Slovak University of Technology in Bratislava
Institute of Information Engineering, Automation, and Mathematics

PROCEEDINGS

17th International Conference on Process Control 2009

Hotel Baník, Štrbské Pleso, Slovakia, June 9 – 12, 2009

ISBN 978-80-227-3081-5

http://www.kirp.chtf.stuba.sk/pc09

Editors: M. Fikar and M. Kvasnica

Ježek, O., Balda, P.: An Embedded Multifunction Board for Automatic Control Applications, Editors: Fikar, M., Kvasnica, M., In *Proceedings of the 17th International Conference on Process Control '09*, Štrbské Pleso, Slovakia, 484–490, 2009.

Full paper online: http://www.kirp.chtf.stuba.sk/pc09/data/abstracts/039.html

AN EMBEDDED MULTIFUNCTION BOARD FOR AUTOMATIC CONTROL APPLICATIONS

O. Ježek *, and **P. Balda** *

* *University of West Bohemia in Pilsen, Faculty of Applied Sciences, Department of Cybernetics, Univerzitni 22, Pilsen, Czech Republic*
fax : +420 377 632 502 and e-mails : {ojezek,pbalda}@kky.zcu.cz

Abstract: This paper describes basic features of an universal embedded board developed in our university. After a brief summarization of the board hardware design, the paper focuses on presented software architecture solution. The modular firmware structure based on input/output driver model and a real time operating system (FreeRTOS in our case) is explained and the usage of the software development kit (SDK) plugged-in the Eclipse development platform is shown. Moreover, a recently developed technique for automatic generation of advanced control algoritms has been ported to this platform. This technique called MicroREX is based on the Micro RexLib function block library. Software generation is demonstrated on two examples.

Keywords: embedded systems, ARM, industrial communications, automatic control, function block, Simulink, REX control system.

1. INTRODUCTION

There are not many universal embedded control devices having open architecture which are suitable for automatic control applications with rich communication capabilities. That is why Georgiev (Georgiev (2007)) developed his own hardware solution. The hardware features are summarized in section 2.

First software applications for board tests have been developed in the diploma project Ježek (2008). But more serious and systematic firmware development technique is described in section 3. It is based on the software development kit (SDK) (section 4) containing libraries and header files of application programming interface (API). Section 5 deals with convenient development tools for maximum simplicity and efficiency of application software development.

Techniques from sections 3-5 can be used for creating of general embedded applications using the board, including automatic control applications. But development and especially maintenance of more complex real-time control applications is not a simple task. Recently a powerful technique for automatic control code generation from function block diagrams has been developed in Balda (2007). The main ideas of this technique, which has been ported to the ARM platform, are presented in section 6.

Overall application design process is demonstrated on an example in section 7.

2. HARDWARE SPECIFICATION

At the beginning of the hardware design phase, there were several requirements:

- Rich communication capabilities for very good interconnection and interoperability – ZigBee wireless communication, Ethernet, USB, RS232, RS485, CAN.
- Universal direct inputs and outputs for process and machine control applications –

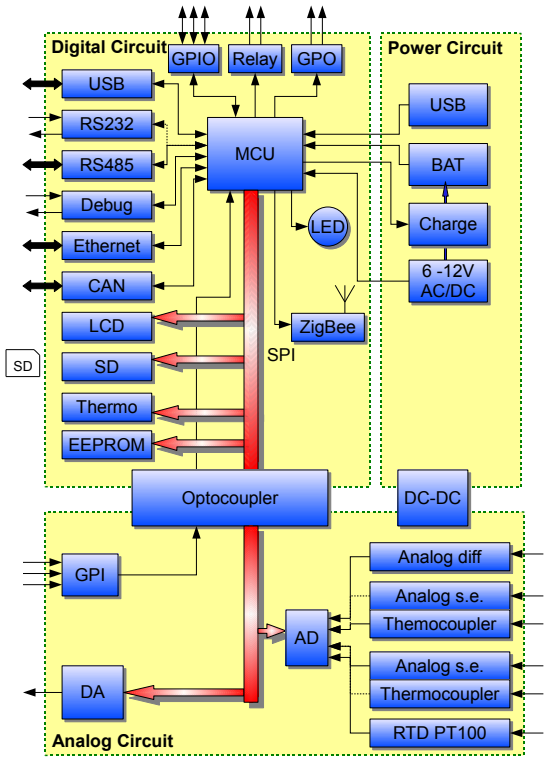

Fig. 1. Hardware structure of the board

digital inputs and outputs, analog, RTD and thermocouple inputs, analog output, etc.

