AN APPROACH TOWARDS INTEGRATING EMBEDDED CODE DEVELOPED USING HETEROGENEOUS PROGRAMMING LANGUAGES

Dr. JKR Sastry¹, Dr. A. Vinya Babu²

¹Department of Computer Science and Engineering, KL University
²Department of Computer Science and Engineering, JNTU Hyderabad

Abstract: Complex embedded systems having different response times that are related to different events happening in real-time have to be addressed through adapting Modular approach. Every important feature is implemented through a different embedded system and the designed is optimized to achieve desired response time. The individually developed embedded systems have to be integrated considering both hardware and software. While doing so the original response time requirements have to be met with. These days many languages which include C, C++, EJAVA, and ASSEMBLERS are being used for the development of embedded systems. When different embedded systems are to be integrated, the Embedded system code developed using different languages has to be integrated into a single language which can be cross compiled for a target Micro Controller and a chosen Real-time operating system. This paper presents a method integrating the ES code developed using different languages into C++ code which can be cross compiled for a target Micro Controller and a real-time operating system.

Keywords: Complex embedded system, heterogeneous ES languages, Software Integration, Integrating ES code

I. INTRODUCTION

It is important that every intelligent feature of a TAG has to be individually dealt with, to check and find its effectiveness while a TAG is responding to the changes taking place in the environment. Development and implementation of individual intelligent features through a different embedded board is absolutely necessary to measure the responsiveness and throughput. However all the individual embedded systems that implements a TAG must be integrated to form into a unified solution.

Today in wireless world a Real-time monitoring Tag plays a significant role in tracking high-value assets and people, in any environmental conditions. An intelligent TAG must have the capability to communicate with the mobile phone and gives the information regarding Tagged items. TAG can be attached to an item that must be protected. TAG should provide the security zone to the valuable assets.

The TAG is a single entity which consists of many components. Each TAG has specific set of features. TAG should be compatible with multiple communication systems and it should work under different standard networks like Bluetooth, Wi-Fi, NFC and Wi-Max. TAG can capture and transmit information regarding low battery state, tamper proofing, attacking, location etc. The Tags must be robust and can be operated in challenging environments. TAG life time depends on battery and it requires more power as it supports many functions and therefore it must have efficient power management system.

Different technologies are used to support each intelligent feature of the TAG. The Hardware and ES software that is used for implementing each of the features must be integrated to form into a single embedded system that supports all the intelligent features. Many integration issues are to be addressed for integrating the hardware, RTOS and ES software.

Multiple languages are used in the design and development of software for embedded systems under the influence of a single real time operating system and a microcontroller. Different languages are used for the development of ES software which includes C, C++, EJAVA, and many of the Assemblers.

Complex embedded systems include reactive as well as transformative functions which are tightly coupled. The two possible approaches for complex embedded systems are a heterogeneous multi-language specification consisting of several differently specified system parts or homogeneous specification using a single general language.

In language-based approach the embedded system has 1 to N subsystems, which may be described in different languages at different levels of abstraction and are individually designed and optimized. The subsystems are coupled as communicating processes via a common protocol.
“backbone.” The backbone protocol can be used for simulation as well as for implementation using communication synthesis. This two-level approach allows using existing tool environments for each of the subsystems. The language-based approach is the state of the art that is reflected in tools such as Co-ware which uses a client server approach to using a remote procedure call. These systems provide libraries with communication primitives that implement the backbone protocol. The individual processes can then be mapped to hardware or software components while the communication protocol is mapped to target system communication primitives, such as bus transactions or operating system functions. Object oriented languages, such as C++ or Java are well suited for this task as they expose just the interfaces with which communication can be undertaken. The language-based approach is a systematic and very flexible approach to system integration of individually designed subsystems.

The compositional approach combines the semantics of the subsystem languages to obtain a common representation; it can be used for global analysis and optimization. Compositional approaches provide deep subsystem integration by combining semantics of specification languages in a unified composition format. In Compositional approach the differently specified subsystems are translated into a composition format and merged to form a homogeneous representation of the complete system. Then, tool-supported system analysis and optimization is performed based on this homogeneous representation.

Integration of several individual modules into a single embedded system is a major issue in the designing of large embedded application. There exist several integration technologies for every integration issue, for every individual technology having its own pros and cons and it is important to investigate the suitable integration technology for developing an intelligent TAG.

