we must naturally be able to enter or load the programs to be tested, and possibly also modify them.

Processors of the 80C51 family possess five different memory ranges altogether:

- the Program Memory
- the external Data Memory
- the internal Data Memory
- the bit-addressable Memory
- the Special Function Registers;

the three latter are partially overlapping. A special **BicWin** window is allocated to each memory range, where the bit-addresable memory is displayed in the Intdata window.

In principle, we can distinguish the following access possibilities for various memory types:

- Loading and saving of memory contents
- Displaying and changing the Special Function Registers
- Displaying and changing memory cells (Extdata, Intdata, Program window)
- Modifying of memory cells, registers and variables
- Initializing and copying of memory blocks
- Mapping, i.e. determining whether the RAM internal to the emulator should be accessed or whether we should access the circuit to be tested
- Disassembled representation of the Program Memory (Assembler window)
- Assembling 80C51 Opcodes

5.2 Saving and loading files

The **BicWin** user interface offers various functions for loading and saving of memory contents with different formats.

We can load:
- Contents of the program and data memories
- Source text files for highlevel language debugging
- Symbol information

and we can save:
- Contents of the program and data memories
- Trace memory contents

For loading and saving of memory contents four different file formats are available.

BicWin can save memory contents of the emulator also as text files, e.g. to evaluate these files with other programs or to edit them with a text editor. Distinguished are
- hexadecimal representation of memory contents (Hexdump)
- disassembled representation of the program memory

It is to be noticed, that significant limits are stated for creating of text files, since the files can otherwise reach a considerable size (several tenthousend lines). At saving the program memory in disassembled form two formats are distinguished:
- in the list format (same contents as in the Assembler window)
- in the sourcetext format the columns with addresses and file contents are omitted, to make the text files usable as source for an assembler

5.3 Access to the Special Function Registers

The contents of the special function registers are displayed in the Register window. The names of the registers are preset, and depend on the CPU type selected in the “Options” dialog. In addition to the real special function registers, the display includes the program counter PC, and the register bank R0..R7.

The registers are separated in groups; each group has an own box with title bar (e.g. Timer). The groups can be moved inside the register window; so even if only a small register window is visible, the most interesting registers can be displayed. The names, addresses and groups are contained in the external file “BICREGS.TXT”, which can be edited if necessary.

The arrow keys or the mouse can be used to select a register and open the “Modify” dialog to enter a new content. Ports are set at once, i.e. the new content appears at the port pins immediately after entering the new value.

Attention:

The internal data memory and the special function registers overlap in the address range 80H..0FFH. In the Intdata window no special function registers can be accessed, but only the internal RAM cells of the processor (if there are any).

5.4 Access to the Program Memory, to the internal and external Data Memories

Memory contents are represented in hexadecimal form in the Extdata, Intdata and Program windows. By means of the arrow keys we can select specific addresses and set at a new value, with the “Got to” function it can be jumped to a new address.

Altogether three fields can be distinguished: the address field, the hexadecimal field and the ASCII field. The cursor can be moved to any position of the hexadecimal and the ASCII field. New memory contents in hexadecimal form or as ASCII character can be entered.

For the individual functions one should keep in mind:

Program:

Entering data is possible only for internal mapped memory ranges.

Extdata:

Entering data is always possible. If the test board should be accessed directly, then the considered memory range must be mapped externally (default setting).

Intdata:

In the address range 80H..0FFH the internal Data Memory overlaps with the Special Function Registers. The Special Function Registers cannot be accessed in the Intdata window.

5.5 Output and Change of single Memory Cells and Registers (Modify)

If a single memory cell or variable shall be aimed accessed without hexadecimal representation of a memory block, the Modify function must be used. This enables also other formats for input and the output, i.e. memory cells can be entered or represented in binary or decimal form, as ASCII characters or as highlevel language variable. Which variable type can be selected depends on the compiler defined under "Options Highlevel".

The function Modify can be invoked directly from the Register, Assembler and Sourcetext window. For that purpose the requested variable must be only selected by a double click with the mouse. If it is a valid variable then one gets the current value immediately and can enter a new value.

As a speciality the function "Modify write only" is still available, which enables the writing to memory cells, without accessing the corresponding address by previous reading. This is necessary e.g. with I/O components, which can change their state already by the reading access.

5.6 Watched variables

With the Watch function memory contents and variables can be displayed in up to four different watch windows. Which variable shall be represented is determined by the “Watch list” dialog. Variable type (e.g. Unsigned) and format (e.g. binary) can be selected.

Up to four windows are available, into which watch variables can be issued (Watch windows 1..4). By the distinction of up to four windows it can be exchanged in an easy way between sets of variables to be issued without altering the definition in the list of watched variables. Only the windows with the desired variables had to be opened or the not relevant windows closed again. For this the key combinations ALT+1 until ALT+4 are provided.

The watch windows are actualized by choice either automatically after each program execution or only at selection of the watch window.

5.7 Initializing and Copying of Memory blocks

The function "Initialize" writes a constant value into a memory block, defined by start address and end address.

By means of "Memory Copy" we can move memory blocks, i.e. copy them. The source and target areas may overlap. As extension hereto there is the function "Copy EPROM", which transfers the program memory contents from the user's circuit into the Emulation RAM (see chapter 4.8 "Mapping").

5.8 Mapping

The memory types program memory and Extdata memory can be mapped. This means that the user can determine with help of the "Memory Map"-function whether

- the RAM internal to the emulator (emulation memory), or else
- a memory possibly present on the circuit to be tested (e.g. an EPROM)

should be accessed.

The default setting for map is an internal mapped program memory and an external mapped Extdata memory.

The **BICEPS** Universal adapter offers as well a mapping possibility. This is true only for the program memory, the Extdata memory is mapped external always. To access the original EPROM, the EPROM of the test board must be removed, replaced by the adapter and be plugged on the adapter. This means, that an emulation is possible with the original CPU and the original EPROM.

5.9 Disassembled representation of the program memory

The Assembler window gives out the contents of the Program Memory in the form of executable commands, i.e. in disassembled form. The code undergoing this representation is always the code executed by the emulation processor, i.e. - depending on the relevant map configuration - the contents of either a memory internal to the emulator or an external one.

One can move through the memory forwards and backwards, using the cursor keys. The line with the current program counter position, i.e. the command to be executed next, is highlighted.

Besides the the representation of the program additional functions are possible:

- Execution of the program
- Set and clear of breakpoints and trace filter ranges
- Modifying of memory cells and variables

For more information see chapter 7.

The **BicWin** Line Assembler (integrated in the Assembler window) serves as comfortable input of 80C51 program code into the program memory RAM. Before entering a mnemonic command, the command currently present in the program memory is displayed in disassembled form. Entering code is only possible in internal mapped memory ranges.

For denoting the addresses of the program memory, the bit memory and the internal and external data memories one can use arbitrary symbols - provided that they are known, i.e. entered into the corresponding symbol table. The predefined symbols known to the **BicWin** Line Assembler are the names of the special function registers (bit and byte addresses). They correspond to the type of processor selected in the “Options” dialog. Furthermore complex arithmetic expressions can be formed at the declaration of addresses and constants, which can contain symbolic names as well as number constants.

