eurek
ELECTRONIC ENGINEERING

IL TUO PARTNER TECNOLOGICO

touchMe SERIES

PICCOLOTOUCH

PiccoloTouch®EK315

USER MANUAL

PiccoloTouch®, it takes a finger, so simple.

Il nostro Sistema Qualità ha ricevuto la certificazione di conformità a norme ISO 9002 dal 1998. Da Gennaio 2010 siamo passatti alla ISO 9001:2008.

Our Quality System was granted the certification of conformity with ISO 9002 standards as September 1998. From January 2010 we have switched over to ISO 9001:2008.

assolutamente **made in italy.**

PiccoloTouch®**EK315**

PiccoloTouch®, it takes a finger, so simple.

Software Developement System in a Linux Environment for Embedded Boards EK315 - Piccolo Touch

Linux Edelin

This documents is referring to a VirttualBox Working Image provided with the EK315 - Piccolo Touch board Board Support Package.

Embedded Linux EK315 - Piccolo Touch
Board Support Package

This document is to be intended as a brief explanation set of the most common descriptions of the available tools and software provided *as is* in the *VirtualBox Image Environment*, just to program/flash the board and to be operatively ready for writing your own firmware.

Technical Specification of board EK340 - Piccolo Touch

PiccoloTouch
Dimensions • 75 mm x 96 mm
Power • 9…12V 300mA via power connector or 5V using USB cable
Display • 3.2” color LCD TFT with LED backlight
Resolution • 240×320 262k colors
Touch Screen • Resistive, 4 wires
Processor • STM32 ARM CORTEX M3
Clock • 72 MHz
Internal Flash Memory • 512kB

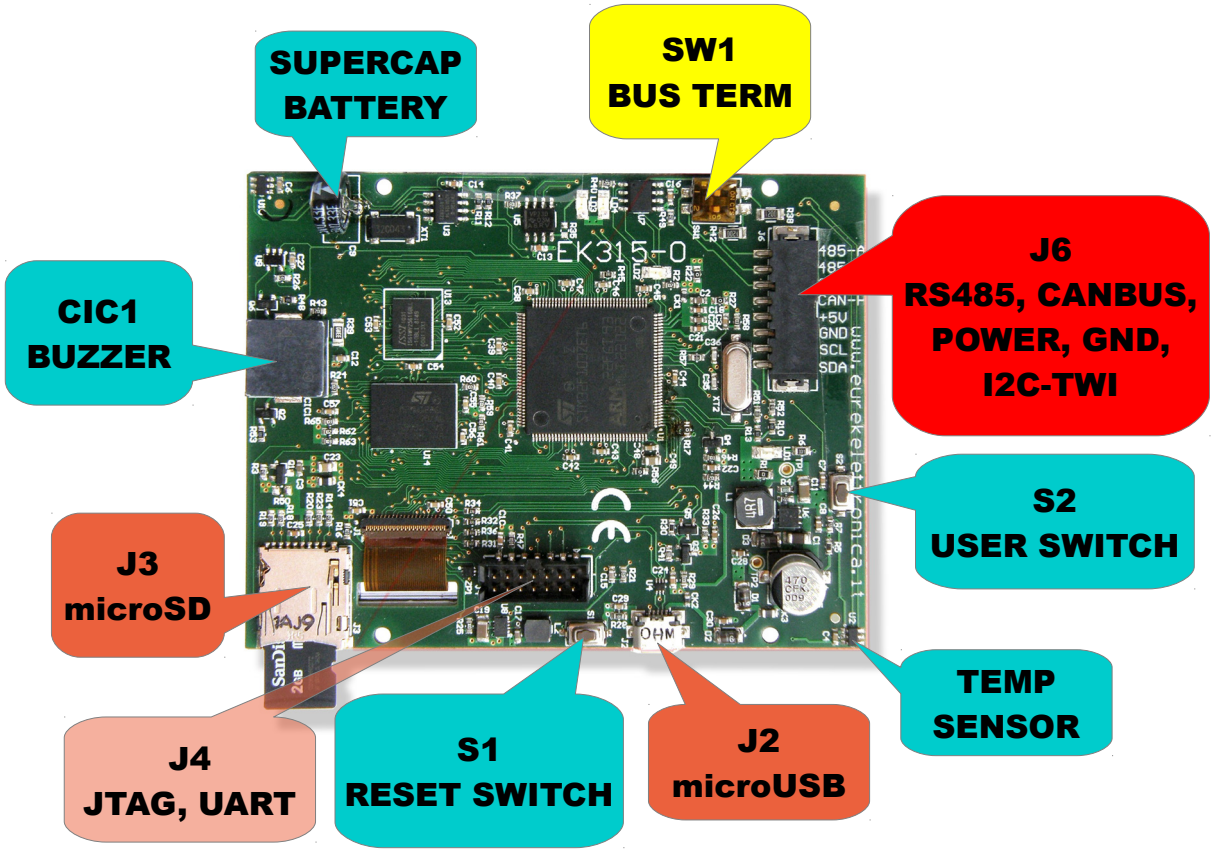
PiccoloTouch
External Flash Memory • 16 MB
Ram Memory • 64 kB (internal) • 512 kB (external)
Ferromagnetic EEPROM • 8k
Temperature Sensor • LM73CIMK-1
Real Time Clock • Internal with SUPERCAP backup
Jtag Port • Available as custom connector
Interfaces • CAN 2.0B (with selectable terminator) • I2C (3.3V) • RS485 (with selectable terminator) • USB (microUSB connector) • microSD
Connector • PHOENIX CONTACT PTSM 8-pin
Available Software • FreeRTOS • EcceGUI Library

PiccoloTouch
File System • FATfs on microSD
Boot Time • Less than 1 second
Sound • Buzzer (for TOUCH Events)
Developer tools • IDE Eclipse with GCC Compiler and OCD Debugger • VirtualBox BSP Image (can be used in Linux, MacOS X or Windows Operating Systems and possibly others)

Connectors and indicators available on board EK315 - Piccolo Touch

On the board there are those connectors/indicators:

- **Master I/O and Power Supply Connector J6** (serial RS485, CANBus 2.0B, I2C Two-wires Interface and PowerSupply pins)
- **Custom Key** SMD
- **System Reset Key** SMD
- **Power Red Led** SMD
- **User Red Led** SMD
- **microUSB** device type AB
- **microSD Connector J3**
- **LCD Connector J1**
- **Bus Terminator Selector SW1** (CANBus / RS485)



Pin Out Connectors

- **Connector J6**

1	RS485 - A
2	RS485 - B
3	CAN - L
4	CAN - H
5	+Vcc (5..12V)
6	GND
7	I2C - TWI SCL
8	I2C - TWI SDA

JTAG Connector J 4

1	nTRST
2	Not Connected
3	TDI
4	nRESET
5	TMS
6	GND
7	TDO
8	Not Connected
9	TCK
10	GND
11	GND
12	UART RX
13	+3.3V
14	UART TX

External Memory

The external flash memory is used mainly as graphic storage for widgets and screens of the GUI.