Modular decomposition is one of the main strategies adapted when a very large size or complex embedded application has to be developed. Modular decomposition of large size embedded applications involves development of each of the model into an individual embedded application. Each module as such deals with one or more specific technologies.

Integration of all the individually developed embedded module based applications becomes thus necessary. The integration issues includes memory integration, module integration and interfacing, device interfacing, hardware integration and isolation, architecture integration, application segmentation which includes hardware and software partitioning, power management, code integration, where several languages are used for developing an embedded application, RTOS integration and migration, integration of task scheduling ,integration of interrupt handling and performance optimization. The integrating of all individual embedded systems into a major embedded system is a complex process as it has to deal with very many integrating issues.

Therefore investigating the integration of the several of the embedded system into a single large embedded application through a challenge and therefore needs to be investigated and suitable solutions are to be presented. Not many solutions exist that address completely the integration of the individual modules into a single embedded system and therefore becomes necessary to investigate a comprehensive integration solution.

When a single embedded system is to be developed by integrating several of the embedded systems considering both Hardware and Software, different issues are to be considered which include different development technologies, performance, throughput, programing languages and different RTOS using which different embedded systems are developed.

Hardware integration and ES software integration are the two most important considerations that must be addressed to start with followed by code optimization. Both the issues are complicated. Hardware integration should consider different types of micro controllers, devices and interfacing of devices with micro controllers etc. Software integration includes the code developed in different languages, usage of different operating systems etc.

Many of the languages these days are predominantly object oriented languages and some of these languages have support for implementing either the C code or Assembler Code using multi language code inclusion interface. Especially the C++ language provides for support for including C and assembler Code.

The integration process involves three phases which include integration, optimization and performance evaluation. The computational time required for each of the code elements can be computed and the response time for handling each of events is calculated by adding the computing times of the code segments that are involved in processing an event.

In this paper, an approach has been presented that help integrating ES code related to different embedded systems developed using C, C++, EJAVA and ARM7 related assembler language.

II. RELATED WORK

Rolf Ernst [1] proposed the use of several languages in the design of embedded systems which are very convenient for application development and optimization but it can become an obstacle on the way to higher design productivity. Most of complex embedded systems are developed using several languages and there is a need to either combine system parts described in different languages or to find a single general language. They have demonstrated two approaches to combine different languages, the language-based approach which couples models in different languages using a fixed communication protocol, and the compositional approach which combines different model semantics on a unified internal representation.

Jerraya [2] proposed specification languages and intermediate models used for system-level design. The system-level specification of a mixed hardware/software application may follow one of two schemes which include homogeneous specification where a single language is used for the specification of the overall system including hardware parts and software parts and another model which uses heterogeneous modeling specific languages suited for
hardware parts and software parts, a typical example is the mixed C-VHDL model.

There exist several system specification languages, each of these languages is more efficient for a given application domain. For instance some of these languages are more adapted to the specification of state-based specification (SDL, State chart), some others are more suited for data flow and continuous computation (LUSTRE, Matlab), while many others are more suitable for algorithmic description (C, C++).

The use of Multilanguage specification requires new validation techniques to be able to handle a multi paradigm model. Instead of simulation, co-simulation and instead of verification co-verification is needed. System specification is based on four basic concepts; they are concurrency, hierarchy, communication and synchronization.

No unique universal specification language exists as on today that support all kinds of embedded applications. A specification language is generally selected according to the application. In fact, the use of more than one language corresponds to an actual need in embedded systems design.

The design of heterogeneous systems may require the combination of several specification languages for the design of different parts of the system. The key issues in this case are Multilanguage validation, co-simulation and interfacing. Multi-language co-simulation aims at executing several models given in different languages in concurrent way. The concept of multi-language specification aims at coordinating different modules described in different languages, formalisms, and notations.

**Thomas Kuhn** [3] has described domain specific languages (DSLs) which are used to design parts of embedded systems. These languages are highly specialized and often tailored to one domain; one single language therefore cannot describe all relevant aspects of systems and system components. This raises the need for heterogeneous modeling approaches that are capable of combining multiple DSLs into holistic system models.

The CompoSE modeling approach focuses on this problem; it does not only cover system modeling with DSLs, but provides also interfacing of language specific generators and harmonization of generated code. The principles of CompoSE together with the integration of an existing modeling language provide the basis for support of code that is developed using heterogeneous programming languages.