The final hardware structure of the board designed by Georgiev (2007) is depicted in Fig. 1. Fig. 2 is the photography of the manufactured board.

Microcontroller (MCU)

- AT91SAM7XC512@48MHz
  - ARM7TDMI® processor
  - 512kB of FLASH memory
  - 256kB of SRAM memory
  - many on chip integrated peripheries

Reset circuitry

- Internal on chip reset controller
- External reset button

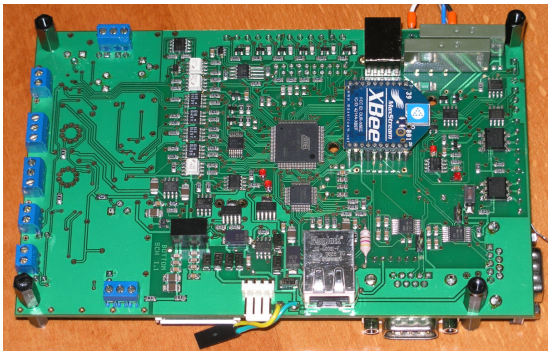

Fig. 2. Photography of the embedded board

Power

- External supply 6 - 12V AC/DC
- 4 battery pack Ni-Mh 4,8V
- USB

Communication interfaces

- RS232 2x - serial line in the RS232 standard
- RS485 - serial line in the RS485 standard multiplexed with one RS232
- USB in device mode
- ZigBee wireless communication module supporting several operation modes
  - ZigBee virtual serial line mode, for connection of two modules
  - ZigBee mesh mode, for connection of several modules in master/slave protocol
- Ethernet 100Mb/s
- Controller Area Network (CAN)

Digital Inputs/Outputs (GPIO, GPI, GPO)

- Magnetic relay 2x 250V/4A
- Optical switch AC/DC 400mA/250V
- Straight digital I/O 3.3V TTL compatible
- Opto-isolated digital inputs

Analog Inputs/Outputs

- Analog output 0 ÷ 4V or -2V ÷ 2V, 13bit resolution
- Analog voltage differential input, 1kHz filter input
- PT100 input
- 2x analog voltage single ended 0 ÷ 4V
- or 2x thermocouple of the J/K type

Configuration of each single ended analog input or thermocouple is done during the final stage of production.

Other

- Small graphic LCD display interface

3. FIRMWARE ARCHITECTURE

Generally, the board can be equipped with arbitrary firmware written for the given processor and used peripheries using several available assembler and/or C-language compilers, e.g. IAR C/C++ Compiler for AMR, Keil C/C++, GNU C/C++, etc. A User can start from scratch or follow the instructions of recommended development tools at the processor producer web site (Atmel (2009)).

For maximum user convenience and speeding up the application development, we decided to implement a software development kit (SDK) for our

