## QuickStart Instructions

# PowerPC Kit

# phyCORE-MPC5554

**Using iSYSTEM winIDEA for PowerPC Development Tool Chain**

Note: The PHYTEC Tool-CD includes the electronic version of the phyCORE-MPC5554 English Hardware Manual

**Edition: January 2013**

A product of a PHYTEC Technology Holding company

In this manual are descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (™) and copyright (©) symbols does not imply that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, PHYTEC Messtechnik GmbH assumes no responsibility for any inaccuracies. PHYTEC Messtechnik GmbH neither gives any guarantee nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. PHYTEC Messtechnik GmbH reserves the right to alter the information contained herein without prior notification and accepts no responsibility for any damages which might result.

Additionally, PHYTEC Messtechnik GmbH offers no guarantee nor accepts any liability for damages arising from the improper usage or improper installation of the hardware or software. PHYTEC Messtechnik GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

| | EUROPE | NORTH AMERICA |
|---|---|---|
| Address: | PHYTEC Messtechnik GmbH<br>Robert-Koch-Str. 39<br>D-55129 Mainz<br>GERMANY | PHYTEC America LLC<br>203 Parfitt Way SW, Suite G100<br>Bainbridge Island, WA 98110<br>USA |
| Ordering Information: | +49 (6131) 9221-32<br>sales@phytec.de | 1 (800) 278-9913<br>sales@phytec.com |
| Technical Support: | +49 (6131) 9221-31<br>support@phytec.de | 1 (800) 278-9913<br>support@phytec.com |
| Fax: | +49 (6131) 9221-33 | 1 (206) 780-9135 |
| Web Site: | http://www.phytec.eu | http://www.phytec.com |

2nd Edition: January 2013

## Index of Figures

# 1 Introduction to the phyCORE-MPC5554 Rapid Development Kit

**This QuickStart provides:**

- General information on the PHYTEC phyCORE-MPC5554 System on Module.
- An overview of iSYSTEM AG winIDEA GNU development tool chain.
- Instructions on how to run example programs on the phyCORE-MPC5554, mounted on the PHYTEC Development Board, in conjunction with iSYSTEM winIDEA tools.

Please refer to the phyCORE-MPC5554 Hardware Manual for specific information on such board-level features as jumper configuration, memory mapping and pin layout. Selecting the links on the electronic version of this document links to the applicable section of the phyCORE-MPC5554 Hardware Manual.

## 1.1 Rapid Development Kit Documentation

This "Rapid Development Kit" includes the following electronic documentation on the enclosed "PHYTEC Tool-CD-ROM":

- PHYTEC phyCORE-MPC5554 Hardware Manual
- PHYTEC phyCORE-MPC5554 QuickStart Instructions
- MPC5554 controller User's Manuals and Data Sheets

## 1.2 Overview of this QuickStart Instruction

This QuickStart Instruction provides a general "Rapid Development Kit" description, as well as software installation hints and example programs enabling quick out-of-the box start-up of the phyCORE-MPC5554 in conjunction with iSYSTEM winIDEA tools. It is structured as follows:

1) The *"Getting Started"* section uses the example program *"Hello"* to demonstrate the download of user code to the phyCORE-MPC5554 external RAM and Flash memory using iSYSTEM' winIDEA Debugger.

## 1.3 System Requirements

Use of this "QuickStart" requires:

- the PHYTEC phyCORE-MPC5554
- the PHYTEC Development Board with the included DB-9 serial cable and AC adapter supplying 5 VDC/min. 1000 mA
- the iSYSTEM iONE-E Emulator unit (or other iSYSTEM Debugger unit you own). iONE-E is the hardware interface between a standard PC USB 1.1 port and the standard (2.54 mm) 14-pin MPC55xx JTAG/ONCE On-Chip-Debug header connector
- the PHYTEC phyCORE-MPC5554 Tool CD-ROM including winIDEA Setup Files for iONE-E
- computer with 800 MHz Pentium compatible microprocessor, 512 MB RAM, CD-ROM drive, USB port, serial, and Ethernet ports, running Windows XP/2000, 350 MB free hard disk space plus project space

For more information and example updates, please refer to the following sources:

http://www.phytec.com - or - http://www.phytec.eu
support@phytec.com - or - support@phytec.de

http://www.isystem.com
sales@isystem.com
support@isystem.com
holger.wild@isystem.com (Field-Application D-A-CH)

http://www.freescale.com

## 1.4 The PHYTEC phyCORE-MPC5554

The phyCORE-MPC5554 module integrates the 32-bit Freescale MPC5554 PowerPC microcontroller on an advanced PCB layout. All applicable controller signals extend to two 200-pin, high-density Molex connectors. In addition to the on-chip memory (2 MByte high speed Flash, 64 kByte SRAM, 32 kByte Cache), the phyCORE-MPC5554 can be populated with 2 MByte to 8 MByte Standard Flash memories (29LV800..320) and 1 MByte to 16 MByte of Sync. BurstSRAM. The external Flash supports direct on-board programming without additional programming voltages. A serial EEPROM, 4 Kbyte (up to 32 KByte), is available for storing operating parameters. I$^2$C Real-Time Clock with a calendar and alarm function serves as a real-time reference.

The phyCORE-MPC5554 is especially suited for applications requiring processing of calculation-intensive algorithms in addition for handling of numerous complex and time critical external signals.

The MPC5554's integrated Signal Processing Extension (SPE) provides DSP-like hardware-level execution of elementary operations with single precision floating point numbers (FPU) and supports Multiply and Accumulate operations with its MAC unit. The two Enhanced Time Processing Units (eTPU) with its 32 channels (signals) each provides flexible co-processing to meet hard real-time requirements. In addition of the processors processing units, the SOM integrates a very high density FPGA device, which enables a very flexible way to generate the application specific interface needs. Interfaces like PCI bus, normal address/data bus, GPIO, LVDS ports etc. can be implemented by software. This FPGA can accommodate additional co-processing and periphery units like I2C-Master Controller, 1-Wire-Bus Controller etc.

Other controller features supported by the SOM include three on-chip CAN 2.0B controllers, two UARTs; SPI Interface, PWM Ports and 40 channel Dual-ADC.

## phyCORE-MPC5554 Technical Highlights

- System On Module in subminiature form factor (84 x 57 mm) according to phyCORE specifications
- all applicable controller and other logic signals extend to two high-density 200-pin Molex connectors
- processor: Freescale embedded PowerPC MPC5554 (up to 132 MHz clock)

## Internal Features of the MPC5554:

- 32-bit PowerPC core, up to 132 MHz CPU speed
- 32 kByte Cache Memory
- SPE Signal Processing Extention (FPU, MAC Unit)
- MMU Memory Management Unit
- DMA Direct Memory Access Controller
- Interrupt Latency <70ns @132MHz
- 64 kByte SRAM; 32kByte capable of battery buffering
- 2 MByte Flash (read while write functionality)
- two UART’s (eSCI) LIN support
- four SPI interfaces (DSPI)
- three CAN 2.0B interfaces
- two TPU Time Processing Units with 32 channels (pins) each
- 24 channels (pins) Timer system (eMIOS) for PWM etc.
- dual 12-bit ADC with 40 (65) channels (ext. MUX)
- multi-purpose I/O signals
- JTAG/ONCE/NEXUS/Nexus test/debug port

## Memory Configuration[1]:

- SRAM: 1 MByte to 16 MByte flow-through synchronous burst-RAM, 32-bit access, 0 wait states, 2-1-1-1 burst mode
- Flash-ROM: 2 MByte to 8 MByte asynchronous standard Flash-EEPROM, 32-bit access
- $I^2C$ Memory: 4 kByte EEPROM (up to 32 kByte, alternatively $I^2C$ FRAM, $I^2C$ SRAM)

## Other Board-Level Features:

- UART: two RS-232 transceivers for channel A and B (RxD/TxD); also configurable as TTL
- CAN: two CAN transceivers 82C250-compatible for channels A and B; also configurable as TTL
- Ethernet: 10/100 Mbit/s LAN91C111
- FPGA: Lattice XP FPGA XP6/10/15 or XP20 device for
  IP cores: e.g. I2C-Master, 1-Wire-Master, UART, SPI etc.
  Programmable bus bridge (simple address-/data bus, PCI-Bus, DDR-RAM etc.)
  84 external GPIO with programmable characteristics (TTL, CMOS, differential logic, LVDS etc.)
  Application specific control logic and clock generation (PLL)
  Embedded memory: Single-/Dualport SRAM, FIFO etc.
  In-system programmable over JTAG-Emulation
- $I^2C$ Real-Time Clock with calendar and alarm function
- JTAG/JTAG/ONCE/NEXUS/Nexus test/debug port
- industrial temperature range (-40…+85°C)

[1]: Please contact PHYTEC for more information about additional module configurations.

## phyCORE-MPC5554 Development Board

The phyCORE-MPC5554 Development Board is fully equipped with all mechanical and electrical components necessary for the speedy and secure insertion and subsequent programming of the phyCORE-MPC5554 module with high-density (0.635 mm pitch) pin header connectors. RS-232 and CAN interface signals extend from the module to dual-stacked DB-9 connectors on the Development Board, while the Ethernet signals extend to a RJ-45 connector.
Two debugging connectors are provided. A reduced 14-pin JTAG/OnCE and a full 38-pin JTAG/OnCE/Nexus connector.

## Development Board Technical Highlights

- Development Board (160 x 100 mm)
- high-density (0.635 pitch) Molex connector for speedy and secure insertion, and subsequent programming, of the phyCORE-MPC5554 System On Module
- socket for +5VDC power supply connectivity
- 2x RS-232 at DB-9 sockets
- 2x CAN interfaces at DB-9 plugs
- RJ-45 Ethernet interfaces
- 10-pin JTAG header to the FPGA device
- 14-pin OnCE/JTAG/Nexus interface
- 38-pin OnCE/JTAG/Nexus interface
- two power LEDs for monitoring the supply voltages
- RESET and IRQ push buttons
- Expansion Bus: all signals routed from implemented phyCORE module to 2 x 200-pin Molex connectors, enabling connectivity to PHYTEC **Add-On hardware**

## 1.5 The iSYSTEM winIDEA Development Tool Chain

The iSYSTEM winIDEA itself is a highly functional Integrated Development Environment (IDE) tool for developers to create, compile, assemble and link optimized embedded systems code for the PowerPC and many other architectures. The winIDEA works together with included or your own GNU CC package or ANY common third party Compiler (like i.e. DIAB-Data/Wind-River, Greenhills, Freescale etc.) and other useful tools (like i.e. EasyCode, Tessy, National Instruments LabView etc.). There are interfaces via "isystem.connect" to any Windows based application includes your own PC applications, Excel and others. After building the code winIDEA has a powerful debug control center with target download and flash programming possibilities together with the iSYSTEM Emulator hardware. For On-Chip debugging this could be an "iC5000" universal On-Chip Debug unit with NEXUS Trace, other iSYSTEM Debugger for MPC5xxx or the attached, limited "iONE-E" connected.

The iSYSTEM winIDEA Integrated Development Environment (IDE) is quite similar to the PC market leader "Microsoft Visual Studio" with extensions and changes for embedded developing and debugging. That IDE is intuitive, easy to use and work in a cohesive manner regardless of compiler or target architecture. This means you can reach more markets and solve more problems faster without invest time and money for learning how new different tools work or work together. iSYSTEM winIDEA IDE graphical interface includes an editor, project manager, class browser, universal command line based compiler interface, source-level and assembly-level debugger.

In addition to keeping all your development tools a mouse-click away, the IDE stores compiler, project, Window and debug settings and tracks all the dependencies for your project, simplifying even the most complex development build. The software debugging in winIDEA are state-of-the art and features syntax highlighting of sources and the power to evaluate structures, complex expressions and such things in the debugger. You may correct/change/add source code within the build-in Editor while debugging your application. You are able to perform Multi-Core (i.e. eTPU C-Source Level) debugging. With iC5000 you would be able to Debug and use the famous NEXUS Trace possibilities.

Now you get a brief overview, please refer the details in the following documents - stored in the winIDEA program directory - after installing winIDEA. Partly they are available as on-line help too.

To start with your works please follow instructions in sector 2.4.

The important documents for you regarding winIDEA are:

- **winIDEA.pdf**
  This is the universal reference and user manual
- **OCD MPC5500.pdf (**refer ...\DOC directory and subdir)
  This is the special manual for MPC55xx [**]
- **IONE123_10.pdf**
  This is the description of your iONE [**]

[**] this should be attached to your iONE-E in printed form

iSYSTEM winIDEA consists of the IDE with interfaces to other tools and the Debugger, which are structured as follows:

**IDE:**

Windows-based Integrated Development Environment housing the following tools in one IDE:

- **Project &Build Manager:**
  it shows the collection of groups, its files and support files necessary and used to build a target loadable output file, invokes the assembler, compiler and linker, pass error messages to the build-in Editor. It interfaces to external make utilities too with lot of possibilities.

| | |
|---|---|
| • **Editor:** | double-clicking on a file in the Project Manager Window opens that file in the Editor. |
| • **C compiler:** | GNU Standard compiler package included (*) |
| • **Debugger:** | Fully active source-level and assembly-level Windows-based debugger for iSYSTEM Debug hardware |

(*) You can adapt own GNU or other common compilers. You may edit your sources with your favorite editor outside of winIDEA too - winIDEA then automatically reload the changed sources.

Upon installation of winIDEA - please select "full" installation - , the executables and all manuals, documents, further examples and GNU Compilers are located in the ***C:\iSYSTEM\winIDEA\2012\*** folder (selectable).

The GNU CC tools are then located at ..\gcc\ppc\. There you will find the GNU Documentation too (..\doc\). All tool commands are easily accessible via intuitive pull-down menus with prompted selections.

The winIDEA handling itself has mostly self-describing user Interface. You have three ways for dealing with it:

- Hot-Keys (re-attachable under "Tools => Customize")
- Icons (Buttons for mouse - simply place mouse over it and wait for function explanation)
- Use "right mouse key" for context sensitive local menus. It makes a difference whether you do it in the middle of a Window or at the margin for example, simply try it out!

As depicted on *Figure 1*, winIDEA’s editor, Project/Build manager with connected C/C++ compiler and linker and winIDEA debugger with flash programming possibilities comprise the complete tool chain. You have - configurable - Windows and Window panes for your project, internal editor, disassembly and CPU main registers, global watches and real-time refreshed watches, local variables and context, build manager and search output, memory contents and special-function registers and toolbars for the main functionality.

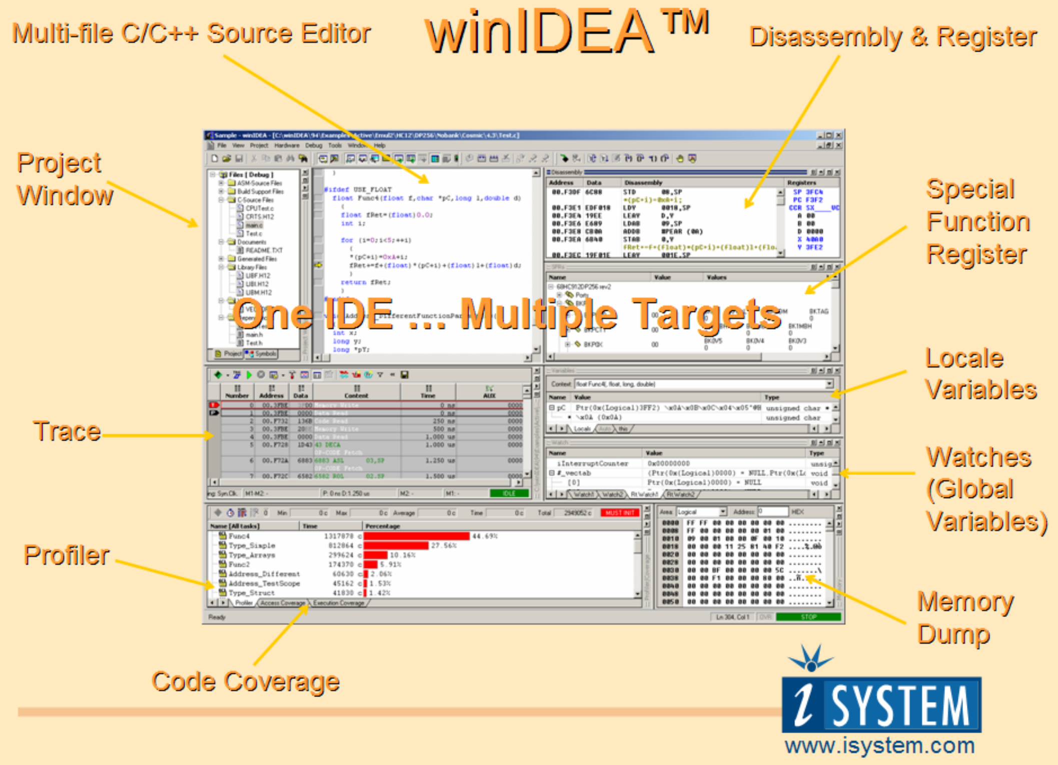


*Figure 1: iSYSTEM winIDEA IDE Architecture i.e. with iC5000*

Please note that you will have no Trace, Profiler or Code Coverage with simple iONE-E - you would need an iC5000. Please ask iSYSTEM directly for a free test run before the trial period end!
As depicted on *Figure 1*, winIDEA's editor, Project/Build manager with connected C/C++ compiler and linker and winIDEA debugger with flash programming possibilities comprise the complete tool chain. You have - configurable - Windows and Window panes for your project, internal editor, disassembly and CPU main registers, global watches and real-time refreshed watches, local variables and context, build manager and search output, memory contents and special-function registers and toolbars for the main functionality.

All IDE commands and functions are accessible via pull-down menus, most of it with short cuts too. All colors and custom key sets for the whole winIDEA parts/Windows can be user-defined.

The integrated **Editor** is a fair and easy-to-use tool to compose and later debug the source code. It includes a context-sensitive coloring of keywords for easy navigation and recognition. Pop-up menus provide an overview of all available functions and navigation to the start of a desired function or its corresponding header file. The search engine enables location of a specified text string within file or within project (scope selectable) and implements find-and-replace operations. At the left selection margin you will find rectangle blocks after download your application code into the target via winIDEA debug hardware. If the compiler has build MCU executable code from that line, there exist the rectangle there. So you can easily control your compiler and source constructs and partly the optimizing of your compiler. But you can do even more. You can select (right mouse key in editor, options…) which function your left mouse key has, when it is above selection margin (Line select, set/reset break or run until). In addition the margin display your Break status (active/ not active) in the source and - with the iSYSTEM Trace tools or Emulators - you get Execution coverage Info's there. It is a multi-file, multi Window editor and every Window within winIDEA can arranged as "docked" or "MDE" or "Mini Frame". With that we support i.e. a very useful second screen on your Windows PC. There is an interface to common Source Control Systems. Anyway: Please refer the manuals for details! If you prefer an other, more powerful Editor (i.e. "Context" as powerful freeware or ECLIPSE as complete Build System or Editor only), the winIDEA Editor will load and synchronize automatically the results of your work.

The **Build System** controls the Project dependencies automatically and invokes the different C/C++ compilers, assemblers, linkers, code converters and many other tools for "perfect" build. After download your results to the target winIDEA automatically recognize any of your changes in a project file (source, headers, …) and will ask you whether you want to "make" new version or continue (i.e. if only comment changed) the debugging. If you agree, winIDEA can do all the necessary things (compile, link, download/flash to the target, run until…) automatically WITHOUT closing the debug session. This depends from your settings in winIDEA.

For syntax check, winIDEA provides pre-processing of single source file. It will update dependencies too.

The **Linker result** is a ***.elf*** file and - via converter or direct - i.e. Freescale S-record output file format could be available too. The winIDEA can download and handle all usual output formats, but only ELF contains the necessary Debug information! So you don't need other files for download usually. And you need normally neither different files for debug and flashing nor other tools for flashing - it is "all in one" tool! For PPC is this preset (compare the examples and GNU Documentation) ELF Format with winIDEA download file option "load program headers physical" (not "sections only"!).

The **WinIDEA Debugger** provides C/C++ (and ADA for some CPU's) source-level, assembly-level and mixed mode debugging. The eTPU's of MPC5554 are supported (refer "Debug => Core" menu). For code exploration, absolute or conditional breakpoints can be accessed or single-step operation can be performed. Step-over, step-into and step-out of code function capabilities are provided. The contents of usual HLL constructs (i.e. arrays, enums, pointers, structs …) can be displayed in different kinds, monitored and contents and kind of presentation manually modified. There exist different global watches, global real-time watches and local variables display Windows including context (= Call-Stack). For internal Flash only it provides unlimited flash breaks for Code.  Be careful, this can worn your number of possible flash cycles for that Chip (check Freescale manual for information). A flash-break within internal Flash read, erase and re-program up to 3 times the smallest possible flash-sector for every break (worst case; if active, reached and then continued execution afterwards).

**winIDEA SDK/Scripting via isystem.connect.**

Besides IDE functionality you are able to use Scripting. Please refer i.e. to "c:\iSYSTEM\winIDEA\2012\sdk", in example for PHYTON script

"c:\iSYSTEM\winIDEA\2012\sdk\iSYSTEM.Python.SDK\examples\" directory. We recommend using PHYTON if you are inexperienced user because it is free of charge, there exists good community support and lot of extensions via WEB. It is usable directly from winIDEA IDE (beside this the free "PYSCRIPTER" GUI for Python strongly recommended for do the JOB as Python development and execution system!). iSYSTEM have build a lot of useful examples (*.py) in the corresponding directory for you.
With installation of winIDEA we add an iSYSTEM library to Standard PYTHON (today Python 2.7 - if you don't have it we install it before) and then you can use everything nicely. If you prefer other supported Script languages (C#, Pearl, …) we provide some examples too, but you have take care for installations on your own. In case of questions please ask holger.wild@isystem.com because Python Scripts open a lot of additional, very useful possibilities for you daily work. It can not be described here.

**iSYSTEM testIDEA unit test.**

Note that any winIDEA contains a basic unit-test feature, called "testIDEA" (needs PYTHON installed). You may invoke it from menu and it opens - after a while - a second IDE (testIDEA GUI) for setup and executes the testcases. Please use the testIDEA internal Help-System to collect more information about. You will find additional information on www.isystem.com. If you are forced to think about automatized tests this could be a very interesting solution - i.e. together with Scripting!

**iSYSTEM winIDEA iONE-E** supports the following debug options of the phyCORE-MPC5554 target.

- A hardware-level debugging is provided via a USB 1.1 PC-to-iONE-E and from that to target connection using the MPC5554 JTAG/OnCE port (OnCE: on-chip emulation). The OnCE itself is a kind of hardware debug module integrated on the processor from Freescale.

- You may download/program/edit the content of the internal MCU flash, the internal RAM, the external RAM and external Flash on PHYTEC MPC5554 Kit with it. Please note that you maybe have to change the external Flash device within this particular example for external Flash because there could be different devices soldered on your target.
- The term "download" means in parallel "initialize debugging tool" too. So with no preceding "download" after power on you usually cannot debug. If your code within flash is unchanged at startup, you may use Icon "load symbols only" and winIDEA "Reset button" instead for saving time of extra download. But usually your debugging starts with first download of actual Code - wherever it will be placed/loaded. The MCU itself at download time is not initialsized in any way (refer to Freescale manual BAM/boot assist module!). So if download target is external Flash or RAM device winIDEA have to use a *.ini file. This is provided and setup correctly within the corresponding PHYTEC examples - if you change something or try to use own constructs you should be aware of this MCU, project and target related settings and how to get the setup into Debugger!
- Please note that there exists no real iSYSTEM MCU Simulator (only a plug-in for the core…) for this MPC5xxx architecture at all, so you need always your PHYTEC Kit or other target for debugging your application code.

To the iSYSTEM winIDEA the following debug parts could be connected:

**iONE-E** This is the small blue box you got with the package. It is an entry level debug tool with following characteristics:
Run up to **90 days AFTER FIRST DOWNLOAD** without restriction with any common compiler output. So you are able to try different commercial Compilers - mostly 30 day versions are usual. You could - if you want - compare delivered standard GNU solution results against commercial Compiler. Their result will be "Standard *.ELF files" too - only with Green-Hills you have to set the Compiler option "dual_debug" to get this standardised output.