**Dr.-Ing** [4) explained about an important trend in embedded system design is increase in system complexity resulting into heterogeneity with respect to different functions and components of an embedded system. This means that functions from different application domains are tightly coupled in a single embedded system. The key problems in the context of multi-language design are the safe integration of the differently specified subsystems and the optimized implementation of the whole system.

Current multi-language design approaches can be classified into co-simulation and compositional approaches. Co-simulation approaches provide a flexible and systematic integration for heterogeneously specified systems. Compositional approaches provide a deeper integration of the different system parts by creating a coherent formalism for the representation of the complete system at a higher level of abstraction. A novel approach to the design of heterogeneously specified, complex embedded systems has been proposed. By assuming a truly heterogeneous Multilanguage specification while nevertheless providing an abstract homogeneous design representation supporting system-wide analysis and optimization, this approach combines the advantages of both co-simulation and compositional approaches.

**Dirk Ziegenbein** [5] have described the System property intervals (SPI) workbench which is an open framework for embedded system analysis and synthesis from heterogeneous specifications. The design of complex embedded systems typically requires combining multiple models of computation for different application domains. Two classes of approaches exist to model such heterogeneous systems. In the first class of approaches, a single super-language is used. In the second class of approaches, multiple models of computation or languages are used, each for a different part of the system. SPI is an internal high-level representation that facilitates global system-level analysis, optimization and synthesis of heterogeneously specified embedded systems.

**R. Ernst** [6] has presented the SPI model which is based on intervals of system properties and is specifically targeted to co-synthesis. Embedded systems typically include reactive and transformative functions, often described in different languages and semantics which are well introduced in various application fields. A large part of the system functionality and components is reused from previous designs including legacy code. There is little hope that a single language will replace this heterogeneous set of languages. A hardware/software co-design process must be able to bridge the semantic differences for verification and synthesis and should accept limited knowledge of system properties.

**Guilherme Bertoni** [7] have stated that the embedded applications, which were originally built on standalone devices, now a day, require a growing integration with other systems through their interconnection with TCP/IP networks. Web Services, which provide a service oriented distributed architecture for the interconnection of systems through TCP/IP networks, have been widely adopted for the integration of business applications, but this sort of integration is still not provided by embedded applications. The feasibility of using Web Services for the integration of embedded applications running on heterogeneous architectures has been presented.

**Abel Marrero P`erez** [8] have presented that the increasing importance of embedded software has produced a shift in the testing activities from system testing towards software testing. They have presented large embedded system integration during testing level and present a novel approach for describing multi-level test case integration from the point of view of the system’s functionality. As a consequence, single test case specifications and implementations are reused throughout the test process, minimizing the test implementation effort and taking advantage of the synergies among test levels.

**NI Lab VIEW** [9] is used to interface measurement and control devices. LabView integrates seamlessly with
thousands of different hardware devices, and helps save development time with convenient features and a consistent programming framework across all hardware. Integrating different hardware devices with traditional tools is littered with time-wasting steps and possible incompatibilities, increasing risk. One has to find the correct drivers for all hardware, and then to figure out how to install them and call them from software. Once the drivers are usable, they have to communicate with the hardware and learn the programming which was appropriate for that particular device. LabView can help save time and frustration by eliminating some of these steps and making others markedly easier. LabView is one software tool that can span all of the hardware components. Drivers are readily available for common hardware devices.

NI Multisim [10] software provides simulation-driven instruments that one can use to drive the circuit, measure the behavior of the circuit, and examine simulation results. These instruments are set, used, and read just like their real-world equivalents. In addition to the components and wires used to capture a circuit, NI Multisim software contains a variety of simulation-driven instruments that wire into the schematic just like the way real instrument on the bench works. These simulation-driven instruments, like their real-world counterparts, are fully interactive so one can change their settings while running a simulation and instantly see new results. Simulation-driven instruments help you take advantage of the full power of simulation without having to be an expert in “SPICE” syntax. LabView extend the simulation and analysis capabilities beyond the instrumentation in Multisim. Multisim as such has no feature in it to simulate the integration of embedded system boards.

The flat and hierarchical Schematic Page Editor of ORCAD (Capture Component Information System (CIS)) [11] combines an intuitive interface with the features and functionality needed to speed design tasks and facilitate circuit creation. For larger, more complex designs, ORCAD Capture supports multi-sheet and hierarchical designs. It also makes hierarchical designs easy to traverse and ensures that all connections are maintained accurately throughout the design. Seamless interfaces establish robust data paths and integration with ORCAD PCB Editor for physical PCB design and with Cadence PSpice® A/D for analog/digital circuit simulation. Seamless bi-directional integration with OrCAD PCB Editor enables synchronization and cross-probing/placing between the schematic and the board, and automated engineering change orders (ECOs) back annotate layout changes, gate/pin swaps, and changes to component names or values.

