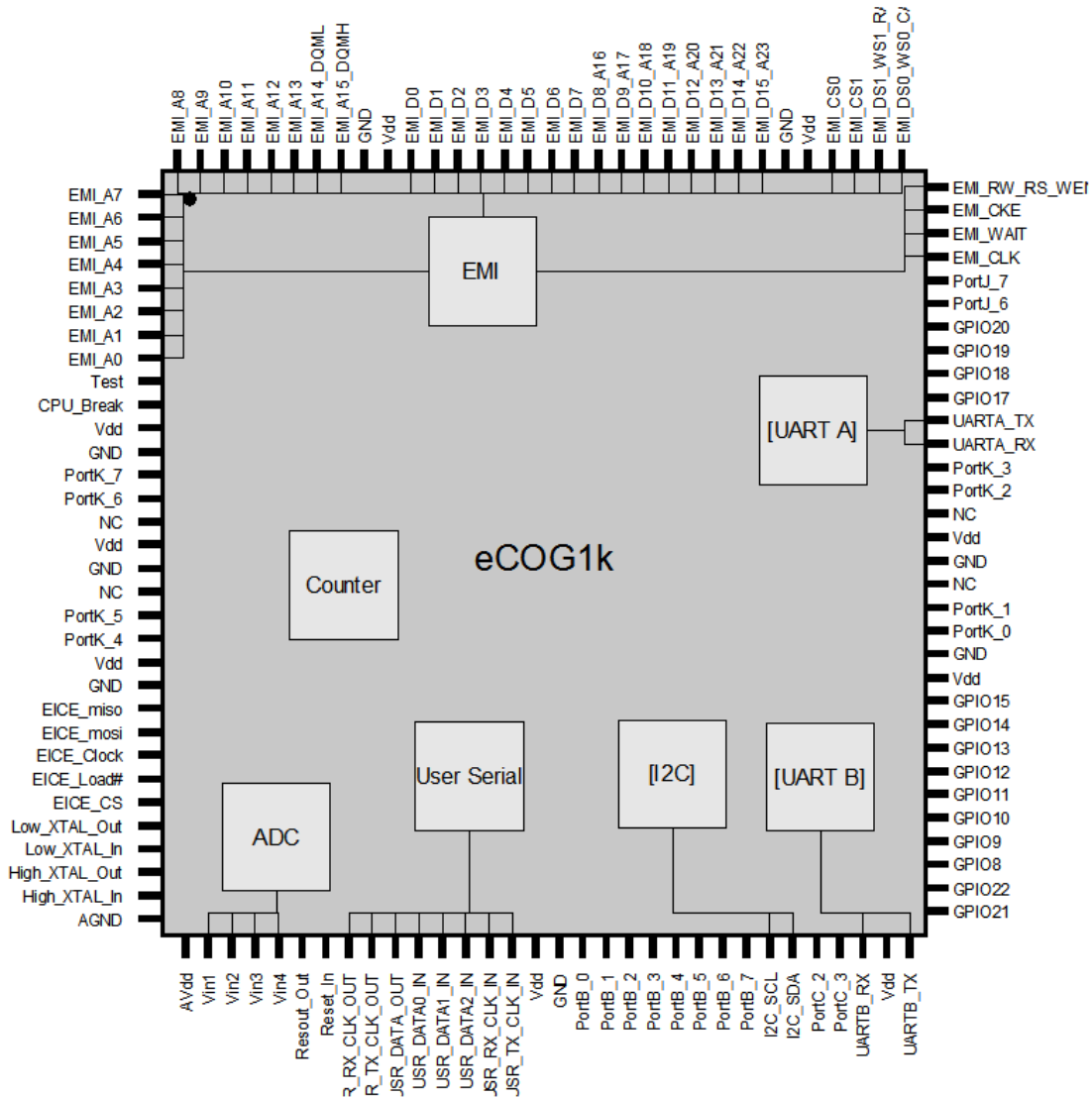




AN040 – Programming External Memory in CyanIDE

Version 1.1

This application note describes how CyanIDE can be extended to support programming of external memory.



Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



Revision History

Version	Date	Notes
V1.0	30/05/2006	First Release
V1.1	03/10/2007	Updated for CyanIDE V1.4.2

Contents

1	Introduction	5
2	Glossary	5
3	Introduction to How CyanIDE Programs Memory	5
4	Installing and Using the Included Extensions	6
5	Software Operation.....	7
6	Customising the Programming Code for Other Devices	9
6.1	JEDEC Standard for Single Supply Flash Devices	9
6.2	Parts Not Software-Compatible with the Above Standard.....	9
7	Programming External SDRAM	10

1 Introduction

This application note describes how CyanIDE can be extended to support programming of external memory, for example the Spansion flash chip on the eCOG1k development board issue 2.1 and the SDRAM on all versions of this board. Source code is provided for these parts.

In addition, this document also discusses how to modify these examples to support other memory devices.

2 Glossary

A table of abbreviations used in this document.

eCOG1	Cyan Technology target micro controller
eICE	Embedded In Circuit Emulator (eICE)
SRAM	Static Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory

3 Introduction to How CyanIDE Programs Memory

CyanIDE can directly read from and write to the internal SRAM of the eCOG1 via the eICE interface. To program other types of memory, a small programming utility is downloaded into the SRAM and then CyanIDE communicates with this program by modifying variables in memory via eICE.

Commands and data are passed to the downloaded programming utility, and then this utility in SRAM handles the actual programming of the external memory device. CyanIDE is supplied as standard with a programming utility that supports this function for the internal flash memory.

4 Installing and Using the Included Extensions

To install the programming extensions included in the accompanying software, unzip the archive file *AN040SW.zip* to the CyanIDE projects directory, then copy the unzipped flash directory into your CyanIDE installation directory. When Windows asks if it should overwrite any existing files, select *Yes* or *OK*.

CyanIDE generates multiple object files, one for each different physical memory area that contains code and/or constant data, according to the information given in the map file. Once the programming extensions are installed, CyanIDE automatically downloads and programs the complete set of object files for both internal and external memory.

The individual programming extensions provide the device-specific functions for writing and programming data into the target memory, for example the erase and write algorithms required for flash memory. The examples in the zip file *AN040SW.zip* provide programming extensions suitable for the external flash memory and SDRAM fitted to the eCOG1k development board. Source code is provided to allow users to modify these programming utilities in order to support different memory devices.

The table below summarises the files provided in the zip file.

File	Function
flash\IROM1K.ram flash\IROM1K.sym	Programming software image and symbol file for internal flash.
flash\EM0Prog.ram flash\EM0Prog.sym	Programming software image and symbol file for external memory attached to chip select CS0. Version of this file included with application note for SDRAM fitted to development boards issue 1.2 and 2.1.
flash\EM1Prog.ram flash\EM1Prog.sym	Programming software image and symbol file for external memory attached to chip select CS1. Version of this file included with application note for flash memory fitted to development board issue 2.1.
Source\IROM1K	CyanIDE project with source code for programming eCOG1k internal flash memory.
Source\EM0RamProg	CyanIDE project with source code for programming external SDRAM on chip select CS0.
Source\EM1FlashProg	CyanIDE project with source code for programming external flash memory on chip select CS1. Supports the Spansion S29AL008D device.

Table 1. CyanIDE programming utility files

5 Software Operation

In this section, the programming utility for the Spansion S29AL008D flash memory device fitted to the development board issue 2.1 is used as an example to describe the operation of the external memory programmers. This is located in `<source\EM1FlashProg>` in the zip file `AN040SW.zip` accompanying this document.

The file `flash.c` contains the interface functionality and generic code that is suitable for most types of flash memory. This makes use of functions in `FlashProg.c` to perform the device-specific low-level write operations. The code is arranged in this way so that in order to modify it to work with a different flash device, it is only necessary to modify `FlashProg.c`.

Once CyanIDE has downloaded the programming software to the internal RAM on the target device, it starts execution and communicates with the code by examining and modifying two global variables via the eICE interface. These variables are defined as follows:

```
// Input and output variables, accesses using eICE
unsigned int command_response;
unsigned int command_data[DATA_BLOCK_SIZE + 2];
```

The variable `command_response` is used by CyanIDE to send commands to be executed, and by the programming software to return response codes. The array `command_data` is used to pass in any necessary arguments for the command specified in `command_response`.

The command codes, responses and interpretation of data are summarised below.