The Nexus Trace Port with Profiler, Code-Coverage, Trace and eTPU Trace is NOT supported with iONE-E solution - for this features exists the iSYSTEM "iC5000" unit. The iONE-E has only USB connection to the PC (USB 1.1 low speed standard, could be connected on any USB Port) and the device itself is powered over the USB line from the Host-PC.

The support for iONE-E is limited to usual warranty and technical function together with the package. In case of questions beside that please ask for, we will find an agreement to support you.

For the Demo-Workspaces and target PHYTEC is your primary support partner, but you may ask holger.wild@isystem.com directly for questions regarding debugging too.

After that 90 day trial period, iONE-E's download code-size is limited to 32 Kbyte! You should ask iSYSTEM at latest after this period for a test-run (up to additional 4 weeks) with the "real" Debug Tools, i. e. "iC5000".

**iC5000.** This is the latest, universal On-Chip Debug & Trace unit for nearly unlimited high-speed and high-end debugging and optional using of NEXUS and ETM Trace-Port technic. It contain a powerful Freescale MPC CPU and Data transfer from/to the PC is very fast (up to 300 times faster then iONE-E) via USB 2.0 high-speed (480 Mbit/sec) and/or Ethernet 100 connection. Because of latest FPGA technic inside the Box is always the same. New MCU family or architecture support is only an additionally cheap adapter for the physical connection and a cheap license for Debugging and maybe for the Trace-Port. The NEXUS Trace of MPC5554 allows you to trace and debug both eTPU's beside the usual Code debug & trace with that kind of units.

If you get this iC5000 tool for evaluation you will find the Demo workspaces with the same name, but "iC5000_" before. In Example: iONE-E Workspace is "**hello_ExtFlash.xjrf**", the same Workspace for iC5000 is named "**iC5000_hello_ExtFlash.xjrf**". It contains a pre-recorded trace so you could have a look what NEXUS Trace could do. To use this please ask iSYSTEM FAE Holger Wild via eMail for further instructions (holger.wild@isystem.com).

**iC3000**. If you own a (older) "iC3000" based Debug unit there are iCARD's available for Debugging of the MPC 5xxx MCU's On-Chip-Debug too. In addition for iC3000 you could have an iTRACE GT as high-end tool for the NEXUS trace possibilities.

# 2 Getting Started

What you will learn with this Getting Started example:

- installing the Rapid Development Kit software
- installing iSYSTEM winIDEA toolchain
- interfacing the phyCORE-MPC5554, mounted on the Development Board, to a host-PC
- connect the iONE-E to the PC USB, set the path to the USB drivers in the winIDEA directory (see below)
- downloading example user code to the phyCORE-MPC5554 external memory
- programming example user code to the MPC5554 On-Chip Flash memory, external SRAM or external Flash memory

## 2.1 Installing Rapid Development Kit Software

- Insert the PHYTEC phyCORE-MPC5554 Tool-CD into the CD-ROM drive of your host-PC. The CD should automatically launch a setup program that installs the required demos, documentation and other support documents. Otherwise the setup program ***setup.exe*** can be manually executed from the root folder of the PHYTEC Tool-CD.

The default destination location is ***C:\PHYTEC***. All path and file statements within this QuickStart Instruction are based on the assumption that you choose the default install paths and drives. If you decide to choose different paths and/or drives you must consider this for all further file and path statements. We recommend that you accept the default destination location.

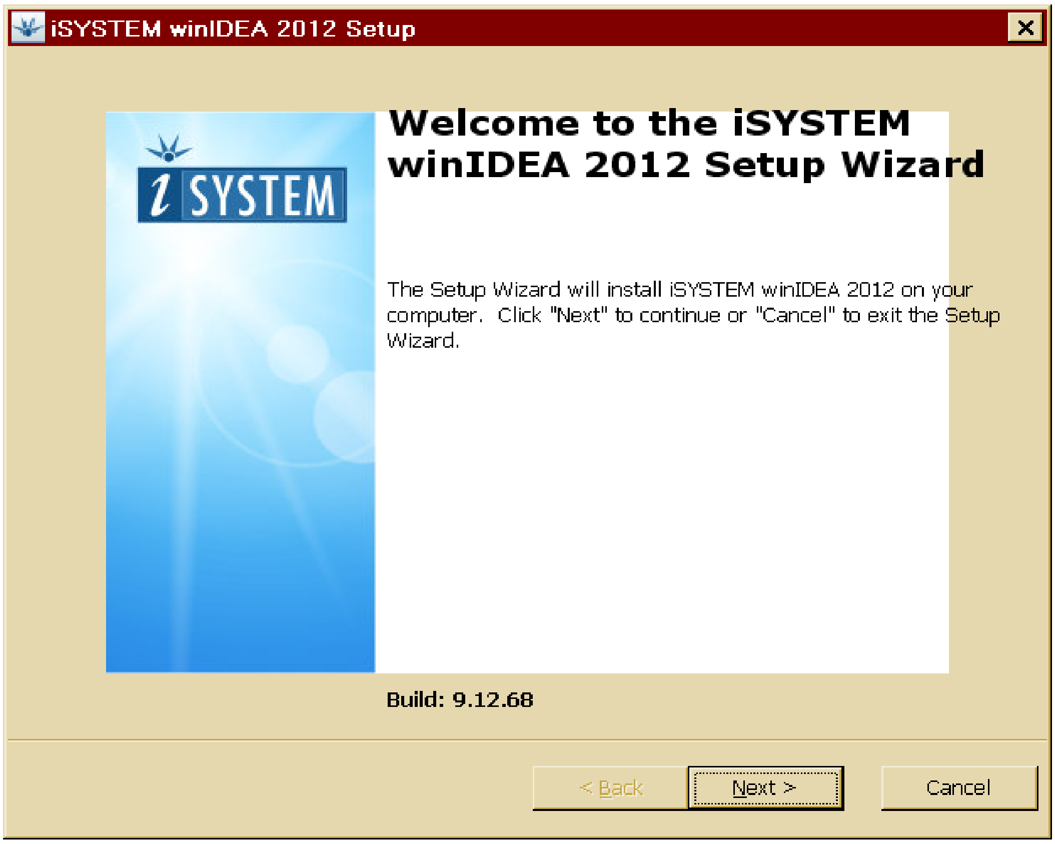
- Follow the instructions in the setup Window.

## 2.2 Installing the iSYSTEM winIDEA Tool Chain

The applicable WinIDEA tool chain software must be installed BEFORE you connect the included iONE-E to any USB Port of your host-PC in order to ensure successful completion of this QuickStart Instruction! Failure to install the proper software could lead to possible version conflicts, resulting in functional problems.

winIDEA 2012 needs PYTHON version 2.7.3. It is included in the installation file, but it will not ask for the destination location. Instead it will always use the default ***C:\Phython27\***. If you want it to be installed to another path, you need get PYTHON 2.7.3 installation file from the web and install it before installing winIDEA 2012.

- Install the iSYSTEM winIDEA tool chain via Windows Explorer from a single file named "***winIDEA2012*_68.exe**". You will find it on the CD within directory "***winIDEA 2012***". For installing you need Administrator rights!

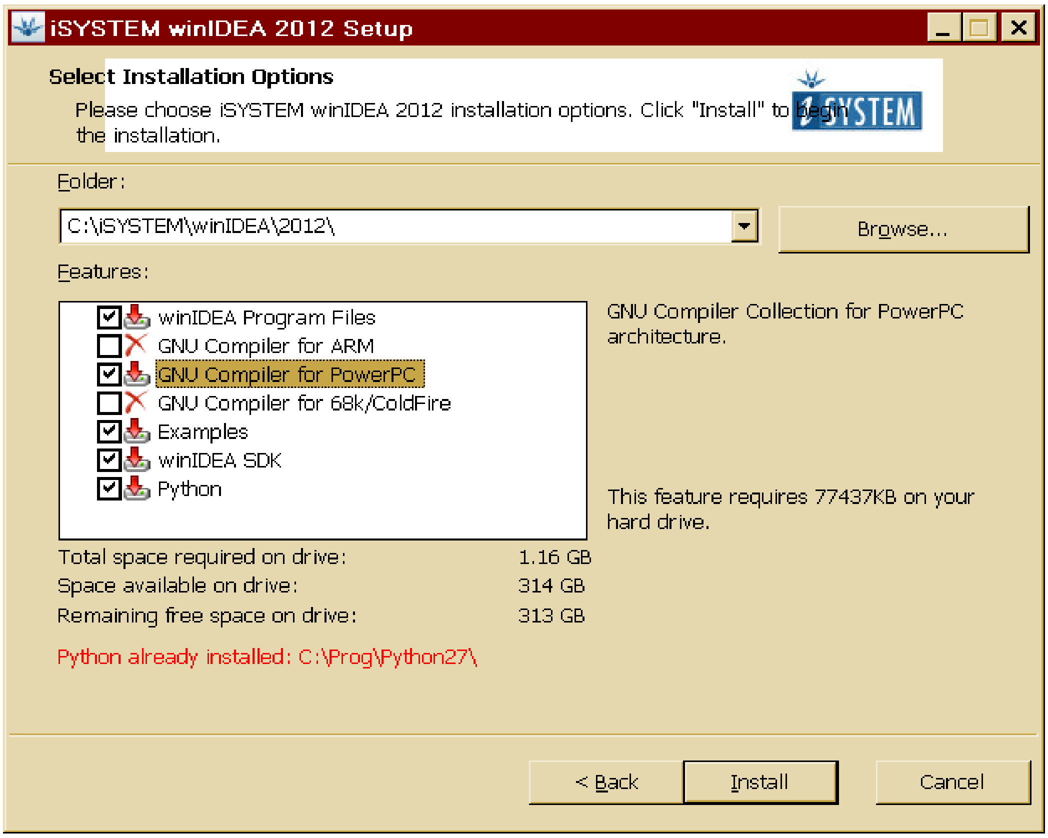


Welcome to the iSYSTEM winIDEA 2012 Setup Wizard

The Setup Wizard will install iSYSTEM winIDEA 2012 on your computer. Click "Next" to continue or "Cancel" to exit the Setup Wizard.

Build: 9.12.68

At first you should get this screen.



iSYSTEM winIDEA 2012 Setup

**Select Installation Options**

Please choose iSYSTEM winIDEA 2012 installation options. Click "Install" to begin the installation.

Folder: C:\iSYSTEM\winIDEA\2012\

Features:

- ☑ winIDEA Program Files
- ☐ GNU Compiler for ARM
- ☑ GNU Compiler for PowerPC
- ☐ GNU Compiler for 68k/ColdFire
- ☑ Examples
- ☑ winIDEA SDK
- ☑ Python

GNU Compiler Collection for PowerPC architecture.

This feature requires 77437KB on your hard drive.

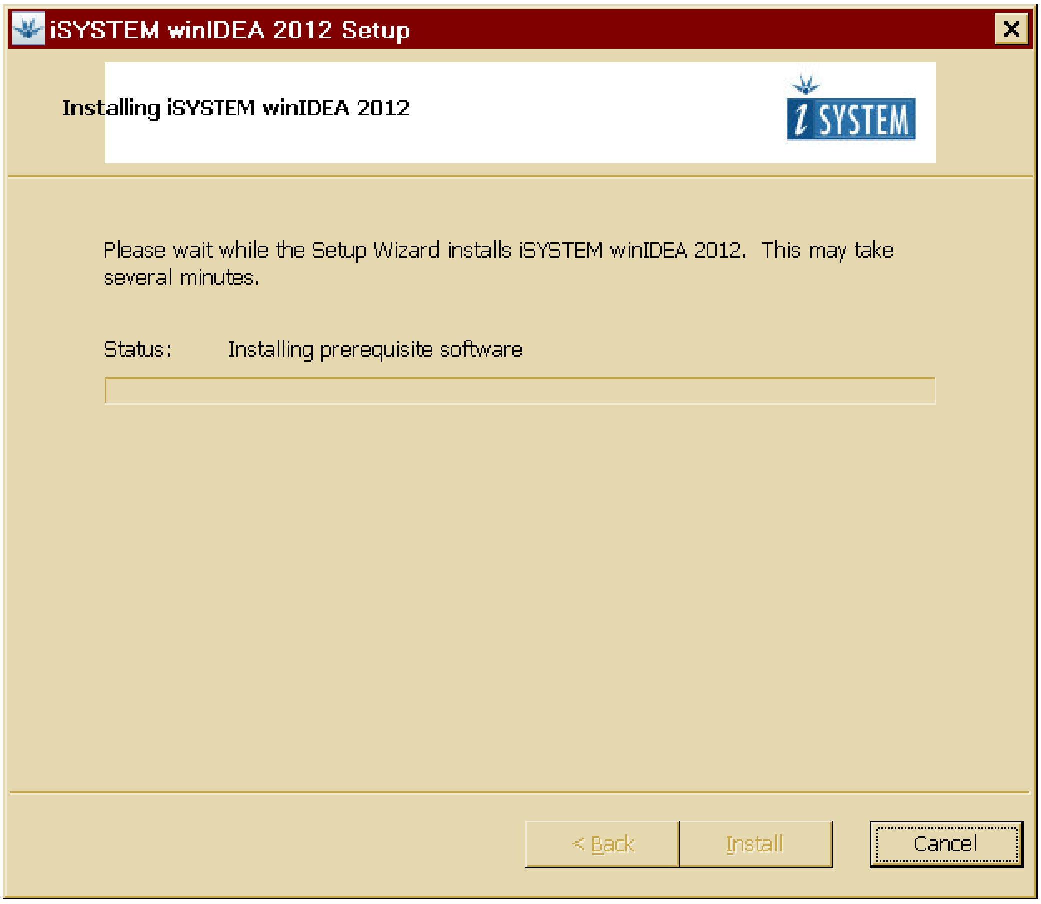
| | |
|---|---|
| Total space required on drive: | 1.16 GB |
| Space available on drive: | 314 GB |
| Remaining free space on drive: | 313 GB |

Python already installed: C:\Prog\Python27\

Choose at least the installing options above (or mark all of them).

The default destination location is suggested during installation, i.e. ***C:\iSYSTEM\winIDEA\2012\***
All path and file statements within this QuickStart Instruction are based on the assumption that you accept the default install paths and drives. If you choose different paths and/or drives you must consider this for all further file and path statements. We recommend that you accept the default destination location and install all options, including PYTHON Version 2.7.3 (default is ***C:\Python27***\).

In this example case PYHTON was installed before seperatly to a different location.

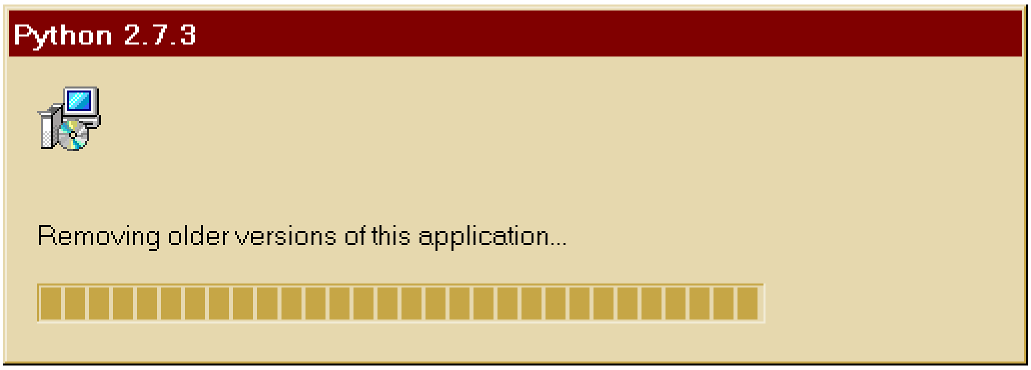


Installing iSYSTEM winIDEA 2012

Please wait while the Setup Wizard installs iSYSTEM winIDEA 2012. This may take several minutes.

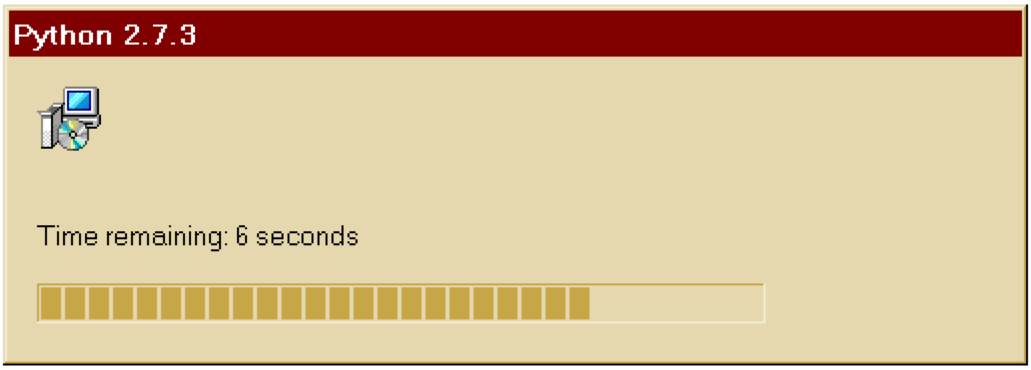
Status: Installing prerequisite software

After pressing "Install" the installation continues like depicted above with the usual "advance" screens.



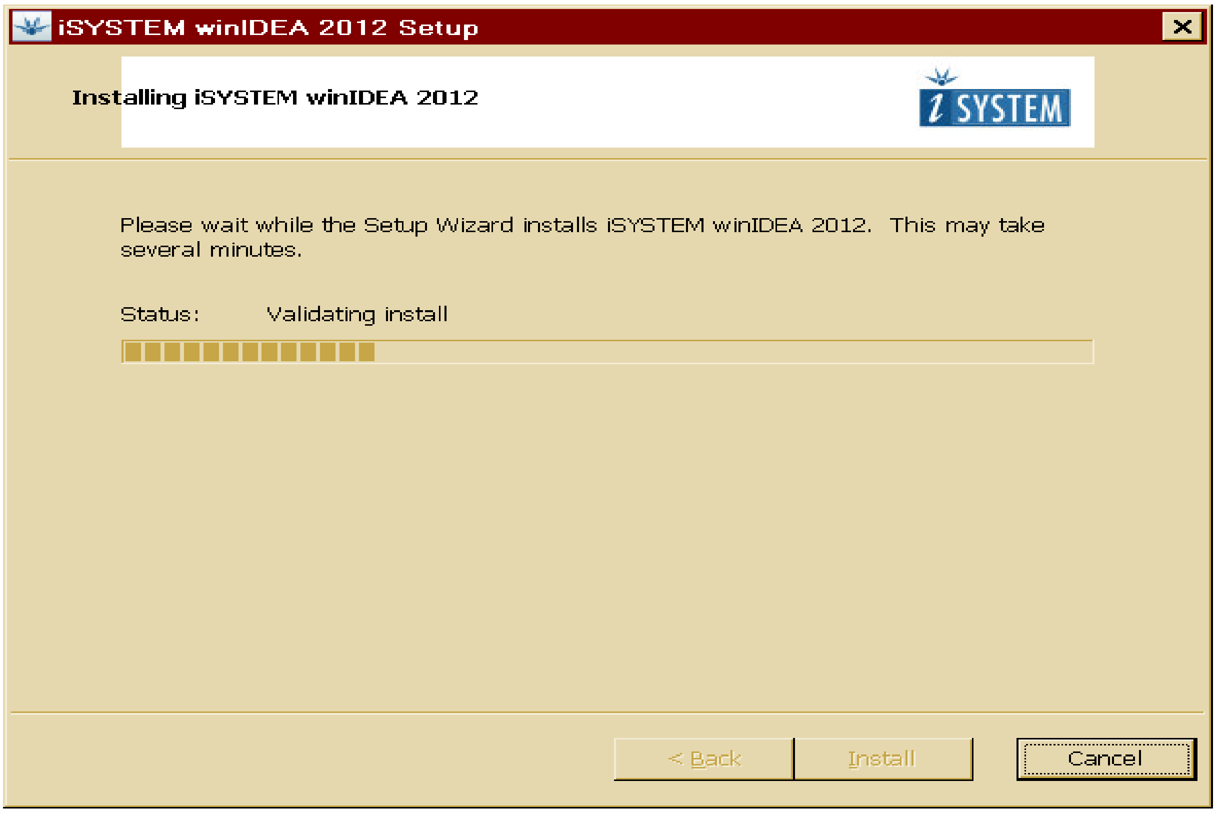
Removing older versions of this application...

If an older version of Phyton has been found on your PC, it will first be removed.


Python 2.7.3

Time remaining: 6 seconds

After that the new one will be installed.

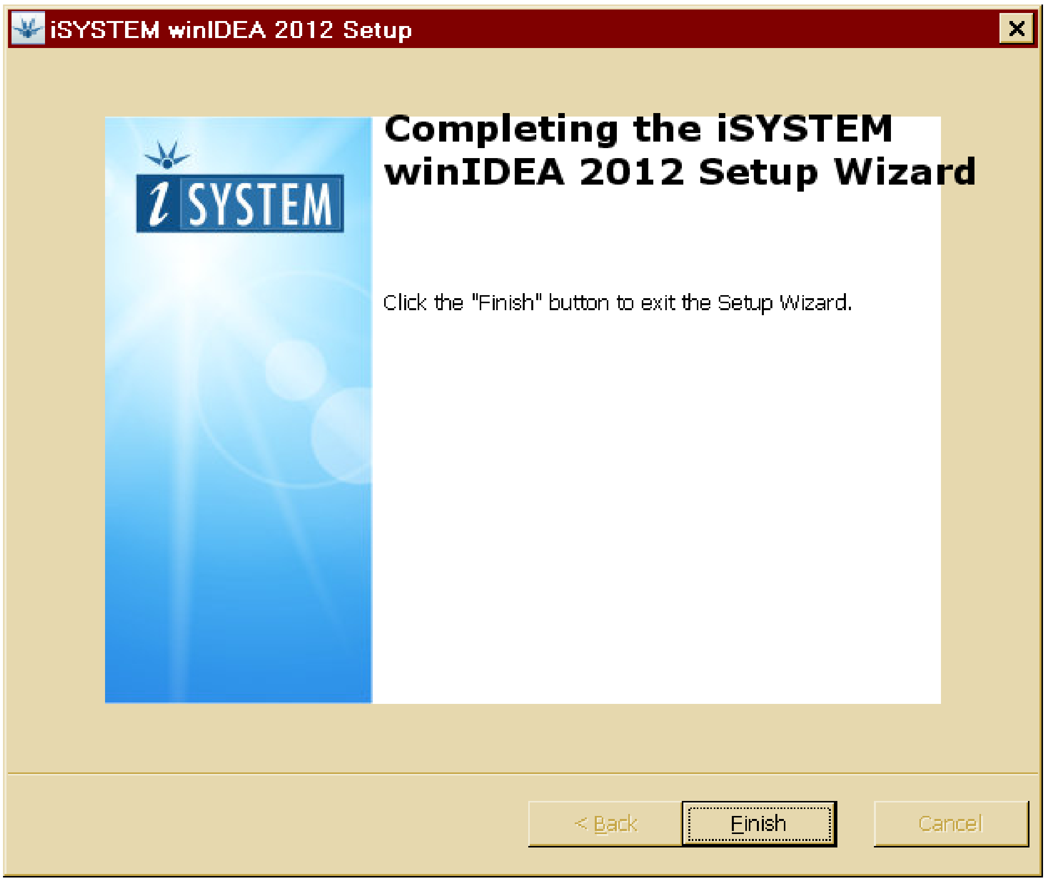

iSYSTEM winIDEA 2012 Setup

Installing iSYSTEM winIDEA 2012

Please wait while the Setup Wizard installs iSYSTEM winIDEA 2012. This may take several minutes.

Status: Validating install

Thereafter standard installations continues.



iSYSTEM winIDEA 2012 Setup

Completing the iSYSTEM winIDEA 2012 Setup Wizard

Click the "Finish" button to exit the Setup Wizard.