Most of the solutions proposed in the literature do not take into the account the heterogeneous Hardware platforms especially the Micro controllers, the languages used for the development of embedded software and real time operating systems used for the development of real time applications.

Sastry [12] have presented a method that helps integrating hardware seamlessly. The method has been is used for development of different embedded boards based on ARM technologies. Sastry [13] have presented another method for integrating ES software which is developed using different languages.

III. An approach towards integrating ES code developed using heterogeneous languages

The following procedure when followed will integrate the ES software developed in C, C++, EJAVA, and ARM7 into C++ Language to run under μcos real-time operating system.

IIIA Code conversion into C++ from EJAVA

1. All the static variable defined in various Java defined classes be copied as global static variables as per C++ syntax
2. The main method included into the class having the name same as the name of the file in which the class is contained be removed and the code be included into the main nonmember function of C++
3. The class that extends the thread class be converted into a TASK and the priority defined for the thread be included into the functional call related to Task creation under μcos real-time operating system.
4. All the classes that are defined in JAVA be copied into the C++ code with proper inheritance syntax used in JAVA (Public, Packaged, Protected) converted into C++ (Public, Packaged, Protected)
5. Tasks are created in C++ through calling μcos function (Task Create) from the main function. The tasks created are similar to the tasks created in the JAVA code through calling a μcos function (Task Create). The inclusion of the classes into the tasks in C++ code is undertaken based on the parameters passed to taskcreate () function.
6. The size of stack for each of the task is derived by computing the size of the parameters passed to the functions that are contained in each of the class and the stack size is used as parameter passed as an argument to taskcreate () function.
7. Drop the common classes especially related to devices which include LCD, Buzzer, Wi-Fi and Bluetooth communication devices and replace function call sequence defined in JAVA with the functional call sequences defined in C++

IIIB Code conversion into C++ from C

1. All the static variable defined in various functions are defined as global static variables as per C++ syntax if the same has not been defined earlier
2. The Code contained in the main method of the C program be included into the main method of C++ program. The RTOS related initialization statements are ignored and task creation related statements are copied after the task creation statements contained in the C++ program
3. Include the extern “c” [function-prototype] statement in respect of all the functions contained in the C program
4. Include namespace statement as C-Name space
5. Copy all the C functions which are not related to nonmember functions
6. Drop the functions especially related to devices which include LCD, Buzzer, Wi-Fi and Bluetooth communication devices and replace function calls through C++ related object initiation and calling the related function through objects initiated.
7. If the name of the function is same as one of the functions of a class and the code is also same then the function is dropped
8. If the name of the function is same, and the code is not the same, then in that case the code is included into the integrating program qualified with the type of intelligent feature
9. If new function appears related to the same device then the function is added to the class as a new function
10. If few functions are added related to a new device then a class is created which is included with the functions related to the device

TIIIC. Code conversion into C++ from Assembler

1. Include all the assembler code using the following syntax into the main function after the all the RTOS initialization instructions are coded.

```c
asm
{
asm code
}
```

IV. EXPERIMENTATION

Three embedded systems that implement 3 distinct intelligent features (Tag identification and Location Identification) have been considered which are developed using C++ and C and the same are integrated using the method stated in the Section III into C++ Language. Each of the ES application has two segments one that works on the TAG side and the other working on the HOST side (Mobile Phone). The code integration shown below is related to TAG side integration. Due to space limitations, the integration of C with C++ considering the Tag identification (C++) and the location identification (C code) embedded systems has been shown below:

