International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 5 2835 - 2838

GUI based Code Generation for Embedded Systems

Vaghela Megha Nareshbhai
Embedded System Design
GTU PG School
Ahmedabad, India
megha.vaghela2012@gmail.com

Gadhia Deep H.
Embedded System Design
GTU PG School
Ahmedabad, India
deepgadhia30@gmail.com

Abstract— This paper presents a novel approach for an Integrated Development Environment (IDE) for generating software for embedded systems using user friendly graphical interface. For the embedded software engineer, starting embedded application development can present a major hurdle. To develop an application for embedded systems there are many technologies and platforms available like RTOS or bare metal programing and many other tools and environment. But these things are conceptually very different. To further complicate matters, these tools are sometimes operating system specific. Embedded application development normally requires development tools that run under Linux according to their application hardware software specification. The programmer therefore needs to first learn the basics of desktop Linux and related tools and their interface subsystems under various kernel images. After all this complications, there are various choices for open source development tools each having its own methods and practices. Although the technology is very powerful, it creates a big complication as one attempts to find a learning path. Thus one should work on techniques which streamlines the whole development process and makes it easy to develop embedded software. A unique user friendly Integrated Development Environment with the help of graphics user interface can be developed for direct code generation which can directly work with targeted embedded hardware.

Keywords- *GUI, Embedded systems, Embedded Software, IDE, Embedded Tool*

*****

I. INTRODUCTION

The complexity associated with the embedded software invites a new, more efficient design approach. An obvious choice is to use well-established component-based design; however, its adoption to design of embedded software has been slow and riddled with difficulties. One of the most significant difficulties here is the tight integration between hardware and software, typical for embedded systems, makes it virtually impossible to model and implement software separately from hardware. However, this approach to embedded software development has been significantly slower than to software development in general [1].

Component-Based Software Engineering (CBSE) is an approach which aims to increase the efficiency in software development by reusing already existing solutions encapsulated in well-defined entities (components) and building systems by efficient composition (which includes constructive, i.e. functional composition).[3] Flow-based programming (FBP) is a programming paradigm that defines applications as networks of "black box" processes, which exchange data across predefined connections by message passing, where the connections are specified externally to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component-oriented. Thus, such a policy can be implemented on unique IDE for efficient code generation for various embedded application. [4]

Its visual programming technique will help us to reduce time to prototype and market to a great extent. The generated code will help you to achieve 100% functionality originally envisaged. It is targeted toward reducing application development time and cost for industry. This tool generates C code from model diagrams, interface connections, and protocol blocks in the form of templates. The generated source code can be used for designing targeted embedded applications. You can tune the generated code using various abstraction layer libraries to make it make it work on actual targeted hardware used in the application.

In this paper we present an idea for a unique environment for embedded software generation using user friendly graphical interface which will generate the code accordingly for targeted hardware.

II. CONCEPT

The concept is to develop such an environment which incorporates the entire development process of application software generation for the targeted embedded hardware. The environment is developed as a project. This is built with the help of open source technologies that make it more reliable and scalable. The project build on top of industrially acclaimed Eclipse OSGI framework, reputed for its scalability and reliability. New feature can be integrated by simply adding new plug-ins, without any change in code. It includes design of a system components architecture representation in a logical format and specification of component behavior. Other important factors can also be included like timing requirement, abstraction glue code, and operating system functionalities, editing and programming rules.

The design is realized in a project on Eclipse framework as an graphical component edition which works as an interface to user. The additional parts include elements and programming elements are designed through diagram blocks that represent different view of elements internally. However, the external view of an components is designed in a way that represent the connection nodes and interfaces. The internal view describes all the properties and setup values for the respective component or a block.

 ISSN: 2321-8169
2835 - 2838

III. SYSTEM ARCHITECTURE

The IDE aims to cover the entire software development process for embedded systems. For realizing the proposed idea, the whole process is divided into various discrete components upon which development is done individually. Once the desired functionality on these components is achieved they can be integrated to perform a conclusive task by developing some glue analysis and filling the functionality gap. The entire concept can be seen as an architecture of these different functional blocks which works in hierarchical manner in order to get the desired outcome. The proposed system architecture is shown in the figure [Figure 1] below.

