ISTANBUL UNIVERSITY ENGINEERING FACULTY
JOURNAL OF ELECTRICAL & ELECTRONICS

YEAR : 2002 (593-599)
VOLUME : 2
NUMBER : 2

OBJECT ORIENTED PROGRAMMING IN MICROCONTROLLER BASED SYSTEMS WITH EXTREMELY LIMITED RESOURCES

Suha FUTACI

Senior Project Manager
European Semiconductor Group
Motorola Semiconductor Products Sector

E-mail: suha.futaci@motorola.com

ABSTRACT

In many microcontroller based embedded system projects the object oriented programming is not considered because of the extremely limited resources and relatively large code generated by the compilers. Usually C, assembly and mixed C-Assembly language programming is dominating these applications. In some cases even the size of the code generated by a very efficient C compiler is unacceptable. So in many projects the advantages of the object oriented programming are not realized.

In this paper, the use of object oriented programming techniques in the absence of sufficient resources is emphasized. There are significant driving factors limiting the cost of the microcontrollers and limiting their resources. On the other hand consumers expects more functions from the microcontroller applications. It is shown that object oriented analysis/design techniques and programming can still be used to the extent that most of the benefits of object-oriented programming can be realized if the suitable mechanisms are employed.

Keywords: *Object-Oriented Programming, Object-Oriented design, Real-time System, Microcontroller*

1. INTRODUCTION

In today's industry, Microcontrollers (MCU's- Microcontroller Units) are playing an important role in the embedded systems. Among the many major applications areas in which MCU's are heavily used are white goods, computer peripherals, electrical appliances, air conditioning systems, remote controls, transportation systems, and car electronics.

A microcontroller is a microprocessor with the processing core and all peripherals, such as, general purpose input-output, RAM, ROM, asynchronous serial communication, keyboard interface and timers, are integrated into one

Received Date : 30.5.2002
Accepted Date: 04.6.2002

integrated circuit, so, in fact, it is a singlechip microcomputer with limited resources.

The microcontrollers are being used in volume, compared to the microprocessors of desktop platforms. By the year 2001 Motorola sold five billion MC68HC705KJ1 microcontroller, only a member of a large family of 8-bit HC05 family of microcontrollers. A white good manufacturer typically uses a million MCU's a year. This brings in the challenge of reducing the cost of microcontrollers. The resources such as memory, central processing unit (CPU) speed are then reduced to the point that they are merely sufficient for the intended applications.
The software of these platforms gets complicated as more functions are added. Simple microcontroller applications are being replaced with more intelligent ones that are even interfacing to Internet. It becomes common to have multiple microcontrollers working in parallel and communicating over a bus, for example, in today's car, it is common to find more than 40 microcontrollers serving as, engine controller, break controller, power steering, driver seat controller, electric mirrors controller, all communicating over a bus. An elevator control system has many microcontrollers. One at the central control unit, many at the buttons on each floor, one or more at the cabinets and again, all are in continuous communication with each other over a communication network. In this area reuse of robust and tested software components of prior applications is of great desire. [12]

We sometimes think these controls should be more intelligent in that we expect more than they are able to deliver to us. In some applications, the associated software deserve more attention than the software we use in other areas. Imagine a microcontroller controlling a medical respiratory support system by receiving sensory information from a pressure sensor and makes an error, or the one serving as oven timer, cooks more than intended. How about the break system happens to execute a subroutine a few milliseconds later. We can give many examples of this kind.

There exist a delicate balance between the capability of applications and the resources that can support the applications. Since the unit cost is very important for the manufacturer and the consumer expects more functions from the applications, the techniques we had used to save resources in the dark ages of computing are all being used in this arena.

Application writers in the process of fitting an application in this platform some times make hard decisions diverting from the holly principles of computer science. Sometimes the architecture of the microcontroller, optimized for cost reduction, forces them to work around. Hence it is in this field where we need to use the object oriented programming methods to increase the software quality, reliability, reusability and maintainability. The research challenges in the field of component-based software engineering for resource-constrained systems are defined in [5].