IVA Tag Identification program in C++

```c
namespace using std;
#include <LPC214x.H>
#include <watermarking.h>
#include "template.c"
#define RS 0x10000
#define RW 0x20000
#define EN 0x40000
#define CR 0x0D

unsigned char msg[] = {'s','e','n','d','i','n','g',' ','t','e','m','p','l','a','t','e',' ','d','e','v','i','c','e',' ','i','d','e','n','t','i','f','i','ed',' ','x',' '};
unsigned char msgf[] = {'t','e','m','p','l','a','t','e',' ','m','a,'t,c,'h','e','d',' ', 'd','e','v','i','c','e',' ',i,'d','e','n','t','i','f','i','ed', 'x',' '};

void Serial_Init(void);
unsigned int Receive(void);
void LCD_init4(void);
void LCD_cmd4(unsigned char);
void LCD_cmd4(unsigned char*);
voidDelayMs(unsigned char*);
inputchar(int);

unsigned long int DATA;
unsigned char Chr;i=0;

intQRCODE = 123;
intwatermarkcode;

void main()
{

DelayMs(10);
buzzerclassbuzzer1;
Serial-port1sp1;
Sp1. Serial_Init();
lcdclasslcd1;
lcd1. LCD_init4();
DelayMs(100);
lcd1. LCD_cmd4(0x80);
lcd1. LCD_puts(msg);

watermarkcode=watermarking (QRCODE);

str = str + "X";
msg1=str;
while(1)
{
i=0;
chr = msg1[i];
while(chr != 'x')
{
serialport. putchar (chr);
i++;
chr = msg1[i];
}
DelayMs(3000);
Chr=serialport1. Receive();
switch(Chr)
{
case 0x33: buzzer1.buzzeroff(); break;
case 0x34: buzzer1.buzzeroff (); break;
}
}

classLCDclass
{
voidLCD_init4(void)
{
IO0DIR = 0xFFFF0000;
LCD_cmd4(0x33);
LCD_cmd4(0x22);
LCD_cmd4(0x22);
LCD_cmd4(0x28);
LCD_cmd4(0x06);
LCD_cmd4(0x0c);
LCD_cmd4(0x01);
}

voidLCD_cmd4(unsigned long intcmd)
{
DATA = ((cmd<<<15) &0x00780000);
IOCLR0 = ((DATA^0xFFFFFFFF) &0x00780000) | RS | RW;
IOSET0 = DATA | EN;
IOCLR0 = EN;
DATA = ((cmd<<<19) &0x00780000);
IOCLR0 = ((DATA^0xFFFFFFFF) &0x00780000) | RS | RW;
IOSET0 = DATA | EN;
IOCLR0 = EN;
```
DelayMs(3);
}

void LCD_dat4(unsigned char byte) {
    DATA = ((byte<<15) &0x00780000);
    IOCLR0 = ((DATA*0x00000000)&0x00780000) | RW;
    ISET0 = DATA | RS | EN;
    IOCLR0 = EN;
    DATA = ((byte<<19) &0x00780000);
    IOCLR0 = ((DATA*0x00000000)&0x00780000) | RW;
    ISET0 = DATA | RS | EN;
    IOCLR0 = EN;
    DelayMs(3);
}

void LCD.puts(unsigned char *string) {
    while(*string)LCD_dat4(*string++);
}

void DelayMs(unsigned intMs) // Task Delay
{
    int delay_cnt;
    while(Ms>0)
    {
        Ms--;  
        for(delay_cnt = 0;delay_cnt<220;delay_cnt++);
    }
}

Class serialPort1
{
    inputchar (inbyte) /* Write character to Serial Port */
    {
        if (ch=='x')
            while (!(U0LSR&0x0c20));
        U0THR = CR; /* output CR */
    }
    return(U0THR = ch);
}

void Serial_Init(void)
{
    PINSEL0 = 0x00000005; /* Enable Txd0 and Rxd0 */
    U0LCR = 0x00000083; /* 8-bit data, no parity, 1-stop bit */
    U0DLL = 0x00000061; /* for Baud rate=9600,DLL=82 */
    U0LCR = 0x00000063; /* /DLAB = 0; */
}

unsigned int Receive(void) /* Read character from Serial Port */
{
    while (!(U0LSR&0x001));
    return(U0RBR);
}

Class buzzerclassbuzzer1
{
    buzzeron()
    {
        ISET0 |= 0x0080;
    }
    buzzeroff()
    {
        IOCLR0 |= 0x0080
    }
}