This part should end with this screen. Please click "Finish".

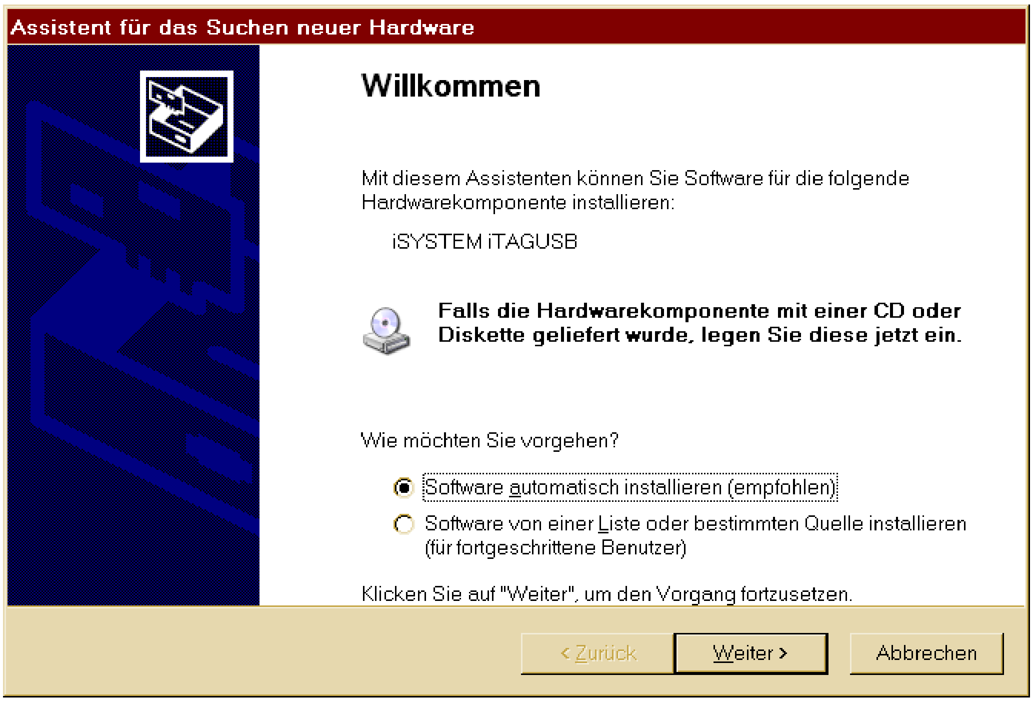
## 2.3 Connecting the iONE-E to your Host-PC

After having winIDEA Software successfully installed, connect the included iONE-E to any USB Port of your host-PC. Windows will recognize it and ask for drivers. Installation should work if you select “automatically”.

The drivers are usually NOT Microsoft verified, but they work and are compatible to any actual updated Windows version. In case Windows asks for certificated driver, you should answer something like "use it anyway".

In case of difficulties with that please ask support@isystem.com for latest drivers and advice because the iSYSTEM iONE-E is a special product build only for PHYTEC and will not be tested with every winIDEA Version and Windows Version!

Please note that you have unfortunately to repeat this procedure for every USB port connection of your PC, if you connect iONE-E (or any other iONE, iC5000, etc.) for the first time to that port. This is a Windows request and unfortunately we cannot change this behavior.



Willkommen

Mit diesem Assistenten können Sie Software für die folgende Hardwarekomponente installieren:

iSYSTEM iTAGUSB

Falls die Hardwarekomponente mit einer CD oder Diskette geliefert wurde, legen Sie diese jetzt ein.

Wie möchten Sie vorgehen?

● Software automatisch installieren (empfohlen)

○ Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)

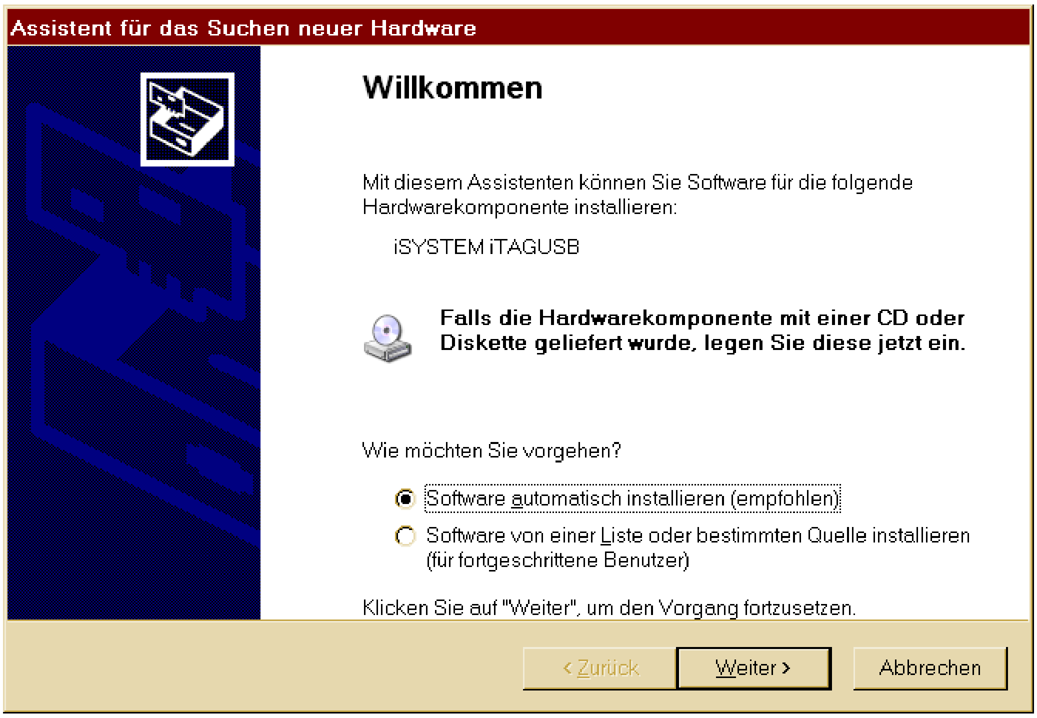
Klicken Sie auf "Weiter", um den Vorgang fortzusetzen.

USB Setup after connection of iONE-E (in winIDEA it is called iTAG USB). Sorry for German dialogs, but these windows depend on the language of Windows-PC and are similar for every supported Windows language.

In case that Windows asks you for searching for updates over the internet, please answer “No!”.

In case that it didn't find the driver, you may take a look here (German XP, probably different when using other Windows versions): ***"C:\Dokumente und Einstellungen\All Users\isystem\drivers\ "***.

You will find "***InstallDriver.exe***" there that could solve issues. If you are not limited from your IT department (you should ask there!) the free tool "USBDeView" (www.nirsoft.net) could solve nearly any USB related trouble. Simply "deinstall" awkward device and reinstall correspondig driver to get it work (of course: here no warranty!).



Assistent für das Suchen neuer Hardware

**Willkommen**

Mit diesem Assistenten können Sie Software für die folgende Hardwarekomponente installieren:

iSYSTEM iTAGUSB

**Falls die Hardwarekomponente mit einer CD oder Diskette geliefert wurde, legen Sie diese jetzt ein.**

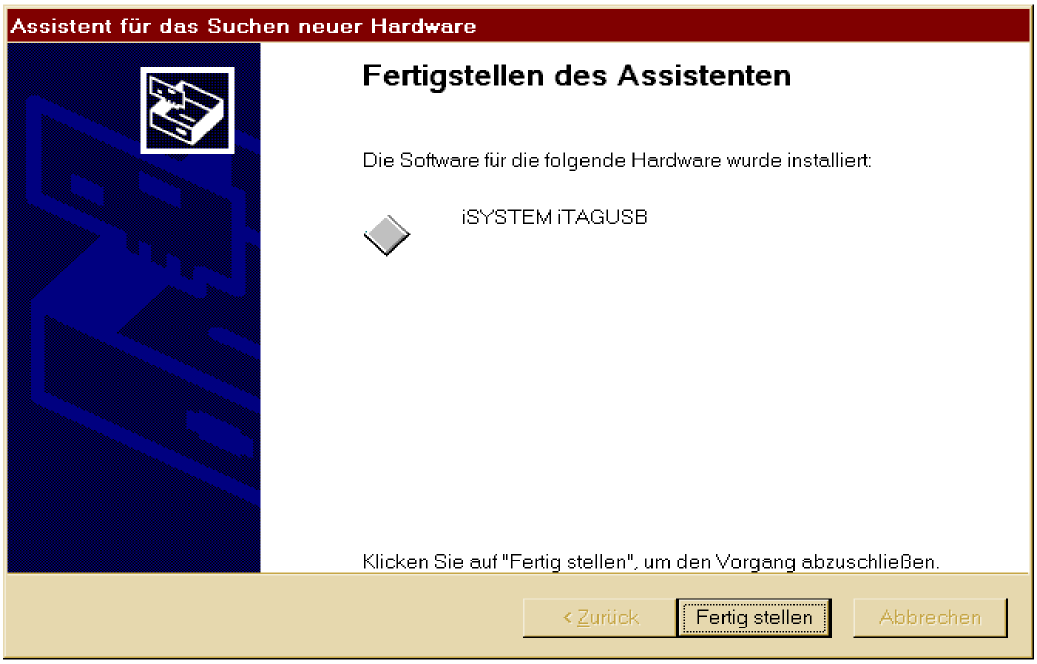
Wie möchten Sie vorgehen?

● Software automatisch installieren (empfohlen)

○ Software von einer Liste oder bestimmten Quelle installieren (für fortgeschrittene Benutzer)

Klicken Sie auf "Weiter", um den Vorgang fortzusetzen.

Please choose "automatic installation".



Assistent für das Suchen neuer Hardware

Fertigstellen des Assistenten

Die Software für die folgende Hardware wurde installiert:

iSYSTEM iTAGUSB

Klicken Sie auf "Fertig stellen", um den Vorgang abzuschließen.

When you get this window, your system is ready to work.

## 2.4 Interfacing the phyCORE-MPC5554 to your Host-PC

Connecting the phyCORE-MPC5554, mounted on the PHYTEC Development Board, to your computer is simple:

- If the phyCORE module (part # PCM-028-xxxx) is not already mounted on the Development Board (part # PCM-979), please mount it pins-down onto the Development Board's receptacle footprint (X1) as shown in *Figure 2* below. Ensure that pin 1 of module matches pin 1 of the receptacle on the Development Board.

Ensure that there is a solid connection between the module's pins and the Development Board receptacle. If the phyCORE module is removed from the Development Board, take precautions to properly mount the module when it is reattached to the Development Board. Pin 1 on the phyCORE module (denoted by the hash stencil mark on the PCB) should be matched to the footprint receptacle on the Development Board marked by "X1" on the PCB. Also take precautions not to damage the connectors when the phyCORE is removed from and inserted onto the Development Board.

- Configure the jumpers on the phyCORE Development Board as indicated in *Figure 2*. This correctly routes the CAN interface signals. *Please see the phyCORE-MPC5554 Hardware Manual for further information on jumper settings.*

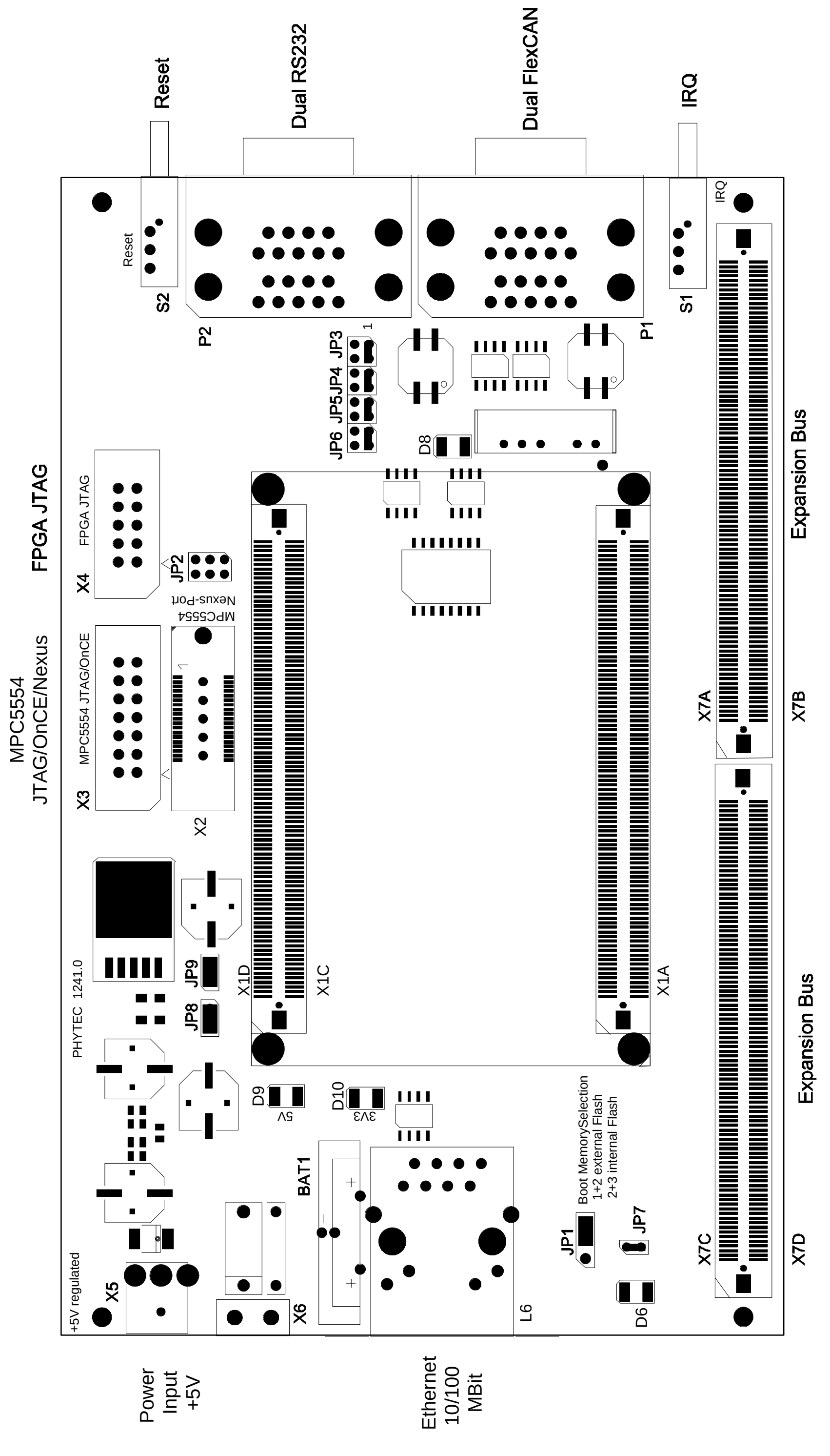


*Figure 2: View of Development Board and Jumper Settings*

- Connect the iONE-E JTAG/OnCE interface cable to the 14-pin JTAG/OnCE header on the Development Board at X3. This connection is used for the communication between the iONE-E and the phyCORE-MPC5554 target hardware.
- Connect the included serial cable to the lower socket P2A of the female double DB-9 connector on the Development Board and to a free serial port of your host-PC. This will enable you to monitor board-host communication via the terminal emulation Window included in the winIDEA surface or a terminal emulation program, such as Windows HyperTerminal.
- Using the included 5VDC power adapter to connect the power socket X5 on the board (*refer to Figure 3 for the correct polarity*). The phyCORE module/Development Board combination requires a 5 VDC@1A ±5 % regulated supply.

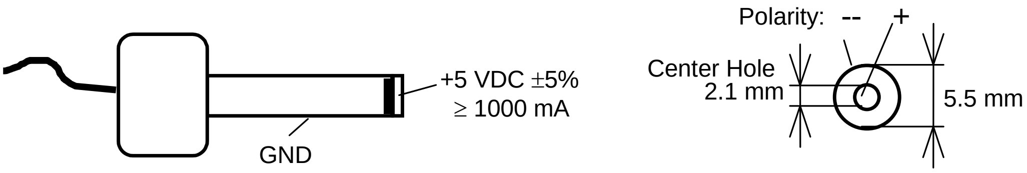


*Figure 3: Connecting the Supply Voltage at X5*

The LED’s D9 and D10 (two green LED’s) should light, indicating that all voltages are supplied to the phyCORE module. D9 indicates the +5V supplied by the attached power supply and D10 monitors the on-board regulated 3V3.

The phyCORE-MPC5554 should now be properly connected via the Development Board to the host-PC and power supply and you are now ready to use the iSYSTEM winIDEA tool chain to establish communication. This phyCORE module/Development Board combination shall also be referred to as "target hardware" or simply "target".

## 2.5 Downloading Example Code with winIDEA IDE

At first please note that later within this manual you will find a chapter “Background information & Troubleshooting”. In case of trouble please refer to there.

If you followed the suggested defaults during installation, you will find some examples here:

C:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\
Quickstart\Demos\Hello

Please note that in case that you want to create your own new project later on, we strongly suggest to copy one of these examples to a new directory and then modify it. You should not use the "Workspace new" option within winIDEA! It would generate a complete empty project without ANY necessary settings, so you probably will “get lost".

Concept of winIDEA is always to deal with a “Workspace”. It consists out of XML-files named **.xjrf* and **.xqrf*, which together (!) represent the Workspace with all important settings for project build, download to desired destination, debug, breakpoint and Window/pane positions as well as particular setup. A Workspace contains only one project.

Settings like fonts, font colours and last opened Workspace are stored in your Windows log-in profile within file *winIDEA.xtrf*. Usually this file is stored in an hidden directory.

The **.lcf* is the Linker command file (it is an ascii text file).

The **.ini* file (ascii text file) is needed because of special MCU issues and downloads to external Flash/RAM.

Check the MCU manual regarding "BAM" (Boot assist module), boot behavior and possibilities, "MMU" and "EBI" (external Bus Interface) if you need to know more. Don't worry about that at this time. Our little Demo Projects will make you familiar with the toolchain and MCU. Later on, you can expand these examples in order to learn more about the powerful PHYTEC MCU Kit without need to change the Linker settings or the INI File. Only keep in mind that the **.ini* file ensures that correct and desired application RESET-entry-point is reached.

For your own applications you need to understand and to setup the BAM first. Otherwise your target will only work correctly when debugger is connected and in use!

For these example applications entry points are setup in the last lines within corresponding **.ini* files (here last line is active):

```
###
// ==> Reset entry according to lcf
// R PC L 0x00000008           // application Entry internal Flash
// R PC L 0x20000008           // application Entry external Flash
R PC L 0x21000008           // application Entry external RAM
###
```

The ***Hello*** example sends a program to the target hardware that, when executed, sends a character string from the RDK back to the host-PC. The character string can be viewed with the terminal emulation Window in the winIDEA or with a terminal emulation program. The program also controls the user LED D6 (red LED) with equal on and off ratio.

To show how to load and run software, the Hello Demo exists in three different versions for the delivered iONE-E (the "iC5000_* are for iC5000 Debug Tool only!):

- ***Hello_ExtRAM.xjrf***
  Running the Hello Demo Project in external SRAM. This application cannot run as "stand alone" (without Debugger connected) because SRAM is volatile Memory. You can expect the fastest execution because RAM needs no wait states during code execution.

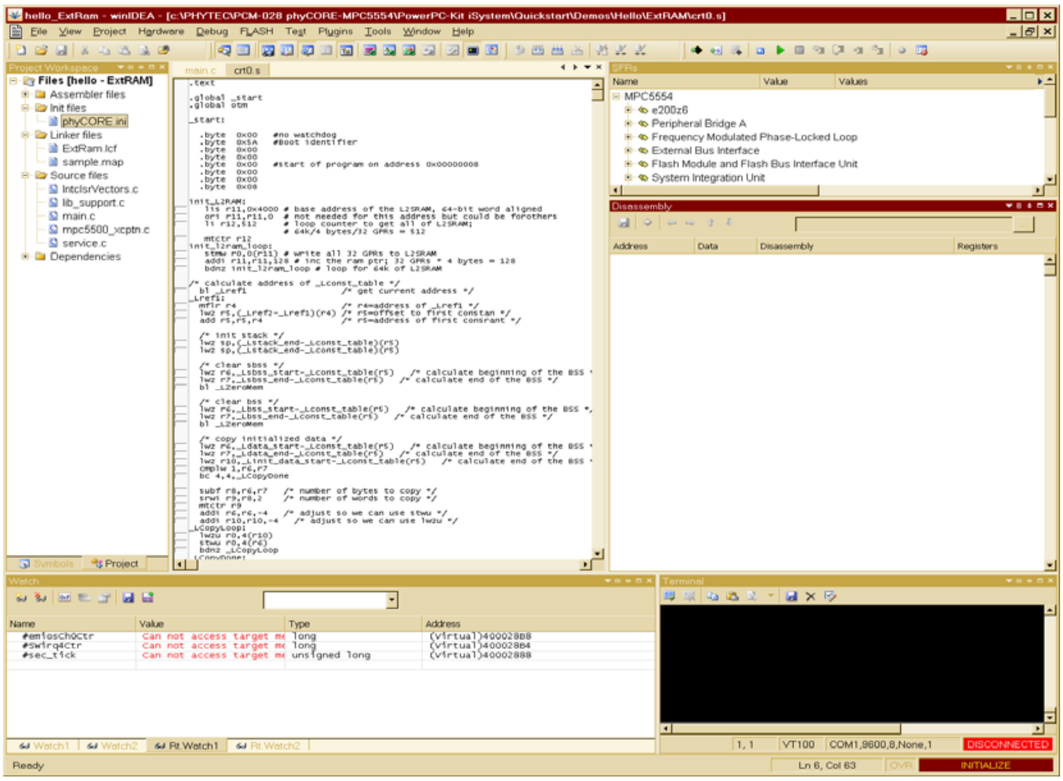
- ***Hello_IntFLASH.xjrf***
  Running the Hello Demo Project in MPC5554 on-chip Flash**.**
  This application could - when flashed - run as "stand alone" too.
  Jumper 1 on Baseboard needs to be set accordingly.

- ***Hello_ExtFLASH.xjrf***
  Running the Hello Demo Project in external Flash memory.
  This application could - when flashed - run as "stand alone" too.
  Jumper 1 on Baseboard needs to be set accordingly, different from IntFlash Demo.

### 2.5.1 Running the Hello Demo Project in external RAM

In this section you will download the Hello Demo to the external Sync. Burst Mode SRAM.