6 Break possibilities

6.1 Basic Information

Breaks are defined in order to stop the program execution of the emulation CPU at certain conditions, interesting for the program test. For example, these condition may be the execution of a certain part of the program, or an access to some specific ranges of the Data Memory. The **BICEPS** emulator offers the following possibilities for defining a break condition:

- Access to one or several addresses or address ranges (64k breakpoint memory)
- State of the data bus by accesses to the data memory and program memory
- State of the external inputs (Logic Probes)
- Event counter for breaks (Break Counter)
- conditional breaks
- Overflow of the real time trace memory

The defined state of the address bus, the data bus and the logic probes forms a break event, where the combination is predetermined

(addresses AND data AND Cycle type)

An event is additionally connected with an event counter (break counter).

Altogether there are two break events “Break1” and “Break2”, which can generate a break alone or together. The following break actions are possible:

- interruption of the real time program execution (break)
- enabling the second break event for conditional breaks
- start or stop of the real time trace
- triggering external devices.

6.2 Break mode

“Break1” and “Break2” can be combined in several ways. Beside OR and AND combination there are conditional combinations, that allows to enable one break event by the otherone. Furthermore there are the two events “Clear1” and “Clear2”, which can reset detected events “Break1” or “Break2”. In this way it is possible for example to define an event, which is only enabled in a special subroutine.

The events “Break1” and “Berak2” are available as signal outputs to trigger external devices (see chapter “Interfaces”).

In the **BicWin** help function you can find several examples of the break and controlling possibilities of the **BICEPS** emulator.

In addition to states of the address and data buses, we can also define a state of the external inputs as a break event. When the signals at the external inputs become effective and what qualities they must possess is described in chapter B.2. The definition is made as binary combination, i.e. as a sequence consisting of the symbols "0", "1" and "x", where to the external inputs 16 characters (corresponding to the 16 inputs) are assigned.. "x" stands for don't care, these bits are not rated.

6.3 Definition of Break Events

The most important function of the break logic of the **BICEPS** emulator is interrupting the program execution at a specific position of the program. This position is defined by its address.

For the four break events “Break1”, “Break2”, “Clear1” and “Clear2” one can define as many addresses or address ranges as one likes. By the definition of the “Cycle type” it is possible to distinguish between Program and Extdata memory accesses; in addition a specific data bus state can be defined. A defined break address is only then valid when during the program execution the defined state of the data bus occurs at the same time.

For accesses to the program memory the options “Opcodes” (default), “Line numbers” and “All accesses” are available. "Opcodes" means, that a break is only then recognized when the first byte of a command is executed (independent of of the value of the read date). This is a very important option, since CPUs of the 80C51 family access addresses, partly unnecessarily, and reject the read data afterwards. With “All accesses” this restriction is omitted; with “Line numbers” the opcodes cycles are restricted to sourcetext line beginnings.

The data bus conditions Read and Write contain the ranges 00-FF as presetting, i.e no restriction to a specific state. Each reading or writing access to an address of the external data memory leads to a valid break event, as far as the address is marked as break address.

Each of the possible break events is assigned an event counter (break counter). In order to effect a break, an event defined by the states of the address bus, the data bus and the logical probes must occur as often as it is predetermined in the break counter. Preset is a value of 1, i.e. the event is immediately valid at its first occurrence. For “Break1” a 24 bit counter is available, but this is reduced if break counter capacity is needed by other events.

7 The Real Time Trace Memory

7.1 Overview

The **BICEPS** emulator can be provided with a Real-Time Trace Memory of a 32k x 96 bit capacity. It is used for recording certain signals during a real time program execution, similar to a logic analyser. Additionally existent is a 32 bit real time counter, which runs synchronously with each program execution and therefore enables exact time measurements.

The data recorded in the trace memory are:

- 24 bits address bus signals
- 16 bits data bus signals (see app. B)
- 16 bits external signals (logical probes)
- 32 bits status of the real time counter (time stamp)
- 8 control signals

On the basis of the traced information one can easily obtain an overview of the run and the time duration of a program or the external signals.

The real time trace memory of the **BICEPS** emulator can store up to 32768 (131072 bei 128k) entries. Each entry has a size of 96 bit and is named a "frame". The frame number is jointly represented in the trace window, where number 0 corresponds to the oldest entry and the largest frame number to the entry traced at last.

Traced are all program steps, i.e. single steps as well as program fragments executed in realtime.

The contents of the real time trace memory is displayed in the “Trace” window. This is also possible during the program execution, without influencing it ("on the fly" access).

The real time trace memory of the **BICEPS** emulator offers the following functions:

- representation of the traced data either as program or as sequences of cycles with time stamp and state of the external inputs
- show context in the Assembler or Sourcetext window while scrolling in the trace window
- calculating of time differences of the traced time stamp
- search functions for traced data
- Sourcetext trace mode (tracing of source text lines)
- selecting of address ranges and access modes for the tracing
- real time filter to enable/disable memory address ranges for tracing
- start and stop of tracing by break events
- Program break by overflow of the trace memory with a predetermined number of frames to be traced
- Performance analysis, i.e. calculating the execution time for different parts of the program

7.2 Time Measurements

The **BICEPS** emulator possesses a 32 bit real time counter, which counts timing marks during a program execution.The time base is switchable between

- 100 ns (measurement range up to ca. 7 min)
- 1 µs (measurement range up to ca. 70 min)
- 10 µs (measurement range up to ca. 12 h)

The actual value of the real time counter is displayed in the status line. It can be set directly to 0.0 with the function "Reset real time counter" or together with a CPU reset with the function "Debug Reset all". The status line displays the time of program execution since the last reset of the real time counter.

Time measurements come in connection with the real time trace memory, since the position of the counter is traced as time stamp. The time of program execution yields to the difference of traced time stamps. Since their calculation from absolute values is too troublesome, the trace window offers several possibilities for forming of time differences:

1. The time stamp at the current cursor position in the trace window can be set to 0. All following time stamp entries are represented then relative to this zero mark.
2. The multiple manual setting to zero of the time stamp can be simplified with the time filter function. In the time filter dialogue conditions can be defined to which the continous time stamp representation starts again with 0.0. Is e.g. the address of a subroutine entered as condition, then the duration of execution of this subroutine is automatically represented in the trace memory window. Besides addresses also conditions for the data bus or the external inputs can be defined as time filter condition.

7.3 Search function

A complex search function allows to find those information in the large amount of traced data, which are interesting for the debugging. Default setting is no restriction, that means all traced frames met this condition. By restricting the search conditions (e.g. address range 1000-1020 instead of 0000-FFFF) the search function becomes more sensible.

In addition to address and data bus contents it is possible to search changes in break events “Break1” and “Break2”.

7.4 Sourcetext Trace Mode

In the sourcetext trace mode only the execution of lines of a loaded highlevel language program is traced instead of the complete program run. Assumption for this is, that a source text as well as the corresponding symbol information were loaded (see chapter 9).

Each line of a highlevel language program is related to a memory range of the program memory. Since in the sourcetext language trace mode only the start of this address range is traced, the real time trace memory can trace up to 32766 sourcetext lines.

When scrolling in the Trace window, it is possible to display automatically the part of the program in the Assembler and Sourcetext window, which belongs to the actual selected trace address.

7.5 Controlling the real time trace

The tracing is controlled by trace filter conditions. Single addresses or address ranges can be defined; also symbolic names, in particular module names, are admissible there. Only accesses to the addresses defined in this way are traced in the real time trace memory of the **BICEPS** emulator.

Three real time filters are available. Default setting is one filter for each cycle type, i.e.

- accesses to the program memory
- writing accesses to the external data memory (Write)
- reading accesses to the external data memory (Read).

For the program memory filter “Opcodes”, “Line numbers” and “all accesses” are distinguished.

It is possible to switch from one filter to another during program execution, controlled by a break event.