/* IVB Location identification code in C */

#include <LPC214x.H>
#include <string.h>
#define RS 0x10000
#define RW0x20000
#define EN 0x40000
#define CR 0x0D

unsigned char msg[] = {"INTELLIGENT TAG "};
/views
unsigned char msg2[32];

void Serial_Init(void);
unsigned int Receive1(void);

void LCD_init4(void);
void LCD_cmd4(unsigned long int);
void LCD_dat4(unsigned char);
void LCD_puts1(unsigned char *, unsigned char);

void DelayMs(unsigned int);

void getUSART1(unsigned char *);
void Get_GPS_USART1(unsigned char *);
inputchar (int);
unsigned long int DATA;
unsigned char Ch, Chr, bi=0,i=0,J=0,Str_GPS[100],*ptr,
LAT]="LAT";LAN[]="LAN";

void main()
{

    DelayMs(10);
    Serial_Init();
    LCD_init4();
    DelayMs(100);
    LCD_cmd4(0x80);
    LCD_puts1(msg,16);
    Get_GPS_USART1(Str_GPS);
    LCD_cmd4(0x80);
    LCD_puts1(LAT,4);
    LCD_puts1(Str_GPS+20,11);
    ptr = ptr+13;
    LCD_cmd4(0x40);
    LCD_puts1(LAN,4);
    LCD_puts1(Str_GPS+33,11);
    bi=0;
    while(bi<50)
    {
         Chr=Str_GPS[bi];
         patchar(Chr);
         bi++;
    }
    bi=0;
}

void Serial_Init(void)
{
    PINSEL0 = 0x00050005; /* Enable Txd0 and Rxd0 */
    U1LCR = 0x00000083; /* 8-bit data, no parity, 1-stop bit */
    U1DLL = 0x00000061; /* for Baud rate=9600,DLL=82 */
    U1LCR = 0x00000063; /* /DLAB = 0; */
    //for Baud rate=9600,DLL=82
    //DLAB = 0;

unsigned int Receive1(void) /* Read character from Serial Port */
{
    while (!(U1LSR&0x01));
    return (U1RBR);
}

void LCD_init4(void)
{
    IOODIR = 0xFFFFFFF;
    LCD_cmd4(0x33);
    LCD_cmd4(0x22);
    LCD_cmd4(0x22);
    LCD_cmd4(0x28);
    LCD_cmd4(0x06);
    LCD_cmd4(0x0c);
    LCD_cmd4(0x01);
}

void LCD_cmd4(unsigned long int cmd4)
{
    DATA = ((cmd4<15) & 0x00780000);
    IOCLR0 = ((DATA<<4)||0x00780000) | RS | RW;
    IOSETO = DATA | EN;
    IOCLR0 |= EN;
    DATA = ((cmd4<19) & 0x00780000);
    IOCLR0 = ((DATA<<4)||0x00780000) | RS | RW;
    IOSETO = DATA | EN;
    IOCLR0 |= EN;
    DelayMs(3);
}

void LCD_dat4(unsigned char byte)
{
    DATA = ((byte<15) & 0x00780000);
    IOCLR0 = ((DATA<<4)||0x00780000) | RS | RW;
    IOSETO = DATA | EN;
    IOCLR0 |= EN;
    DATA = ((byte<19) & 0x00780000);
    IOCLR0 = ((DATA<<4)||0x00780000) | RS | RW;
    IOSETO = DATA | EN;
    IOCLR0 |= EN;
    DelayMs(3);
}

void LCD_puts1(unsigned char *string, unsigned char n)
{
    while(*string) & ((n--)>0)
        LCD_dat4(*string++);
}

void gets_USART1(unsigned char *string)
{
    // The String Must Ended with "Carriage return" i.e "Enter key"
    unsigned char i=0, j=0;
    do
    {
        *(string++)=Receive1();
        J = *(string+i);
        i++;
        while((j==~u') & (J==~u'));
        i++;
        *(string+i) = '0';
    }

    voidGet_GPS_USART1(unsigned char *GPS_Str) / * Read GPS Str */
    {
    ptr = strstr(GPS_Str,"GPRMC")+19;
    }

    voidDelayMs(unsigned intMs)
    {
        int delcst_n;
        while(Ms>0)
        {
            Ms--;
            for(delcst_n = 0; delcst_n<220; delcst_n++);
        }
        inputchar(intCh) /* Write character to Serial Port */
    }