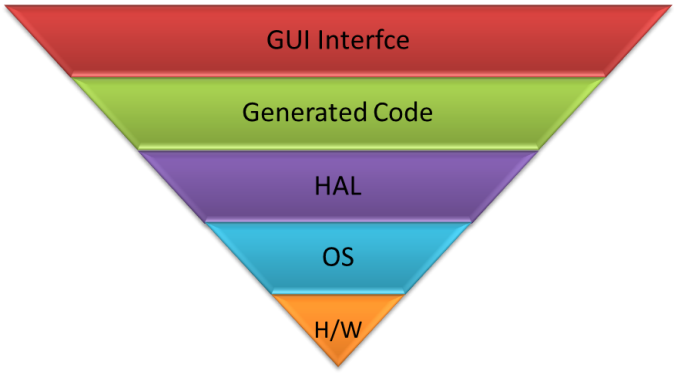

Figure 1. Proposed System Architecture

As shown in figure there are lots of different concepts are associated with this idea. The most important aspect of the project is the GUI (Graphical User Interface) which will provide the visual platform to the user equipped with drag and drop functionality. This will make the entire process much simpler as compared to the one where user needs to write the entire code from scratch. However this is much simpler task as compared to other complex functional block as there are many options available for creating such models which are going to be used in embedded systems. As there will not be any complex graphics processing associated with it and thus this is the simplest one to implement.

Once the code has been generated according the graphical representation made by user, it is also important to make sure the code is in an appropriate form in which form the user has made it. Additionally this generated code has to work properly with the hardware, thus there must be a hardware abstraction layer which defines the protocol about how that code is going to communicate with the given hardware. Board support packages would help us produce the glue code for the same.

This entire work is intended to make this code generation process easy and thus there is no point of providing the user with many complex graphical blocks and ultimately again makes this entire process tedious. Thus for handling events and interrupts the glue layer of operating systems is used for complex processing. And lastly the entire work can be tested on some evaluation board with the use of several small application test cases.

The entire developed environment is shown in figure [Figure 7] below which generates the code as per the graphical representation in the editor.

IV. FRAMEWORK

A. *GUI Editior*

GUI is the most important part of the environment however there are many JAVA based technologies are available for the same. Beside that in our system, we do not require any complex graphical design and thus any existing technology will work for our simple embedded graphical models.

XML (Extensible Markup Language) is popular options now are days for building up a GUI elements. It defines a set of rules for encoding documents in a format which is both human-readable and machine-readable.

The entire palette has been developed consists of rich set of elements which will help the programmer's job easy and which gives variety options for efficient programming. User can choose the programing elements and other functional and logical blocks from the palette [Figure 2] and can draw a logical architecture for the desired application software in the workspace [Figure 3].

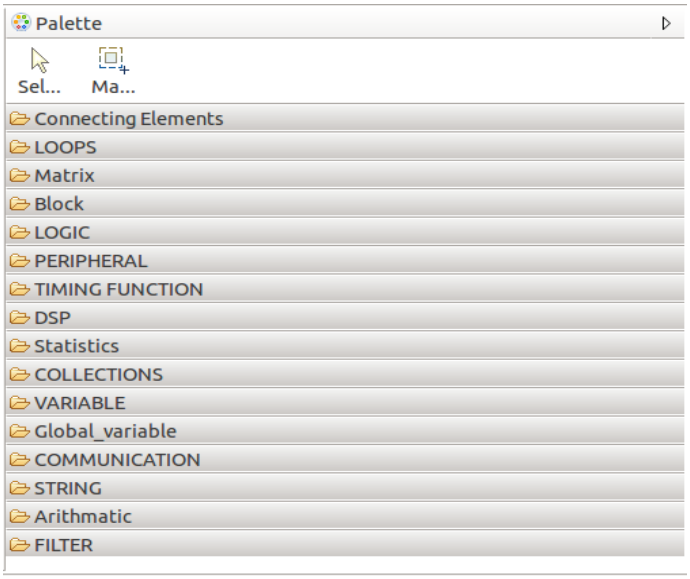

Figure 2. Component Palette of IDE

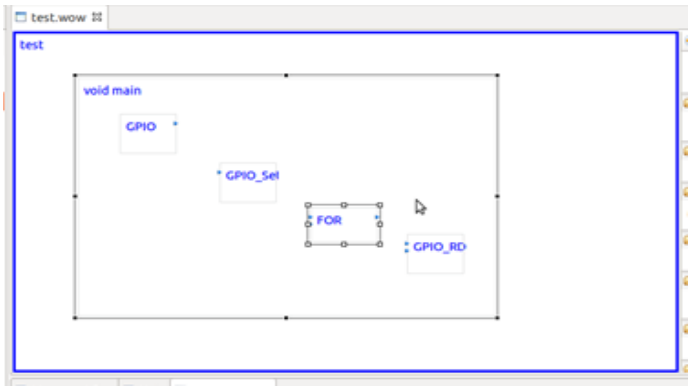

Figure 3. Componet Editior (Workspace for Software Design)

The important criterion here is to use the palette elements in a logical form to create a function software code as per the requirement. However the components and its relationship are designed in a way that it validates the principle to flow based programming. Because in FPB, the relation between processes is cooperative, rather than hierarchical, processes can easily be moved from one program or application to another or even from one machine to another [4].