The **BICEPS** emulator allows to start or stop the real time tracing at predefined events. For this purpose break events are used. It can be defined, which action should be released by break events. Possible are:

1. Start of tracing at program execution start
   Stop of tracing at program execution end
2. Start of tracing at program execution start
   Stop of tracing at an event
3. Start of tracing at an event
   Stop of tracing at program execution end
4. Start of tracing at an event
   Stop of tracing at another event
5. Start of tracing at program execution start
   Switch to another filter at an event
   Stop of tracing at program execution end

8 Program Execution

8.1 Overwiev

The **BICEPS** emulator offers various functions for executing a user's program.

- Execution of a program in real time (Run)
- Execution of a single step
- Execution of a single step, but with the condition that subroutines are executed in real time (Jump over calls)
- Generation of a hardware reset of the emulation CPU (Reset processor)
- Execution of a program in real time until a break address is reached
- Set and clear of breakpoints

8.2 Execution of a program in real time, Reset

One of the essential functions of an in-circuit emulator is the execution of programs under real time conditions until the program's run is stopped. In the simplest case the break follows after executing a single step of the program (Single Step). However, the real time relation can be obtained only by executing several program steps.

By means of the "Run"-function we initialize the break logic and start the program.

The execution of a program in real time is indicated by a light diode in the field "Running" on the front panel of the emulator. The real time trace memory can be accessed, while the program is executed, and its contents displayed without

disturbing the program execution. An interruption of the program can be performed

- by the user with the mouse or the keyboard or
- by occurrence of a break condition.

After a break, the actual values of the open windows and the status line are displayed. Very important for program debugging are the register window with the actual PC and registers and the actual part of the program in the Assembler window. After a break, an access to all memory ranges is possible in order that the current memory contents could be changed.

The function "Reset processor" resets the emulated CPU by executing a hardware reset.

8.3 Execution of single Steps

The emulated CPU can execute a user's program either in real time or in single steps. In the first case, the program is started up and executed until a break occurs. In case of single steps, the internal state of the processor is displayed after each program step.

The "Jump over Calls" function allows to execute single steps with executing of subroutines in real time. This means: if the next command to be executed is a CALL opcode, then a break point is set immediately after this opcode, and the program is started up in real time; otherwise a normal single step is executed. If the processor fails to return from the subprogram, then the program execution must be stopped by the user.

Single steps are traced in the real time trace memory as programs executed in real time are.

8.4 Debugging with the Assembler and Sourcetext window

The most important debugging functions can be executed and monitored directly in the Assembler and Sourcetext window. This functions are:

- Reset of the processor
- Execution of a single step, i.e. an assembler opcode or a sourcetext line
- Single step with "Jump over Calls"
- Execution of the program in real time
- Program execution to the cursor, i.e. the command marked by the cursor is provided with a breakpoint and the program is started
- Setting the program counter PC on the command marked by the cursor
- Setting and clearing of breakpoints
- Marking of program ranges and setting/clearing of trace filters for this program ranges. By this feature it is possible to disable the tracing of program loops whose execution leads to an overflow of the trace memory
- Opening the “Modify” dialog by a double click of the mouse with the selected variable name.
- representation of another part of the program

Most of this functions can be executed directly by a click with the mouse in the mouse palette or by a function key. The remaining functions can be activated via the local context menue.

In combination with the Trace window it is possible to show the actual part of the program in the Assembler or Sourcetext window when scrolling in the Trace window. The address of the just selected trace frame is shown as highlighted bar in the Assembler or Sourcetext window. The user can watch the traced program flow in a representation with a moving bar just like the execution of single steps.

By deactivating the option “Follow program counter” it is possible to freeze a representation of the Assembler or Sourcetext window. The shown part of the

program is not adapted when the program counter is changed by execution of the program.

The following lines are shown highlighted:
1. the actual program counter address
2. breakpoints
3. disabled trace filter ranges
4. the address of the just selected frame in the trace window
5. marked areas.

9 Sourcetext Debugging

9.1 Introduction

The **BicWin** user interface supports a program test on the base of highlevel language debugging. This means that the source text of programs, which is written in a highlevel language (e.g. C51) or in assembler language, can be loaded by **BicWin** and is represented during the debugging process.

The sourcetext is shown in the Sourcetext window. All debugging features known from the Assembler window are available in the Sourcetext window, too (e.g. execution of the program line by line). In the real time trace memory sourcetext lines instead of assembler opcodes can be traced. Highlevel language variables are represented and altered corresponding to their type definition.

The switching between Assembler and Sourcetext window makes it possible to execute a program in different “resolution”.

The assumption is, that the relation between the line numbers of the original source text and the addresses of the executable code is known. The used compiler or assembler software must generate the necessary symbolic and line number information.

9.2 Invoking the Compiler, Loading a Program

The **BicWin** user interface uses different files for handling sourcetext during debugging. They are:

- a file with the program code and the symbolic information
- one or more files containing the source text

For the cooperation with the **BicWin** software it is important to correctly select the various options of the compiler, so that the program and symbol files could be created in a form readable by the emulator. Very important are the line number symbols because they are needed for the relation of program code and source text line. To access highlevel language variables, symbolic names and typedef information are needed. Check in your compiler manual, which options must be set to generate this information (e.g. Keil-C51: objectextend). The function “Load status” of the **BicWin** software shows how many symbol names and line numbers were loaded.

After starting up the emulation software **BicWin**, a program to be tested is loaded by means of the function "File Load”. This function loads the program file as well as the symbol file and the source text, if source text information are contained in the symbol file. More sourcetext modules are loaded by the function “File Open”. It should be noticed that the program file and the source text are loaded from the directories determined under "Options Directories".

9.3 Program execution

The loaded source text is represented in the Sourcetext window. Every line of the text can be selected by the mouse or the cursor keys. The line with the actual program counter, i.e. that line of the source text, which is to be executed next, is optically highlighted.

The Sourcetext window offers the same functions for executing a program and for debugging as the Assembler window (see chapter 8.4). The execution of a program in a high-level language does not differ in principle from that of an assembler program. The essential differences are the representation of the current program sector and the execution of single steps.

Breakpoints can be set only on the lines which have been translated into the corresponding program code by the compiler, so they cannot be set e.g. on empty lines, or on comment lines. If either the program counter of the CPU or a breakpoint cannot be set on some line, since no corresponding line number symbol for this line has been generated by the compiler, then an error message is given.

It can always be switched between Assembler window and Sourcetext window and therewith between the highlevel language representation and the assembler one. We can also determine in this way whether in case of single steps an assembler command is executed or whether the program will be executed line by line (i.e a single step corresponds to a line of the source program). Highlevel language single steps are only executed, when the Assembler window is not selected and the program counter points to a line number.

9.4 Handling of several modules

If the program is composed of several source text files (modules), more than one text files can be represented in the Sourcetext window. With the function “File Open” every text file can be loaded.

If the option “Follow program counter” is activated (default setting), more sourcetext files are reloaded automatically when the program counter points to another module after a break or a single step.

More than one Sourcetext window can be opened. In this windows the option “Follow PC” is switched off to enable one window for the actual program section and other windows for fixed program representations.

Switching between different modules is possible only, if sourcetext file names and module names are contained in the symbol table.

10 Options for P0/P2 Emulation

An in-circuit-emulator needs the bus signals of the emulated controller, to control the emulation memory, the real time trace memory ect. So it is no severe problem to emulate applications which use an external bus. More difficulties arise if true single chip applications without an external bus have to be emulated, because port pins are lost if used as bus signals. Therefore the bus signals must be connected in another way or the ports must be regenerated. For 80C51 controllers this is true for ports P0 and P2.