- Launch the iSYSTEM *winIDEA* by double-clicking the *winIDEA* icon or by selecting *winIDEA IDE* from within the *winIDEA* program group. Escape from the workspace selection window.
- Select *File | Workspace \ Open Workspace* from the *winIDEA* menu bar. Navigate to ***C:\PHYTEC\PCM-028 phyCORE-MPC5554\ PowerPC-Kit iSystem\Quickstart\Demos\Hello\ExtRAM*** and open the project file ***Hello_ExtRAM.xjrf***.
- The ***Hello_ExtRAM.xjrf*** project will open with a predefined Window arrangement.



hello_ExtRam - winIDEA - [c:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demos\Hello\ExtRAM\crt0.s]

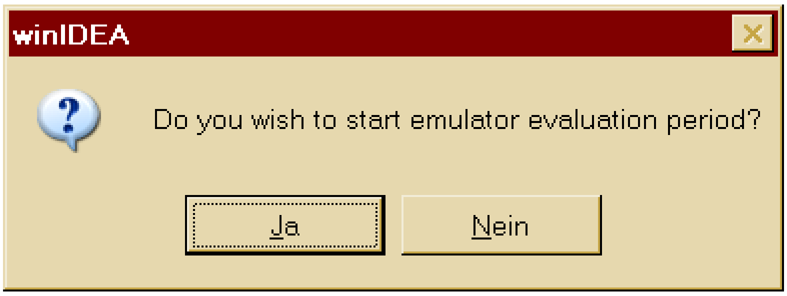
```
.text

.global _start
.global otm

_start:

  .byte  0x00   #no watchdog
  .byte  0x5A   #Boot identifier
  .byte  0x00
  .byte  0x00
  .byte  0x00   #start of program on address 0x00000008
  .byte  0x00
  .byte  0x00
  .byte  0x08

init_L2RAM:
   lis r11,0x4000 # base address of the L2SRAM, 64-bit word aligned
   ori r11,r11,0  # not needed for this address but could be forothers
   li r12,512     # loop counter to get all of L2SRAM;
                  # 64k/4 bytes/32 GPRs = 512
   mtctr r12
init_l2ram_loop:
   stmw r0,0(r11) # write all 32 GPRs to L2SRAM
   addi r11,r11,128 # inc the ram ptr; 32 GPRs * 4 bytes = 128
   bdnz init_l2ram_loop # loop for 64k of L2SRAM

/* calculate address of _Lconst_table */
  bl _Lref1                     /* get current address */
_Lref1:
  mflr r4                       /* r4=address of _Lref1 */
  lwz r5,(_Lref2-_Lref1)(r4) /* r5=offset to first constan */
  add r5,r5,r4                  /* r5=address of first consrant */

  /* init stack */
  lwz sp,(_Lstack_end-_Lconst_table)(r5)
  lwz sp,(_Lstack_end-_Lconst_table)(r5)

  /* clear sbss */
  lwz r6,_Lsbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS *
  lwz r7,_Lsbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* clear bss */
  lwz r6,_Lbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS *,
  lwz r7,_Lbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* copy initialized data */
  lwz r6,_Ldata_start-_Lconst_table(r5)   /* calculate beginning of the BSS *
  lwz r7,_Ldata_end-_Lconst_table(r5)   /* calculate end of the BSS */
  lwz r10,_Linit_data_start-_Lconst_table(r5)   /* calculate end of the BSS *
  cmplw 1,r6,r7
  bc 4,4,_LCopyDone

  subf r8,r6,r7   /* number of bytes to copy */
  srwi r9,r8,2    /* number of words to copy */
  mtctr r9
  addi r6,r6,-4   /* adjust so we can use stwu */
  addi r10,r10,-4   /* adjust so we can use lwzu */
_LCopyLoop:
  lwzu r0,4(r10)
  stwu r0,4(r6)
  bdnz _LCopyLoop
_LCopyDone:
```

| Name | Value | Type | Address |
|---|---|---|---|
| #emiosCh0Ctr | Can not access target me | long | (virtual)400028B8 |
| #Swirq4Ctr | Can not access target me | long | (virtual)400028B4 |
| #sec_tick | Can not access target me | unsigned long | (virtual)40002888 |

```
1, 1   VT100   COM1,9600,8,None,1   DISCONNECTED
Ready   Ln 6, Col 63   OVR   INITIALIZE
```

On the left side you should see the docked Window "project workspace". Ensure that you select "project" and not "symbols" pane there (symbols pane could be interesting later during debugging).

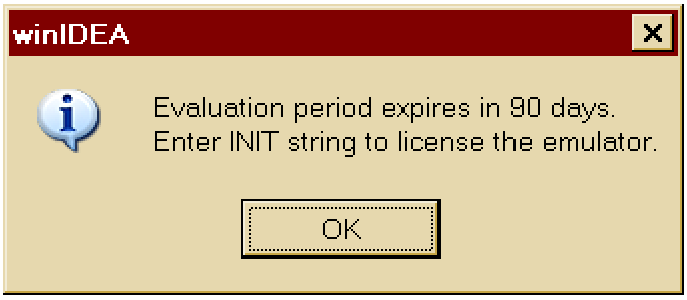
- Alternatively you can open the demo by browsing to the *Hello_ExtRAM.xjrf* file with Windows Explorer or another File Manager tool. Double-click to invoke the winIDEA with the demo project.
- Perform the build process by clicking the *Make* icon at the tool bar or select *Project* | *Make* from the winIDEA pull-down menu (or press F7 key in standard-setting).
- The Make process generates - if no error occurs - an actual output binary (***hello.elf***) in subdirectory /DEBUG/ of the project Workspace directory. The "OUTPUT" build/search status Window will appear within winIDEA while the project is being compiled and linked. You should/may close it if no error occurs, otherwise you can navigate (right mouse key => "Error" in local menu or simply press function key "F4") to your source file errors. Note that origin linker errors and any build message appear in the output Window pane *Tools*.
- Start the download process by clicking the download icon at the tool bar (or press key combination Ctrl+F3). If this will rise an error "Error 40: Communication port not found", then please refer to troubleshooting section.
- On the first download ever with iONE-E it asks to start the evaluation period. You have to select "yes" for continue and with that you start the iONE-E evaluation period (90 days) automatically.



winIDEA

Do you wish to start emulator evaluation period?

Thereafter you will be reminded as countdown for the remaining days of using with the full featured Debug tool on every Debug session.

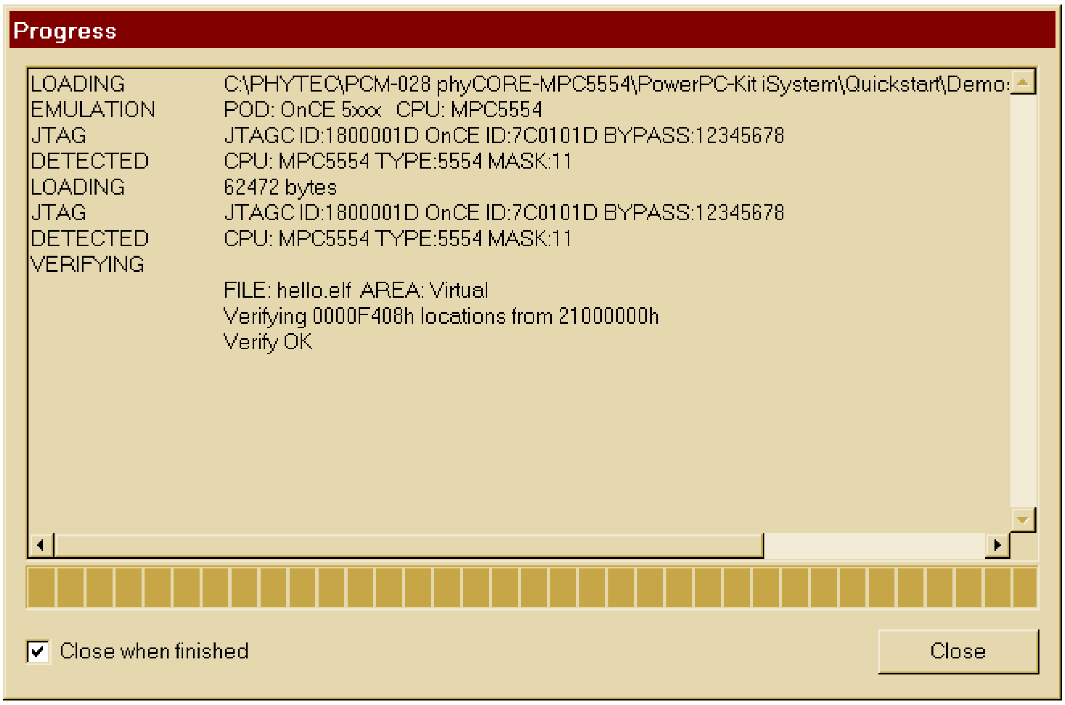


winIDEA

Evaluation period expires in 90 days.
Enter INIT string to license the emulator.

In this example the download is performed to the external RAM. The appropriated target initialization for the debugger to reach the RAM and execute from it are in the *.ini file contained in the project folder.

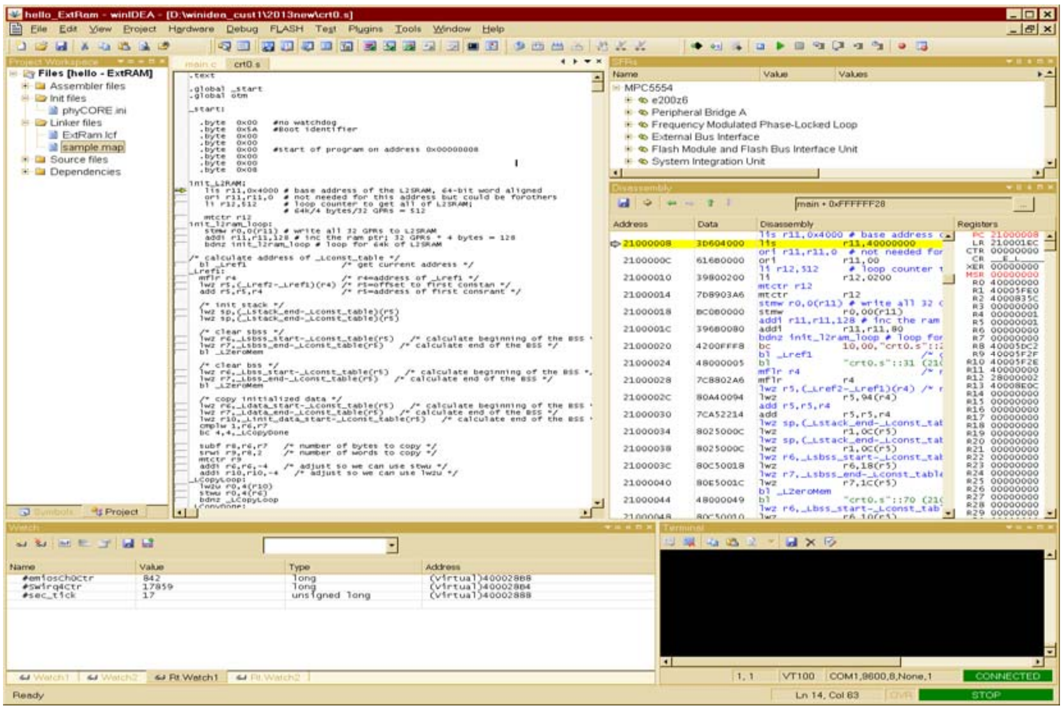
The following status Window will appear during the download and should look similar when successful (you may uncheck "close when finished" during download to verify it without precipitance).



Progress

```
LOADING      C:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demo:
EMULATION    POD: OnCE 5xxx  CPU: MPC5554
JTAG         JTAGC ID:1800001D OnCE ID:7C0101D BYPASS:12345678
DETECTED     CPU: MPC5554 TYPE:5554 MASK:11
LOADING      62472 bytes
JTAG         JTAGC ID:1800001D OnCE ID:7C0101D BYPASS:12345678
DETECTED     CPU: MPC5554 TYPE:5554 MASK:11
VERIFYING
             FILE: hello.elf  AREA: Virtual
             Verifying 0000F408h locations from 21000000h
             Verify OK
```

☑ Close when finished



```
.text

.global _start
.global otm

_start:

  .byte   0x00   #no watchdog
  .byte   0x5A   #Boot identifier
  .byte   0x00
  .byte   0x00
  .byte   0x00   #start of program on address 0x00000008
  .byte   0x00
  .byte   0x00
  .byte   0x08

init_L2RAM:
   lis r11,0x4000 # base address of the L2SRAM, 64-bit word aligned
   ori r11,r11,0  # not needed for this address but could be forothers
   li r12,512     # loop counter to get all of L2SRAM;
                  # 64k/4 bytes/32 GPRs = 512
  mtctr r12
init_l2ram_loop:
   stmw r0,0(r11) # write all 32 GPRs to L2SRAM
   addi r11,r11,128 # inc the ram ptr; 32 GPRs * 4 bytes = 128
   bdnz init_l2ram_loop # loop for 64k of L2SRAM

/* calculate address of _Lconst_table */
  bl _Lref1                   /* get current address */
_Lref1:
  mflr r4                     /* r4=address of _Lref1 */
  lwz r5,(_Lref2-_Lref1)(r4) /* r5=offset to first constan */
  add r5,r5,r4                /* r5=address of first consrant */

  /* init stack */
  lwz sp,(_Lstack_end-_Lconst_table)(r5)
  lwz sp,(_Lstack_end-_Lconst_table)(r5)

  /* clear sbss */
  lwz r6,_Lsbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS *
  lwz r7,_Lsbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* clear bss */
  lwz r6,_Lbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS *,
  lwz r7,_Lbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* copy initialized data */
  lwz r6,_Ldata_start-_Lconst_table(r5)   /* calculate beginning of the BSS *
  lwz r7,_Ldata_end-_Lconst_table(r5)   /* calculate end of the BSS */
  lwz r10,_Linit_data_start-_Lconst_table(r5)   /* calculate end of the BSS *
  cmplw 1,r6,r7
  bc 4,4,_LCopyDone

  subf r8,r6,r7   /* number of bytes to copy */
  srwi r9,r8,2    /* number of words to copy */
  mtctr r9
  addi r6,r6,-4   /* adjust so we can use stwu */
  addi r10,r10,-4   /* adjust so we can use lwzu */
_LCopyLoop:
  lwzu r0,4(r10)
  stwu r0,4(r6)
  bdnz _LCopyLoop
_LCopyDone:
```

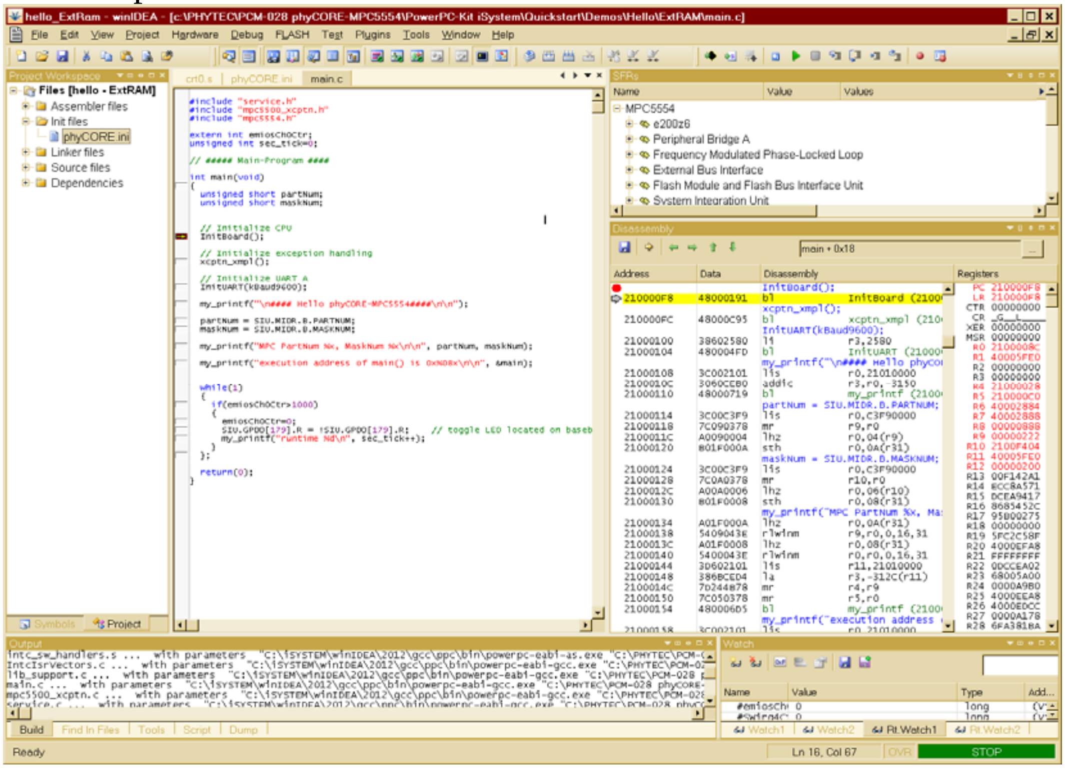
```
Address     Data       Disassembly                                    Registers
                       lis r11,0x4000 # base address c                 PC 21000008
21000008    3D604000   lis            r11,40000000                     LR 210001EC
                       ori r11,r11,0  # not needed for                CTR 00000000
2100000C    61680000   ori            r11,00                           CR __E_L____
                       li r12,512     # loop counter t                XER 00000000
21000010    39800200   li             r12,0200                        MSR 00000000
                       mtctr r12                                       R0 40000000
21000014    7D8903A6   mtctr          r12                              R1 40005FE0
                       stmw r0,0(r11) # write all 32 c                 R2 4000835C
21000018    BC0B0000   stmw           r0,00(r11)                       R3 00000000
                       addi r11,r11,128 # inc the ram                  R4 00000001
2100001C    396B0080   addi           r11,r11,80                       R5 00000001
                       bdnz init_l2ram_loop # loop for                 R6 00000000
21000020    4200FFF8   bc             10,00,"crt0.s"::2                R7 00000000
                       bl _Lref1                   /* c                R8 40005DC2
21000024    48000005   bl             "crt0.s"::31 (21(                R9 40005F2F
                       mflr r4                     /* r               R10 40005F2E
21000028    7C8802A6   mflr           r4                              R11 40000000
                       lwz r5,(_Lref2-_Lref1)(r4) /* r                R12 28000002
2100002C    80A40094   lwz            r5,94(r4)                       R13 40008E0C
                       add r5,r5,r4                                   R14 00000000
21000030    7CA52214   add            r5,r5,r4                        R15 00000000
                       lwz sp,(_Lstack_end-_Lconst_tab                R16 00000000
21000034    8025000C   lwz            r1,0C(r5)                       R17 00000000
                       lwz sp,(_Lstack_end-_Lconst_tab                R18 00000000
21000038    8025000C   lwz            r1,0C(r5)                       R19 00000000
                       lwz r6,_Lsbss_start-_Lconst_tab                R20 00000000
2100003C    80C50018   lwz            r6,18(r5)                       R21 00000000
                       lwz r7,_Lsbss_end-_Lconst_tabl                 R22 00000000
21000040    80E5001C   lwz            r7,1C(r5)                       R23 00000000
                       bl _LZeroMem                                   R24 00000000
21000044    48000049   bl             "crt0.s"::70 (21(               R25 00000000
                       lwz r6,_Lbss_start-_Lconst_tab                 R26 00000000
21000048    80C50010   lwz            r6,10(r5)                       R27 00000000
                                                                      R28 00000000
                                                                      R29 00000000
```

| Name | Value | Type | Address |
|---|---|---|---|
| #emiosCh0Ctr | 842 | long | (virtual)400028B8 |
| #SwIrq4Ctr | 17859 | long | (virtual)400028B4 |
| #sec_tick | 17 | unsigned long | (virtual)40002888 |

- Once download is complete, the source code Window will show the actual location of the program counter from where to start the software execution (green bar in disassembly Window above).

In the high level language Window this location is marked with a arrow on left margin. The exact occurrence depends on your setup - the screen dumps here are changed for better visibility! You may change this in every Window with right mouse key => options. The high level language Window shows at the moment also assembler code due to the entry point of the project is located in a startup file that is written in assembler code. Because GNU CC creates "HLL Line Info" for assembler too you see on the left margin of Source Window rectangles. Later you will see this again within C Source for every line which was translated to MCU instructions. So it is easy to follow the way from C-Source (or C++) to executable Object Code and within the Disassembly Window of winIDEA you have a display in "mixed mode". The next step will run the code to the main() function because there is a breakpoint set (active).

- Run the code by clicking on the ▶ icon or press F5. This will execute the startup code and the processor is running until the first breakpoint is reached.



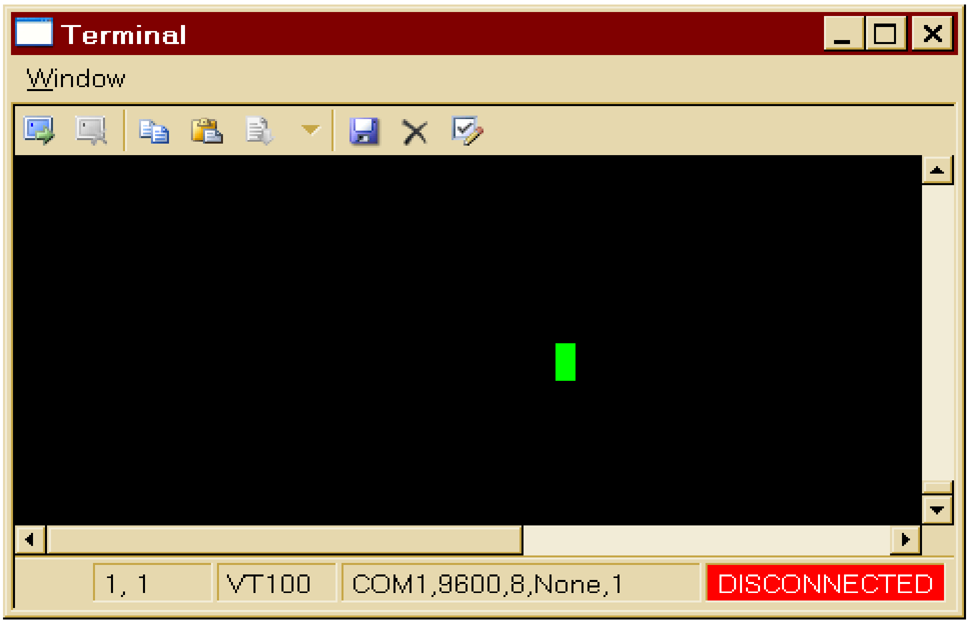
hello_ExtRam - winIDEA - [c:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demos\Hello\ExtRAM\main.c]

```
#include "service.h"
#include "mpc5500_xcptn.h"
#include "mpc5554.h"

extern int emiosChOCtr;
unsigned int sec_tick=0;

// ##### Main-Program ####

int main(void)
{
  unsigned short partNum;
  unsigned short maskNum;


  // Initialize CPU
  InitBoard();

  // Initialize exception handling
  xcptn_xmpl();

  // Initialize UART A
  InitUART(kBaud9600);

  my_printf("\n#### Hello phyCORE-MPC5554####\n\n");

  partNum = SIU.MIDR.B.PARTNUM;
  maskNum = SIU.MIDR.B.MASKNUM;

  my_printf("MPC PartNum %x, MaskNum %x\n\n", partNum, maskNum);

  my_printf("execution address of main() is 0x%08x\n\n", &main);


  while(1)
  {
    if(emiosChOCtr>1000)
    {
      emiosChOCtr=0;
      SIU.GPDO[179].R = !SIU.GPDO[179].R;    // toggle LED located on baseb
      my_printf("runtime %d\n", sec_tick++);
    }
  };

  return(0);
}
```

Now the processor has stopped at the breakpoint before the InitBoard() function. Breakpoints can be placed by setting the focus to a code line and pressing F9 (or press right mouse key and select corresponding entry from local menu!). The grey shaded area left the code line will then filled red (colours may differ and could be changed). Pressing F9 again remove the breakpoint.

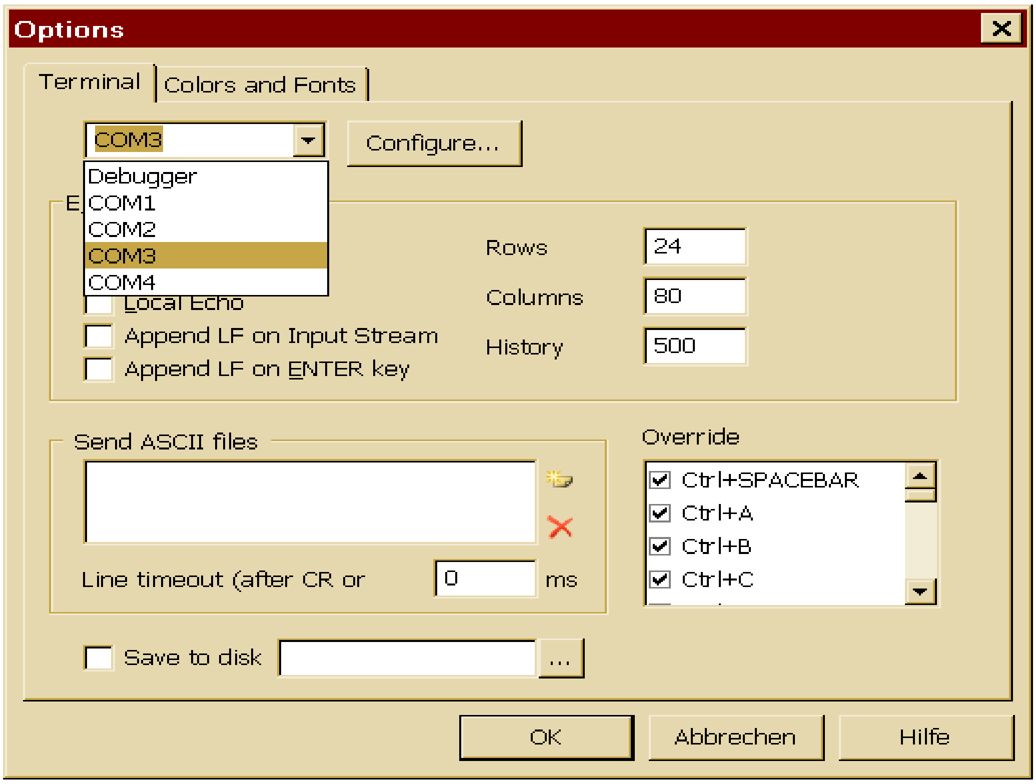
- Before running the complete demo like described, we need a terminal Window to show the output that will send over the serial port A (connector P2A at the development board; refer to section ***2.4***).