B. Code Generation

One of the prominent ideas behind this entire concept is abstraction approach in terms of designing layered architecture of BSP of any hardware which will be reachable through some common code generated by an IDE is which is hardware independent. Thus the entire architecture can be attained by following a simple process of putting two additional layers after the generated code which helps the same to interact with the targeted hardware. [Figure 4]

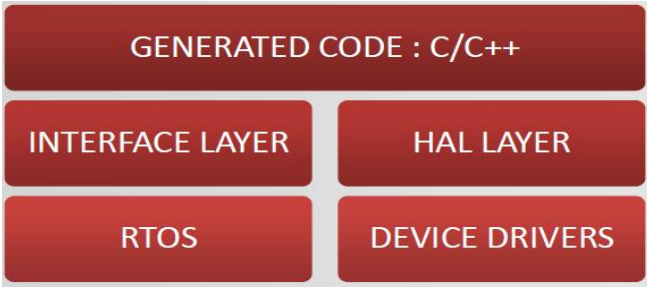

Figure 4. Generated Code Abstraction

This is an important aspect where we can make the generated code simple and hardware independent by introducing some interfaces into the BSP of a targeted hardware. Thus the IDE generated code is not any hardware specific and it interacts with the board specific code via various abstraction layers. This will benefit the environment in terms of graphical to code representation by making it easier. However, the main challenge is to develop interface libraries for each and every peripheral available on the board. In terms of complex application where the operating system functionality is needed, the same interface abstraction approach is implemented to incorporate all the functionalities of desired operating system. Once the architectural design has been made the associated code is generated in terms of .c files which represent the desired functionality of an embedded application. The related documents like header files and editor snapshot is also generated which helps the user to understand and modify the code in case of any errors or changes.

V. RESULTS

The best way to validate the proposed study is through implementing it to any embedded application and draws some fruitful conclusions. Thus, the software development process described above has been tested in the development of peripheral interfaces of the STM32 F3 Discovery Board. We have been developing libraries for every particular peripherals and one to one mapping is required to make a full fledge IDE in which any program can be written. It means you have to write an interface and wrapper for every available peripheral on the board.

Thus, we have been so far able to test and generate test cases for most of the peripheral on this board. The generate code successfully works on a board and is able to interact with the hardware by making changes in its graphical representation.

Consider a small example of GPIO peripheral on STM board which needs to be initialized as per the code given below [Figure 5].

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE, ENABLE);

GPIO_InitTypeDef GPIO_InitStructure;

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOE, &GPIO_InitStructure);
```

Figure 5. GPIO Hardware Specific Code

When this peripheral is initialized by the graphical representation, the hardware independent code is generated accordingly as shown in figure [Figure 6]. This both code works exactly with the same fashion on the hardware when downloaded.