The **BICEPS** emulator supports the following modes of the emulated controller:

1. Applications with external bus. P0 and P2 operate as address and data bus.

2. If P0 and/or P2 operate in I/O mode (no external bus), the BICEPS emulator uses the **Softhook** emulation technique developed by Brendes. The I/O information of ports P0 and P2 are generated by the emulator and not by the controller. Nearly all bit and most byte operations to P0 and P2 can be emulated in real time. Using the Keil C51 compiler, **Softhook** compatible code can be generated.

10.1 The BICEPS Softhooks

For the controller, which can not support bondout or hardware-hook mode emulation for ports P0 and P2, the **BICEPS** emulator offers **Softhook** emulation. This method emulates the port P0 and P2 and the opcodes accessing this ports; the port registers are located in the emulator hardware and not in the controller. During program execution P0/P2 operations are detected and executed in real time by the emulator while the controller excutes opcodes which have exactly the same clock cycles as the emulated opcode. Therefore the emulated controllers works like a single chip, although it is running in external bus mode.

Most opcodes accessing P0 or P2 – including nearly all bit opcodes - can be emulated. Those opcodes which can't be emulated directly (e.g. MOV P0,A) are replaced by 3 byte intrinsic functions. The C51 compiler can be forced to generate **Softhook** compatible code; nearly no loss in performance can be detected. If the **Softhook** compatible opcodes are used only, Assembler programming is possible without problems (see chapter 10.3)

The basic function of the **SoftHook** methode is:

1. Regeneration of ports P0 and P2 and the operations accessing these ports by emulator hardware (port-replacement-unit PRU).
2. The code generated by compiler or assembler is checked automatically for **Softhook** compatibility. Source text lines, which are not compatible, can be selected by mouse click. Normaly more than 95% of all C51-statements with P0/P2 accesses can be emulated without changes.
3. The generated hex file can be programmed in the controller or loaded in the emulator directly.
4. Therefore the result is no or a minimal loss in perfomance.
5. The controller operates during emulation in normal bus mode. All P0/P2 opcodes are recognized by the **BICEPS** emulator in real time. They are replaced by compatible opcodes, which have the same execution time and which can trigger the port replacement unit.

It is important to know for the user:

- **no difference in code size or timing between emulator and controller**
- **no or minimal loss of performance**
- **C51 compiler compatible**

10.2 Compatibility with C51-Compiler

The following method is used to make sure that the C51 compiler generates **Softhook** compatible code:

After the link process, it is checked, wether the generated code is **Softhook** compatible. If a “MOV portbit,C” opcode is detected (the only bit operation which cannot be emulated directly), a “_nop_” statement has to be inserted in the corresponding source text line. This statement is compiled to a “NOP” opcode.

For incompatible byte access opcodes the **BICEPS** emulator supports four routines to access P0 and P2. This are:

Set port byte: WriteP0, WriteP2
Read port byte: ReadP0, ReadP2

This function calls have to be used in source lines with a incompatible code generation; the right **Softhook** intrinsic functions (3 bytes, see chapter 10.3) are inserted then automatically.

Examples:

C51 statement	replaced by	Softhook Intrinsic function (3 Bytes)
P2_3 = bitvar;	P2_3 = bitvar; _nop_;	MOV P2.3,C NOP
P0 = variab;	WriteP0(variab);	MOV P0,R7 MOV A,R7

This four routines requires that they are executed in register bank 0. If port accesses with different register bank are needed, the following routines have to be used:

Set port byte: WriteP0RegBank, WriteP2RegBank
Read port byte: ReadP0RegBank, ReadP2RegBank

Configure register bank to use in options dialog on register card hardware.

The Softhook emulation method is optimized for the Keil C51 compiler. The user interface **µVision** is extended by a special tool: the **Softhook** checker. It checks the code generated by the compiler and generates the right hex and object file. If an incompatible opcode is detected, an error message is

presented; the corresponding line of the source text can be selected directly by mouse click. In this line the right function call (WritePx, ReadPx) or an "_nop_;" statement has to be used.

If the compiler, linker and Softhook checker have been executed without error messages, the generated code can be used to program the controller or to load in the emulator. **There are not different program versions for emulator and controller.**

Because only a few source text lines have to be altered and because in this cases only 3 bytes for the intrinsic functions are necessary, there is no or a minimal loss in performance, which is not significant normally.

10.3 Hints for assembler programming

The Softhook checker utility makes sure that compiled programs are **Softhook** compatible. It can be used for A51 assembler programs, too. But the assembler programmer has to know, which 8051 opcode can be emulated and which not. The opcodes can be classified in three classes:

1. can be emulated without restrictions
2. can be emulated by an intrinsic function
3. can not be emulated.

The following tables show, which bit and byte opcodes with P0/P2 accesses can be emulated by **Softhooks**:

Opcode	can be emulated	intrinsic function
CLR bit	yes	-
SETB bit	yes	-
CPL bit	yes	-
MOV C,bit	yes	-
ANL C,bit	yes	-
ANL C,/bit	yes	-
ORL C,bit	yes	-
ORL C,/bit	yes	-

JB bit,rel	yes	-
JNB bit,rel	yes	-
JBC bit,rel	yes	-
MOV bit,C	with intrinsic function	MOV bit,C NOP

table 10.1 bit opcodes

Opcode	can be emulated	intrinsic function
MOV A,Px	yes	-
MOV direct,Px	yes	-
MOV Px,#data	yes	-
ANL/ORL/XRL Px,#data	yes	-
ANL/ORL/XRL A,Px	yes	-
ADD/ADDC/SUBB A,Px	yes	-
INC/DEC Px	yes	-
DJNZ Px,rel	yes	-
CJNE A,Px,rel	yes	-

MOV Px,A	with intrinsic function	MOV R7,A
		MOV Px,R7
		(register bank 0)

MOV Px,R7	with intrinsic function WritePx();	MOV Px,R7
		MOV A,R7
		(register bank 0)

MOV R7,Px	with intrinsic function ReadPx();	MOV 07H,Px (register bank 0)
ANL/ORL/XRL Px,A	no	-
MOV Px , Ri/@Ri/direct	no	-
MOV Ri/@Ri , Px	no	-
PUSH/POP Px	no	-
XCH A,Px	no	-

table 10.2 Byte opcodes

Additionally the following opcode sequences with P0/P2 accesses are recognized as Softhooks:

Opcode sequence	can be emulated
MOV bit,C CLR A	yes
MOV A,#data MOV Px,A	yes
CLR A MOV Px,A	yes
MOV Ri,#data MOV Px,Ri	yes

table 10.3 Emulatable opcode sequences

Not executable code in program memory (e.g. constant tables accessed by MOVC instructions) can detected wrongly as Softhook only, if the assembler generates no line number symbols. In this case

- two 00 bytes can appended to the constant table or
- the first opcode following the constant table can be marked by a label with the beginning "_sh_" (e.g. "_sh_label1")

10.4 Specialties for Winbond Turbo51-Controllers (77Exx)

The Turbo51 controllers of Winbond have a faster program execution than standard 80C51 CPUs. Therefore other intrinisc functions are necessary for this controllers.