  If not already opened as "floating Window" (again: right mouse key within Windows for "local menu" and you may i. e. change the "Window type"): Open and configure the "Terminal Window" of the winIDEA by selecting *View|Terminal* over the menu bar or click to the ■ icon.



1, 1 | VT100 | COM1,9600,8,None,1 | DISCONNECTED

For configuration open the Options Icon within the terminal Window and select the corresponding Host PC serial port number. You are not limited to COM1 (default) to COM4, simple overwrite the selection field (i.e. COM7) instead of pull down selection.



COM3

E

□ Local Echo
□ Append LF on Input Stream
□ Append LF on ENTER key

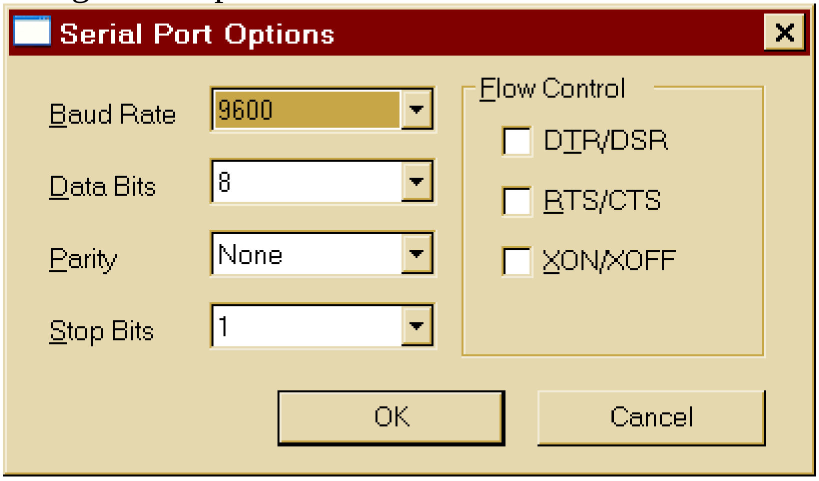
Rows 24
Columns 80
History 500

Line timeout (after CR or 0 ms

☑ Ctrl+SPACEBAR
☑ Ctrl+A
☑ Ctrl+B
☑ Ctrl+C

□ Save to disk ...

Press the ***Configure*** button adjust the Baud Rate to 9600 and set other settings like depicted below.



Baud Rate: 9600

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control

□ DTR/DSR

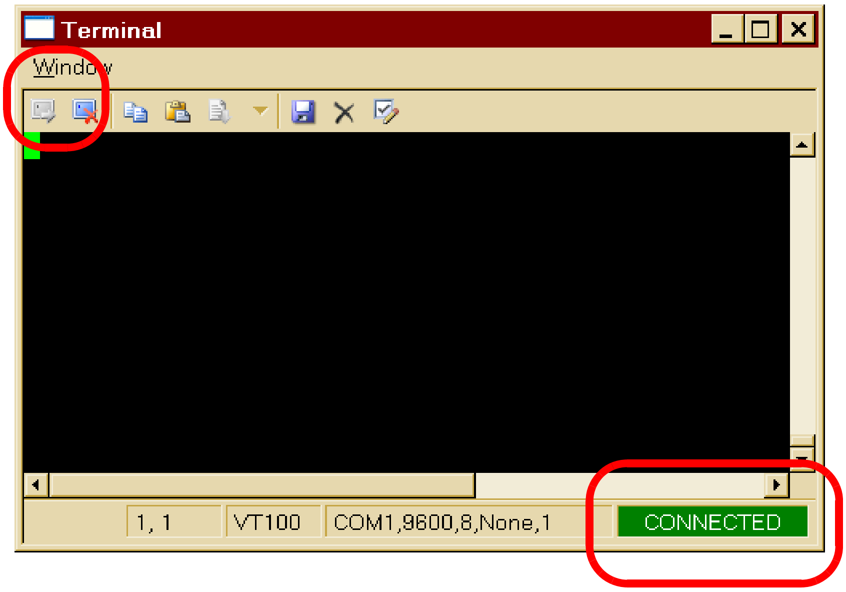
□ RTS/CTS

□ XON/XOFF

Close the Configure Window and Options Window by confirming with the OK button.

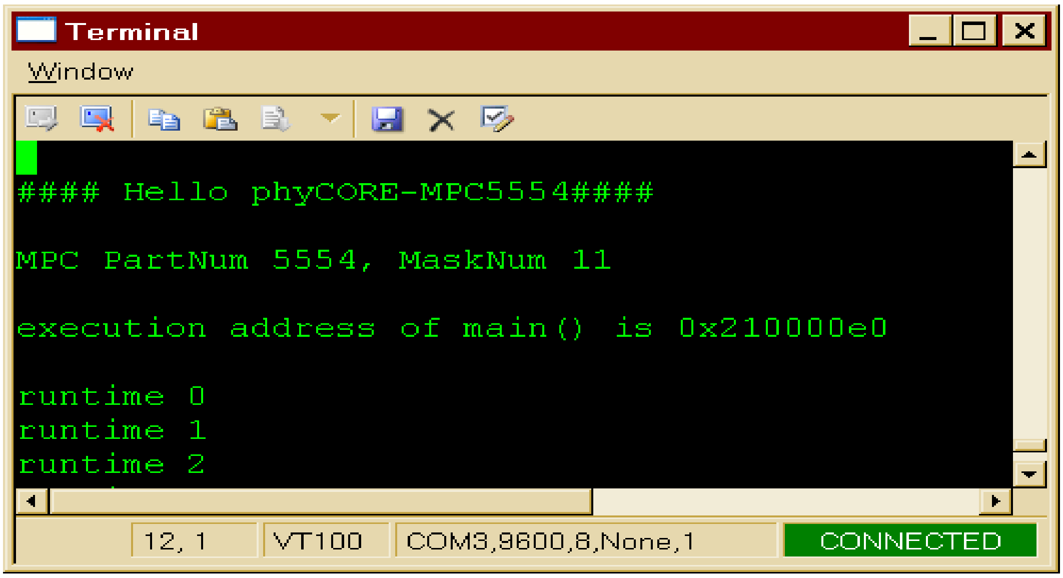
To establish the serial port communication press the ***Connect*** button within the Terminal Window icon bar.

With a successful opened port the Connect Icon disappears and the status in the right corner changes to ***CONNECTED***.



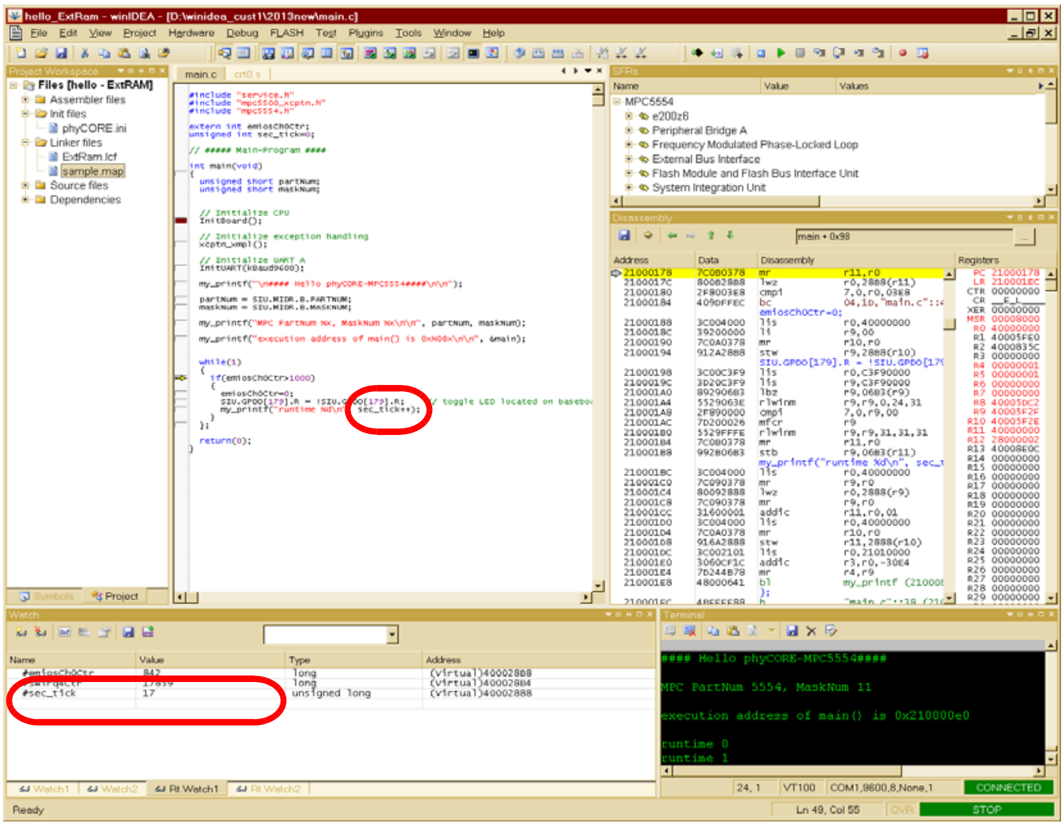
1, 1 | VT100 | COM1,9600,8,None,1 | CONNECTED

- Run the code by clicking on the ▶ icon or press F5. This will restart the code execution beginning with the InitBoard() function with full processor speed. The status of the processor is shown in the lower right corner of the winIDEA Window. It is now changed from ***STOP*** to ***RUN*** and Terminal Window starts display like



```
#### Hello phyCORE-MPC5554####

MPC PartNum 5554, MaskNum 11

execution address of main() is 0x210000e0

runtime 0
runtime 1
runtime 2
```

Function "main()" is located at external RAM address depicted above.



```
#include "service.h"
#include "mpc5500_xcptn.h"
#include "mpc5554.h"

extern int emiosCh0Ctr;
unsigned int sec_tick=0;

// ##### Main-Program ####

int main(void)
{
  unsigned short partNum;
  unsigned short maskNum;


  // Initialize CPU
  InitBoard();

  // Initialize exception handling
  xcptn_xmpl();

  // Initialize UART A
  InitUART(kBaud9600);

  my_printf("\n#### Hello phyCORE-MPC5554####\n\n");

  partNum = SIU.MIDR.B.PARTNUM;
  maskNum = SIU.MIDR.B.MASKNUM;

  my_printf("MPC PartNum %x, MaskNum %x\n\n", partNum, maskNum);

  my_printf("execution address of main() is 0x%08x\n\n", &main);


  while(1)
  {
    if(emiosCh0Ctr>1000)
    {
      emiosCh0Ctr=0;
      SIU.GPDO[179].R = !SIU.GPDO[179].R;       // toggle LED located on basebo
      my_printf("runtime %d\n", sec_tick++);
    }
  };

  return(0);
}
```

```
Address   Data      Disassembly                                Registers
21000178  7C0B0378  mr        r11,r0                           PC  21000178
2100017C  800B2888  lwz       r0,2888(r11)                     LR  210001EC
21000180  2F8003E8  cmpi      7,0,r0,03E8                      CTR 00000000
21000184  4090FFEC  bc        04,1D,"main.c"::4                CR  __E_L____
                    emiosCh0Ctr=0;                             XER 00000000
21000188  3C004000  lis       r0,40000000                      MSR 00008000
2100018C  39200000  li        r9,00                            R0  40000000
21000190  7C0A0378  mr        r10,r0                           R1  40005FE0
21000194  912A2888  stw       r9,2888(r10)                     R2  4000835C
                    SIU.GPDO[179].R = !SIU.GPDO[179            R3  00000000
21000198  3C00C3F9  lis       r0,C3F90000                      R4  00000001
2100019C  3D20C3F9  lis       r9,C3F90000                      R5  00000001
210001A0  89290683  lbz       r9,0683(r9)                      R6  00000000
210001A4  5529063E  rlwinm    r9,r9,0,24,31                    R7  00000000
210001A8  2F890000  cmpi      7,0,r9,00                        R8  40005DC2
210001AC  7D200026  mfcr      r9                               R9  40005F2F
210001B0  5529FFFE  rlwinm    r9,r9,31,31,31                   R10 40005F2E
210001B4  7C0B0378  mr        r11,r0                           R11 40000000
210001B8  992B0683  stb       r9,0683(r11)                     R12 28000002
                    my_printf("runtime %d\n", sec_t            R13 40008E0C
210001BC  3C004000  lis       r0,40000000                      R14 00000000
210001C0  7C090378  mr        r9,r0                            R15 00000000
210001C4  80092888  lwz       r0,2888(r9)                      R16 00000000
210001C8  7C090378  mr        r9,r0                            R17 00000000
210001CC  31600001  addic     r11,r0,01                        R18 00000000
210001D0  3C004000  lis       r0,40000000                      R19 00000000
210001D4  7C0A0378  mr        r10,r0                           R20 00000000
210001D8  916A2888  stw       r11,2888(r10)                    R21 00000000
210001DC  3C002101  lis       r0,21010000                      R22 00000000
210001E0  3060CF1C  addic     r3,r0,-30E4                      R23 00000000
210001E4  7D244B78  mr        r4,r9                            R24 00000000
210001E8  48000641  bl        my_printf (21000                 R25 00000000
                    );                                         R26 00000000
210001EC  4BFFFF88  b         "main.c"::38 (210                R27 00000000
                                                               R28 00000000
                                                               R29 00000000
```

| Name | Value | Type | Address |
|---|---|---|---|
| #emiosCh0Ctr | 842 | long | (virtual)400028B8 |
| #swirqctr | 17859 | long | (virtual)400028B4 |
| #sec_tick | 17 | unsigned long | (virtual)40002888 |

```
#### Hello phyCORE-MPC5554####

MPC PartNum 5554, MaskNum 11

execution address of main() is 0x210000e0

runtime 0
runtime 1
```

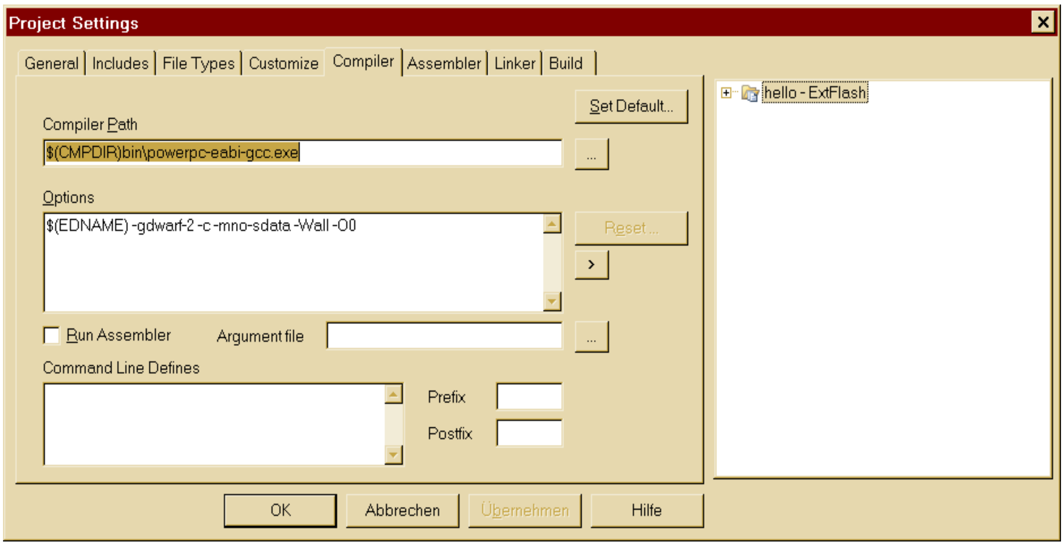
In the Terminal Window you can see some status messages and the output of a second counter. Also see the User LED D6 located on the baseboard between the Ethernet plug and the Expansion Bus Connector. The LED changes once a second.

The watch Window shows some pre-defined Real-time Watches. The debugger communicates via the JTAG/OnCE module of the Processor and use the "Real-time-watch" Feature. This allows reading of global variables contents without remarkable influence of the MCU (compare Freescale manual).

Note the ***sec_tick*** variable. This variable is incremented in the demo once a second dependent on the ***emiosCh0Ctr*** counter variable that is incremented by an interrupt service routine once a millisecond.

<u>Important remark:</u>

The projects are build with Compiler optimize setting "-O0" which means "no optimize" at all. If you change this Compiler optimization level it could happen that important parts of source code will be ignored for translation and application misbehaves if you not use "volatile" keyword accordingly!



Compiler Path: $(CMPDIR)bin\powerpc-eabi-gcc.exe

Options: $(EDNAME) -gdwarf-2 -c -mno-sdata -Wall -O0

□ Run Assembler Argument file

Command Line Defines

Prefix

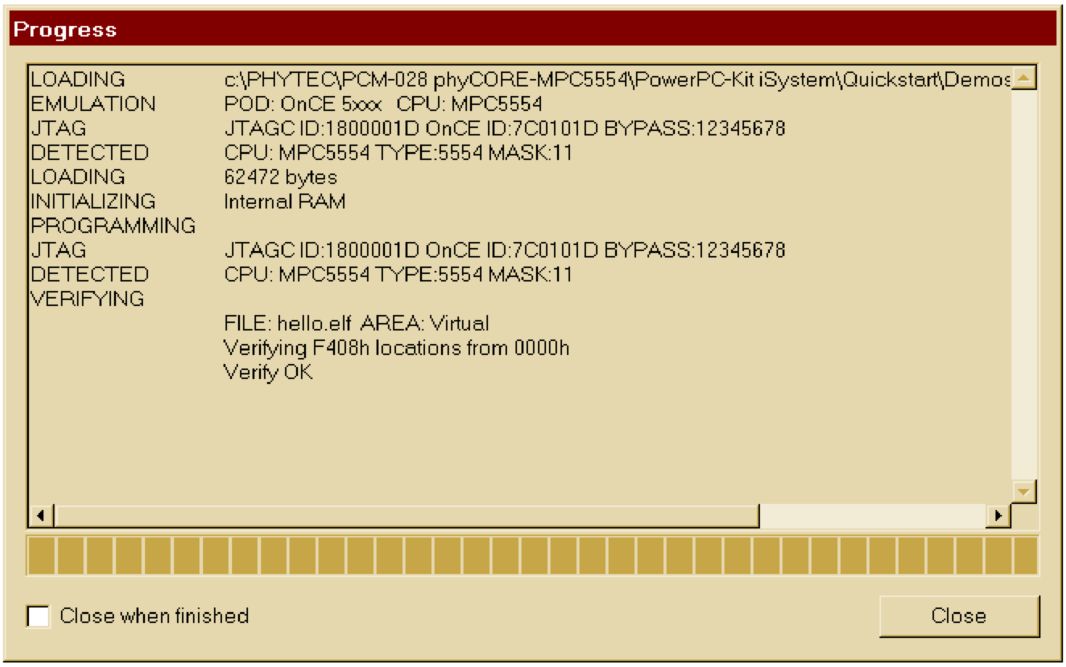
Postfix

For the GNU CC there are different optimize levels (0,1,2,3,s at least - refer GCC manual) and it is set [here -O0] within menu ***"Project - Settings, pane Compiler"*.**

### 2.5.2 Running the Hello Demo Project in MPC5554 On-Chip Flash

In this section you will download and program the Hello Demo to the MPC5554 on-chip Flash memory.

- Make sure Jumper JP1 on the phyCORE-MPC5554 Development Board is closed at position 2+3 (*refer to Figure 2*).
- Open the ***Hello_IntFLASH.xjrf*** and build the output file ***hello.elf*** as described in section ***2.5.1***.
- To program the ***hello.elf*** to the internal processor Flash perform a download by selecting the icon ➡ or press Ctrl+F3.



Progress

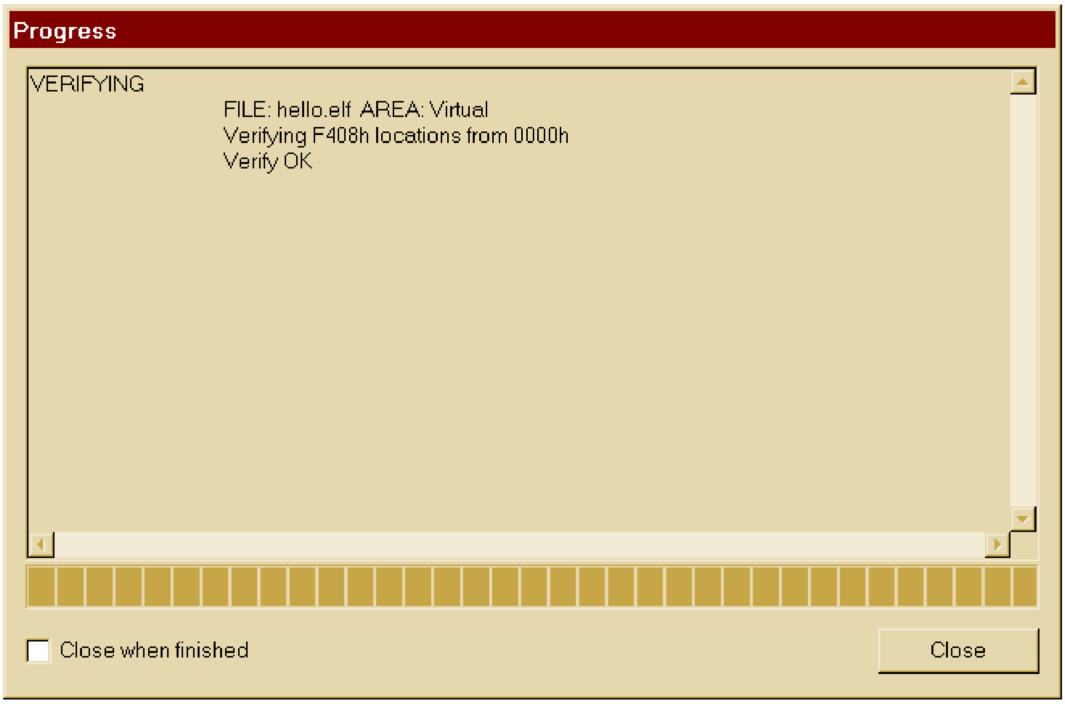
```
LOADING         c:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demos
EMULATION       POD: OnCE 5xxx   CPU: MPC5554
JTAG            JTAGC ID:1800001D OnCE ID:7C0101D BYPASS:12345678
DETECTED        CPU: MPC5554 TYPE:5554 MASK:11
LOADING         62472 bytes
INITIALIZING    Internal RAM
PROGRAMMING
JTAG            JTAGC ID:1800001D OnCE ID:7C0101D BYPASS:12345678
DETECTED        CPU: MPC5554 TYPE:5554 MASK:11
VERIFYING
                FILE: hello.elf  AREA: Virtual
                Verifying F408h locations from 0000h
                Verify OK
```

□ Close when finished

winIDEA automatically first erase and then burns the binary to the internal CPU Flash during download process.

Verify is performed on-the-fly while flashing and - if set like above - as a extra compare file : flash content after all. Above "close when (successful) finished" was unchecked to take the screen shot).