2.OBJECT-ORIENTED PROGRAMMING IN EMBEDDED REAL TIME SYSTEMS

The research for applying object oriented programming in real time embedded systems focuses on object-oriented design by using extended UML specifications [9] and programming in Embedded C++ (EC++) [2], [3],[4] or Java[6], [10] on these platforms.
Today object-oriented design is mainly done using UML (Unified Modeling Language). The timing constraints in real time systems necessitate extending the UML to express the real time behavior. For example sequence diagrams describing the interactions between system components may be annotated with timing constraints. [9] MAST_RT View, a new view in the UML description of a system, models its real time behavior. [8]. There is an effort going on in OMG (Object Management Group) companies for elaborating the UML for real time. [1] The component based software engineering aiming at applying component technology in resource-constrained-embedded systems is an active research area. [5]

On the programming side, EC++ (Embedded C++) proves to be a good enhancement of traditional C++ for saving embedded resources. In a comparison done between an EC++ library and a standard C++ library, it was found out that the size of executables generated by EC++ are 33% to 90% of traditional C++ executables in most of the cases [11]. If the code plus data is compared, this figure ranges between %15 and %48. [11]. More material can be found at the

web site of "Embedded C++ Technical Committee"[2],[3],[4].

Java, a widely used object oriented language, is also being targeted for embedded real time platforms. J2ME (Java Version 2 Micro Edition) as defined in Sun Micro Systems Inc. white paper [6] uses a small JVM called KVM that still needs 128K to 512KB of memory.

3. REALIZING THE BENEFITS OF OBJECT-ORIENTED PROGRAMMING WITH EXTREMELY LIMITED RESOURCES

In the world of programming languages, standards are immediately emerging after the introduction of a new concept. We begin to produce more and more restrictive derivatives of the original idea. At the end, we begin to believe that an application without strictly conforming to these standards cannot be classified under this category

Let's assume that we have had highly self-disciplined programmers and we do not need to bind them with object oriented programming constructs of an object oriented programming language. We give them an object oriented design document, possibly produced by using UML. If they write the application in assembly code, as objects interact by exchanging messages while their private data and internal code are properly encapsulated, would the resulting software be classified as an object oriented one?

It is conceivable for the programmers to program under some constraints to achieve the well-established benefits of object oriented programming, but enforcing these constraints under a high level programming language is only a way of doing this.

In the microcontroller world we are somewhat forced to find another way to do it and we should not give up because of we do not have the object oriented programming language generating that compact code for us. At this level we are dealing with the MCU's typically having 128 to 512 Bytes of RAM, 2 to 8 Kbytes of Flash memory/ROM(Read only memory), ADC (Analog Digital Converter), Timer and general purpose I/O(Input-Output).

3.1.MOTOROLA MICROCONTROLLER ARCHITECTURE

Motorola microcontroller architecture inherently supports object oriented programming, enabling us to view functional modules as active objects (Figure-1). Each functional module in the microcontroller can work in parallel with the others having a different thread of execution and can generate interrupts on the key events. It has all the critical data pertaining to its function in its registers. So each functional module can be viewed as hard-wired active object, capable of accepting some messages and responding with the corresponding actions or by returning some results in its registers.

Take the Analog to Digital Converter, when it is set to make continuous conversion with ADC interrupt is enabled, it works stand alone, gets the input from the input channel, performs the conversion and generates an interrupt when the conversion is completed. Let's wrap it with the object abstraction now. When a proper environment is set up, it can be thought of getting a message to set some of its registers (internal variables) to specific values and beginning to act accordingly by continuously converting the analog signal levels at the input channel to the digital values and putting the results in its data register (a private variable). The interrupt generated at the end of each conversion can be used to invoke other objects by sending them messages.

In this view, (Figure 1) if we give the programmer a set of messages/method calls that are acceptable by the functional modules behaving like objects and the attributes of these objects, we can have a infrastructure of objects making up the microcontroller hardware. Than the programmers will begin to think in object oriented way about the functional modules of the MCU.