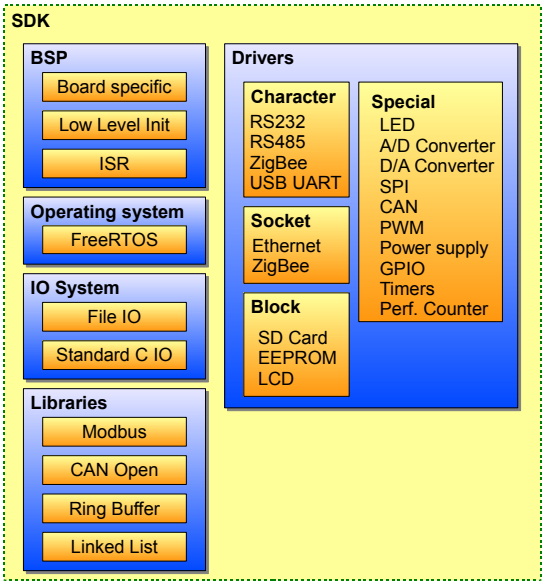

Fig. 3. SDK modular architecture

board. Real-time operating system issues are crucial for predictable behavior of automatic control applications. At present, there are many operating systems for embedded applications with various functionalities implemented. For the present version of SDK, FreeRTOS (FreeRTOS (2009)) has been chosen. FreeRTOS has very small memory footprint but also a limited functionality. It primarily implements preemptive task scheduler but neither interrupt handling services nor input/output system interface. Therefore, both functionalities has been implemented into our SDK.

4. SOFTWARE DEVELOPMENT KIT (SDK)

Simply said, our SDK is a zip file containing three folders:

`doc` – SDK documentation in several formats (compiled HTML `*.chm` and portable document format `*.pdf`)
`inc` – C-Language header files `*.h`
`lib` – Compiled libraries `*.a`

Architecture of SDK is modular, each module is represented by at least single header file and exactly one library file. SDK modules are divided into four categories (see Fig. 3) according to their characters:

Board Support Package (BSP) provides board interface for board abstraction. Microcontroler specific functionality and interrupt handling interface are provided by BSP.
Operating System (OS) category contains the particular operating system kernel libraries. For our case of FreeRTOS, also interrupt handling extension routines are included.
Input/Output System (I/O system) libraries implement standard device and file handling interface. This interface can also be used for most hardware drivers (see below). The I/O system is based on the idea of virtual file system. Moreover, I/O system functions are mapped to standard C-language input/output.
Hardware Drivers are libraries which support onboard hardware devices (e.g. RS232 line or A/D converter). Drivers handle device interrupts (hide them to the user) and usually implement I/O system interface because of unified access to various devices.
Libraries are the other miscellaneous libraries which can be used for development of the SDK based applications, e.g. Ring buffer, Modbus, etc.

Hardware drivers can be further (more detailed) divided into four groups corresponding to the device type:

Character – for character oriented devices such as RS232, RS485, USB, ZigBee in virtual serial line mode,
Socket – for BSD socket oriented devices such as Ethernet or ZigBee in the mesh mode,
Block – for block oriented devices such as disks (SD Card, EEPROM, LCD),
Special – for devices not conforming to any previous category (timers, LED, A/D and D/A converters, General Purpose I/O (GPIO), Pulse Width Modulation (PWM) outputs, Serial Peripheral Interface (SPI), Controller Area Network (CAN).

SDK development is a long run work. At present, BSP, OS and I/O system libraries has been developed. The development states of hardware drivers are the following:

Finished : RS232, RS485, GPIO, LED, Zigbee in serial line mode.
Being tested : Ethernet, A/D and D/A Converter, Performance Counter, SPI, Power Supply.
Planned : ZigBee in mesh mode, CAN, PWM, Timers, all block drivers.

Note that SDK is distributed without source code because it contains large parts of code developed without any public or grant support. However, source code organization corresponds to individual libraries mentioned.

5. DEVELOPMENT TOOLS

The present version of SDK is compiled with IAR Systems C/C++ compiler for ARM (IAR (2009)). This compiler and the corresponding linker is used for applications building process. Compiler and linker (and the other tools) can be called from command line or IAR Embedded Workbench