MicroSD Support

Any microSD Card inserted into the connector can be used safely using the FatFS API access. It has to be formatted with ***FAT32 FileSystem*** type. Up to 16Gb can be used. *Other sizes could be used but are not tested.*

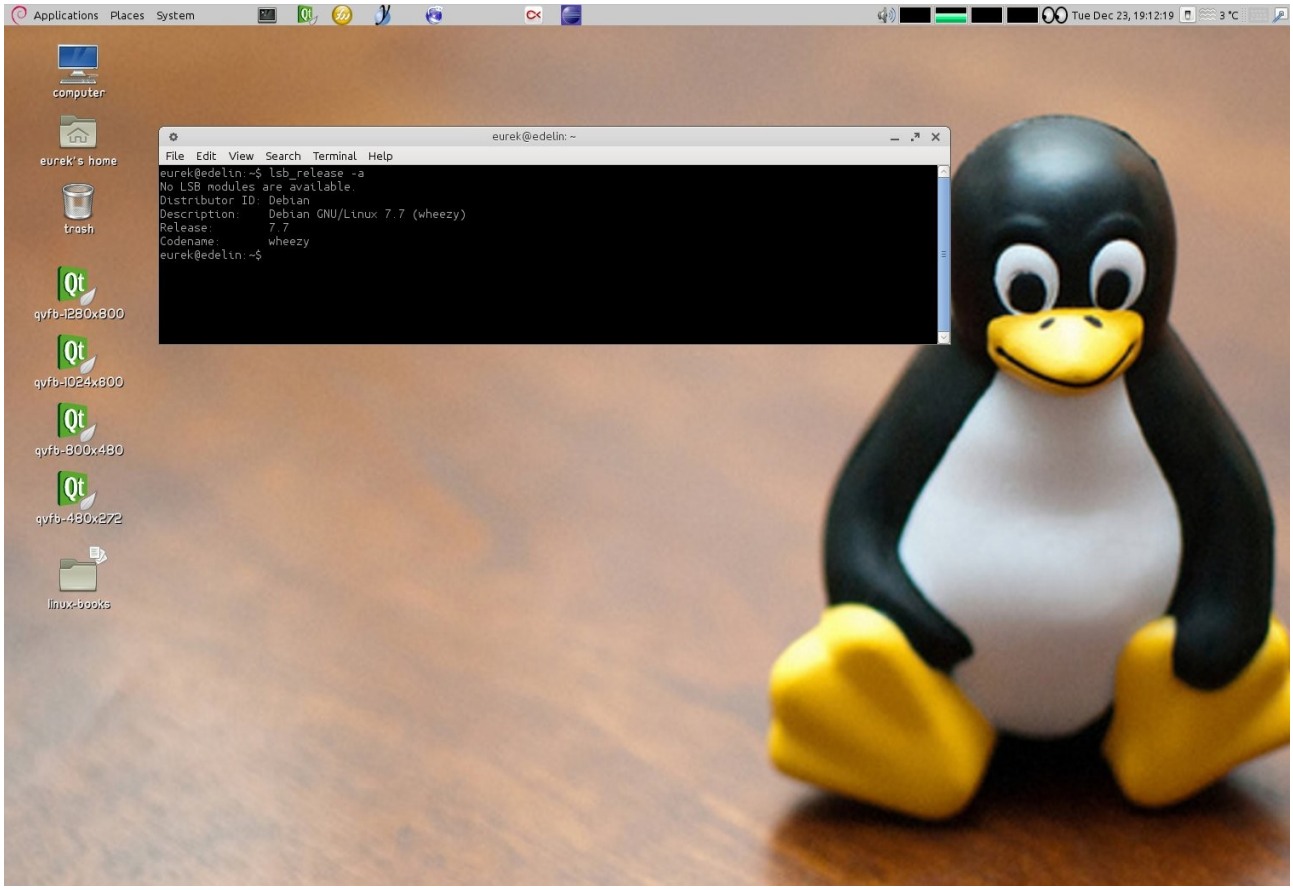
Board Support Package (BSP)

Along with the board **EK315 - Piccolo Touch** a VirtualBox Virtual Machine Image is provided: in this way all software is already included within and no other software is to be installed in your PC. Simply installing the latest VirtualBox software (https://www.virtualbox.org/wiki/Downloads) in your PC and importing this VirtualBox Image and in few moments you will be ready to program you application.

Minimum System Requirements:
CPU : x86 Pentium 4 Dual Core / Quad Core / Eight Core
RAM : 2 GB or more (4 Gb recommended)
VIDEO : 1280x1024 24bit or higher
INTERNET : FLAT Internet Connection
DISCO : at least 100 GigaBytes of free space

The Linux distro is a personalized version of **Debian Wheezy 7.** It offers high configuration flexibility and a very big software repositories for any need.

Login:
user name: *eurek*
password: *eurek*



PREPARING THE VIRTUAL MACHINE

Simply go to the **VirtualBox** site (or if you prefer you can use **VMWare** too, but this tutorial will cover only the first one):

https://www.virtualbox.org/wiki/Downloads

Download and install the latest version you can find (we are currently using the **4.3.26**) or use your system packaging tools to install it in the correct way for your Operating .