The Winbond Turbo51 controllers need more program bytes for **Softhook** compatibility. For this reason in C51 statements additional „_nop_;“ statements must be inserted, if a **Softhook** intrinsic function is used:

Original C51 statement	C51 statement (Winbond 77x)	Intrinsic function (Winbond 77x)	Hex codes	Bytes / Cycles
Px = (...);	WritePx(...); _nop_;	SJMP $+2 MOV Px,R7	80 00 8F 80/A0	4 / 5
Px_y = (...);	Px_y = (...); _nop_; _nop_;	MOV bit,C SJMP $+2	92 8y/Ay 80 00	4 / 5

Assembler programmers can use the following opcodes:

Original opcode	Bytes / Cycles	Softhook compatible opcodes	Hex codes	Bytes / Cycles
MOV Px,A	2 / 2	SJMP $+2 MOV Px,A	80 00 F5 80/A0	4 / 5
MOV Px.y,C	2 / 2	MOV Px.y,C SJMP $+2	92 8y/Ay 80 00	4 / 5

Contrary to standard 8051 controllers, the reading access

```
MOV  Ri,Px
```

can be emulated without restrictions.

10.5 Specialties for Maxim/Dallas 80C320/323/520/530 and TI MSC1210 controllers

The Maxim/Dallas controllers and the MSC12xx controllers of Texas Instruments have a faster program execution than standard 80C51 CPUs. Therefore other intrinisc functions are necessary for this controllers.

The Dallas/Maxim and TI MSC12xx controllers need more program bytes for **Softhook** compatibility. For this reason in C51 statements additional „_nop_;“ statements must be inserted, if a **Softhook** intrinsic function is used:

Original C51 statement	C51 statement (MSC12xx)	Intrinsic function (MSC12xx)	Hex codes	Bytes / Cycles

Px = (...);	WritePx(...);
	nop;
		MOV Px,R7
		SJMP $+1
		MOV R7,A
			8F 80/A0
			80 FF
				4 / 6

Px_y = (...);	Px_y = (...); _nop_; _nop_; _nop_;	MOV bit,C SJMP $+2 NOP	92 8y/Ay 80 00 00	5 / 6

Assembler programmers can use the following opcodes

Original opcode	Bytes / Cycles	Softhook compatible opcodes	Hex codes	Bytes / Cycles
MOV Px,A	2 / 2	SJMP $+2 MOV Px,ACC	80 00 85 E0 80/A0	5 / 6
MOV Px.y,C	2 / 2	MOV Px.y,C SJMP $+2 NOP	92 8y/Ay 80 00 00	5 / 6

Contrary to standard 8051 controllers, the reading access

```
MOV  Ri,Px
```

can be emulated without restrictions.

10.6 Advantages and disadvantages of Softhooks

The advantages of **SoftHook** emulation are:

- No difference between emulation and controller **(**timing and code size)
- Bit- and byte-access to P0 and P2
- Compatible with C51-compilery by use of intrinsic functions
- No critical bus timing for higher frequencies, because a lower bus clock is used compared with the hardware-hook mode.
- Independent of silicon vendors, suits all controllers with 80C51 architecture
- lower price

The disadvantages are:

- Current 80C51 programs have to be checked by the the Softhook utility for compatibility
- At some few program parts there can be a minimal loss in performance (typical 1 cycle)

10.7 Necessary conditions for BICEPS Softhooks

At the moment the **BICEPS** Softhook routines are compatible with Keil-C51-compiler, i.e. parameters are transfered in register R7 of registerbank 0.

The expansion tool “Softhook checker” is integrated in the **µVision** user interface by definig it as „Run User Program #1“ in the menue „Project/Options for Target/Output“.

For the **Softhook** functions (ReadPx and WritePx) 4 dummy addresses are used (0FFF8H-0FFFBH). The correct definition files for the C51-compiler (see files Shook.h, Shook.a51) and a sample program come with the **BicWin** de-bugging software. The IAP entry point at 0FFF0H can be used without restrictions.

The following files are on the disc:

ShChecker.exe
: checkes a linked OMF or HEX file for Softhook compatibility and generates the correct hex file for programming

ShChecker.txt
: Describes softhook checker incl. command parameters

Shook.h, Shook.a51, Shook.obj:
: Definition file for Softhook addresses

Shdemo.omf
: Demo program for Softhook use with softhook intrinsic functions with the files ShDemo.c, ShDemo.obj, ShDemo.uv2, ShDemo.opt

ShRotP51:
: Demo pogram for Softhook use with the files ShRotP51.c, ShRotP51.obj, ShRotP51.uv2, ShRotP51.opt

The project- and option-files of Keil µVision are correctly defined in that way, that ShChecker is called after link process.

10.8 Softhook sample program

here you find a C51 sample program with typical bit- and byte-accesses to P0 and P2. The example shows, which routines can be used and which 80C51 code is generated:

```
void main (void)
{
  uchar i;

  while (TRUE)
  {
    P2_3 = 1;
    if (P0_4) i=1;
    else i=i+1;
    i = P2;
    P0 = i;
  }
}
```

The **Softhook** checker detects a problem with the line `P0 = i;` , only:

```
Demo.C(20): ERROR: MOV P0,A can't be emulated: replace with: WriteP0()
```

The statement in this line is replaced by an intrinsic function call:

```
void main (void)
{
  uchar i;

  while (TRUE)
  {
    P2_3 = 1;
    if (P0_4) i=1;
    else i=i+1;
    i = P2;
    WriteP0(i);
}
```

The following final program is generated by the compiler:

```
; FUNCTION main (BEGIN)
?C0001:
      SETB    P2.3
      JNB     P0.4,?C0003
      MOV     i,#01H
      SJMP    ?C0004
?C0003:
      INC     i
?C0004:
      MOV     A,P2
      MOV     i,A
      MOV     R7,i
      MOV     A,R7  ; Intrinsic Function
      MOV     P0,A
      SJMP    ?C0001
      RET
; FUNCTION main (END)
```

11 BICEPS-Debug-Connector

11.1 Basic information

By use of the pin headers described in this chapter it is possible to connect the test board and the **BICEPS** emulator by a flat ribbon cable. The controller is not replaced, a special POD is not needed.