In some embedded MCU applications there are more than one MCU communicating over a data communication network. We can now extend the object-oriented conceptualization of the MCU functional modules to the MCU level and think MCU as a higher-level object, so MCU's as active objects interact with each other over the network and cooperatively serve for completing the real time task.

In summary we should treat the control and data registers of a functional module as variables of the module. The methods we should device should be able to set the values of these variables or return their content. Further, the interrupt service routines associated with the events generated by the module can do internal state changes or invoke the methods of the other objects wrapping the other functional modules. The programmer should be able to design any other object for some computation not associated with any physical functional module.

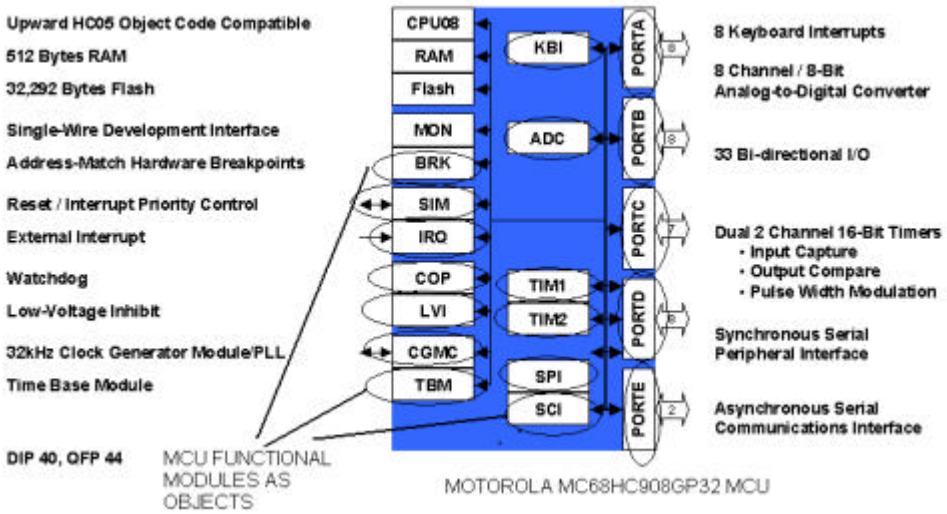

Figure 1. MCU functional modules viewed as objects.

3.2. USING ASSEMBLY LANGUAGE

We can design our system in object oriented way by using UML tools. Based on this design we can move into programming, which language should then be used?

The limited resources such as 2 Kbytes of Flash Memory, 128 bytes of RAM, may force us to use Assembly language. Assembly language is error prone and carries all sorts of problem sources in its structure. At this point we can write the program as if the restrictive structures of an object oriented programming language exists. We can integrate some syntactic sugaring and implement parameter passing by using macros. So it is kind of applying modular-programming techniques in assembly language programming in the absence of a block structured high level language. By doing so, we can give the feeling that the methods of the objects are called and we can still hide data from the programmer by forcing him/her to use only the methods of the objects to access internal data, The programmer writes the program in terms of method calls only and that is easy to check.

A brief example using Motorola HC08 Microcontroller Assembly Language versus a EC++ program doing the same thing may help us understand the above-discussed technique. In this example we took the Motorola MC68HC908GP32 microcontroller and viewed the ADC functional module of the microcontroller as an object having a private data space (ADC control and data registers), and methods accessing, reading or modifying its data. A short segment of an Embedded C++ program is created declaring the necessary “Adc” class and its instance “adcobj”. Some references to the methods are made, for changing the state the of the “adcobj”. (Figure- 2)

Another program is written in Motorola assembly language by using P&E Micro [13] CASM08Z macro assembler. (Figure-3) In this program macro names are defined in such a way that, they look like the dotted notation of method calls of object with the name ADC. The subroutines corresponding that functions are

called from within macros. This allows the use of macro names like we use function names for calls in a high level language, preventing JSR assembly language instruction be used with the subroutine label. The parameter passing is done by using macro parameter mechanism and making use of the stack. In this mechanism macro can be thought to be receiving the parameters. It pushes them on stack and calls its subroutine. Subroutine pops them from the stack and makes use of them. Subroutine returns the control to the instruction immediately following subroutine call in macro. Together with the way of using macro names and macro parameter passing, our main program's look and feel is more like a high-level language program rather than assembly language program. After getting used to this style of programming the programmer begins to think in object oriented way and can use object oriented design documents as the base for programming.