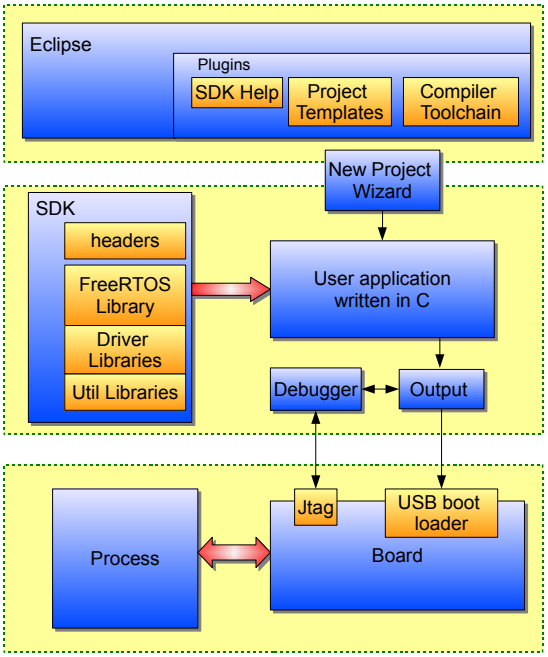

Fig. 4. Board development environment

IDE (integrated development environment) can be used.

IAR Embedded Workbench is not a very convenient development environment compared with the best development environments in the market as Visual Studio, Eclipse or NetBeans. Therefore, we chose Eclipse platform due to its adaptability using plug-ins. The brief structure of our development environment is depicted in Fig. 4.

Eclipse IDE has been extended by a special plug-in to facilitate the board application development. This plug-in can be used for generation of empty projects supporting all tools necessary for the board application development. The plug-in can also be used for development of existing and/or new SDK libraries.

Our plug-in contains a wizard for automatic generation of the board SDK based applications as well as the necessary settings of all tools (compiler, linker, debugger, Eclipse IDE itself). Let's explain the wizard capabilities in a more detail. Creation of a new SDK C/C++ project from Eclipse `File/New/C Project` or `File/New/C++ Project` menu activates the Eclipse wizard multi page dialog. Specification of `Project name` as `<project_name>` and `Project type` as `emSDK application` and pressing the `Next` button activates further pages contained in our plug-in with the additional information:

- Basic Settings page with Author identification field, short project description, etc.
- Driver selection page, which enables to specify all drivers including in the application firmware, see Fig. 5.

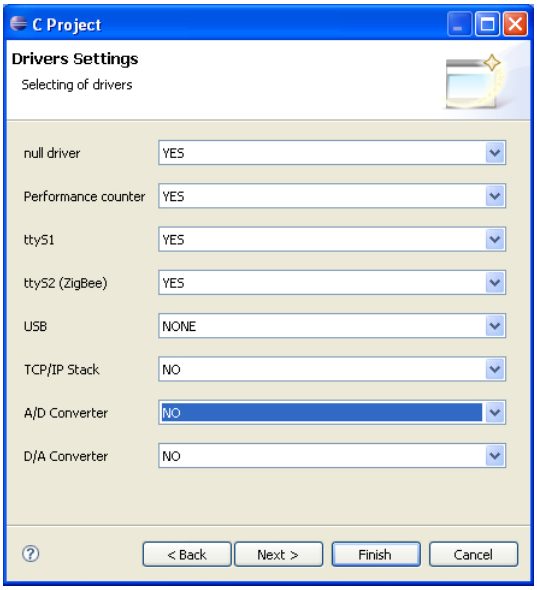

Fig. 5. SDK wizard driver selection page

- Configuration selection page, which enables to choose the generation of Debug or Release or both versions of the project.

After finishing the project wizard, it generates the project settings containing among other things the list of required static libraries. The following files are generated automatically:

`config.h` – File containing definitions of which drivers (libraries) are included and which are excluded from the firmware. The example of this file, which corresponds to the drivers selections in Ffig. 5, is depicted in Fig. 6.

`sysConfig.c` – File containing the initialization sequence of all SDK drivers and libraries. Initialization depends on conditional variables defined in the `config.h` file. File `sysConfig.c` should not be modified by the user.

`<project_name>.c` is an main file of the project. It contains the `main()` function, which is the first developer modified function called.

`<project_name>.eww` is the IAR workspace file. This file is used to debug the project using the IAR Embedded Workbench.

The project also contains other automatically generated files. But the detailed descriptions of these files are out of the scope of basic application programming techniques and of this paper.