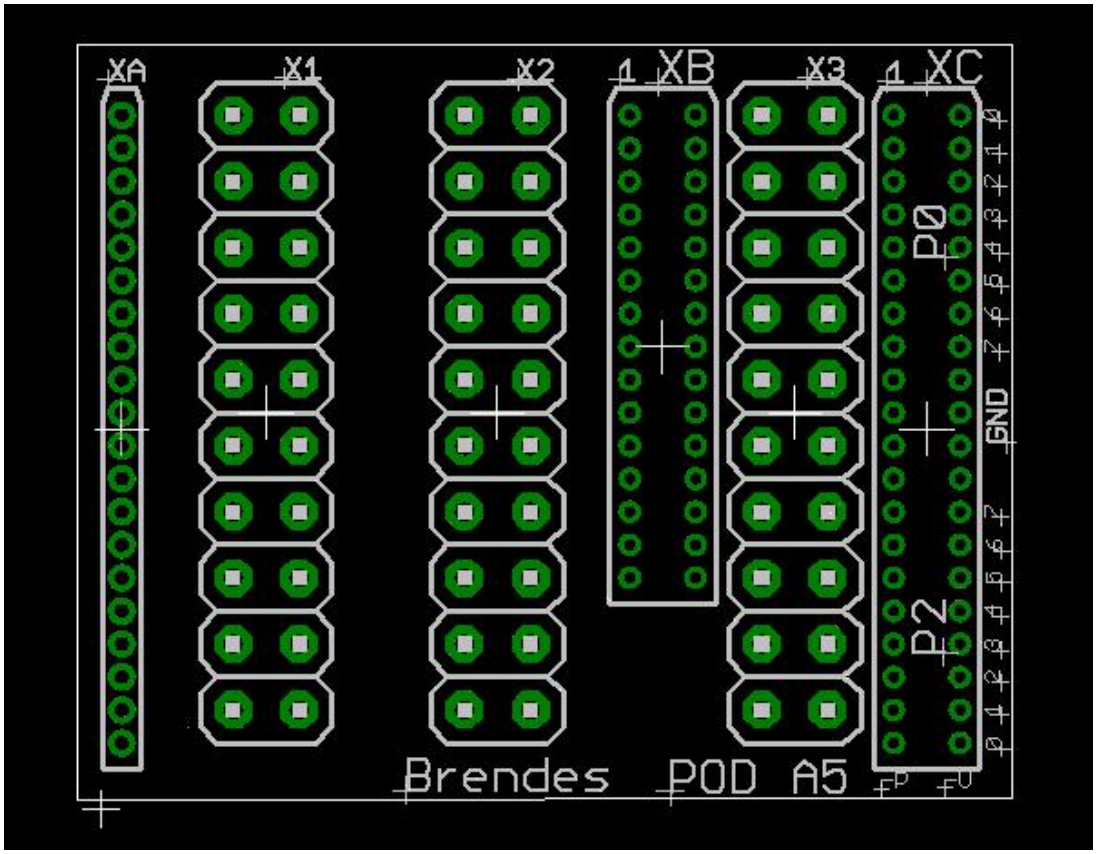
Some signals of the controller have to be separated from the rest of the circuit. Their connection is done via the debug connector. The signals which are connected directly with the controller have the extension „_CPU“. The signals which are connected with the rest of the circuit have the extension „_Board“ .

If the board is to be operated without emulator, the emulator cable is removed and replaced by short-circuit jumpers, i.e. „_CPU“- and „_Board“-signals are connected.

It can be choosen between two pin header types:

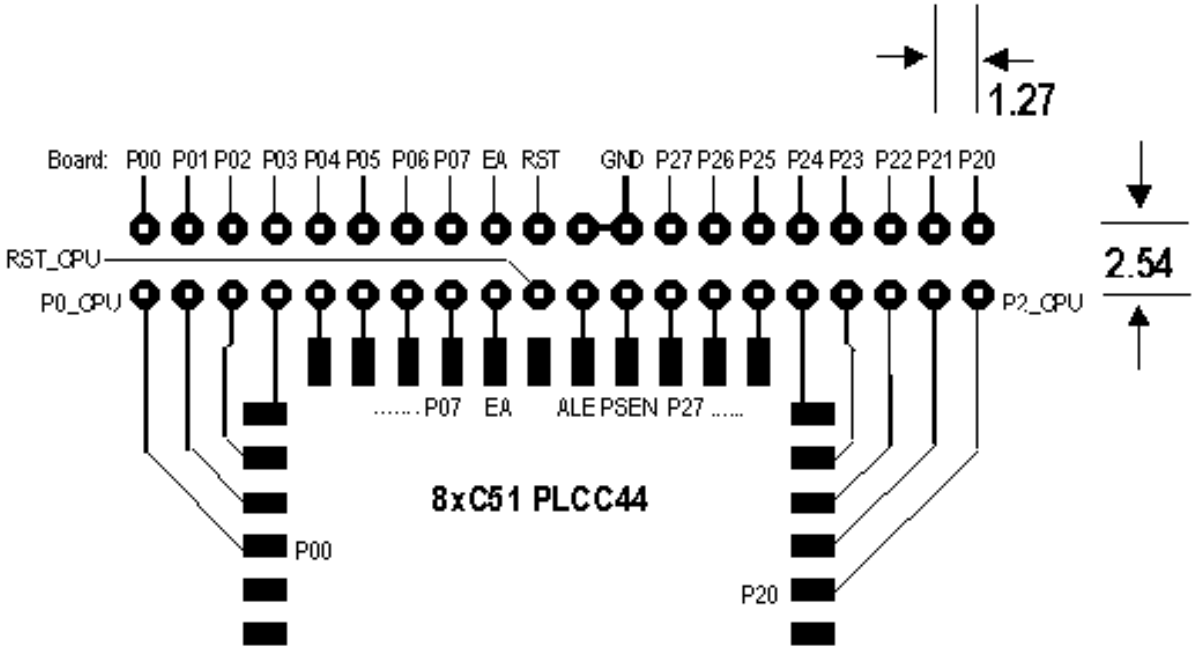
1. 40 pin **BICEPS** debug connector emulates bus and true single chip applications
2. 30 pin older ICE-connect adapter designed for applications with external bus. Softhook generated P0/P2 signals are available on an additional connector

The following picture shows the adapter board (labeled A5) with both possible connectors:



XC : **BICEPS** debug connector, 40 pin
XB : Former debug connector, 30 pin (ICE-connect)

PCB layout for 40 pin connector and PLCC44 controller:



11.2 Applications without external bus P0=Port, P2=Port (Softhook emulation)

XC high-density pin header, 40 pins, grid 1.27/2.54

P00_CPU	1	2	P00_Board
P01_CPU	3	4	P01_Board
P02_CPU	5	6	P02_Board
P03_CPU	7	8	P03_Board
P04_CPU	9	10	P04_Board
P05_CPU	11	12	P05_Board
P06_CPU	13	14	P06_Board
P07_CPU	15	16	P07_Board
EA_CPU	17	18	EA_Board
Rst_CPU	19	20	Rst_Board
ALE_CPU	21	22	GND
PSEN_CPU	23	24	GND
P27_CPU	25	26	P27_Board
P26_CPU	27	28	P26_Board
P25_CPU	29	30	P25_Board
P24_CPU	31	32	P24_Board
P23_CPU	33	34	P23_Board
P22_CPU	35	36	P22_Board
P21_CPU	37	38	P21_Board
P20_CPU	39	40	P20_Board

Separated signals: EA, RST, P00..P07 and P20..P27

11.3 Applications with external bus P0=Bus, P2=Bus or P0=Bus, P2=Port (Softhook emulation)

XC high-density pin header, 40 pins, grid 1.27/2.54

P00_CPU	1	2	
P01_CPU	3	4	
P02_CPU	5	6	
P03_CPU	7	8	
P04_CPU	9	10	
P05_CPU	11	12	
P06_CPU	13	14	
P07_CPU	15	16	
EA_CPU	17	18	EA_Board
Rst_CPU	19	20	Rst_Board
ALE_CPU	21	22	GND
PSEN_CPU	23	24	GND
P27_CPU	25	26	P27_Board
P26_CPU	27	28	P26_Board
P25_CPU	29	30	P25_Board
P24_CPU	31	32	P24_Board
P23_CPU	33	34	P23_Board
P22_CPU	35	36	P22_Board
P21_CPU	37	38	P21_Board
P20_CPU	39	40	P20_Board

Separated signals: EA, RST, P00..P07 and P20..P27

11.4 BICEPS ICE-connect

For compatibility with older debug connectors (ICE-connect), the **BICEPS** emulator adapter board A5 has an additional 30-pin connector. It combines the ICE-connect (designed for bus mode only) with the Softhook emulation technique.

P0/P2 static I/O signals can be additionally connected via pins 2..16 (P00..P07) and 26..40 (P27..P20) of XC .

