sAPI (*simple*API), a hardware-independent C library for Embedded Systems programming

Ing. Eric Nicolás Pernia
Departamento de Ciencia y Tecnología
Universidad Nacional de Quilmes (UNQ)
Quilmes, Buenos Aires, Argentina
eric.pernia@unq.edu.ar / ericpernia@gmail.com

Mg. Ing. Félix Safar
Departamento de Ciencia y Tecnología
Universidad Nacional de Quilmes (UNQ)
Quilmes, Buenos Aires, Argentina
felix.safar@unq.edu.ar

***Abstract*— A library design is presented aimed to standardize C language programming on microcontroller-based platforms. This library defines a simplified Application Programming Interface (API) that abstracts the most common modes of use of typical peripherals found in current microcontrollers in the marketplace. In this way, it becomes possible to program them with no need of knowing details on the underlying architecture. This design promotes and enables hardware-independent programming, reducing overall complexity of embedded systems development. Reference implementation on the EDU-CIAA-NXP platform is detailed.**

Keywords—sAPI; API; HAL; Embedded Systems; Microcontroller; C microcontroller library;

I. INTRODUCTION AND BACKGROUND

Currently the majority of embedded systems programming based on microcontrollers is written in C language, using libraries for handling the processing core (or cores) and peripherals. Consequently, a C library is an integral part of any embedded microcontroller-based system design, even in cases when programming is done in another high-level language.

In this paper we present a library design to program microcontroller-based Embedded Systems platforms in a simplified way. It allows the most common modes of typical peripherals of a microcontroller. It also outlines main features in a reference implementation on the EDU-CIAA-NXP platform [1].

This work is part of a group of initiatives promoted by the *Universidad Nacional de Quilmes* [2] to collaborate within the framework of the CIAA Project [3], to facilitate the development and teaching of Embedded Systems in Argentina. Other initiatives include: Ladder PLC programming environment (IDE4PLC [4]), Java programming for CIAA [5], and Firmata4CIAA [6] to facilitate graphical blocks programming in BYOB Snap! [7] language for use in secondary schools.

II. MOTIVATION AND CONTEXT

There are a great variety of microcontroller-based platforms in the market, and while they all expose microcontrollers with similar characteristics and compatible peripherals, it is observed that their libraries are very different. This is because each manufacturer and/or associated companies offer their own libraries written in C language, which are designed with strong dependence upon the underlying microcontroller architecture that these platforms contain.

There are also many libraries that achieve good hardware abstraction across platforms, but none of them have been adopted as a general standard. Among several, it can be cited:

- In the automotive industry there is a standard called AUTOSAR [8] aimed for the standardization of automotive electronic systems. This architecture proposes very extensive and complex to implement library definitions, making difficult to learn and use. Also, it is not yet been extended to other industries.
- Driver libraries based on POSIX [9] (can be found on systems with embedded Linux such as Raspberry Pi [10]). Its abstraction has been very useful for the standardization of drivers for PC's with Unix-compatible operating systems. However, it is so far from the physical hardware that in practice is not very used to program embedded systems.
- Embedded systems hobbyists have pushed standardization of programming through a library known as Wiring [11] which is available for multiple platforms (such as the popular Arduino [12]). Although this library achieves great ease of use and fast learning, it contains some technical inaccuracies that provoke unwanted vices in the learning of programming microcontrollers. It also lacks an API definition for the use of timers, peripherals widely used in microcontroller applications.
- Currently there are many manufacturers of microcontrollers that have acquired licenses for the

manufacturing of microcontrollers with processing cores of Cortex architectures [13] designed by the company ARM [14]. For them there is a standard library called CMSIS [15] (Cortex™ Microcontroller Software Interface Standard) to program of the processing cores and interrupt controllers but does not extend to peripherals where each manufacturer seeks to differentiate itself from its competitors.

- Other well-known companies in the field of embedded systems programming such as Microchip [16] provide hardware-dependent libraries and automated code generators. These tools, aimed to accelerate the development timeframes, do not perform very well when an application requires more advanced modes, making programming difficult since some configurations steps over the others cause coding problems.

From previous examples it is observed that for the realization of a standard library satisfying different users a balance must be achieved among following requirements:

- Size of API definition.
- Hardware dependency.
- Level of abstraction.
- Complexity of learning and use.
- Peripherals and supported modes.
- Scalability.

Due to these reasons, in order to overcome above exposed difficulties it was decided to elaborate an API definition of a standard library for microcontrollers in C language.