More precisely, the `main()` function, as the starting function of the SDK based application, is called immediately after the C-language run-time library initialization, OS initialization and initializations of the drivers selected. Fig. 7 shows an example of the `<project_name>.c` file. Example shows how to resend data received through wireless ZigBee to standard, output. Standard output is mapped to `/dev/ttyS0` (Debug port) by default. Note, that the `main()` function is running

```
#ifndef CONFIG_H_
#define CONFIG_H_

/* Power driver */
#define CONF_PWR            1

/* Serial drivers */
#define CONF_TTYS0          1
#define CONF_TTYS1          1
#define CONF_TTYS2          1

/* Null driver */
#define CONF_NULL           1

/* Performance counter driver */
#define CONF_PFC            1

/* USB driver */
#define CONF_USB            0
/* Ethernet TCP/IP driver*/
#define CONF_TCPIP          0

/* Analog drivers */
#define CONF_ADC            0
#define CONF_DAC            0

#define CONF_STD_ON_TTYS0   1

#endif /*CONFIG_H_*/
```

Fig. 6. Example of `config.h` file

as a FreeRTOS task with the highest possible priority. Other FreeRTOS tasks can be created from the `main` function.

6. ADVANCED CONTROL ALGORITHMS

The technique described in the previous sections allows the programmer to develop wide range of applications for our board, e.g. communication applications, user interface application and also real-time control applications. The common feature of this technique is hand writing of all the code, except the automatically generated application template.

However, manual editing of control algorithms is a very error prone (and annoying) process. Much more convenient is to design especially advanced control algorithms in some graphical environment, e.g. in Matlab Simulink. Consequently, an automatic generation technique of C-language source code from the function block diagram designed in Matlab Simulink (or RexDraw) has been developed (Balda (2007)). This technique was slightly enhanced a year later and ported to B&R Automation PLCs (Balda and Schlegel (2008)). The same technique is now also available for our board in the form of C-language static library and set of header files. It supports almost 90 different function blocks from more than 130 function blocks of the REX control system (REX Controls (2008)).

```
/* standard headers */
#include <stdio.h>
#include <stdint.h>

/* SDK headers */
#include <FreeRTOS/FreeRTOS.h>
#include <FreeRTOS/task.h>

#include <ios/io.h>

/* Definitions */
#define DELAY_MS 50

/* main function is always a task  */
/* for the SDK base applications   */
int main(){
    uint32_t data;

    /* open ZigBee serial line */
    HANDLE ttyZB;
    ttyZB = open("/dev/ttyS2", 0, 0);

    for(;;){
        /* Read data,
           wait indefinitely */
        read(ttyZB, &data,
             sizeof(uint32_t));

        /* Write message to std out */
        printf("Redirect data: %d",
               data);

        /* wait */
        vTaskDelay(DELAY_MS);
    }
}
```

Fig. 7. Example: sending ZigBee data to standard output

7. MICRO REX EXAMPLE

Automatic generation of controls algorithm is demonstrated on the example depicted in Fig. 8 stored in Matlab-Simulink model file (`.mdl`)[1]. Note that the example uses an advanced sliding mode heating/cooling controller with autotuner (SMHCCA, see Schlegel and Mertl (2006), REX Controls (2008)).

The model file from Fig. 8 is processed by RexComp (the REX control system compiler), which generates the C-language source code (`.c`) containing three functions:

`InitControlAlgorithm()` – function for initialization of function block data structures, setting configured function blocks parameters and running initialization functions of each particular block in the control algorithm.

`MainControlAlgorithm()` – function implementing single execution of the control algorithm. This function should be called periodically with the period specified in the `SLEEP` block (in

[1] The `.mdl` files can be also created by the RexDraw program, which is the design environment of REX.