Constant Name	Value	Description
<i>Command codes, written via eICE</i>		
CMD_PROGRAM_DATA	0x8001	This command is used to program 512 bytes of memory. The first word in <code>command_data</code> is the start address, the next 256 words are the data to be programmed and the final word is a CRC checksum.
CMD_ERASE_SECTOR	0x8002	Requests a sector erase function. The sector to be erased is specified by the 16 most significant bits of its 24-bit address, relative to the base of the flash device. This is passed in the first word of <code>command_data</code> contains.
CMD_ERASE_ALL	0x8003	Requests a device erase function.
<i>Response codes, read via eICE</i>		
RESP_RESET	0x0000	Initial value of <code>command_response</code> before any commands are executed.
RESP_OK	0x0001	The command was successful.
RESP_ERR	0x0002	The command was not successful.
RESP_INVALID_DATA	0x0003	The parameter(s) specified in <code>command_data</code> were not valid for the function called. For example, the address specified for the <code>CMD_PROGRAM_DATA</code> function was beyond the end of the flash device.
RESP_INVALID_COMMAND	0x0004	The command is not recognised.
RESP_CRC_FAIL	0x0005	Returned for <code>CMD_PROGRAM_DATA</code> when the calculated CRC value for the block of data in <code>command_data</code> does not match the value sent with the data.

Table 2. Programming utility command and response codes

As the programmer code is written to run from only internal RAM, it does not include the default configuration file but instead performs its own initial hardware configuration from the `main()` function. The main function loops forever, checking for a command in the variable `command_response`, then calls the command interpreter to handle the requested operation.

Commands are distinguished from responses by the most significant bit being set. When the requested operation is completed, `command_response` is set to indicate the outcome of the operation and execution returns to the command interpreter loop in the main function to wait for the next command.

6 Customising the Programming Code for Other Devices

The file *FlashProg.c* contains code that generates the actual sequence of command words required to perform device-specific flash erase and programming operations. The version included with this application note has been specifically written to work with the Spansion S29AL008D fitted to the eCOG1k development board issue 2.x.

6.1 JEDEC Standard for Single Supply Flash Devices

The Spansion S29AL008D is fully compatible with the JEDEC standard for single supply flash memory devices, so it should be possible to program other parts based on this standard simply by modifying the device sector information at the start of *FlashProg.c*:

```
#define NUMBER_OF_SECTORS 19

Segment_Map_t const Segment_Map[NUMBER_OF_SECTORS] =
{
    { 0x000000, 0x001FFF },
    { 0x002000, 0x002FFF },
    { 0x003000, 0x003FFF },
    { 0x004000, 0x007FFF },
    ...
    { 0x078000, 0x07FFFF }
};
```

This Spansion device has 19 sectors, other devices may have more or fewer sectors, in which case the above `#define` statement needs to be modified.

The `Segment_Map` array, shown in part, defines the start and end address of each sector. This needs to be updated for different size devices and may require changes for devices of the same size with different sector maps.

Many devices that are based on the above standard have been described as “AMD compatible”, as AMD was the best-known vendor producing devices based on this standard. Spansion is the new brand name for devices made by the AMD/Fujitsu joint venture flash memory manufacturer.

6.2 Parts Not Software-Compatible with the Above Standard

While the JEDEC standard for single supply flash chips is probably the most widely adopted standard for NOR flash chips, many flash chips are not based on it.

In addition, some vendors have produced chips that, while fully compatible with the JEDEC single supply flash pinouts, do not follow the same programming sequences. These parts require more work to use them with this code.

7 Programming External SDRAM

SDRAM can be read from and written to in the same way as internal SRAM. However, before performing any accesses, both the SDRAM chip and the eCOG1k SDRAM controller need to be correctly initialised to reflect the memory size, bus width, bank layout, CAS latency etc. for the specific device(s) used.

This initialisation must be performed by the downloader in the same way as for flash. Example code to do this for the parts fitted to our development board is included in the zip file *AN040SW.zip*. This code is based on the flash programmer but is significantly simpler as it is no longer necessary to perform operations such as sector erase or to use special sequences to program data.

Both the issue 1.2 and issue 2.1 Development Boards are fitted with a 16MByte external SDRAM chip, typically one manufactured by Micron or Samsung. The example SDRAM programmer should work with other parts that are the same size and have the same number of rows, columns, banks etc. as these parts. To support other parts, the code requires further modification. The source code is located in the accompanying zip file in the subdirectory *\source\EMORamProg*.

CyanIDE resets the target device after downloading code. This causes the eCOG1k SDRAM controller to cease issuing refresh commands to the SDRAM and may cause corruption of the data downloaded to the SDRAM. For this reason, it is necessary to ensure that the SDRAM is in its self-refresh mode after each command executed in the downloader.