```
gpioInterface = Stm32GPIOInterface;
GPIOConfig config;
config.pins = (1 << 11);
config.port = E;
config.mode = OUTPUT;
config.outtype = PP;
config.type = PuPdUP;
gpioInterface.InitGPIO(&config);
```

Figure 6. GPIO Hardware Specific Code

However the execution time may differ as there are more numbers of abstraction layers as compared to normal BSP thus it may cause problem in terms of time critical applications. In this case peripheral interface library is to be optimized for reducing execution time.

VI. CONCLUSIONS AND FUTURE WORK

We have presented here a novel design approach for developing a tool which aims to generate code for targeted embedded systems while using a user friendly graphical interface. We have here evaluated a small prototype of the perspective integrated development environment and presented small case studies on some of the peripherals to validate the entire concept.

Ongoing work on this is to facilitate this entire work with additional functionality in terms of component palette and software mapping rules for generation efficient code for targeted hardware. Addition to that we will also work on the user's ease of operation while using the environment and possible combinations for which logical code should be generated.

REFERENCES

[1] Thomas A. Henzinger and Joseph Sifakis, "The Embedded Systems Design Challenge" in FM 2006: Formal Methods: 14th International Symposium on Formal Methods, 2006

[2] Jimmie Wiklander, Jens Eliasson, Andrey Kruglyak, Per Lindgren, Johan Nordlander, "Enabling Component-Based Design for Embedded Real-Time Software", Journal of Computer, VOL. 4, NO. 12, December 2009

ISSN: 2321-8169
2835 - 2838

[3] S. Sentilles, A. Pettersson, D. Nystrom, T. Nolte, P. Pettersson, and I. Crnkovic, "SAV EIDE- a tool for design, analysis and implementation of component-based embedded systems. In Proc. ICSE'09, pages 607–610, 2009

[4] J. Paul Morrison J, "Flow Based Programming" Regular Technical Paper in Journal of Application Developers' News, 2010

[5] T. G. Moreira et al, ―Automatic code generation for embedded systems: From UML specifications to VHDL code, ―in Proc. of 8th IEEE International Conference on Industrial Informatics, IEEE Computer Society, 2010, pp. 1085–1090.

[6] Luo and Z. Huang. "Embedded C code generation and embedded target development based on RTW-EC". In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, pages 532 – 536, Chengdu, July 9-11 2010.

[7] M. A. Wehrmeister, E. P. Freitas, C. E. Pereira, and F. Ramming, F., "GenERTiCA: A Tool for Code Generation and Aspects Weaving". 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), Orlando, 2008.

[8] S. Karus and M. Dumas, "Designing Maintainable XML Transformations", in 14th European Conference on Software Maintenance and Reengineering (CSMR), 2010.

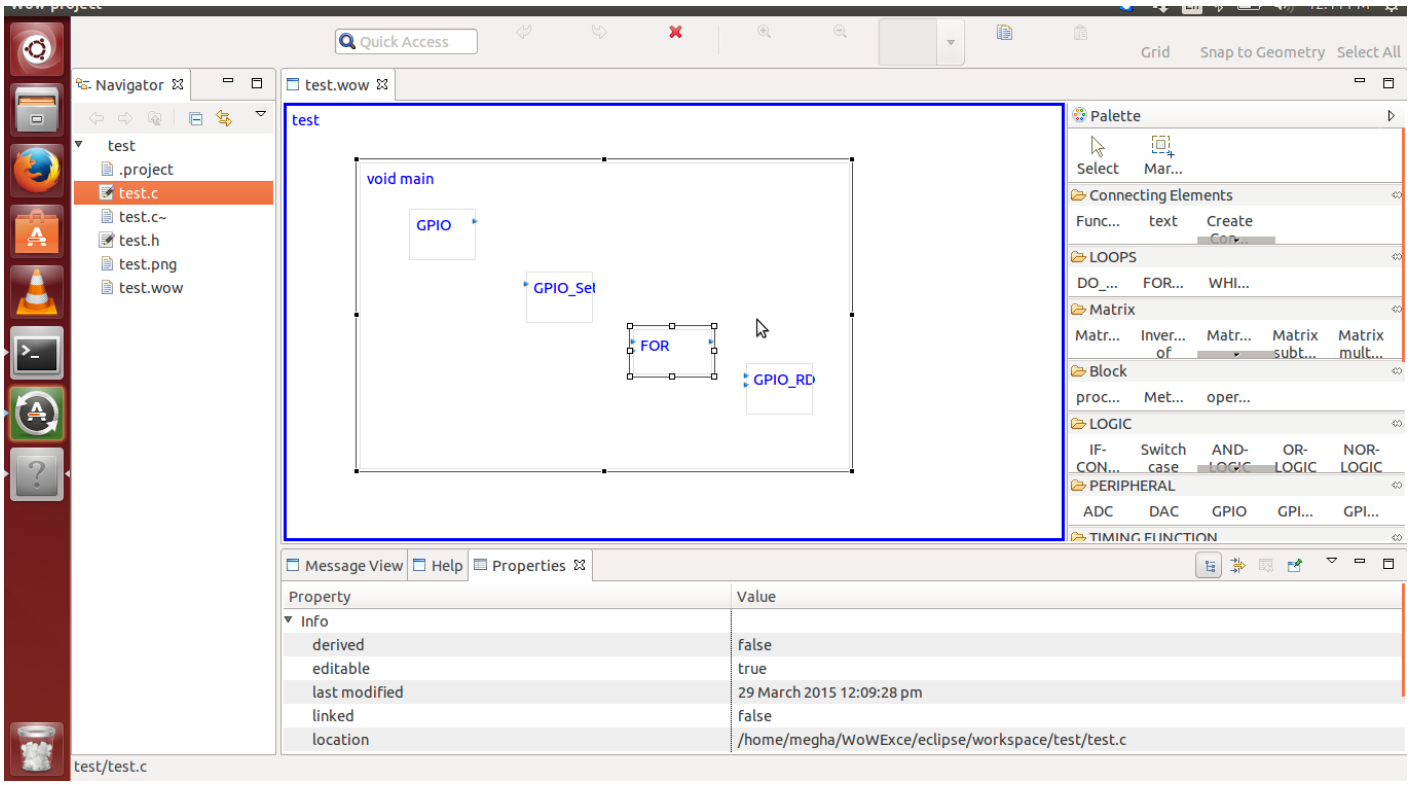

Figure 7. Comeplete Environment Representation