XB Standard pin header, 30 pins, grid 1.27/2.54 mm

GND	1	2	AD0_CPU
AD1_CPU	3	4	AD2_CPU
AD3_CPU	5	6	AD4_CPU
AD5_CPU	7	8	AD6_CPU
AD7_CPU	9	10	GND
A8_CPU	11	12	A9_CPU
A10_CPU	13	14	A11_CPU
A12_CPU	15	16	VCC
A13_CPU	17	18	A14_CPU
A15_CPU	19	20	GND
PSEN_CPU	21	22	PSEN_Board
RD_CPU	23	24	RD_Board
WR_CPU	25	26	WR_Board
Rst_CPU	27	28	Rst_Board
GND	29	30	ALE-CPU

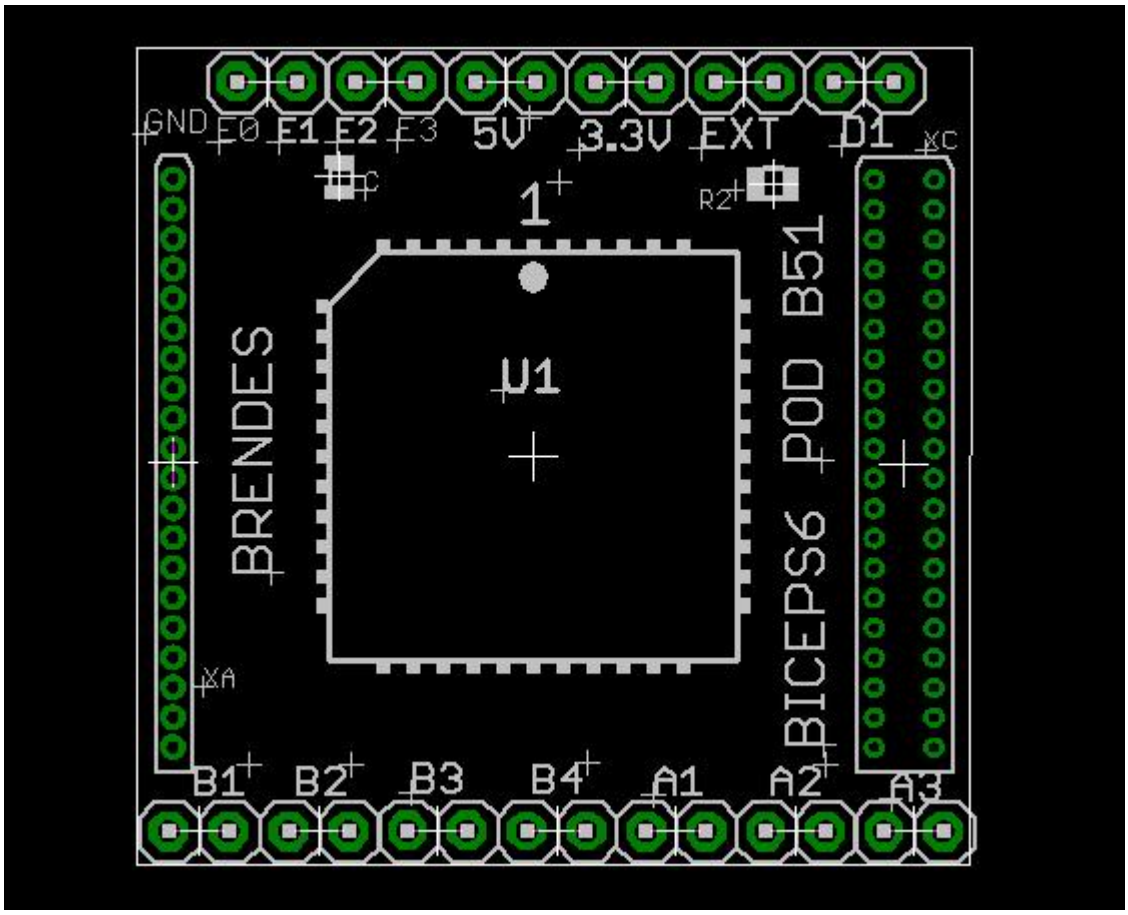
Condition: EA_CPU = GND
Separated signals: RST, PSEN, RD and WR

12 Processor Adapters (PODs)

For different controller derivates and package variants there are different PODs with the controller emulated, and converters. By replacing the POD the **BICEPS** emulator is adapted to the controller.

12.1 POD B51 for controllers in PLC44 package

This POD is used for all controllers with standard pinout. It has an PLCC44 output, which can be plugged directly in a PLCC socket. DIL and QFP layouts are contacted by additional converters. The POD B51 is plugged directly under the adapter board A5 (see chap. 11.1).



Jumper settings:

CPU clock (Jumper A)
A1 : Emulator clock (default)
A2,A3: external clock or crystal
(XTAL pins are connected directly with the board)

RD- and WR signals (Jumper B)
B1, B2, B4: P3.7 = RD, P3.6 = WR (default)
B3: P3.7 = Port, P3.6 = Port

Power supply of the controller on the POD
5V: internal 5V of the emulator
3.3V: internal 3.3V of the emulator (default)
EXT: external power supply by the user board

External reset input (Jumper D)
D1: set: PullDown,
open: PullUp (default)

Additional connector:

E0..E3: Banking signals A16..A19

12.2 More processor adapters

All processor adapters of former Brendes emulators named **BICEPS-V** and **BICEPS-IV** can be used. Because these PODs have larger mechanical dimensions, the adapter board A5 must be replaced by the board A4.

Available are:

1.	POD B51-4	for all standard PLCC44 controllers
2.	POD B51CC	for Atmel89C51CC01/02/03/AC2/AC3
3.	POD BC51CC-52	for Atmel89C51CC03 (PLCC52)
4.	POD B51-68	for Atmel 8xC51RD2/ED2 (PLCC68)
5.	POD B5131USB	for Atmel 89C5131 (PLCC52)
6.	POD B51SND1	for Atmel 8xC51SND1 (QFP80)
7.	POD BuC8xx	for AnalogDevices ADμC812/816/824
8.	POD B552	for Philips 8xC552
9.	POD B592	for Philips 8xC592
10.	POD B537	for Infineons C515, C517, 80C535, 80C537

12.2.1 Jumper settings

CPU clock (Jumper A)

A1=2-3:	Emulator clock (oscillator of partA, default)
A1=1-2, A2:	external clock or crystal (XTAL pins are connected directly with the board)

RD- and WR signals (Jumper B)

B1, B2, B4:	P3.7 = RD,	P3.6 = WR	(default)
B3:	P3.7 = Port,	P3.6 = Port	

Power supply of the adapter (Jumper C)

JC=1-2:	internal power supply (default)
JC=2-3:	exteral power supply by the user board

Reset logic (Jumper D)

D1=2-3, D2=2-3:	Standard-80C51-RESET, high activ (default)
D1=1-2, D2=1-2:	Inverted reset: /RESET, low activ (for example 8xC591)

For details of the mechanical dimensions and jumper locations refer to the BICEPS-V emulator manual.

13 External Program Memory Connector (Universal Adapter)

A special form of debug connector of the **BICEPS** emulator is called Universal adapter: It has the same pin layout as a standard memory circuit. So it can be plugged in a socket of an external program memory (Flash Eprom) for example.

Besides the signals of the program memory socket two further signals are of importance:

- The emulator gets control about the processor on the circuit via a reset output. The RESET signal has to be fed into the circuit or connected to the RESET pin of the processor.
- The ALE signal of the processor must be taken off.

These two necessary connections are realized by means of two additional cables (with test clipses), which are connected to the Universal adapter.