III. Description of sAPI library

A. *Methodology of work*

In order to develop the sAPI library the following practices were chosen:

- Use of Git repository [17], within github site [18].
  - Development workflow through Fork and Pull Requests on github site.
  - Issues are opened within github for each feature to design and/or improve.
- Code production and documentation:
  - Writing files with .h extension for each module, documented using Doxygen [19] tool, and later writing the .c extension files with their implementation.
  - In addition, a document with source code styles is defined.
- General documentation (outside source code) written in Markdown [20] language, which allows exporting contents to html, latex, and pdf formats, among others.

B. *Design principles*

Based upon aforementioned items and other good practices, it was decided to use the following premises to guide the design:

- Support programming of core processing and main peripherals along with most common modes of operation.
- Possess a level of abstraction sufficient to become independent of microcontroller hardware while maintaining the identity and key concepts of each module that contains the microcontroller.
- Keep moderate the extension of API definition extension.
- Use of simple names to facilitate learning and utilization.
- Explicitly specify the dependency between library modules and the use of physical resources in each particular implementation for each platform.

C. *Library's architecture*

An application based upon sAPI library contains at least the software layers described in Figure 1.

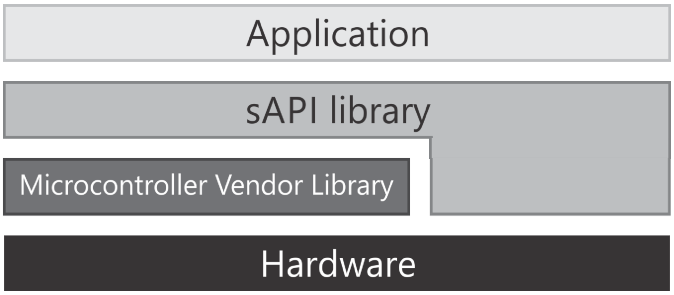

Figure 1, software layers for an application based upon sAPI library.

The sAPI library must isolate the user application from the underlying hardware by forming an abstraction layer of the hardware. Depending on the particular implementation, existing libraries provided by the microcontroller manufacturer of a given platform may be used.

The sAPI library model in turn consists of software modules, grouped into the software layers that are summarized in Figure 2.

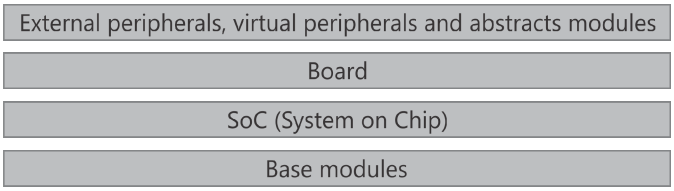

Figure 2, sAPI library software layers.

The characteristics of each layer are shown below. Then the composition of a library module (generic) is exposed.

1) External peripherals, virtual peripherals and abstract modules

The main feature of this type of modules is that specific structures must be assigned because they do not exist until they are "created" and "initialized". Among these, the following are included:

- External peripherals connected to the platform to be used, such as an I2C or SPI connected chips and/or modules.
- Virtual peripherals, for example, a software-implemented I2C that uses GPIOs hardware to operate.
- Abstract modules: they represent modules that incorporate a higher level of abstraction which allow performing advanced tasks such as management of timing of an operative system and management of buffers. These modules are highly portable as they are designed without hardware dependency.

2) Board

It contains definitions at full platform level, by defining, among other things, the names of platform physical connectors (pins, terminals, and other connectors) that are associated with lower level modules.

3) SoC (System on Chip)

This layer groups modules that model the processing core (or cores, for the case of multicore systems) and the internal physical peripherals of the platform. These modules having physical existence in the microcontroller of the platform, and are not to be "created", rather they are only "initialized". In other words, they always exist within the program; they have defined fixed names and cannot be destroyed during the program. These are:

- CORE: models a processing core.
- GPIO: models a single general purpose I/O pin (pin) as well as a set of pins (port). The written or read value of a pin is of the Boolean type.
- ADC: models an Analog-to-Digital converter peripheral. The read value is of unsigned integer type.
- DAC: models a Digital-to-Analog converter peripheral. The written value is of unsigned integer type.
- TIMER: models a Timer/Counter peripheral.
- RCT: models a Real Time Clock peripheral.
- UART: models a serial communication Universal Asynchronous Receiver Transmitter peripheral.
- SPI: models a Serial Peripheral Interface peripheral.
- I2C: models an Inter-Integrated-Circuit peripheral.

4) Base modules

These model common parts associated with the entire library and correspond to:

- Data types.
- General board initialization.