To import a virtual machine, you need to start VirtualBox. On the host where you installed Oracle VDI and VirtualBox, on the desktop select the **Applications** menu, then the **System Tools** menu, and then **Oracle VM VirtualBox**. Alternatively, you can run the **VirtualBox** command in a terminal. The Oracle VM VirtualBox Manager is displayed, as shown:

Oracle VM VirtualBox Manager

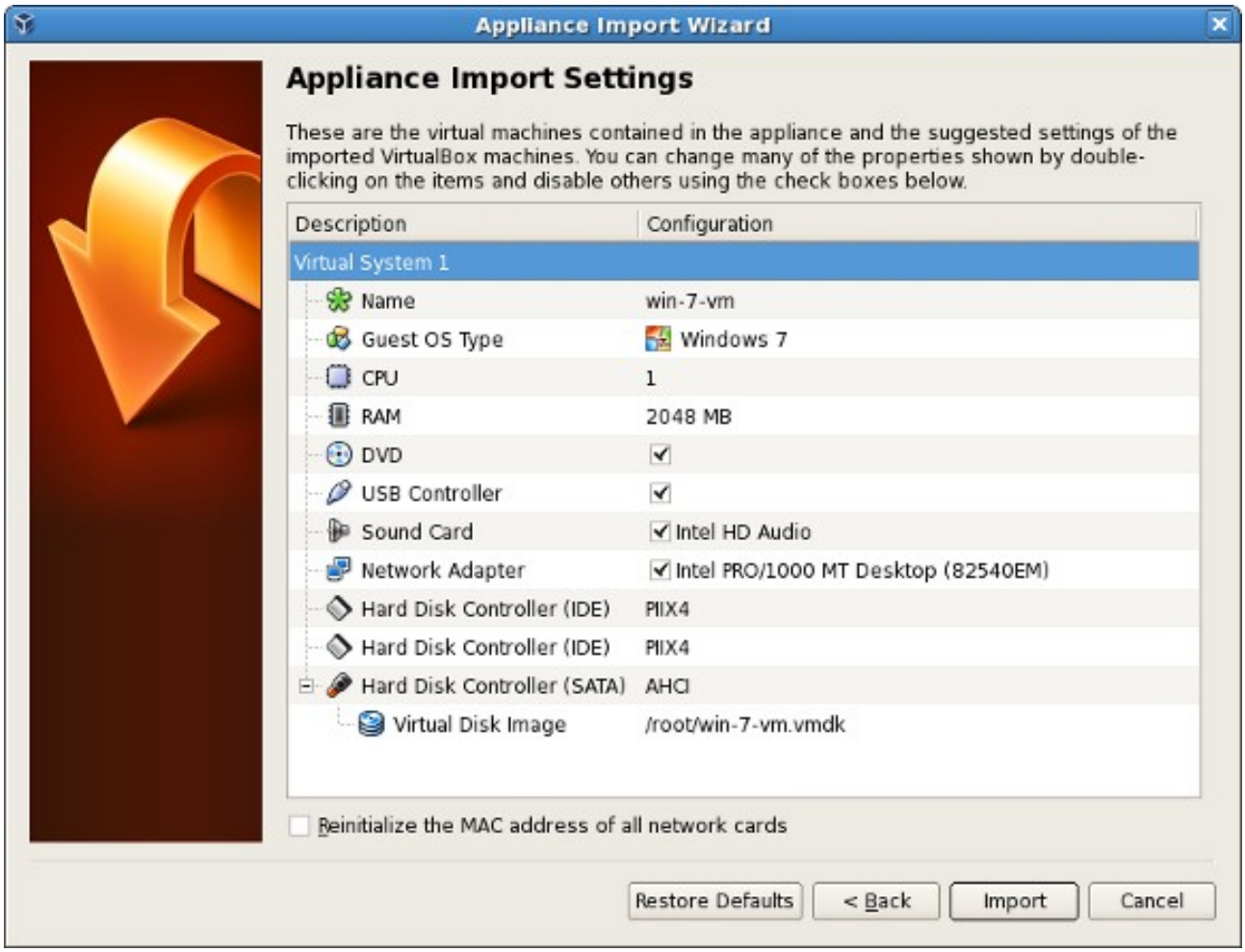
In the **File** menu, select **Import Appliance**. The Appliance Import wizard is displayed in a new window, as shown:

Appliance Import Wizard

Click **Choose**, browse to the **USB Stick** location containing the virtual image file provided with the the purchasing of the board-kit **Piccolo Touch EK315** and select it.

This will be the virtual machine you want to import, and click **Open**. The Appliance Import Settings step is displayed as shown:

Appliance Import Settings



Make any adjustments you want to the displayed settings (you can also change the settings later) and click **Import**. The Appliance Import Wizard is closed and after a few moments, the imported virtual machine is listed in Oracle VM VirtualBox Manager.

After the import, select the imported virtual machine and in the toolbar click the **Settings** button. Review the virtual machine settings to make sure that the virtual machine has the hardware it needs to operate. Make sure that the virtual machine has a CD/DVD drive.

Once you have reviewed the settings, select the imported virtual machine and in the toolbar click the **Start** button. Verify that the virtual machine works.

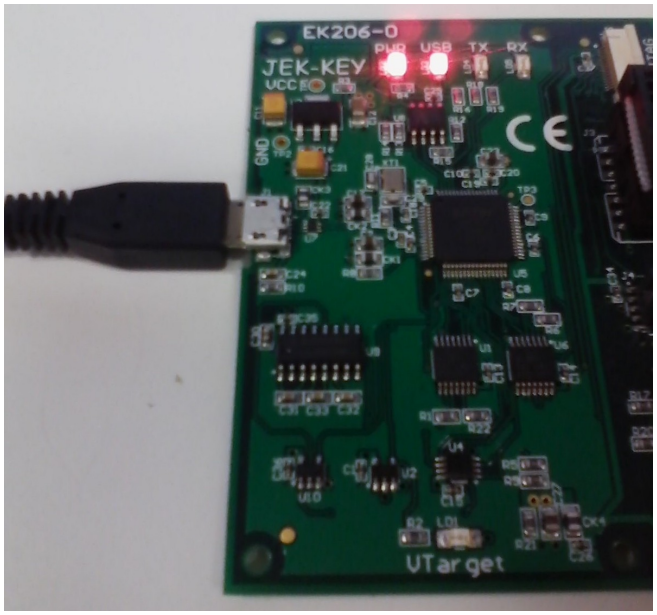
For your best performance please add as many processors you can give to the Virtual Machine and install the maximum RAM available to the Virtual Machine to achieve the best tradeoff between speed and power.

Now you can start to play with the virtual machine Linux Edelin BSP!