13.1 Memory circuit type

The address bus signals A0...A15 are taken via the Universal adapter, the data bus signals D0...D7 as well as the /PSEN signal via the /CS or the /OE pin of the program memory socket. For this a DIL plug with 32 pins is provided. PLCC versions can be adapted by means of a corresponding PLCC converter DIL28-PLCC32 or DIL32-PLCC32.

If the program memory does not use all 16 address lines, i.e. the program memory is <64k, this option has to be set in the “Options” menue of the **BicWin** software.

EPROM up to 64k, DIL28, PLCC32 (nc 1,12,17,26):

Pin number DIL	Pin number PLCC	Signal	Pin number DIL	Pin number PLCC	Signal
3	2	A15	30	32	
4	3	A12	29	31	A14
5	4	A7	28	30	A13
6	5	A6	27	29	A8
7	6	A5	26	28	A9
8	7	A4	25	27	A11
9	8	A3	24	25	/OE
10	9	A2	23	24	A10
11	10	A1	22	23	/CS
12	11	A0	21	22	D7
13	13	D0	20	21	D6
14	14	D1	19	20	D5
15	15	D2	18	19	D4
16	16	GND	17	18	D3

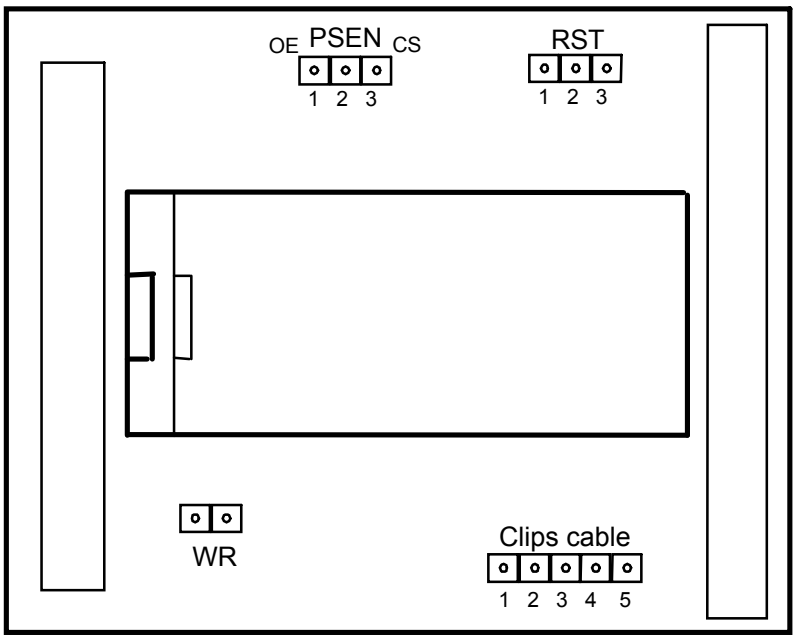
The DIL32 layout is compatible with the DIL28 layout: Pins 1,2,31,32 remain unconnected

EPROM larger than 64k, DIL32, PLCC32:

Pin number DIL	Pin number PLCC	Signal	Pin number DIL	Pin number PLCC	Signal
1	1	A18	32	32	
2	2	A16	31	31	/WR
3	3	A15	30	30	A17
4	4	A12	29	29	A14
5	5	A7	28	28	A13
6	6	A6	27	27	A8
7	7	A5	26	26	A9
8	8	A4	25	25	A11
9	9	A3	24	24	/OE
10	10	A2	23	23	A10
11	11	A1	22	22	/CS
12	12	A0	21	21	D7
13	13	D0	20	20	D6
14	14	D1	19	19	D5
15	15	D2	18	18	D4
16	16	GND	17	17	D3

If a banking memory application is emulated (program memory >64k), the address lines A16..A18 are connected via this program memory socket.

13.2 Jumper Settings



Reset logic (Jumper RST)

RST=1-2: Inverted reset: /RESET, low activ (for example 80C535)
RST=2-3: Standard-80C51-RESET, high activ (default)

Connection of /PSEN signal (Jumper JP1)

PSEN=1-2: /PSEN exists at pin 24 of the socket (/OE) (default)
PSEN=2-3: /PSEN exists at pin 22 of the socket (/CS)

WR-signal (Jumper WR)

set: WR is used to write to program memory (Flash circuits)
removed: WR is not used for program memory (default)

13.3 Connections via Clips Cable

The Universal adapter of the **BICEPS** emulator has up to 5 connections for signals, which cannot be contacted directly via the EPROM socket. Two signals have to be connected in any case; the remaining signals are optional:

1: RESET-Out (necessary)
2: RD (optional)
3: WR (optional)
4: RESET-In (optional)
5: ALE (necessary)

Reset Output (Clips cable 1)

The feeding of a reset signal into the circuit under test is absolutely necessary to control the processor of the circuit. It is to notice here that the Reset signal output is not disturbed by the user board. Admissible is e.g. a voltage monitoring and reset device with an open collector output. Problems can occur with an electrolytic capacitor parallel to the reset signal, it must be eliminated during the test phase.

Must be connected!

Write and Read (Clips cables 2,3)

The connections 2 (/RD) and 3 (/WR) are provided for systems with external data memory. If the signals are connected, accesses to the external data memory can be defined as break condition and traced in the real time trace memory. If the /WR-signal is connected via the socket (jumper /WR), clips cable 3 is not necessary.

Input for external Reset (clips cable 4)

Not supported.

ALE signal (clips cable 5)

Must be connected!

Appendix

A BicWin short keys

A.1 Function keys

		ALT	CTRL	SHIFT
F1	Help			
F2	CPU Reset	Reset all	Clear trace	Macro short key
F3	Search Again			Macro short key
F4	Run to Cursor	Terminate	Close window	Macro short key
F5	Go to		Go to symbol	Macro short key
F6	Next window			Previous window
F7	Single step			Macro short key
F8	Jump over Call			Macro short key
F9	Run			Macro short key
F10	Menue			Context menue

A.2 ALT-keys

key	function	type
Alt+A	Assembler	(window)
Alt+B	Break	(dialog)
Alt+C	Command	(window)
Alt+D	Debug	(menue)
Alt+E	Trace	(menue)
Alt+F	File	(menue)
Alt+H	Help	(menue)
Alt+I	IntData	(window)
Alt+K	Break	(menue)
Alt+M	Macro	(menue)
Alt+N	Window	(menue)
Alt+O	Options	(menue)
Alt+P	Program	(window)
Alt+R	Register	(Fenster)
Alt+S	Sourcetext	(window)
Alt+T	Trace	(window)
Alt+V	Views	(menue)
Alt+W	Watch	(menue)
Alt+X	ExtData	(window)
Alt+Y	Memory	(menue)
Alt+1	Watch 1	(window)
Alt+2	Watch 2	(window)
Alt+3	Watch 3	(window)
Alt+4	Watch 4	(window)

B External Inputs

Up to 16 external digital inputs (logic probes) are available; the signals are traced in the real-time trace memory or can be used as break inputs (see chapter 6 and 7). The inputs have pull-up-resistors.

GND	1	2	
	3	4	
EXT15	5	6	EXT14
EXT13	7	8	EXT12
EXT11	9	10	EXT10
EXT9	11	12	EXT8
EXT7	13	14	EXT6
EXT5	15	16	EXT4
EXT3	17	18	EXT2
EXT1	19	20	EXT0

To connect the external inputs to the target board, the emulator case must be opened. A 20-pin flat ribbon cable is connected to conncetor X4 on the BICEPS emulator board.