    voidSerial_Init(void);
    unsigned int Receive(void);
    voidLCD_init4(void);
    void LCD_cmd4(unsigned long int); 
    voidLCD_dat4(unsigned char);
    void LCD_puts1(unsigned char *, unsigned char);
    voidDelayMs(unsigned int);
    void gets_USART1(unsigned char *);
    void Get_GPS_USART1(unsigned char *);
    inputchar(int);
    unsigned long int DATA;
    unsigned char Ch, Chr, bi=0, i=0, j=0, Str_GPS[100], *ptr,
    LAT[]="LAT:\",LAN[]="LAN:\
    unsigned int intelligentissueNo=0;
    unsigned char msg[] = ("<INTELLIGENT TAG >");
    unsigned char msg[2][32];
    voidSerial_Init(void);
    unsigned int Receive(void);
    voidLCD_init4(void);
    void LCD_cmd4(unsigned long int);
    voidLCD_dat4(unsigned char);
    void LCD.puts1(unsigned char *, unsigned char);
    voidDelayMs(unsigned int);
    inputchar(int);
    unsigned long int DATA;
    unsigned char Chr,i=0;
    int QRCode = 123;
    int watermarkcode;

    namespace using std;
    #include <LPC214x.H>
    #include <watermarking.h>
    #include "template.c"
    #define RS 0x10000
    #define RW 0x20000
    #define EN 0x40000
    #define CR 0x0D
    namespace Location + Tag identification code in C++

    namespace using std;
    #include <LPC214x.H>
    #include <watermarking.h>
    #include "template.c"
    #define RS 0x10000
    #define RW 0x20000
    #define EN 0x40000
    #define CR 0x0D
    unsigned char msg[] = ("<INTELLIGENT TAG >");
    unsigned char msg[2][32];
    voidSerial_Init(void);
    unsigned int Receive(void);
    voidLCD_init4(void);
    void LCD_cmd4(unsigned long int);
    voidLCD_dat4(unsigned char);
    void LCD.puts1(unsigned char *, unsigned char);
    voidDelayMs(unsigned int);
    void gets_USART1(unsigned char *);
    void Get_GPS_USART1(unsigned char *);
    inputchar(int);
    unsigned long int DATA;
    unsigned char Ch, Chr, bi=0, i=0, j=0, Str_GPS[100], *ptr,
    LAT[]="LAT:\",LAN[]="LAN:\
    unsigned int intelligentissueNo=0;
    unsigned char msg[] = ("<sending template... >");
    unsigned char msg[] = ("<template matched device identified x >");
    voidSerial_Init(void);
    unsigned int Receive(void);
    voidLCD_init4(void);
    void LCD_cmd4(unsigned long int);
    voidLCD_dat4(unsigned char);
    void LCD.puts1(unsigned char *, unsigned char);
    voidDelayMs(unsigned int);
    inputchar(int);
    unsigned long int DATA;
    unsigned char Chr,i=0;
    int QRCode = 123;
    int watermarkcode;
void main()  
{
  DelayMs(10);
  buzzerclassbuzzer1;
  gpsclassgps1

  Serial-port1sp1;

  Sp1. Serial_Init();
  lcdclasslcd1;
  lcd1. LCD_int4();
  DelayMs (100);
  Led1. LCD_cmd4(0x80);
  Led1. LCD_puts(msg);

  watermarkcode=watermarking (QRCODE);

  string str (watermarkcode);
  str = str + ‘X’;
  msg1=str

  while(1)
  {
    i=0;
    chr = msg1[i];
    while(chr != ‘x’)
      {
        serialport. putchar (chr);
        i++;
        chr = msg1[i];
      }
  }

  DelayMs (3000);

  Ch=serialport1. Receive ();
  switch(Chr)
  {
    case0x33: buzzer1.buzzeroфф(); break; /* buzzer on-from mobile press 3*/
    case0x34: buzzer1.buzzeroфф(); break; /* buzzer off-from mobile press 4*/
  }
}

voidLCD_class(void)
{
  if (intelligentIssueNo = 2)
  {
    IO0DIR = 0xFFFFFFF;
    LCD_cmd4(0x22);
    LCD_cmd4(0x28);
    LCD_cmd4(0x06);
    LCD_cmd4(0x0c);
  }

  if (intelligentIssueNo = 1)
  {
    IO0DIR = 0xFFFFFFF;
    LCD_cmd4(0x33);
    LCD_cmd4(0x22);
  }

  voidLCD_cmd4(unsigned long intcmd)
  {
    DATA = ((cmd<15) &0x00780000);
    IOCLR0 = (DATA^0xFFFFFFF)&0x00780000) | RS | RW;
    IOSET0 = DATA | EN;
    IOCLR0 |= EN;
    DATA = ((cmd<15) &0x00780000);
    IOCLR0 = (DATA^0xFFFFFFF)&0x00780000) | RS | RW;
    IOSET0 = DATA | EN;
    IOCLR0 |= EN;
    DelayMs(3);
  }