HOW TO CONNECT TO THE DEBUG PORT ON EMBEDDED SYSTEM EK315 - PICCOLO TOUCH

Step 1: Connect the JTAG Cable to the board on at the **connector J4** as shown:

Step 2: Connect the microUSB cable to the microUSB port on the **JTAG Key EK206**



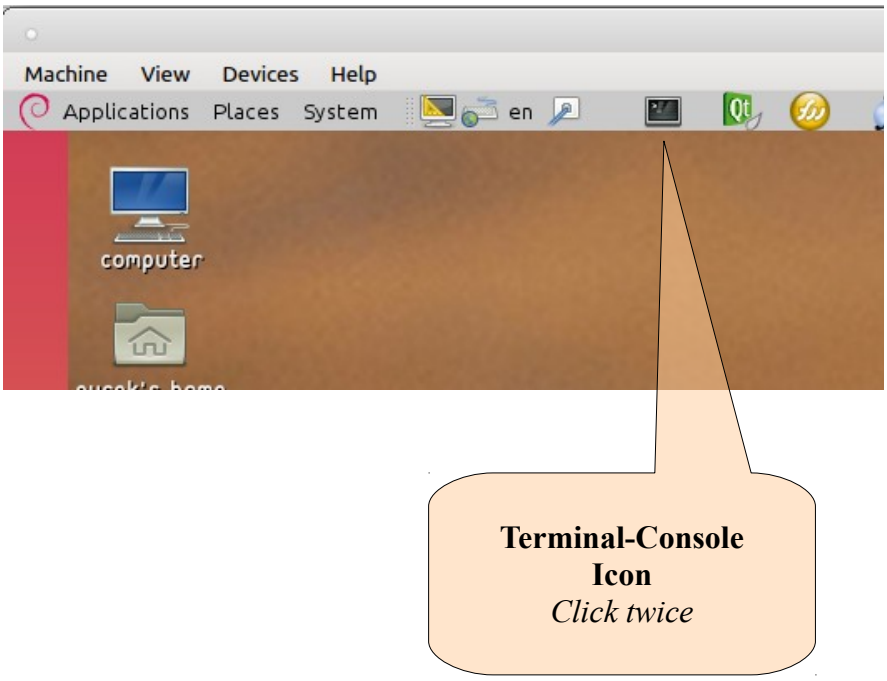
Step 3: Connect the USB Host Type End to the PC

Step 4: Using the **Devices** dialog into the VirtualBox Menu, select to connect the:

USB Devices --> Eurek srl Amontec JTAGKey [700]

in this way the USB port is passed thru your PC to the VirtualBox System

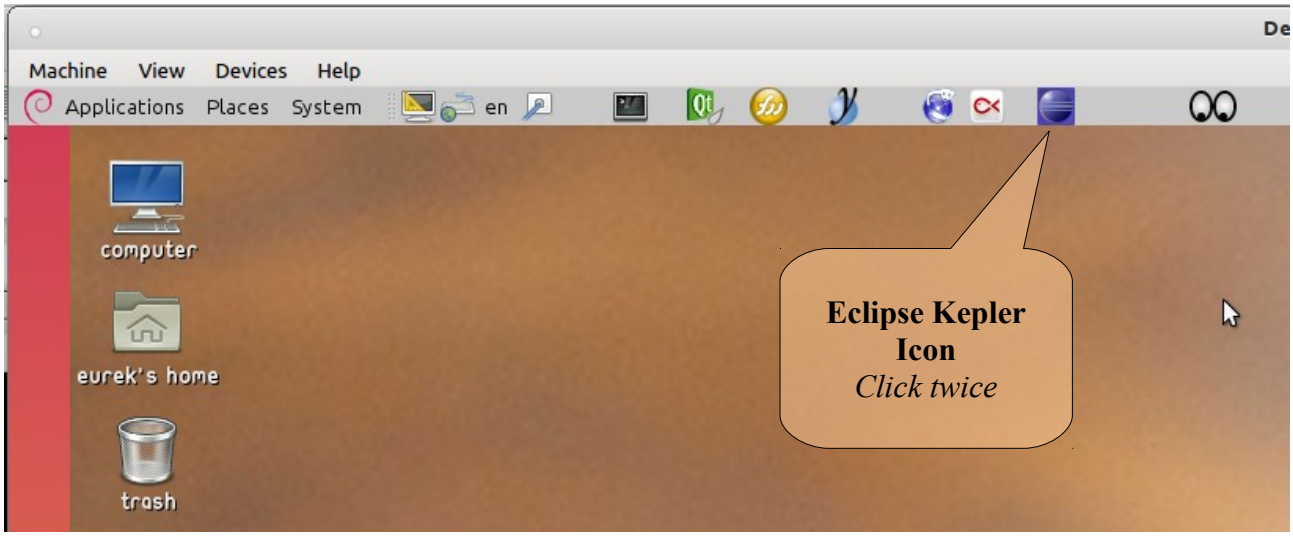
Step 5: Simply open a console/terminal and write: **minicom ttyJEK0-115200** and you will be connected to the debug port on the embedded board. Using all ***minicom*** shortcuts and commands you will be able to save your log and so forth...



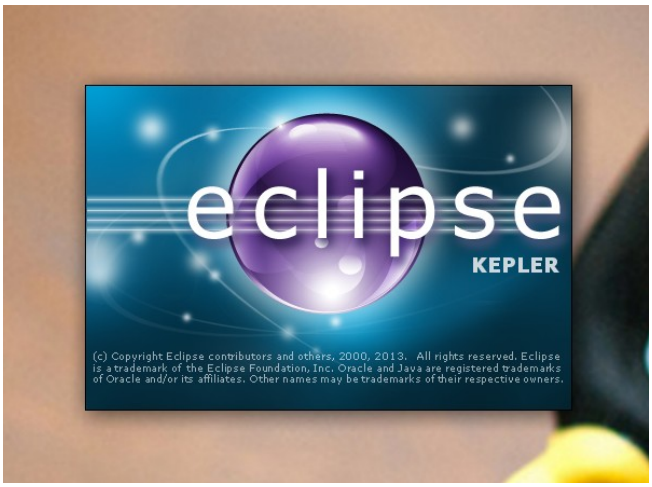
BUILDING AN EXAMPLE APPLICATION

In this section we will teach you how to open an **Eclipse ®** Project and how to build an example application on PC and eventually flash the same application into the embedded board to test its capabilities.