5) Composition of a library module

Each module contains a defined set of data types, properties, and methods. Each of the modules contains a set of properties that are grouped into a structure associated with the module. This is defined as a data type with the following structure:

```
typedef struct{
   <type> property1;
   <type> property2;
   ...
}<module>_t;
```

This structure is the area where the module is mapped and is accessible from the public API only as a named index. An enumerated type associated with the name of the module is then defined to access it by name (on physical peripherals, e.g. I2C0, SPI1, UART4, etc.). This type definition is below:

```
typedef enum{
   <MODULE0>,
   <MODULE1>,
   ...
}<module>_t;
```

Typical properties of a physical peripheral are:

- configParamName: peripheral configuration value.
- power: turning it on or off.
- value: value to read or write.
- eventName: some event associated to the peripheral (either interrupt or other).
- eventNameCallback: a structure with a pointer to function and pointer to parameter that the user can pass to that function (either interrupt or other).

Finally, a set of access methods for each module, including:

- Module initialization. This turn on the module if necessary and initialize it with the most typical configuration used:

```
<module>Init(
   <MODULEi>,
   <mostCommonPropertyValue> |
```

```
   <MODIFY_FLAG1> | ... |
   <MODIFY_FLAGn>
);
```

- Property values assignment:

```
<module><PropertyName>Set(
   <MODULEi>,
   <propertyValue> | MODIFY_FLAG1 |
      MODIFY_FLAG2(value) | ... |
      MODIFY_FLAGn
);
```

- Property values getter:

```
<propertyValue> = <module><PropertyName>Get(
                                    <MODULEi> );
```

- In order to simplify coding, most modules have an alternative API with read, write and configure methods:

```
<propertyValue> = <module>Read<PropertyName>(
                                    <MODULEi> );

<module>Write<PropertyName>( <MODULEi>, value );

<module>Config(<MODULEi>, param_1, ..., param_n);
```

D. *Reference implementation*

A reference implementation of sAPI library has been carried out on the EDU-CIAA-NXP platform of CIAA project (based on the NXP LPC4337 microcontroller [21]) . This implementation currently (as of April 2017) has the following modules:

1) Modules contained in reference implementation

External peripherals:

- External peripherals map.
- Seven segment Displays.
- Matrix Keypad.
- Angular Servo SG90 (0 to 180°).
- Magnetometer sensor (compass) HMC5883L.

Abstract modules:

- Delays:
  - Inexact Delay (for academic use).
  - Blocking Delay timer based.
  - Non Blocking Delay timer based.
- Buffers:
  - FIFO/LIFO buffer.
  - Circular buffer.
- Serial outputs (Print):
  - Console serial output.
  - Debugging serial output.

Virtual peripherals:

- Software I2C (using GPIO´s).

Board:

- Board peripheral map.
- Platform general initialization (Board).

SoC:

- Internal peripheral map.
- Processing core (CORE).
- General purpose Input/Output (GPIO).
- Universal Asynchronous Tx/Rx (UART).
- Analog-to-Digital converter (ADC).
- Digital-to-Analog converter (DAC).
- Inter-IC serial bus (I2C).
- Real Time Clock (RTC).
- Low Power Modes (Sleep).
- Timer:
  - Periodic Interrupt (Ticker).
  - Count to overflow (Overflow).
  - Count to match (Match).
  - Input capture (InputCapture).
  - Pulse Width Modulation (PWM).

Base Modules:

- Data Types.
- Timer Periodic Interrupt to use as time base in time-triggered Operating Systems (Tick).

Each module includes one or more sample programs to demonstrate library utilization.

2) Application examples

In this section we expose two examples to show the sAPI library.

Blinky basic example:

```
// BlinkyExample.c
#include "sapi.h"
```

```
int main( void )
(
   boardInit();
   while(TRUE)
   {
      gpioToggle( LED1 );
      delay( 500 );
   }
)
```

Blinky tick event-based example:

```
// BlinkyTickExample.c
#include "sapi.h"

// Function that execute at every tick event
bool_t tickCallback( void* )
(
   if( tickRead() >= 500 ){
      gpioToggle( LED1 );
      tickWrite(0);
   }
)

int main( void )
(
   boardInit();
   // Configure tick event callback function
   tickEventCallbackSet( tickCallback );
   // Configure tick event at 1 ms rate
   tickInit(1);
   while(TRUE)
   {
      // Do other stuff
   }
)
```

E. *Partial implementation in other platforms*

There are also partial implementations of the same library for the following platforms:

- Industrial Automation and control board CIAA-NXP [22] (an industrial computer board with NXP LPC4337 microcontroller).
- LPCXpresso 1769 (NXP LPC1769 microcontroller) development board [23].
- Atmel ATMega32A [24] custom development Board.
- Intel Quark D-2000 Development Board [25].
- Microchip PIC18LF46K22 [26] custom development Board.
- Silicon Labs Happy Gecko Development Kit [27].
- Silicon Labs Pearl Gecko Development Kit [28].