But you may use additionally a extra verify procedure by selecting ***Debug | Verify Download*** from the main menu bar at any time you like. This feature of winIDEA compares again the Download File with its target address area in the memory and your progress picture should look like this (usually check "Close when finished - if any Error here this Window remains open)



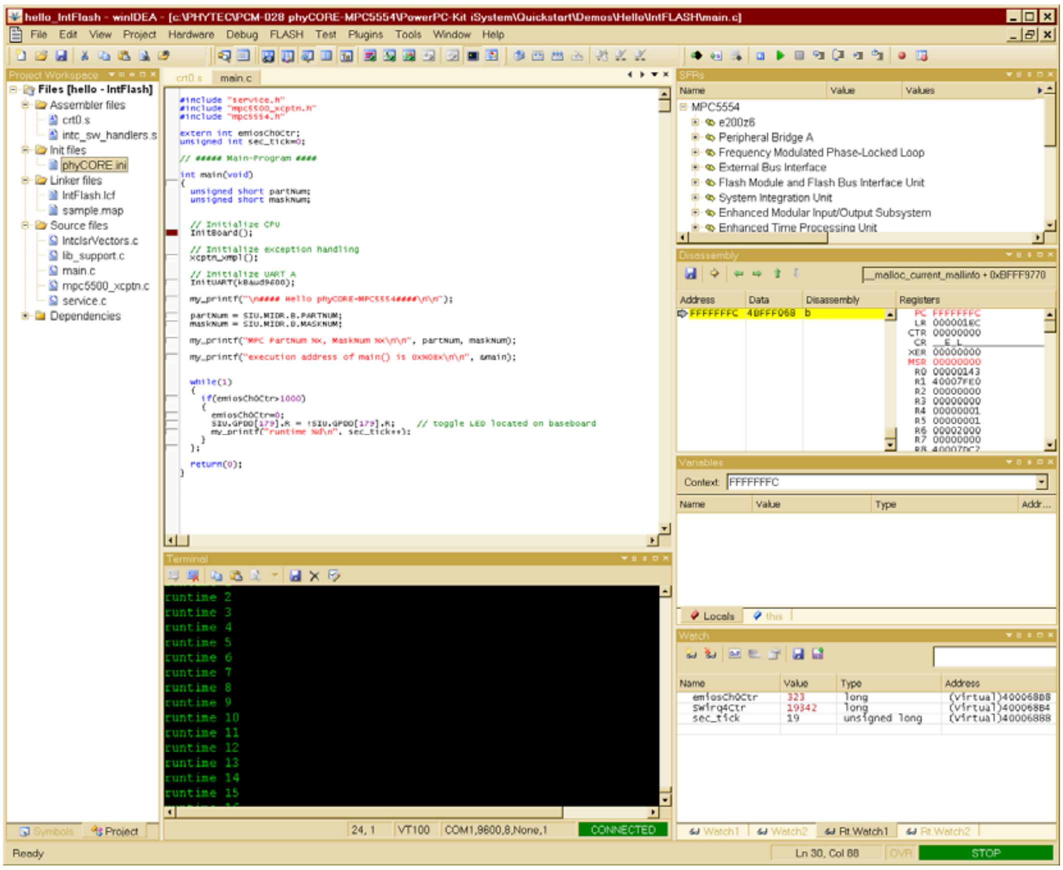
```
VERIFYING
                FILE: hello.elf  AREA: Virtual
                Verifying F408h locations from 0000h
                Verify OK
```

It is independent of the memory type and will work for ROM or RAM.

Note that winIDEA is able to change directly the Content of any kind of RAM content and internal Flash (i.e. patch Code or change Constants is possible with that feature).

You have now successfully downloaded and burned the ***Hello*** example program to the MPC5554 on-chip Flash memory.

The next steps are to start the ***Hello*** application with the debugger or use it stand-alone.



```
#include "service.h"
#include "mpc5500_xcptn.h"
#include "mpc5554.h"

extern int emiosCh0Ctr;
unsigned int sec_tick=0;

// ##### Main-Program ####

int main(void)
{
  unsigned short partNum;
  unsigned short maskNum;


  // Initialize CPU
  InitBoard();

  // Initialize exception handling
  xcptn_xmpl();

  // Initialize UART A
  InitUART(kBaud9600);

  my_printf("\n#### Hello phyCORE-MPC5554####\n\n");

  partNum = SIU.MIDR.B.PARTNUM;
  maskNum = SIU.MIDR.B.MASKNUM;

  my_printf("MPC PartNum %x, MaskNum %x\n\n", partNum, maskNum);

  my_printf("execution address of main() is 0x%08x\n\n", &main);


  while(1)
  {
    if(emiosCh0Ctr>1000)
    {
      emiosCh0Ctr=0;
      SIU.GPDO[179].R = !SIU.GPDO[179].R;     // toggle LED located on baseboard
      my_printf("runtime %d\n", sec_tick++);
    }
  };

  return(0);
}
```

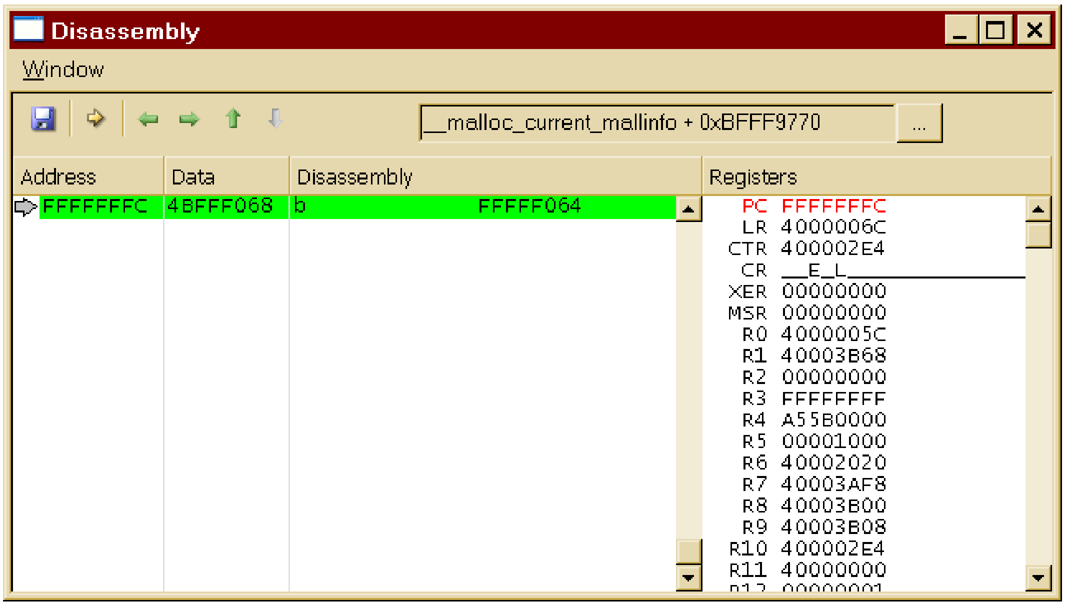
```
runtime 2
runtime 3
runtime 4
runtime 5
runtime 6
runtime 7
runtime 8
runtime 9
runtime 10
runtime 11
runtime 12
runtime 13
runtime 14
runtime 15
```

__malloc_current_mallinfo + 0xBFFF9770

```
Address   Data      Disassembly   Registers
FFFFFFFC  4BFFF068  b              PC  FFFFFFFC
                                   LR  000001EC
                                  CTR  00000000
                                   CR  __E_L
                                  XER  00000000
                                  MSR  00000000
                                   R0  00000143
                                   R1  40007FE0
                                   R2  00000000
                                   R3  00000000
                                   R4  00000001
                                   R5  00000001
                                   R6  00002000
                                   R7  00000000
                                   R8  40007DC2
```

| Name | Value | Type | Address |
|---|---|---|---|
| emiosCh0Ctr | 323 | long | (virtual)400068B8 |
| swirq4Ctr | 19342 | long | (virtual)400068B4 |
| sec_tick | 19 | unsigned long | (virtual)40006888 |

- Running the "Hello" with the Debugger
  After download/burn procedure the processor is stopped at the first code instruction (if not set to other desired options within winIDEA Debug/Download File/options). Prior executing the hello demo code the processor runs its internal BAM (Boot Assist Module) code.
  This code starts at physical address 0xFFFFFFFC, pre-initializes the processor and reads the first two 32-bit words (see crt0.s file) from flash address 0x00000000. The first 32-bit word code contains a boot identifier and the second is the start address (0x00000008) of the application ("hello" demo). After finishing the BAM processor part branches to that address and would run the application code.



__malloc_current_mallinfo + 0xBFFF9770

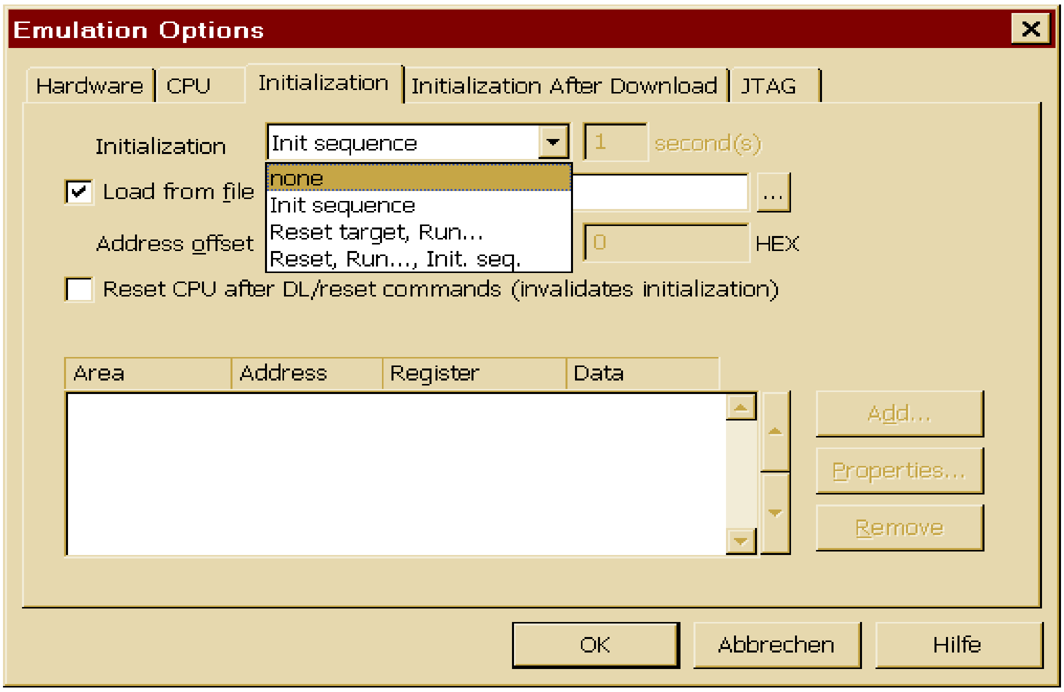
| Address | Data | Disassembly |
|---|---|---|
| FFFFFFFC | 4BFFF068 | b FFFFF064 |

```
Registers
 PC  FFFFFFFC
 LR  4000006C
CTR  400002E4
 CR  __E_L_
XER  00000000
MSR  00000000
 R0  4000005C
 R1  40003B68
 R2  00000000
 R3  FFFFFFFF
 R4  A55B0000
 R5  00001000
 R6  40002020
 R7  40003AF8
 R8  40003B00
 R9  40003B08
R10  400002E4
R11  40000000
R12  00000001
```

But for your comfort we use the INI File too, in this case Execution Point is set to 0x00000008 at the end of INI file to avoid that confusingly (but correct) BAM-Start disassembly Window picture above after RESET/Download.

If you want really see this origin MCU behavior (important to know for own and/or stand-alone projects!) disable using INI File (attempt only for that internal Flash Demo please) in winIDEA Hardware => Emulation Options => Initialization (see next picture), change from "Init sequence" to "none". You can change it back at every time to get it comfortable again.

If you build own applications they have to run - after successful debugging period - always without any INI File or they would not run without Debug tool.



Emulation Options

Initialization: Init sequence

☑ Load from file

Address offset: 0 HEX

□ Reset CPU after DL/reset commands (invalidates initialization)

Anyway: To start the execution click on the ▶ icon or press F5. This and the following are the same procedures as described in the previous section.

- Stand-alone running
  Check that jumper JP1 on the baseboard is configured to 2+3 (*refer to Figure 2*) and press the reset button S2 near the double DB9 serial connector. The system will re-start out of the on-chip Flash memory.
  Or re-connect the power supply cable to X5. The target performs automatically an power-on reset and starts the hello application out of the on-chip internal Flash.

### 2.5.3 Running the Hello Demo Project in external Flash

In this section you will download program the Hello Demo to external Flash memory and will execute it from there.
Using external Flash memory for program execution, development and parameter storage have some advantages compare to limited internal MCU flash. The size of external Flash device could be very much bigger (important for RTOS, Graphics, complex protocols i.e.). It could usually cover very much more re-flash cycles before getting worn out. It could be used/mixed together with the internal Flash (i.e. internal Flash for startup the system and basic routines, external Flash used for big applications).
Now we want to look on it for program execution.

- Make sure Jumper JP2 on the phyCORE-MPC5554 Development Board is closed at position 1+2 (*refer to Figure 2*).
- Open the ***Hello_ExtFLASH.xjrf*** and build the output file ***hello.elf*** as described in section ***2.5.1***.

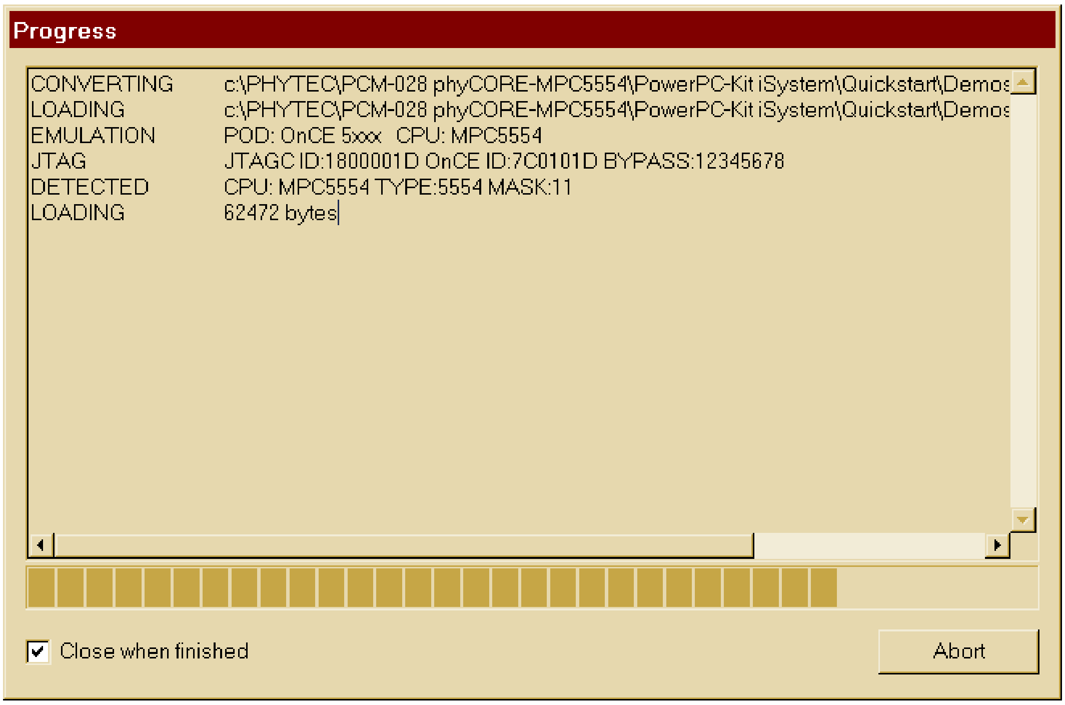
(***) Because PHYTEC has different Flash types soldered in different sizes please check FIRST which kind of flash are mounted (usually two decives for covering the 32bit bus with 16bit-wide each) before continue! To change this default setting in winIDEA please refer to **External Flash setup** on Section 2.5. If you verified the correct external Flash type set there you may continue here with the comfortable, easy way.

Standard 2012 is this kind of flash type, the **bold** one is setup in the Example.

**2MB phyCORE Modul (Flash-1, 2x IM615: S29AL008J70BFI010)**

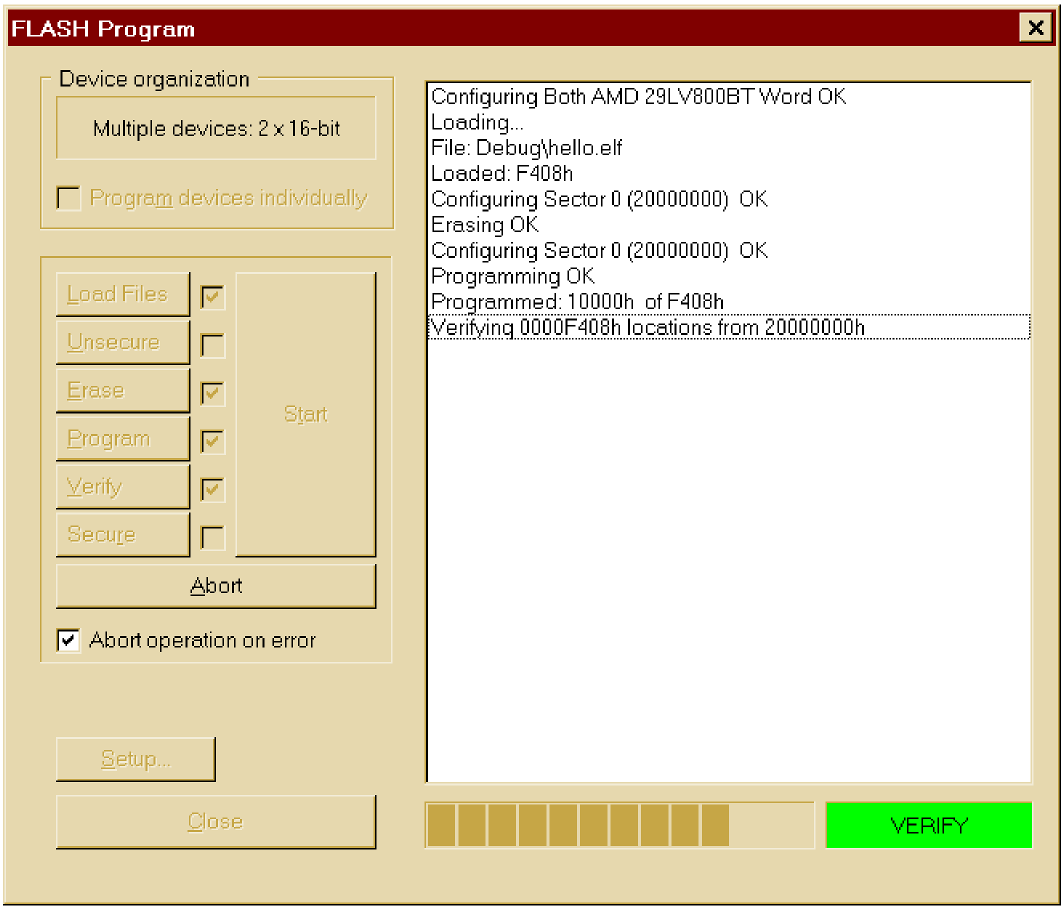
8MB phyCORE Modul (Flash-3, 2x IM435: S29AL032D90BFI030)

- Easy way: Simply press usual winIDEA Download key. Because of the setting it loads the application fist (Debug Info).
- Then it invokes the winIDEA external Flash part automatically. This looks like depicted on next page. Please be patient, it takes about 78 seconds (41 sec. i.e. for iC5000) total to finish the Flash process for external flash.



```
CONVERTING     c:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demos
LOADING        c:\PHYTEC\PCM-028 phyCORE-MPC5554\PowerPC-Kit iSystem\Quickstart\Demos
EMULATION      POD: OnCE 5xxx   CPU: MPC5554
JTAG           JTAGC ID:1800001D OnCE ID:7C0101D BYPASS:12345678
DETECTED       CPU: MPC5554 TYPE:5554 MASK:11
LOADING        62472 bytes
```

☑ Close when finished



Device organization

Multiple devices: 2 x 16-bit

☐ Program devices individually

Load Files ☑
Unsecure ☐
Erase ☑
Program ☑
Verify ☑
Secure ☐

☑ Abort operation on error

```
Configuring Both AMD 29LV800BT Word OK
Loading...
File: Debug\hello.elf
Loaded: F408h
Configuring Sector 0 (20000000)  OK
Erasing OK
Configuring Sector 0 (20000000)  OK
Programming OK
Programmed: 10000h  of F408h
Verifying 0000F408h locations from 20000000h
```

How this works? It reads configuration from Menu Flash/Program and Download File setting, combine it and use the stored information to

- configure the external Flash type (***)

- use download file and its information, load first the Debug info

- erase the contents (in this case only involved sectors) ext. Flash

- program the application code inside the external Flash

- verify it

- and return & continue to usual download part for downloading the symbol information

- close the Flash Program Window.

You have now successfully downloaded and burned the ***Hello*** example program to the external Flash memory.
The next steps are to start the ***Hello*** application with the debugger or use it stand-alone.

- Running the Hello with the Debugger

  First click to the Download Icon ➡ to initialize the project for debugging. Then run the code by clicking on the Run Icon ▶ or press F5. This will execute the startup code (crt0.s) and the processor is running until the first breakpoint is reached near the InitBoard() function in main.c. Press again F5 to go ahead with code execution. Now you can see the output in the terminal Window and the blinking LED as in the previous sections.

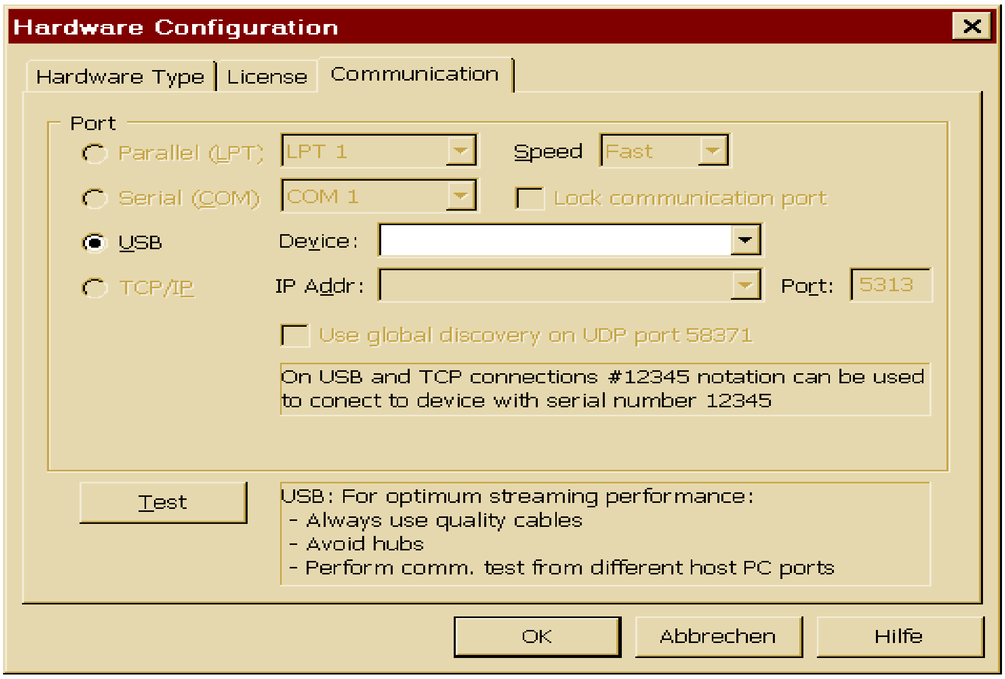
- Stand-alone running
  Check that jumper JP1 on the baseboard is configured to 1+2 (*refer to Figure 2*) and press the reset button S2 near the double DB9 serial connector. The system will re-start out of the external Flash memory. Or re-connect the power supply cable to X5. The target performs automatically an power-on reset and starts the hello

application out of the external Flash. Address reported is 0x200000E0 for function "main()" in that case.