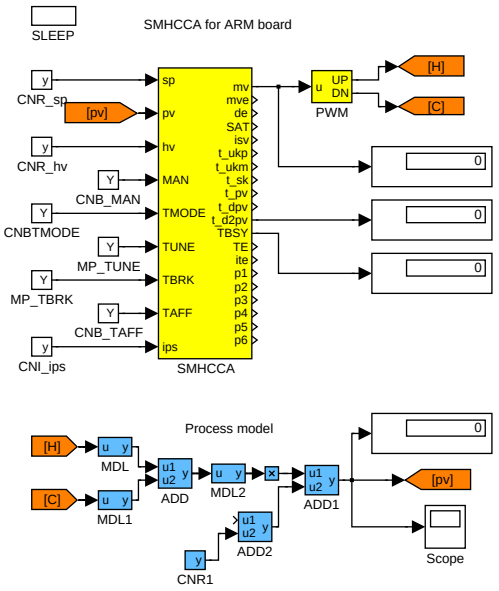

Fig. 8. Control algorithm example in Simulink

Fig. 8). This function sequentially calls main functions of particular blocks and implements the flow of signals (block connections, "wires") in the control algorithm.

`ExitControlAlgorithm()` – termination function of the control algorithm. This function sequentially calls the termination functions of the particular blocks in the algorithm.

```
#include "config.h"
#include <FreeRTOS/FreeRTOS.h>
#include <FreeRTOS/task.h>
#include <stdio.h>

#include "BlkMacrs.h"
#include "AppMacrs.h"

#include "fbSMHCCA.h"

extern ds_SMHCCA SMHCCA;

const portTickType
    tickPeriod = 100; // 0.1 [s]

int main()
{
    InitControlAlgorithm();

    lastWakeTime = xTaskGetTickCount()
        - tickPeriod + 1;
    while (1)
    {
        vTaskDelayUntil(&lastWakeTime,
                        tickPeriod);

        MainControlAlgorithm();
        printf("%.6f\r\n", SMHCCA._pv);
    }

    ExitControlAlgorithm();
}
```

Fig. 9. MicroREX application main function

All three functions respects the block order performed by RexComp. Usage of the automatically generated functions is demonstrated on the `main()` function in Fig. 9. In this case, each control period the `SMHCCA` controller process variable is printed to the standard output. Code of the three functions `Init/Main/ExitControlAlgorithm` is not presented here for brevity reasons.

8. CONCLUSIONS

The paper presents a new board designed for embedded automatic control and rich communication applications. Especially the designed software (firmware) architecture is introduced based on the developed SDK and its integration to Eclipse IDE. The MicroREX technique (formerly called Micro RexLib) can be used for the development of advanced control strategies.

The authors believe that the proposed methodology is suitable for education of automatic control algorithm implementation aspects, embedded and real-time micro-controller programming. The approach will be used for teaching more than 100 students in the "Embedded control systems" course in the Faculty of Applied Sciences at the University of West Bohemia in Pilsen, Czech Republic.

ACKNOWLEDGMENTS

The work has been supported by Czech Ministry of Industry and Trade, project No. MPO FI-IM5/030. This support is gratefully acknowledged.

References

Atmel. Atmel web page: `http://www.atmel.com`. 2009.

P. Balda. Automatic conversion of advanced control algorithms of the REX control system to various embedded platforms of microcontrollers. In *Automatizace, regulace a procesy (in Czech)*, pages 47–54, Prague, 2007. Dimart.

P. Balda and M. Schlegel. BRRexLib – Function block library for B&R Automation PLCs. In *Process Control 2008*, pages 1–7, Kouty nad Desnou, 2008. University of Pardubice.

FreeRTOS. The FreeRTOS.org Project web page: `http://www.freertos.org`. 2009.

V. Georgiev. Atmel ARM based embedded board hardware design. Partially financed by MATEO "Industrial Controllers" subproject, 2007.

IAR. IAR Systems web page: `http://www.iar.com`. 2009.

O. Ježek. *Controller for embedded control applications: Basic design. Diploma project (in Czech).* 2008.

REX Controls. *Function blocks of the REX system – Reference Guide (in Czech).* REX Controls, Pilsen, Czech Republic, 2008.

M. Schlegel and J. Mertl. New control strategies for heating/cooling processes. In *Process Control 2006*, pages 1–12, Pardubice, 2006. University of Pardubice. ISBN 80-7194-860-8.