IV. USE CASES

The first version of the sAPI library (made in 2015 within the context of the Java [5] project) has been used by the first author as an example of hardware abstraction layer in university courses. These courses are: the postgraduate course "*Microprocessor Programming*" within the "*Carrera de Especialización en Sistemas Embebidos"* (CESE) of FI-UBA [29] where the first author has been Professor in charge for three editions, and the course "*Digital Systems*" within the Engineering Degree in "*Ingeniería en Automatización y Control Industrial"* (IACI) of the "*Universidad Nacional de Quilmes"* (UNQ) [2] where the author currently serves as Instructor.

Starting in 2016, it was decided to use the library as part of the "*Cursos Abiertos de Programación de Sistemas Embebidos"* (CAPSE [30]) organized by ACSE [31]. In this way the library has been extended considerably to explain the use of all typical microcontrollers' peripherals with impressive learning results for students both at advanced levels and those who take their first steps in learning microcontroller programming.

In addition, sAPI library was made available to anyone since it is freely available published online under a modified BSD [32] license on the author's github site [33] and has been adopted by a significant number of users.

Finally, in December 2016 it was decided to use the library as a standard library for the CIAA Project platforms. This led to a thorough review and improvement of it currently being worked on.

V. CONCLUSIONS AND FUTURE WORK

Based upon the results obtained in using the library for teaching different courses at different levels and the acceptance of multiple programmers for their individual projects, it is believed that it has been possible to design a library that allows programming in C language in a simplified form in microcontroller-based embedded platforms without the need to know in detail their architecture. In particular, for experienced programmers, the sAPI library is able to streamline the development of applications through its direct utilization, extension or reconfiguration according to their needs; or even taking it as a basis to understand the programming of a particular platform architecture by reviewing its source code.

Within the field of teaching embedded systems programming, it allows concentrating the efforts of novice students in understanding programming applications on embedded systems as a whole, rather than learning the programming of a single particular platform, making it possible to understand the important concepts regardless of the underlying hardware. Then, once training progress a student can understand how the library is made for a particular platform and take it as an example of a hardware abstraction layer. In this way, it is emphasized at all times the fundamental fact of designing applications independent of hardware, a real benefit given the speed of change of these hardware devices in current marketplace.

The library is still being ported to other microcontrollers and improving its definition. It is expected to complete by the end of this year 2017 the whole library implementation for the following platforms:

- CIAA-NXP (full-blown industrial level board).

- Pico-CIAA [34] (NXP microcontroller).
- CIAA-PIC (Microchip microcontroller).

It is also intended to work within near future in the development of a simulator, capable of running a program written in C using sAPI library, on a virtual board created within a PC environment, to facilitate teaching of programming without having the need of an available microcontroller's physical platform.

ACKNOWLEDGMENT

First To Martín Ribelotta whose experience and insight has been and continues to be much valuable support for the revision process of the library.

To Dr. Ing. Ariel Lutemberg and Dr. Ing. Pablo Gómez who relied on using sAPI for the courses at CAPSE.

To the coordinators of Proyecto CIAA who relied and decided to adopt sAPI as a standard library, in particular to general coordinator Esp. Ing. Pablo Ridolfi.

To the students of CESE (FI-UBA), IACI (UNQ), and CAPSE, who used and received it very enthusiastically boosting its development.

To the Departamento de Ciencia y Tecnología (UNQ), led by Dra. Alejandra Zinni for supporting Embedded Systems engineering projects.

Last, to Ing. Leonardo Gassman who proposed the sAPI name for the library during the Java Project at UNQ.

REFERENCES

[1] EDU-CIAA-NXP. Educational board version of "Computadora Industrial Abierta Argentina", of "Proyecto CIAA". Available: 2017-05-01. [Online]: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:edu-ciaa:edu-ciaa-nxp

[2] Universidad Nacional de Quilmes. Institutional web available: 2017-05-01. [Online]: http://www.unq.edu.ar

[3] "Proyecto CIAA", "Computadora Industrial Abierta Argentina,". Available: 2017-05-01. [Online]: http://www.proyecto-ciaa.com.ar/

[4] IDE4PLC: Ladder PLC programming environment. Available: 2017-05-01. [Online]: http://ide4plc.org/