First of all select the **Eclipse Kepler** icon to startup the **Integrated Development Environment IDE:**

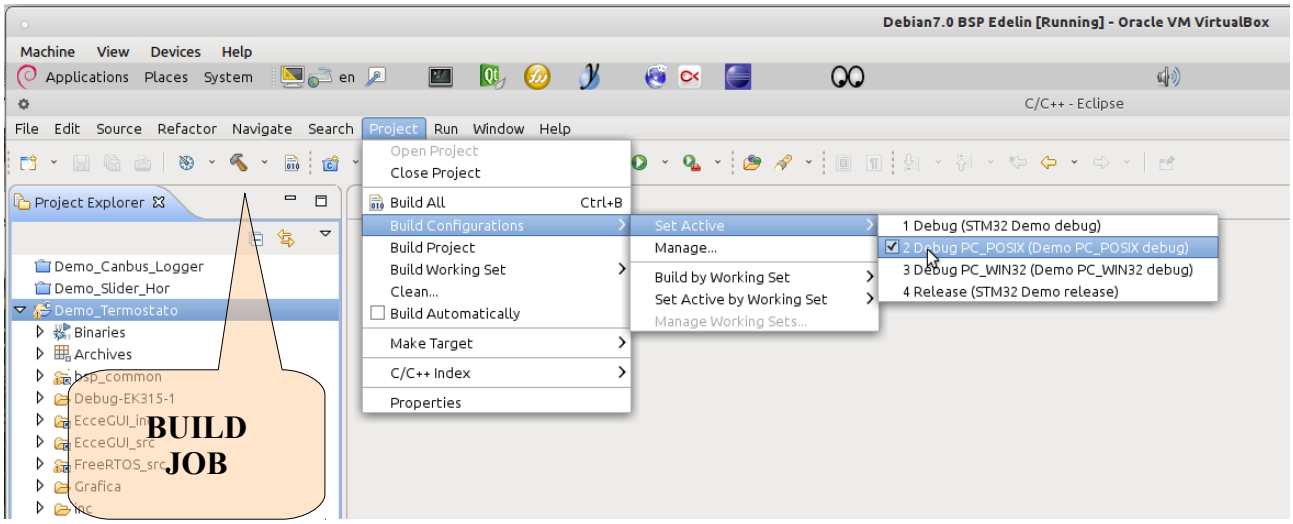


After few seconds it will appear a so-called *splash screen* and a dialog:



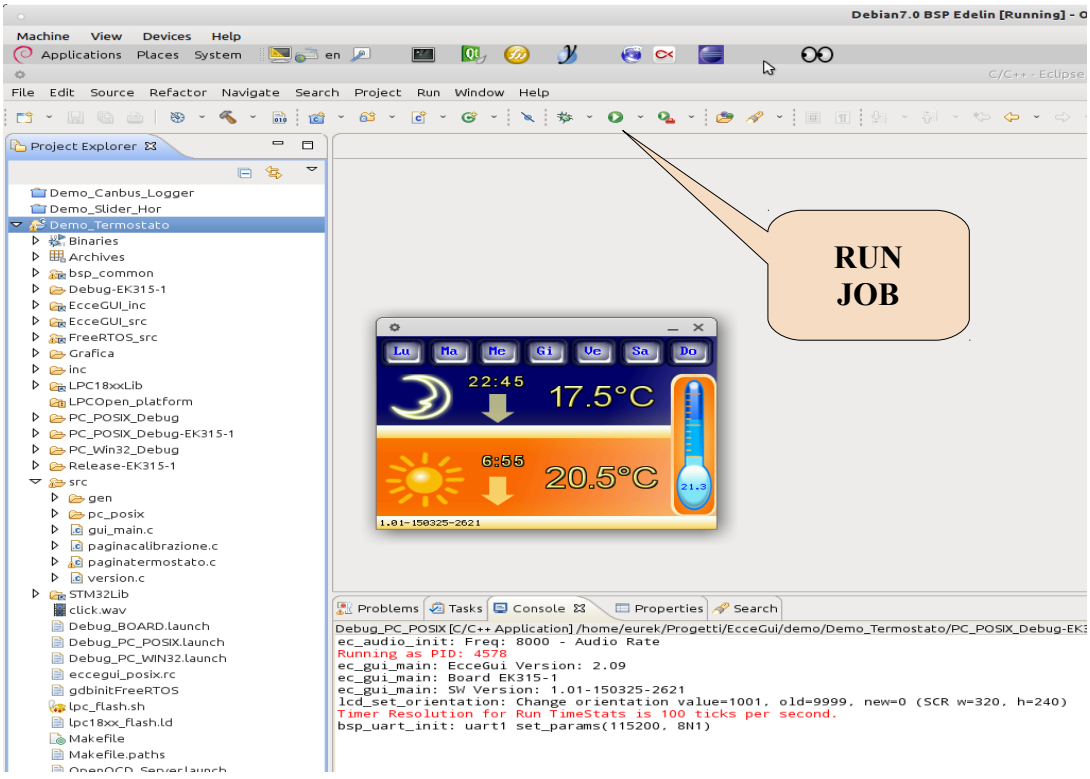
The dialog will ask you to select which workspace to use for the IDE Session, so simply press **OK.** During developing various project it will be useful to adapt workspaces for every project so to keep the environment clean for each build. In our case **only the default workspace is configured**, so please keep this behaviour in mind if change something.

On the left panel please select **Demo_Termostato** and after *click-the-right* button of the mouse and select **Open Project**.
Now select the target system you want to test this firmware: let us begin with a ***PC POSIX*** emulation, just to see how hard is to develop and build a simple application with this system!