## 2.6 iSYSTEM Background Information and Troubleshooting.

If "Error 40: Communication port not found" rise with new Debug device at beginning, then please select Hardware \ Hardware… from the winIDEA menu bar, navigate to pane Communication and clear content of field Device.

Leave this field blank and press the "Test" button. Now PC and iONE-E exchange Data packages. If this works fine, then you can connect any iONE-E to that Workspace. Reason is, that the winIDEA and the blue iONE-box tightly coupled. With that feature you could connect several boxes on same PC to different winIDEA - i.e. for Multi-Processing.



○ Parallel (LPT) LPT 1 Speed Fast

○ Serial (COM) COM 1 □ Lock communication port

● USB Device:

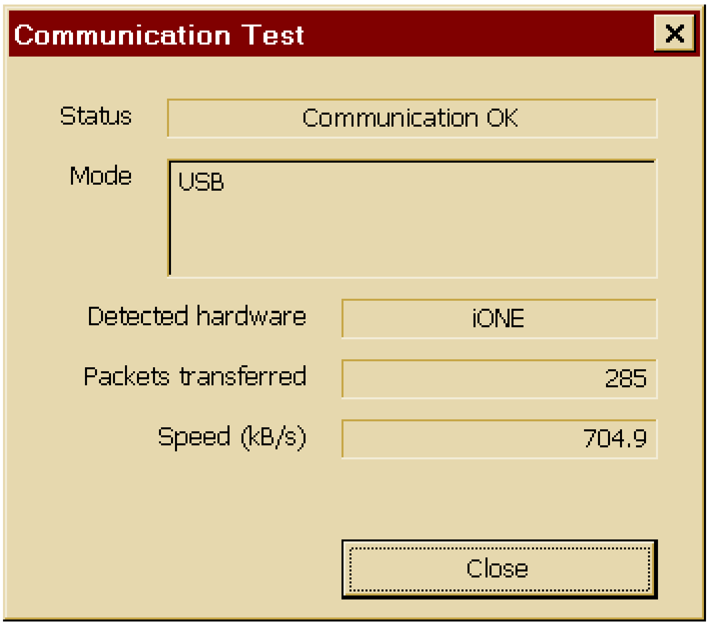
○ TCP/IP IP Addr: Port: 5313

□ Use global discovery on UDP port 58371

On USB and TCP connections #12345 notation can be used to conect to device with serial number 12345

USB: For optimum streaming performance:
- Always use quality cables
- Avoid hubs
- Perform comm. test from different host PC ports

Clicking on Button ***Test*** should show a working USB-connection now. Click ***close*** and ***ok*** to return to ***winIDEA*** main Window and click the download icon again. It should work now.



Communication Test

Status: Communication OK

Mode: USB

Detected hardware: iONE

Packets transferred: 285

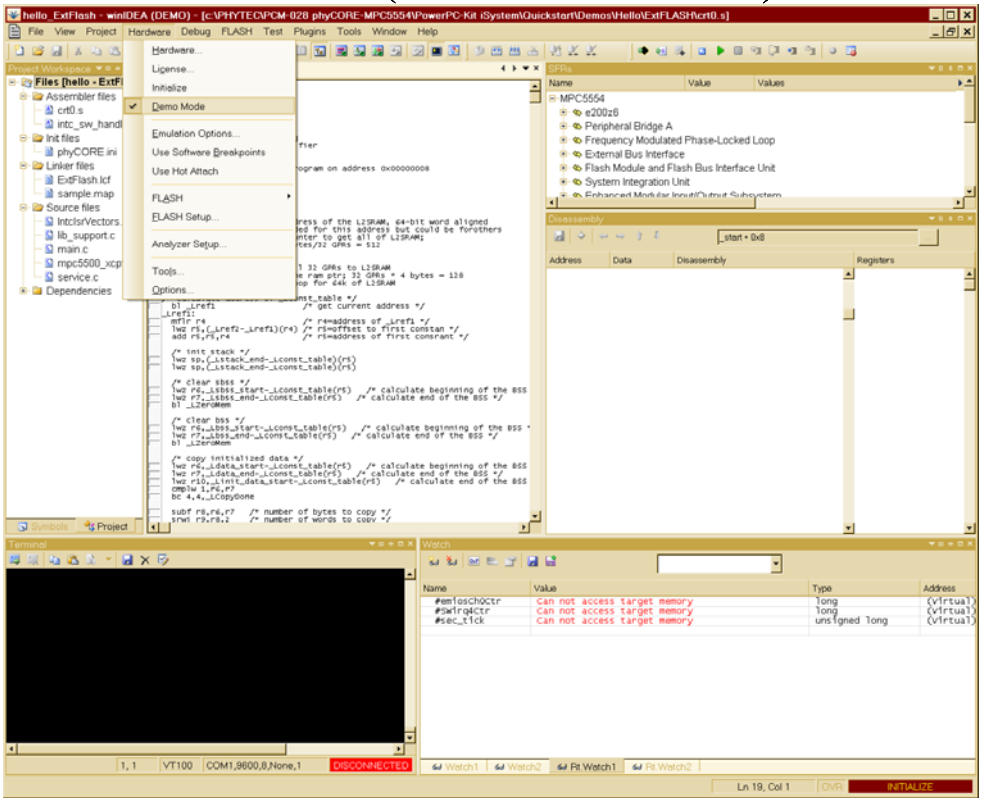
Speed (kB/s): 704.9

**"Hardware not found" // Communication Error // Freeze**

If you have this kind of "Communication Error" or "Hardware not found" messages during your work simply disconnect and reconnect the iONE-E from USB Cable, but let it off for about 5 seconds. Because iONE-E is only a Debugger for Evaluation purpose it has neither RESET key nor complex Hardware inside. So like all simple tools with no external power supply it could be, that it "hangs up" or winIDEA freeze. Disconnect, wait short and re-connect solve these problems.

If Debugger hang up constantly repeatable with your own or changed sample projects but work with the described, unchanged examples, it could be that your application code is wrong linked. Download to "nowhere" or out of boundaries and jump to it usually cause that kind of trouble. To verify your project you could analyse the *.MAP file (build from the Compiler, in the Examples you find it under "Linker Files" in Project Workspace Window of winIDEA.

You may take a look in those cases without using target with the winIDEA "Demo Mode" (Hardware - Demo Mode).



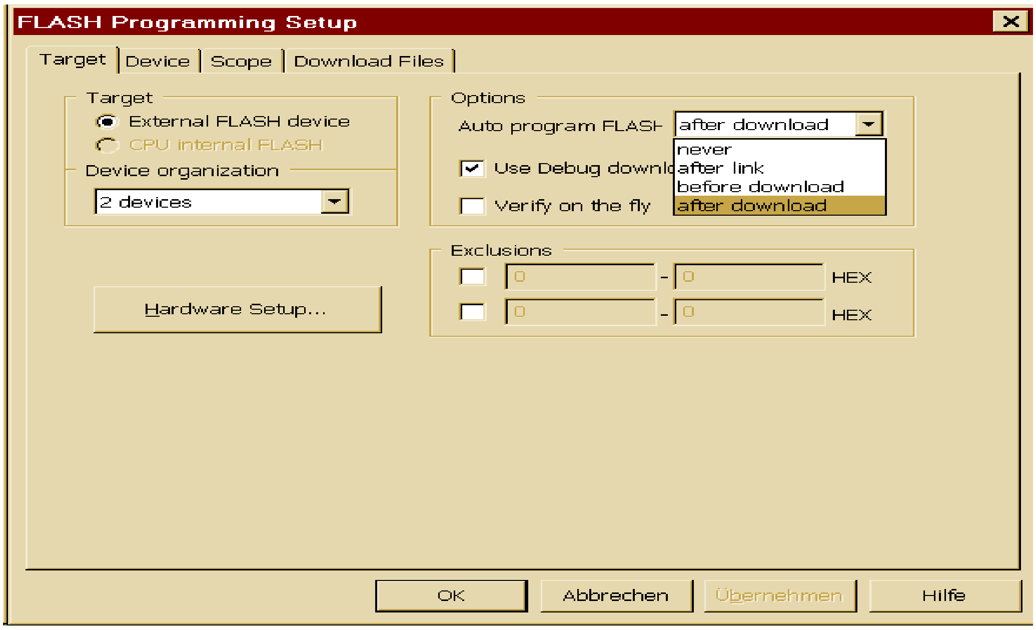
```
fier
ogram on address 0x00000008

ress of the L2SRAM, 64-bit word aligned
ed for this address but could be forothers
nter to get all of L2SRAM;
tes/32 GPRs = 512

l 32 GPRs to L2SRAM
e ram ptr; 32 GPRs * 4 bytes = 128
op for 64k of L2SRAM

nst_table */
  bl _Lref1                   /* get current address */
_Lref1:
  mflr r4                     /* r4=address of _Lref1 */
  lwz r5,(_Lref2-_Lref1)(r4) /* r5=offset to first constan */
  add r5,r5,r4                /* r5=address of first consrant */

  /* init stack */
  lwz sp,(_Lstack_end-_Lconst_table)(r5)
  lwz sp,(_Lstack_end-_Lconst_table)(r5)

  /* clear sbss */
  lwz r6,_Lsbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS
  lwz r7,_Lsbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* clear bss */
  lwz r6,_Lbss_start-_Lconst_table(r5)   /* calculate beginning of the BSS *
  lwz r7,_Lbss_end-_Lconst_table(r5)   /* calculate end of the BSS */
  bl _LZeroMem

  /* copy initialized data */
  lwz r6,_Ldata_start-_Lconst_table(r5)   /* calculate beginning of the BSS
  lwz r7,_Ldata_end-_Lconst_table(r5)   /* calculate end of the BSS */
  lwz r10,_Linit_data_start-_Lconst_table(r5)   /* calculate end of the BSS
  cmplw 1,r6,r7
  bc 4,4,_LCopyDone

  subf r8,r6,r7   /* number of bytes to copy */
  srwi r9,r8,2    /* number of words to copy */
```

| Name | Value | Type | Address |
|---|---|---|---|
| #emiosCh0Ctr | Can not access target memory | long | (virtual) |
| #SwIrq4Ctr | Can not access target memory | long | (virtual) |
| #sec_tick | Can not access target memory | unsigned long | (virtual) |

In that case the neither physical iONE-E nor target is used (you have to switch off automated external Flash download), but implemented loader load your Application into PC RAM. There you may expect it, including disassembling. But again: This is NOT a Simulator, so stepping or so is impossible. Don’t forget to uncheck the "Demo Mode" afterwards for continue with normal Target debugging with iONE-E!

## 2.5.1 External Flash setup (***).

- To setup & program the ***hello.elf*** "manually" to the external Flash open the Flash Programmer Window by selecting ***Flash | Setup*** from the main menu bar of the winIDEA.


FLASH Programming Setup

Target | Device | Scope | Download Files

Target

● External FLASH device

○ CPU internal FLASH

Device organization: 2 devices

Options

Auto program FLASH: after download

☑ Use Debug downl

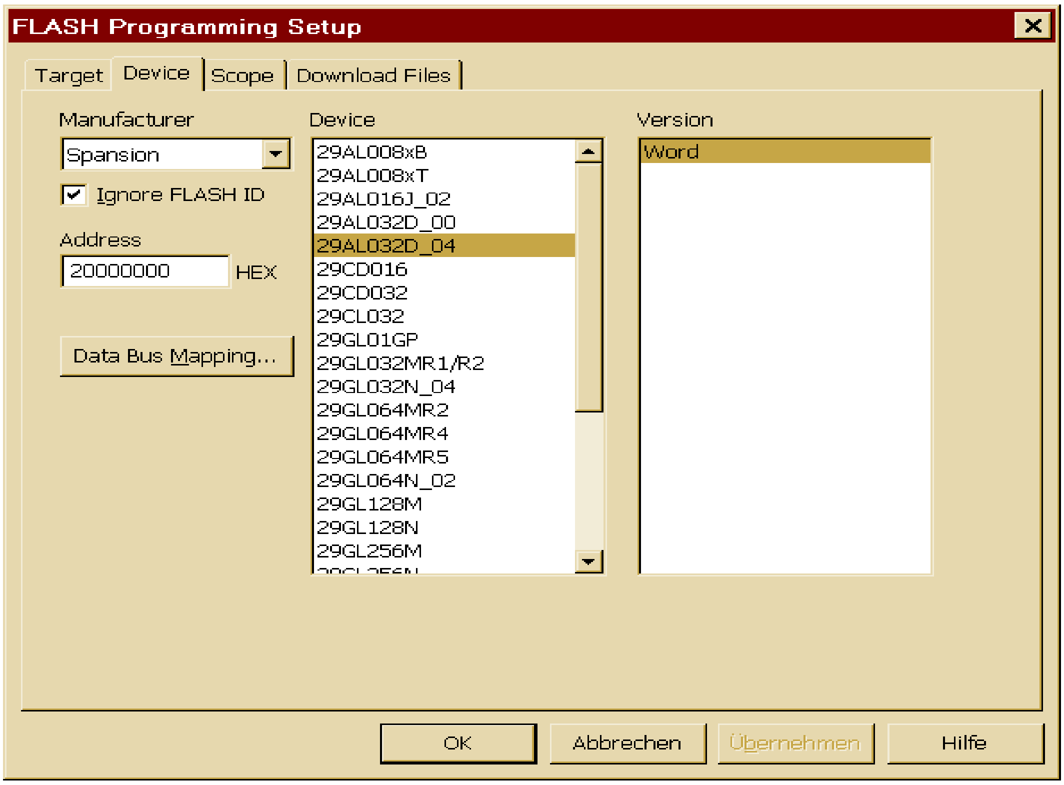
□ Verify on the fly

Exclusions

□ 0 - 0 HEX

□ 0 - 0 HEX

- Check the pane "Device" and compare with your physically soldered device on the Target.


FLASH Programming Setup

Target | Device | Scope | Download Files

Manufacturer: Spansion

☑ Ignore FLASH ID

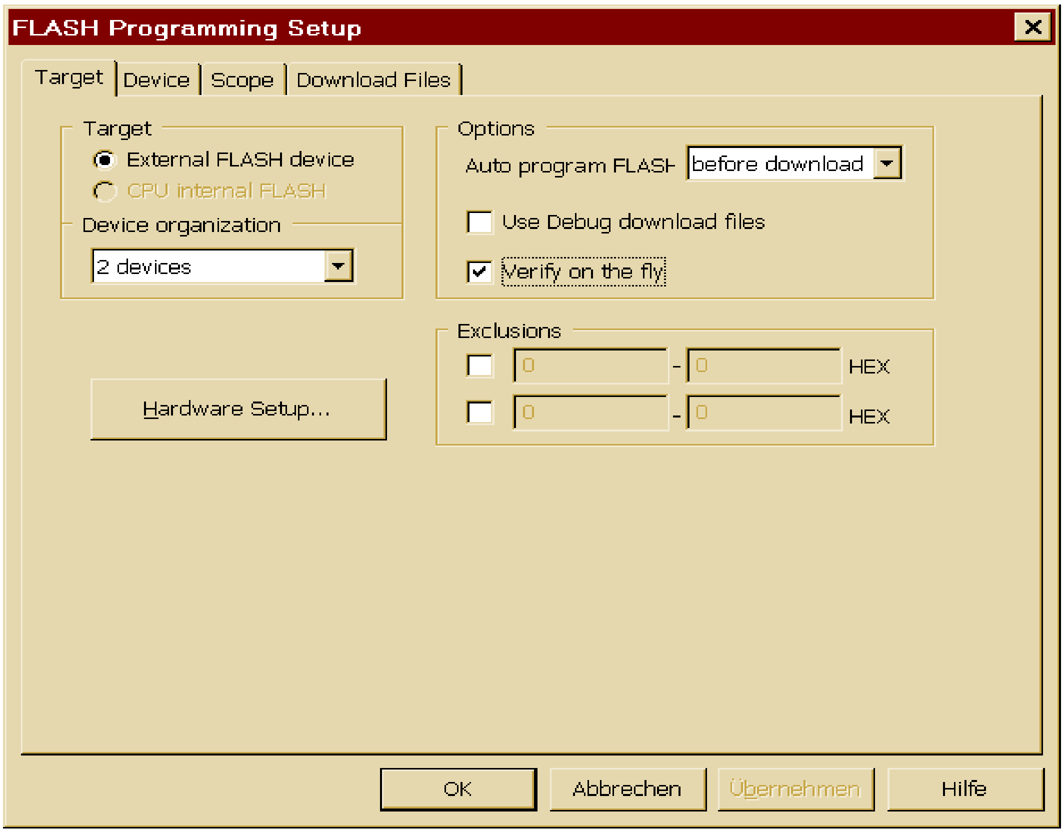
Address: 20000000 HEX

Device: 29AL032D_04

Version: Word

- Above you find an overview as example. "Word (16bit)" is usually correct if - like here - are different possibilities. If necessary please change ONLY here. If you don't find your corresponding device please write this to Support (support@isystem.com), but add the description of that missed device.

- If you don't like the "automated" process described in external Flash example or if you want to do more complex things winIDEA can do that. So you could have one file for external Flash, other for internal Flash and a File with Parameters for RAM. In that case "automatic" cannot do this.
- So uncheck the "Use Debug download files" within FLASH Programming setup first.


FLASH Programming Setup

Target | Device | Scope | Download Files

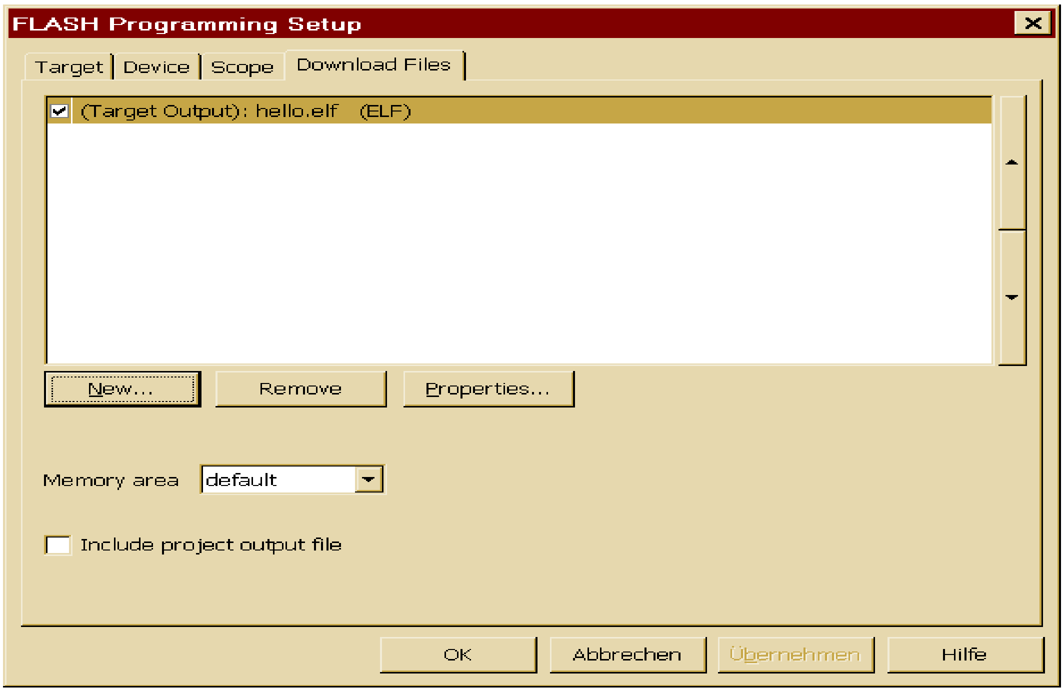
Target
● External FLASH device
○ CPU internal FLASH

Device organization: 2 devices

Options
Auto program FLASH: before download
□ Use Debug download files
☑ Verify on the fly

Exclusions
□ 0 - 0 HEX
□ 0 - 0 HEX

- Change to pane "***Download Files***" within FLASH Programming setup. Add your desired file(s) there. This will be flashed to the external Flash.



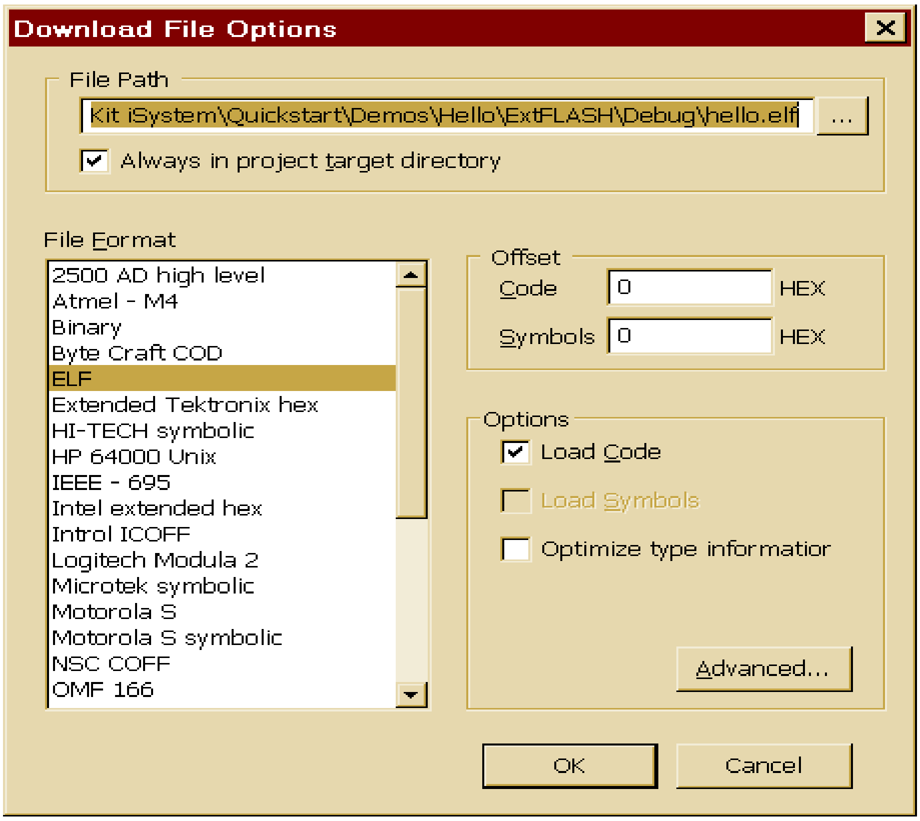
FLASH Programming Setup

Download Files

☑ (Target Output): hello.elf (ELF)

Memory area: default

□ Include project output file



Download File Options

File Path: Kit iSystem\Quickstart\Demos\Hello\ExtFLASH\Debug\hello.elf

☑ Always in project target directory

File Format:

- 2500 AD high level
- Atmel - M4
- Binary
- Byte Craft COD
- **ELF**
- Extended Tektronix hex
- HI-TECH symbolic
- HP 64000 Unix
- IEEE - 695
- Intel extended hex
- Introl ICOFF
- Logitech Modula 2
- Microtek symbolic
- Motorola S
- Motorola S symbolic
- NSC COFF
- OMF 166

Offset

Code: 0 HEX

Symbols: 0 HEX

Options

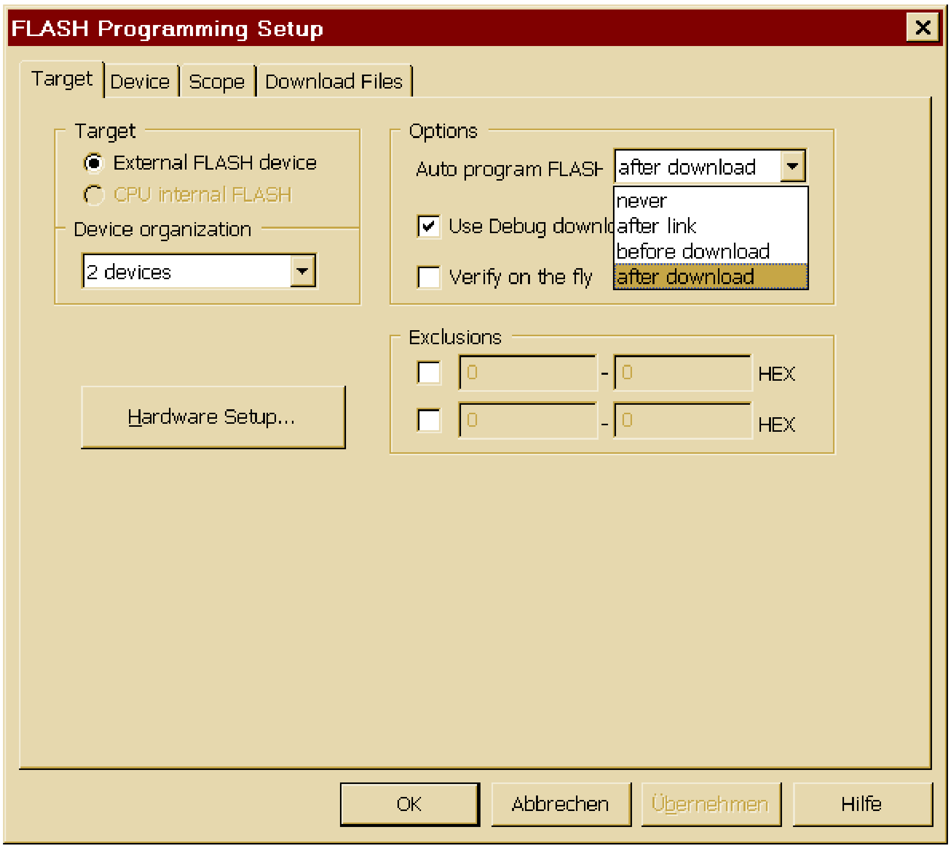
☑ Load Code

□ Load Symbols

□ Optimize type informatior

- You can see that here only CODE without Debug info will be stored into external Flash.