[5] Pernia, E. N., Gomez, P. M., Martos, E. I. P. I., & Sager, G. (2015). "Desarrollo de Firmware y Software para programar la CIAA en lenguaje JAVA con aplicación en entornos Industriales". Available: 2017-05-01. [Online]: http://laboratorios.fi.uba.ar/lse/tesis/LSE-FIUBA-Trabajo-Final-CESE-Eric-Pernia-2015.pdf

[6] Firmata4CIAA. A program to run a Firmata Client on EDU-CIAA-NXP board. Available: 2017-05-01. [Online]: https://github.com/ciaa/Firmata4CIAA

[7] BYOB Snap! Programming language. Available: 2017-05-01. [Online]: http://snap4arduino.org

[8] AUTOSAR (AUTomotive Open System ARchitecture). Available: 2017-05-01. [Online]: https://www.autosar.org/

[9] Why POSIX for embedded systems? Article available: 2017-05-01. [Online]: http://www.qnx.com/developers/docs/660/index.jsp?topic=%2Fcom.qnx.doc.neutrino.sys_arch%2Ftopic%2Fintro_Why_POSIX.html

[10] Raspberry Pi platform. Available: 2017-05-01. [Online]: https://www.raspberrypi.org/

[11] Wiring: open-source programming framework for microcontrollers. Available: 2017-05-01. [Online]: http://wiring.org.co/

[12] Arduino platform. Available: 2017-05-01. [Online]: https://www.arduino.cc/

[13] ARM Architectures. Available: 2017-05-01. [Online]: https://www.arm.com/products/processors/instruction-set-architectures/index.php

[14] ARM. Available: 2017-05-01. [Online]: https://www.arm.com/

[15] CMSIS: CMSIS - Cortex Microcontroller Software Interface Standard. Available: 2017-05-01. [Online]: https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

[16] Microchip. Available: 2017-05-01. [Online]:http://www.microchip.com/

[17] Git repository Reference. Available: 2017-05-01. [Online]: https://git-scm.com/docs

[18] Github site. Available: 2017-05-01. [Online]: https://github.com/

[19] Doxygen. Generate documentation from source code. Available: 2017-05-01. [Online]: http://www.stack.nl/~dimitri/doxygen/

[20] Markdown language. Available: 2017-05-01. [Online]: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

[21] NXP LPC4337 microcontroller Available: 2017-05-01. [Online]: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/lpc-cortex-m-mcus/lpc4300-cortex-m4-m0/32-bit-arm-cortex-m4-m0-mcu-up-to-1-mb-flash-and-136-kb-sram-ethernet-two-high-speed-usb-lcd-emc:LPC4337FET256

[22] CIAA-NXP platform. Available: 2017-05-01. [Online]: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:hardware:ciaa_nxp:ciaa_nxp_inicio

[23] LPCXpresso 1769 development kit (NXP LPC1769 microcontroller) Available: 2017-05-01. [Online]: https://www.embeddedartists.com/products/lpcxpresso/lpc1769_xpr.php

[24] Atmel ATMega32A microcontroller. Available: 2017-05-01. [Online]: http://www.microchip.com/wwwproducts/en/ATmega32A

[25] Intel Quark D-2000 Development Board Available: 2017-05-01. [Online]: http://www.intel.la/content/www/xl/es/embedded/products/quark/mcu/d2000/overview.html

[26] Microchip PIC18LF46K22 microcontroller. Available: 2017-05-01. [Online]: http://www.microchip.com/wwwproducts/en/PIC18F46K22

[27] Silicon Labs Happy Gecko Development Kit. Available: 2017-05-01. [Online]: http://www.silabs.com/products/mcu/32-bit/efm32-happy-gecko

[28] Silicon Labs Pearl Gecko Development Kit. Available: 2017-05-01. [Online]: http://www.silabs.com/products/mcu/32-bit/efm32-pearl-gecko

[29] Postgraduate course "Microprocessor Programming" within the "Carrera de Especialización en Sistemas Embebidos" (CESE) of FI-UBA. Institutional web available: 2017-05-01. [Online]: http://laboratorios.fi.uba.ar/lse/cursos.html

[30] "Cursos Abiertos de programación de Sistemas Embebidos" (CAPSE). Institutional web available 2017-05-01. [Online]: http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=educacion:cursos:cursos_programacion_ciaa

[31] ACSE. Institutional web. Available 2017-05-01. [Online]: http://www.sase.com.ar/asociacion-civil-sistemas-embebidos/

[32] Modified BSD 3 clause license. Available: 2017-05-01. [Online]: https://opensource.org/licenses/BSD-3-Clause

[33] First author's github site. Available: 2017-05-01. [Online]: https://github.com/epernia/sapi