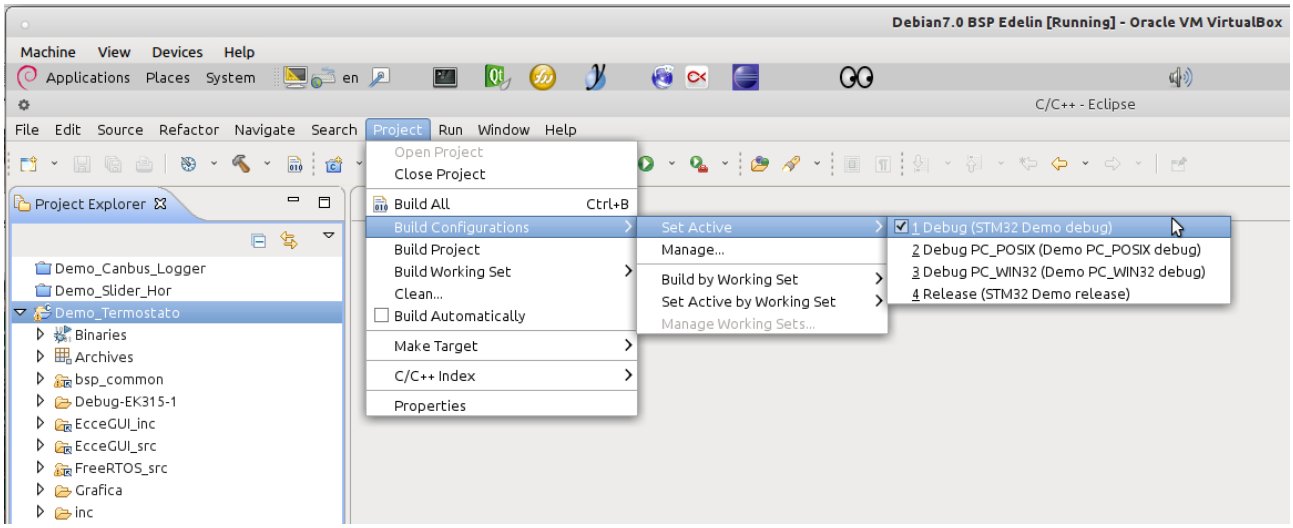


After that, you can build the executable code, pressing the ***hammer icon*** on the left side of the command tools and awaiting the finishing of the compiling job.

And now, simply pressing the **RUN ICON and selecting DEBUG_PC_POSIX** watch our application running on the PC screen!!

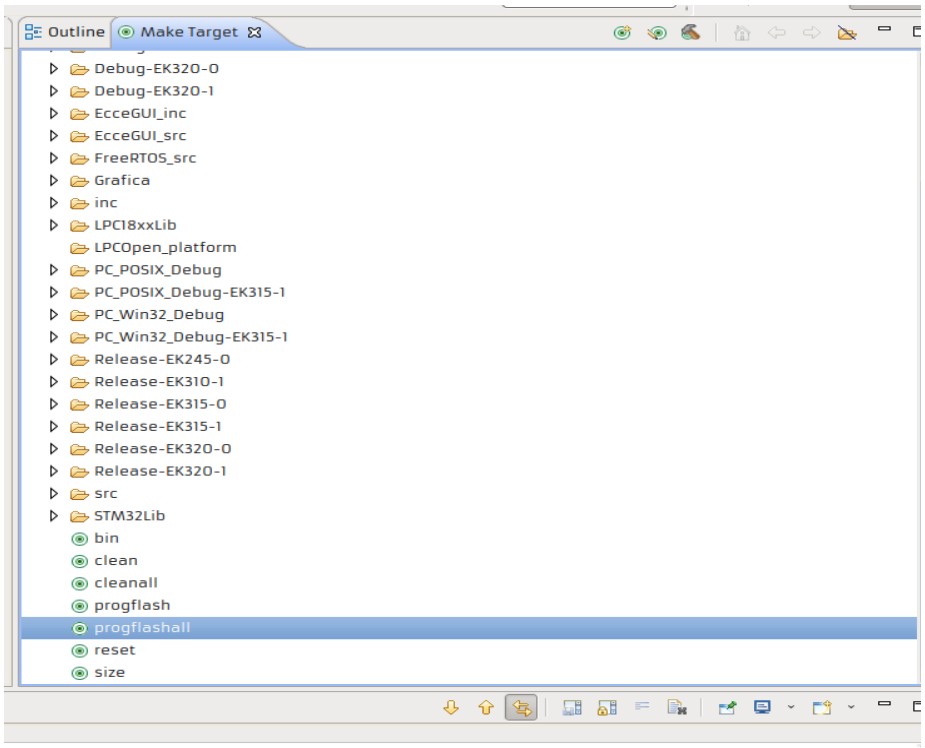


In the same way you can select a **Build Configuration** for **EK315 - Piccolo Touch** (*STM32 CPU*) and test the same application in the **real board**!!!



And in the same way press the ***hammer icon*** to build for the targeted processor/board...
In this way the building process will take a little longer due to the toolchain (cross-compiling) but after few moments (depends on the real CPU power of your system) the executables will be ready to be passed to the board.

Now opening the right panel ***Make Target*** select the **progflashall** entry. With this command you will erase and flash all internal and external flash to ensure the proper application data and code to be flashed into storage memories.



Now the process will be time consuming due to the *slowiness of the emulation USB side of the VirtualBox System*, but nevertheless it worth the awaiting... You can check-out what is happening in the lower window of Eclipse...

After that you can see on the terminal (*minicom*) window all messages coming from the debug port and using your finger, you can **activate / deactivate** icons and numbers!!

There is a **Doxygen** documentation on the desktop, so try to look at the API Library of the EcceGui.

Easy?