  voidLCD_dat4(unsigned char byte)
  {
    DATA = ((byte<15) &0x00780000);
    IOCLR0 = (DATA^0xFFFFFFF)&0x00780000) | RW;
    IOSET0 = DATA | RS | EN;
    IOCLR0 |= EN;
    DATA = ((byte<15) &0x00780000);
    IOCLR0 = (DATA^0xFFFFFFF)&0x00780000) | RW;
    IOSET0 = DATA | RS | EN;
    IOCLR0 |= EN;
    DelayMs(3);
  }

  voidLCD_puts1(unsigned char *string, unsigned char n)
  {
    while(*string) &(& (~n==0))
      LCD.dat4(*string++);
  }

  voidDelayMs(unsigned intMs) // Task Delay
  {
    intdelay_cnt;
    while(Ms>0)
    {
      Ms--;
      for(delay_cnt = 0;delay_cnt<220;delay_cnt++);
    }
  }

  voidinputchar (inch) /* Write character to Serial Port */
  {
  }

  if (intelligentIssueNo=2)
  {
    if (ch == ‘x’)
      {
        while ((U0LSR&0x20));
        U0THR = CR; /* output CR */
      }
  }

  }
while (!(U0LSR&0x20));
return (U0THR = ch);
}

if (intelligentIssueNo=1)
{
while (!(U0LSR&0x20));
return (U0THR = Ch);
}

voidSerial_Init(void)
{
PINSEL0 |= 0x00000005;
// Enable Txd0 and Rxd0
U0LCR = 0x00000083;
// 8-bit data, no parity, 1-stop hit
U0DLL = 0x00000061;
// for Baud rate=9600, DLL=82
U0LCR = 0x00000003;
// DLAB = 0;
}

unsigned int Receive(void) /* Read character from Serial Port */
{
while (!(U0LSR&0x20));
return (U0RBR);
}

Class buzzerclass buzzer1
{
buzzeron()
{
IOSET0 |= 0x0080;
}
buzzero()
{
IOCLR0 |= 0x0080
}
}

classgpsclass
{
void gets_USART1(unsigned char *string) // Receive a batch of characters via USART1, without interrupt
{
// The String Must Ended with "Carriage return" ie "Enter key"
unsigned char i=0,j=0;

do

{ *(string++)=Receive1();
  j = *(string++);
  i++;
}

while((J!='a') & (J!='v'));

*(string++)='0';

voidGet_GPS_USART1(unsigned char *GPS_Str)
{
gets_USART1(GPS_Str);
while( !(strstr(GPS_Str,"GPRMC"))) gets_USART1(GPS_Str);
ptr = strstr(GPS_Str,"GPRMC") + 19;
}

V. CODE OPTIMIZATION THROUGH PERFORMANCE EVALUATION

When code is integrated, it is quite possible that the response times which are originally designed for individual embedded system must be effected. It is necessary to compute the response time for each of the event occurring. The method proposed by Sastry [14] can be used for estimating the performance of the integrated system. All the events are identified and the sequence in which tasks are executed are pre-decided or traced from the integrated code. The computational time in respect of the code element contained within each of the tasks is calculated adding them all together to get the computational time of each of the tasks.

When an event happens, certain hardware devices and software components are used for processing the events. The processing through Hardware devices and Software components takes place in a sequence affecting a specific flow sequence. Response time for each of the event is calculated considering the latency times of HW and processing time of the software considering the sequence in which the HW and software elements are used for processing the events.

If the computed response times are not in acceptable ranges, then the design process has to consider the up-gradation of either the hardware or reducing the delay times of the software or through following any of the design principles as stated by Simon [15].

VI. CONCLUSIONS

When a complex embedded system is developed by modularization and developing multiple embedded systems, an integration approach has to be adapted for developing unified hardware and software. It is necessary to consider many languages using which the ES software is developed for individual embedded systems and an integration strategy is adapted. The integration issue must consider all the language specific issues which are designed to run under specific RTOS and a Microcontroller. In this paper an assumption is made that all the functions of RTOS are made available as different library packages which are included into the integrated code. C++ is used as integration language as C++ has language support for inclusion of C and Assembler code related to a specific Micro controller.

REFERENCES


[10]. mathworks.in, 2009, "Linking Systems”, www.mathworks.in/products/simulink,