- In case of part of the application code you have to add that download file within menu ***Debug - files for Download - Download files***, but change properties to only "Load Symbols" there!
- If content of external Flash does not change often it is a good idea to flash external content manually. Because dealing with external Flash on simple iONE-E is performed via JTAG Port of MCU OCD only and with the USB 1.1 Low Speed (1.5 Mbit/s) connection this takes a very long time (i.e. 90 seconds her for the example). With iC5000 i.e. with its Hi-Speed (480 Mbit/s) it is possible to use a "UMI Monitor" and block operations instead of byte-after-byte. This would dramatically lower the flash-time consumption for external Flash.
- For invoke the flash process always manually change "Auto program Flash" option within FLASH programming setup screen to "never".



FLASH Programming Setup

Target

● External FLASH device

○ CPU internal FLASH

Device organization: 2 devices

Options

Auto program FLASH: after download

never
after link
before download
after download

☑ Use Debug downl

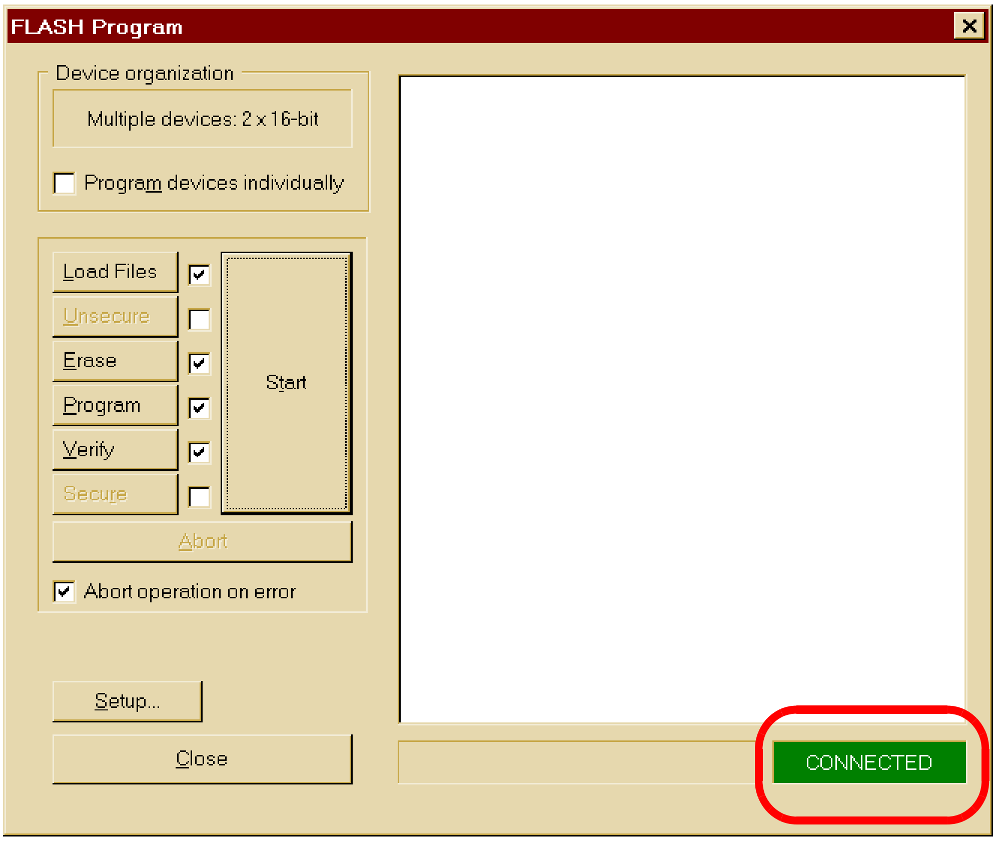
□ Verify on the fly

Exclusions

□ 0 - 0 HEX

□ 0 - 0 HEX

- Then you may select "Program" from FLASH menu. It should depict "connected" or the setup is wrong.



FLASH Program

Device organization

Multiple devices: 2 x 16-bit

□ Program devices individually

☑ Load Files

□ Unsecure

☑ Erase

☑ Program

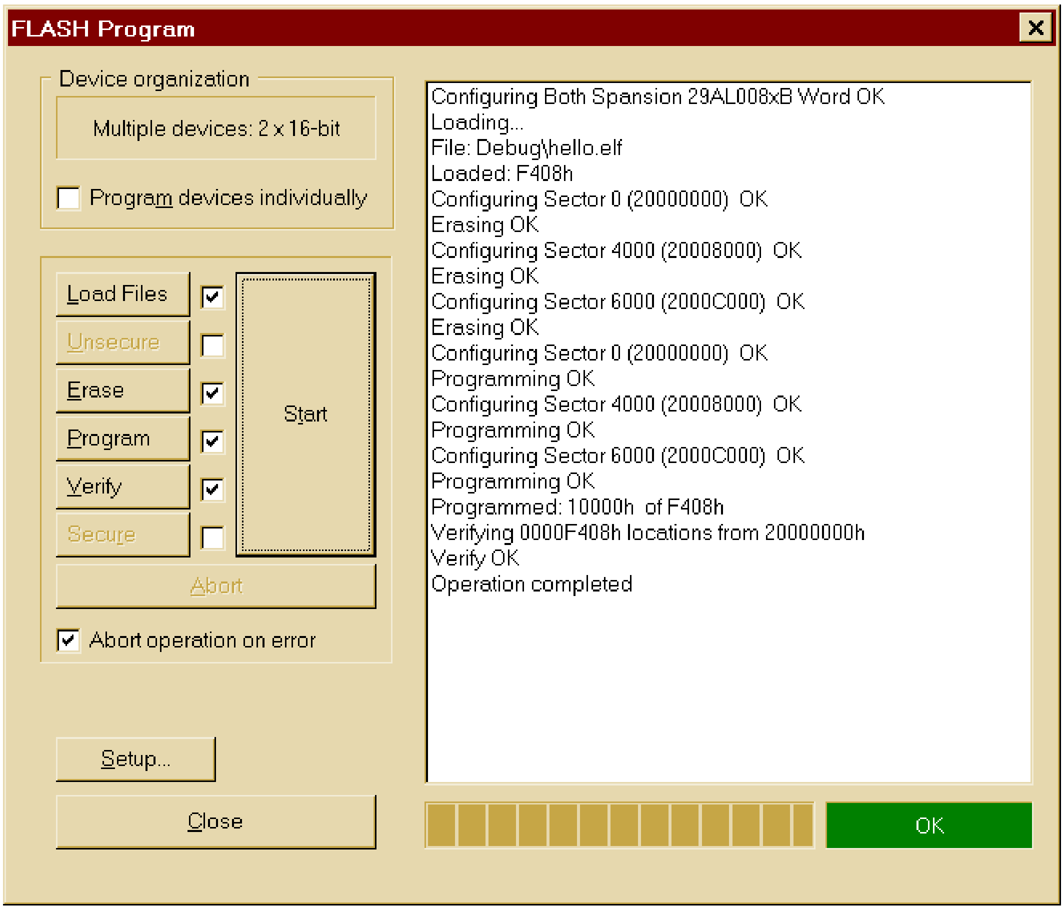
☑ Verify

□ Secure

☑ Abort operation on error

CONNECTED

- Now you are able to perform the flash process by pressing "Start" button or do it step-by-step with the corresponding buttons on left side of that Window. This is very useful in case of trouble for investigations.
- For "daily" development process which using external Flash device it is more convenient to invoke the flash programming prior the debugger Initialize/Download procedure automatically. This is the reason why example was set in that way.



```
Configuring Both Spansion 29AL008xB Word OK
Loading...
File: Debug\hello.elf
Loaded: F408h
Configuring Sector 0 (20000000)  OK
Erasing OK
Configuring Sector 4000 (20008000)  OK
Erasing OK
Configuring Sector 6000 (2000C000)  OK
Erasing OK
Configuring Sector 0 (20000000)  OK
Programming OK
Configuring Sector 4000 (20008000)  OK
Programming OK
Configuring Sector 6000 (2000C000)  OK
Programming OK
Programmed: 10000h  of F408h
Verifying 0000F408h locations from 20000000h
Verify OK
Operation completed
```

- Anyway the Window after flash process should looks like this successful procedure.

### 2.5.2 Speed-up Flash process for external Flash devices

- Programming of external Flash from Debugger via JTAG (OCD) Debug Port only is usually very slow. Reason is that only the MCU "under debug" has access to those external devices via its external Bus system and it takes a lot of time to program "Byte after Byte".
- This procedure could be improved if a "Flash Monitor" application would build and executed in MCU RAM. It acts like a standard application, operate block oriented and could handle "verify on the fly" (immediately compare Flash content with internal block buffer after buffer is programmed). It would be possible - if MCU have

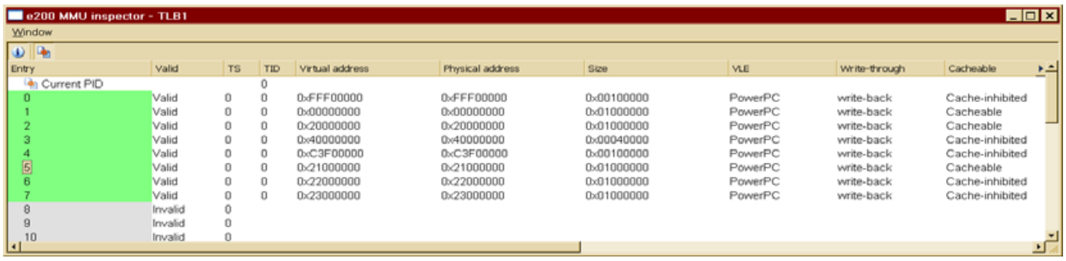
enough RAM for it - that this small program handles everything, including speed up MCU (max. SysClk), initialize Bus system, set correct amount of wait-states and so on. The "Flash Monitor" will be loaded from the Debugger into the RAM of target MCU and then Debugger starts its execution. With certain state variables there is a handshake and communication so Debugger will send the Data for target external flash in blocks.

The "Flash Monitor" speeds up the whole process, but there are some drawbacks. (Today it is not existing for PHYTEC kit, sorry)

- It must be build for **every** external flash device and it has to consider different kind of connection (byte, word, 1, 2 or more devices) from the external Flash devices to the MCU Bus. This must repetitive for every MCU - and there are different MCU's with different size of internal RAM (not always enough for a good Job!).
- In case of NAND Flash devices it gets additional target specific and more complicated. There is reason for customers using in most cases a RTOS to deal with usual "SD" Storage Flash Cards and their complex File System.
- This in total is a lot of effort and Tool vendors could not do it "by the way" without extra charge, not at once and immediately "perfect". He would need a target, best way is a loan of customers target to develop, build and improve that "Flash Monitor".
- Because iSYSTEM AG has more than 27 years experience in this quick market they provide a free-of-charge "do it yourself" solution for experienced developers. If this not fit the expectations iSYSTEM could do all that tasks above for extra charge NRE.
- If you performed a "Standard installation" like suggested you could take a look into "UMI SDK" (universal monitor interface) of winIDEA. You will find documentation and briefly examples here: "c:\iSYSTEM\winIDEA\2012\sdk\iSYSTEM.UMI.SDK\Documentation\UMI.pdf"
- Like personal experiences - of course not transferable without restrictions - user could expect about 5 times faster flash process for external Flash, maybe better.

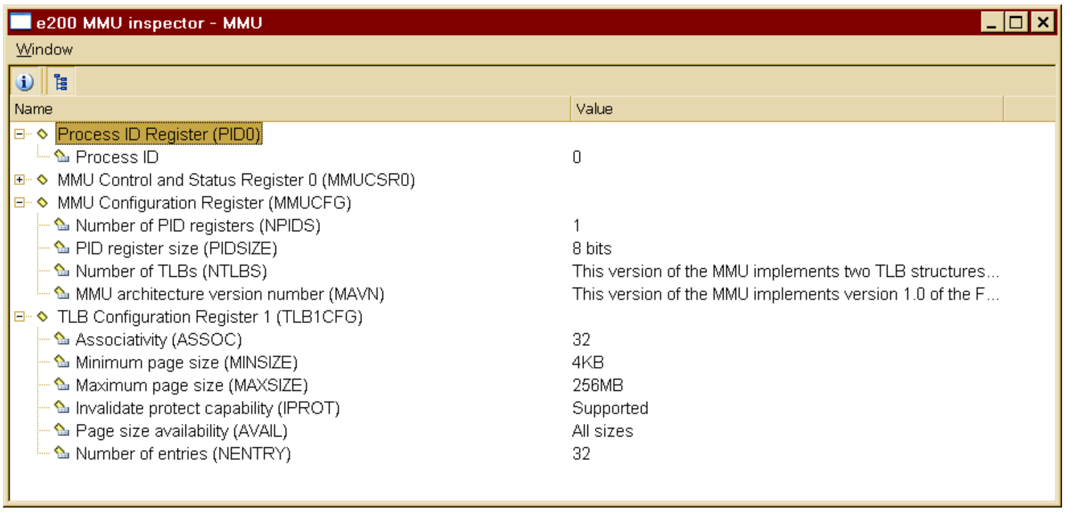
### 2.5.3 Plug-ins for winIDEA.

- winIDEA provide a plug-in Interface and you are able to write additional own plug-ins (compare again the "SDK" Section of your winIDEA installation directory)
- For the MPC5554 exist possibilities to see more MCU internals. This is i.e. important for dealing with MCU Caches.
- Some of plug-ins are for your comfort, like "run duration" (this is not comparable with measurement with the precise NEXUS trace, only a simple counter for time!). You could add own plug-ins - see \SDK.



| Entry | Valid | TS | TID | Virtual address | Physical address | Size | VLE | Write-through | Cacheable |
|---|---|---|---|---|---|---|---|---|---|
| Current PID | | | 0 | | | | | | |
| 0 | Valid | 0 | 0 | 0xFFF00000 | 0xFFF00000 | 0x00100000 | PowerPC | write-back | Cache-inhibited |
| 1 | Valid | 0 | 0 | 0x00000000 | 0x00000000 | 0x01000000 | PowerPC | write-back | Cacheable |
| 2 | Valid | 0 | 0 | 0x20000000 | 0x20000000 | 0x01000000 | PowerPC | write-back | Cacheable |
| 3 | Valid | 0 | 0 | 0x40000000 | 0x40000000 | 0x00040000 | PowerPC | write-back | Cache-inhibited |
| 4 | Valid | 0 | 0 | 0xC3F00000 | 0xC3F00000 | 0x00100000 | PowerPC | write-back | Cache-inhibited |
| 5 | Valid | 0 | 0 | 0x21000000 | 0x21000000 | 0x01000000 | PowerPC | write-back | Cacheable |
| 6 | Valid | 0 | 0 | 0x22000000 | 0x22000000 | 0x01000000 | PowerPC | write-back | Cache-inhibited |
| 7 | Valid | 0 | 0 | 0x23000000 | 0x23000000 | 0x01000000 | PowerPC | write-back | Cache-inhibited |
| 8 | Invalid | 0 | | | | | | | |
| 9 | Invalid | 0 | | | | | | | |
| 10 | Invalid | 0 | | | | | | | |

This is the "TLB" table



| Name | Value |
|---|---|
| Process ID Register (PID0) | |
| Process ID | 0 |
| MMU Control and Status Register 0 (MMUCSR0) | |
| MMU Configuration Register (MMUCFG) | |
| Number of PID registers (NPIDS) | 1 |
| PID register size (PIDSIZE) | 8 bits |
| Number of TLBs (NTLBS) | This version of the MMU implements two TLB structures... |
| MMU architecture version number (MAVN) | This version of the MMU implements version 1.0 of the F... |
| TLB Configuration Register 1 (TLB1CFG) | |
| Associativity (ASSOC) | 32 |
| Minimum page size (MINSIZE) | 4KB |
| Maximum page size (MAXSIZE) | 256MB |
| Invalidate protect capability (IPROT) | Supported |
| Page size availability (AVAIL) | All sizes |
| Number of entries (NENTRY) | 32 |

- This is MMU Inspector plug-in. You find a "table walk" within menu ***winIDEA - Debug***.

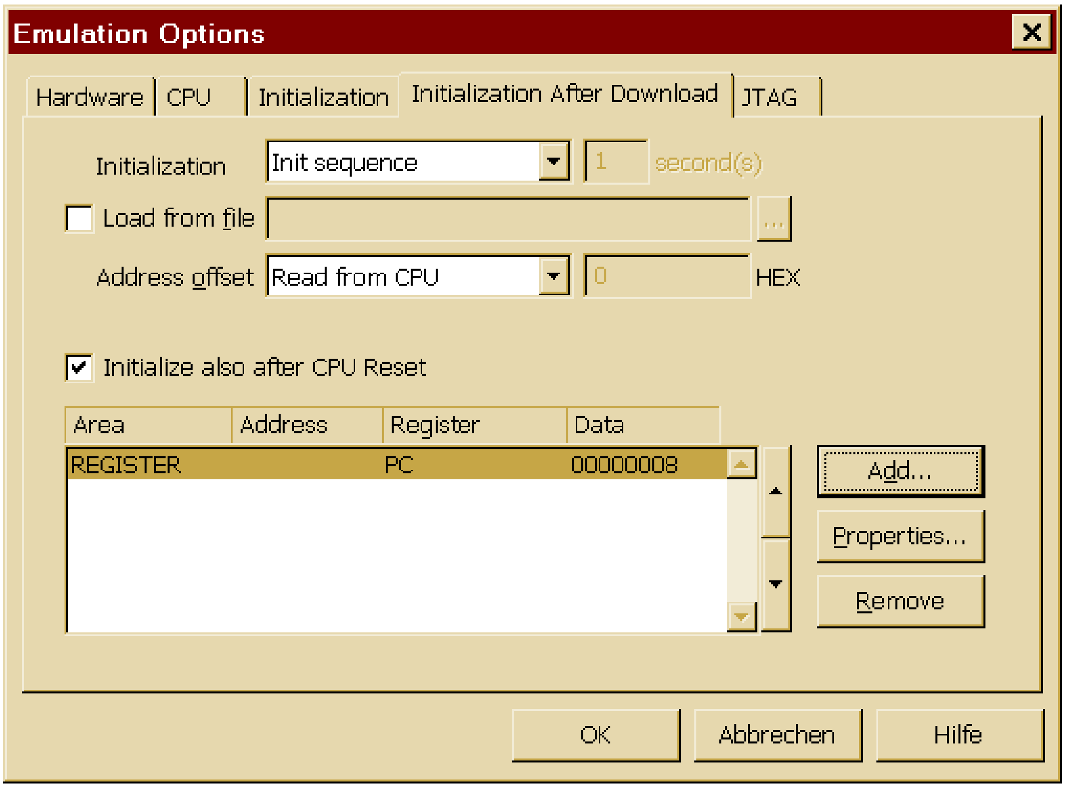
### 2.5.4 Consider your new, own applications.

- As you learned in the past chapters and the hardware manual you have to pay attention for some prerequisites if you build an own working application. We will give you a quick overview to important items to avoid usual, MCU related pitfalls especially if you come from another MCU, i.e. older one or 8 bit. This is a common overview, not special for the MPC55xx only and not detailed!

- eTPU. iSYSTEM Tools could support both eTPU's of MPC5554 for debugging of Assembler and C (with iC5000 Trace possible too!). For building Code for eTPU you have to use and buy a special C-Compiler (Ashware/Bytecraft). In case of questions please send an eMail to holger.wild@isystem.com.
- First of all the "ancient" MCU's were "ready to work" after power-on and cold-start reset. Today MCU are equipped with ECC RAM. Before using it you have to write to every address to set the ECC bit correctly or you will get an error (Job of "ctrs.s", but check and be aware of it).
- Next are you have to deal correctly with MMU (and maybe MPU) because some MCU's has only partly access to the Flash (i.e. 4 Kbyte instead of 2 MByte after Reset). This should be again covered by "ctrs.s" code, but not always comprehensive because these startup files are usually universal. Please check the Freescale Application Notes, i.e. AN2865 and for better understanding AN4239.
- Note that you have not only ramp-up the PLL from Crystal Clock to reach maximum MCU performance. You have to enable the MCU Caches for Code-Execution and/or Data access.
- Don't expect that peripherals (like CAN) are ready to use. Because of lowering the Power consumption nearly all latest MCU's don't power this part nor connect SysClock to it after Reset. Your applications have to take care of it.
- Especially Debugger gets in trouble just with item before. If you imagine that you work on Monday, initialize the SFR i.e. for CAN and leave the SFR Window open. You open winIDEA again on Tuesday and after download the Disassembly window point to "se_illegal" command. Single step fails in that case. What's happened? Because your application is not started yet the CAN is

nor powered neither clocked. But because SFR Window is left open, winIDEA have to readout this and the MCU quite immediately with trap and point to "se_illegal" instruction. To avoid that scenario you should close this kind of windows before next attempt, repeat the download, start application and re-open the Window later. Alternatively you may write/extend an INI File that attaches the CAN on the internal power and clock before download. Then everything works fine. In final application you don't use Debugger and your program starts immediately. Then everything runs fine without Code change. There are other possibilities to reach that goal (Python Script, Reset & Run for certain time before stopping application…) but you should aware of those issues.

- Testing application without Bootloader - example. As you learned in chapters before you may setup execution point within INI file. Alternatively PYTHON Script would to the Job too - automate the whole process: Download, set maybe break, patch variables automatically for test purpose, execute from different address…. If you have no Bootloader ready but need to write Code for it you may use winIDEA for change the Entry point - i. e. execute from different location than MCU usually do. For that you may set a Register PC (and/or other Registers/SFR's) within "Initialization after Download". In the example below Application starts always from 0x00000008.



Initialization: Init sequence | 1 second(s)

☐ Load from file

Address offset: Read from CPU | 0 HEX

☑ Initialize also after CPU Reset

| Area | Address | Register | Data |
|---|---|---|---|
| REGISTER | | PC | 00000008 |

Document: phyCORE-MPC5554 QuickStart Instructions
Document number: L-695e_2, January 2013

**How would you improve this manual?**

______________________________________________

______________________________________________

______________________________________________

**Did you find any mistakes in this manual?** Page

______________________________________________

______________________________________________

______________________________________________

**Submitted by:**

Customer number: ______________________________

Name: ______________________________

Company: ______________________________

Address: ______________________________

______________________________

**Return to:**

PHYTEC Technologie Holding AG
Postfach 100403
D-55135 Mainz, Germany
Fax : +49 (6131) 9221-26