RS485 UART port on EK315 - Piccolo Touch

Example on how accessing the ***RS485*** uart on the **Piccolo Touch** board:

```
// --- ECCEGUI INCLUDES ---
#include "ec/types.h"
#include "ec/input_event.h"
#include "ec/platform.h"
#include "ec/layers.h"
#include "ec/version.h"
#include "ec/touch_screen.h"

#include <string.h>

#include "bsp_board.h"
#include "bsp_uart.h"
#include "dbg_usart.h"
#include "errorcodes.h"

#include "version.h"

#include <string.h>

//Do not modify the following values!
//They could lead the program to crash!!
const int ec_gui_main_stack_size = 512;
const int ec_gui_main_priority = 2;
const int ec_init_log_level = 7;
const uint32_t ec_init_log_mask = DBG_MASK_MAIN|DBG_MASK_PLATFORM;
const int ec_event_queue_size = 8;

static int uart_setup(int baudrate, int bits, int stop, t_parity parity)
{
      struct uart_params par;
      int uart_fd = uart_open(1);
      MY_DBG(1, DBG_MASK_MAIN, "Opening uart\n");
      if (uart_fd < 0)
      {
            MY_DBG(0, DBG_MASK_MAIN, "Error open uart\n");
            ec_critical_error(0, "Error open uart device");
      }
      else
      if (uart_fd != fd_dev_uart1)
            ec_critical_error(uart_fd, "Error Bad Uart fd!");

      par.delay_before_send = 10; //100usec
      par.delay_after_send = 10;  //100usec
      par.flags = RS485_MODE_RXEN;
      par.baudrate = baudrate;
      par.bits = bits;
      par.stop = stop;
      par.parity = parity;
      uart_setparams(uart_fd, &par);
      return uart_fd;
}

static void uart_end(int fd)
{
```

```
	if (fd >= 0)
		uart_close(fd);
}

/* A single string must be received within 3 seconds @ 115200 */

#define TIMEOUT_MS    3000
#define KEY_STRING    "Command RX"

static void serial_demo(void)
{
	int baudrate = 115200;
	int stop = 1;
	int bits = 8;
	t_parity parity = BSP_PARITY_NONE; /* BSP_PARITY_ODD, BSP_PARITY_EVEN */
	int fd;
	int ret, len;
	unsigned int rx_idx;
	static char rx_buffer[256];
	bool found;
	unsigned int j;
	unsigned int counter = 0;

	fd = uart_setup(baudrate, bits, stop, parity);

	// Clear/Reset all FIFOs (TX/RX)
	uart_tcdrain(fd);
	uart_tcflush(fd, true /* flush_rx */, true /* flush_tx */);

	// This program waits on receiving from UART RS485 for a given string
	// then write back a known answer back to the caller

	for (;;)
	{
		rx_idx = 0;
		len = ArraySize(rx_buffer) - rx_idx - 1;
		// read with timeout of 3 seconds
		ret = uart_read_tm(fd, (uint8_t *)&rx_buffer[rx_idx], &len,
				TIMEOUT_MS);

		if (ret == OK)
			rx_idx += len;

		rx_buffer[rx_idx] = '\0';
		if (rx_idx > 0)
		{
			// Looking for a given KEY_STRING length
			len = strlen(KEY_STRING);
			MY_DBG(3, DBG_MASK_MAIN, "* Buffer:\n%s\n *\n", rx_buffer);

			//Looking for keystring
			found = false;
			for (j = 0; j < rx_idx - len && !found; j++)
			{
				MY_DBG(4, DBG_MASK_MAIN, "Compare (%d):\n%s\n%s\n", j,
					KEY_STRING, &rx_buffer[j]);
				if (strncmp(&rx_buffer[j], KEY_STRING, len) == 0)
```

```
                        found = true;
                }

                MY_DBG(2, DBG_MASK_MAIN, "Found = %d\n", found);
                if (found)
                {
                        // if command is found write on serial
                        const char *tx = "COMMAND EXECUTED";
                        MY_DBG(1, DBG_MASK_MAIN, "Command sent\n");
                        len = uart_write(fd, (const unsigned char *) tx,
                                        strlen(tx));
                }
                else
                {
                        // Increase errors' counter
                        counter++;
                        MY_DBG(0, DBG_MASK_MAIN, "Unknown command!\n");
                }
        }
        if (counter > 10)
        {
                MY_DBG(0, DBG_MASK_MAIN, "Too much errors. Quitting\n");
                break;
        }
    }
    // When finshing release fd for uart and exit...
    uart_end(fd);
}

void ec_gui_main(void *unused)
{
    int major, minor;

    (void)unused;

    //dump on serial debug the EcceGui Library version
    ec_get_version(&major, &minor);
    MY_DBG(1, DBG_MASK_MAIN, "EcceGui Version: %d.%02d\n", major, minor);
    MY_DBG(1, DBG_MASK_MAIN, "Board %s-%d\n", BOARD_NAME, BOARD_REV);
    MY_DBG(1, DBG_MASK_MAIN, "SW Version: %s\n", get_version_str());

    ec_post_task_init();
    post_app_initialize();

    ec_enable_touch_tick(true);

    ec_enable_timertick(true);

    DBG_SETMASK(DBG_MASK_MAIN|DBG_MASK_PLATFORM);
    DBG_SETLEVEL(4);

    // serial_demo exit on error(s)
    serial_demo();

    ec_platform_exit(EXIT_POWER_OFF);
}
```

CANBUS port on EK315 - Piccolo Touch

Example on how accessing the ***CANBUS*** on the **Piccolo Touch** board:

```
// --- ECCEGUI INCLUDES ---
#include "ec/types.h"
#include "ec/input_event.h"
#include "ec/platform.h"
#include "ec/layers.h"
#include "ec/version.h"
#include "ec/touch_screen.h"

#include <string.h>

#include "bsp_board.h"
#include "bsp_canbus.h"
#include "dbg_usart.h"
#include "errorcodes.h"

#include "version.h"

#include <string.h>

//Do not modify the following values! They could lead the program to crash!!
const int ec_gui_main_stack_size = 512;
const int ec_gui_main_priority = 2;
const int ec_init_log_level = 7;
const uint32_t ec_init_log_mask = DBG_MASK_MAIN|DBG_MASK_PLATFORM;
const int ec_event_queue_size = 8;

static int canbus_setup(int bitrate)
{
      struct can_params par;
      int can_fd = can_open(1);
      MY_DBG(1, DBG_MASK_MAIN, "Opening canbus\n");
      if (can_fd < 0)
      {
            MY_DBG(0, DBG_MASK_MAIN, "Error open can\n");
            ec_critical_error(0, "Error open can device");
      }

      par.bitrate = bitrate;
      if (can_setparams(can_fd, &par) != OK)
      {
            can_close(can_fd);
            can_fd = -1;
      }
      return can_fd;
}

static void canbus_end(int fd)
{
      if (fd >= 0)
            can_close(fd);
}
```

```
static int canbus_tester(int can_fd)
{
	unsigned long checksum_r;
	int i, rval;
	int nrmsg;
	struct can_frame tcf, rcf;

	assert_param(can_fd >= fd_dev_can1 &&
			can_fd <= fd_dev_can1 + BSP_CAN_NUM);

	nrmsg = 0;
	checksum_r = 0L;

	MY_DBG(6, DBG_MASK_CANBUS, "Waiting for messages...\n");

	while (1)
	{
		int k;

		for (k = 0; k < 200; k++)
		{
			i = can_read(can_fd, &rcf, sizeof(rcf));
			if (i > 0)
				break;
			else
				ec_delay(10);
		}
		if (i == 0)
		{
			// Nothing to receive...
			// At least once?
			if (nrmsg > 0)
			{
				break;
			}
			else
			{
				// No message received, never!
				// Awaiting some more...
				MY_DBG(8, DBG_MASK_CANBUS, "Nothing to receive\n");
			}
		}
		else
		if (i != sizeof(rcf))
		{
			MY_DBG(0, DBG_MASK_CANBUS, "Error on reading\n");
		}
		else
		{
			if (rcf.can_id == 0x100 && rcf.can_dlc == 8)
			{
				if (rcf.data[2] == 0xff &&
					rcf.data[3] == 0x00 &&
					rcf.data[4] == 0xaa &&
					rcf.data[5] == 0x55 &&
					rcf.data[6] == 0x01 &&
					rcf.data[7] == 0xff)
```

```
                        {
                              // Message received. Do checksum calculation
                              checksum_r += rcf.data[0];
                              checksum_r += rcf.data[1];
                              nrmsg++;
                        }
                        else
                        {
                              MY_DBG(0, DBG_MASK_CANBUS, "Bad DATA in msg ");
                        }
                  }
                  else
                  {
                        // This message is not for us...
                        MY_DBG(7, DBG_MASK_CANBUS, "Message not for us\n");
                  }
            }
      }

      MY_DBG(6, DBG_MASK_CANBUS, "n.RX: %4d --> Sending message...\n", nrmsg);
      // Send the calculated checksum with the rcv message
      tcf.can_id = 0x101;
      tcf.data[0] = checksum_r / 256;
      tcf.data[1] = checksum_r % 256;
      tcf.data[2] = 0xff;
      tcf.data[3] = 0x00;
      tcf.data[4] = 0xaa;
      tcf.data[5] = 0x55;
      tcf.data[6] = 0x01;
      tcf.data[7] = 0xff;
      tcf.can_dlc = 8;
      i = can_write(can_fd, &tcf, sizeof(tcf));
      if (i != sizeof(tcf))
      {
            MY_DBG(0, DBG_MASK_CANBUS, "Error on writing (canbus_write)\n");
            rval = -1;
      }
      else
      {
            uint32_t key;

            MY_DBG(6, DBG_MASK_CANBUS, "Awating for key...\n");
            // Awating the message from ID 0x102
            while (1)
            {
                  int k;

                  for (k = 0; k < 50; k++)
                  {
                        i = can_read(can_fd, &rcf, sizeof(rcf));
                        if (i > 0)
                              break;
                        else
                              ec_delay(10);
                  }
                  if (i == sizeof(rcf))
                  {
```

```
                        if (rcf.can_id == 0x102)
                        {
                              MY_DBG(6, DBG_MASK_CANBUS, "Received!!!\n");
                              break; // message ok, so exit while-loop...
                        }
                        else
                        {
                              //message is not for us, drop it...
                              MY_DBG(7, DBG_MASK_CANBUS, "Drop\n");
                        }
                  }
            }

            // Check the correctness of the key (0xdeadbeef)
            key = (uint32_t)rcf.data[0] << 24 |
                        (uint32_t)rcf.data[1] << 16 |
                        (uint32_t)rcf.data[2] << 8 |
                        (uint32_t)rcf.data[3] << 0;
            MY_DBG(7, DBG_MASK_CANBUS, "Verifying 0x%08lx\n",
                  (unsigned long) key);
            if (key == 0xDEADBEEF)
            {
                  MY_DBG(6, DBG_MASK_CANBUS, "Correct Key received OK\n");
                  rval = 0;
            }
            else
            {
                  MY_DBG(6, DBG_MASK_CANBUS, "Wrong Key received ERROR\n");
                  rval = -1;
            }
      }

      return rval;
}

static void canbus_demo(void)
{
      int counter;
      int can_fd = canbus_setup(125000);

      if (can_fd >= 0)
      {
            for (;;)
            {
                  if (canbus_tester(can_fd))
                        counter++;
                  if (counter > 10)
                        break;
            }
            canbus_end(can_fd);
      }
}

void ec_gui_main(void *unused)
{
      int major, minor;
```

```
	(void)unused;

	//dump on serial debug the EcceGui Library version
	ec_get_version(&major, &minor);
	MY_DBG(1, DBG_MASK_MAIN, "EcceGui Version: %d.%02d\n", major, minor);
	MY_DBG(1, DBG_MASK_MAIN, "Board %s-%d\n", BOARD_NAME, BOARD_REV);
	MY_DBG(1, DBG_MASK_MAIN, "SW Version: %s\n", get_version_str());

	ec_post_task_init();
	post_app_initialize();

	ec_enable_touch_tick(true);

	ec_enable_timertick(true);

	DBG_SETMASK(DBG_MASK_MAIN|DBG_MASK_PLATFORM);
	DBG_SETLEVEL(4);

	// canbus_demo exit on error(s)
	canbus_demo();

	ec_platform_exit(EXIT_POWER_OFF);
}
```

From now you are invited to understand the source code, apply changes you want and obviously write your own killer-application from scratch!

Hint: To build **your own** project into workspace, simply **clone** any existing project and change the **Makefile** accordingly (mainly for adding and removing source C code and H headers)...

Please feel-free to contact us for any comment, questions or anything regarding this document and typo errors...

That's all folks!

PiccoloTouch, Great performances. Minimal amount of space.

TECHNICAL FEATURES*

Dimensions	75 mm x 96 mm
Power	9...12V 300mA via power connector or 5V using USB cable
Display	3.2" color LCD TFT with LED backlight
Resolution	240x320 262k colors
Touch Screen	Resistive, 4 wires
Processor	STM32 ARM CORTEX M3
Clock	72 MHz
Flash Memory	512kB (internal) 16 MB (external)
Ram Memory	64 kB (internal) 512 kB (external)
Fram	8k
Temperature Sensor	LM73CIMK-1
Real Time Clock	Internal with SUPERCAP backup
Jtag Port	Yes with custom connector
Interfaces	CAN 2.0B (with selectable terminator) I2C (3.3V) RS485 (with selectable terminator) USB (microUSB connector) microSD
Connector	PHOENIX CONTACT PTSM 8-pin
Real-Time OS	YES, FreeRTOS
File System	YES, FATfs
Graphics Library	YES, ECCE GUI
Bootloader	USB mass storage emulation (includes internal flash, external flash, SD-CARD)
Boot Time	Approx 1 sec
Sound	Buzzer
Developer tools	IDE Eclipse with GCC Compiler and OCD Debugger Emulation on PC with Linux or Windows

ACCESSORIES

JEK-KEY-EK206	USB JTAG Emulator with debug UART
Box (Optional)	BX315
Dimensions	81 mm x 128 mm h. 30 mm (max)
Material	ABS
Color	Warm white

* soggetto a modifiche senza preavviso // *subject to change without notice*

eurek
ELECTRONIC ENGINEERING

Eurek s.r.l.
Via Celletta 8/b
40026 Imola (BO) - Italy
www.eurek.it

Tel: **0542 609120**
Fax: **0542 609212**
P.IVA: **00690621206**
CF: **04020030377**