There is no need to say, the size of generated code in the case of assembly language program in (Figure-2), is 44 bytes and 144 bytes for the EC++ program as shown in (Figure-3) using Metrowerks CodeWarior HC08 C/EC++ compiler.

```
#pragma DATA_SEG SHORT _DATA_ZEROPAGE
      class Adct
      { public:
        Adct()
         {adscr= (volatile unsigned    char*)(0x3c);
         adcData= (volatile unsigned char*)(0x3d); }
       unsigned char readData( )
            { return *adcData; }  // Return the converted value
       void setInterruptBit()
            {(*adScr) =(*adScr) | 0x40; }  // Enable ADC interrupt
       void resetInterruptBit()
            {(*adScr)=(*adScr)&0xBF; }    //Disable ADC interrupt
       void setConversionBit()
            {(*adScr)|=0x20; }   // Set continuous conversion bit
       void resetConversionBit()
            {(*adScr)&=0xdf; }  //Reset continuous conversion bit
       void selectChannel(unsigned char x)
            {(*adScr)&=0xe0;     // Select ADC channel
             (*adScr)|=x; }
      private:
           volatile unsigned char*  adscr ;    // ADC status control register
           volatile unsigned char*  adcdata;  //ADC data register  };
     int main( )
     {
        Adct adcobj;
        unsigned char data;
        adcobj.resetConversionBit();
        adcobj.selectChannel(5);
        adcobj.resetInterruptBit();
        data=adcobj.readData();
     }
```

Figure 2. EC++ program

```
#include "gpregs.inc"
    ORG   $FFFE
    dw  Main    ; Reset Vector
$MACRO adc.ReadData
   JSR Adcreaddata ; call subroutine, get the result in accumulator
$MACROEND
$MACRO adc.SetInt
   JSR Adcsetinterruptbit ; call subroutine
$MACROEND
$MACRO adc.ResInt
   JSR Adcresetinterruptbit ; call subroutine
$MACROEND
$MACRO adc.SetCon
   JSR Adcsetconversionbit ; call subroutine
$MACROEND
$MACRO adc.ResCon
   JSR Adcresetconversionbit ; Call subroutine
$MACROEND
$MACRO adc.SelChan a
   CLRH
   LDHX #%1  ; get the parameter and push it onto the stack
   PSHX
   JSR Adcselectchannel ; Call  subroutine
   AIS #2  ; adjust the stack pointer on return
$MACROEND
         ORG $EC00         ; MAIN PROGRAM BEGINS
 Main:   adc.ResCon          ; Set ADC to single conversion
         adc.SelChan  5      ; Select ADC channel 5
         adc.ResInt          ; Turn ADC interrupt off.
         adcReadData         ; Read the result
         .........
       ; MAIN PROGRAM ENDS
  Adcselectchannel:     AIS #2                ; Adjust the stack pointer
                        PULX                ; Get the parameter
                        MOV X+,ADSCR        ; assign it to the ADSCR (register)
                        AIS #-3             ;Adjust the stack pointer
                        RTS                 ; Return form subroutine
Adcreaddata:             LDA ADR            ;  Load the result to accumulator
                        RTS                 ;  Return
Adcsetinterruptbit:      BSET 6,ADSCR       ; Set interrupt bit at ADSCR
                        RTS                 ; Return
Adcresetinterruptbit:    BCLR 6,ADSCR       ; Clear interrupt bit at ADSCR
                        RTS                 ; Return
Adcsetconversionbit:     BSET 5,ADSCR       ; Set continuous  conversion
                        RTS                 ;  Return
Adcresetconversionbit: BCLR 5,ADSCR         ;Set single conversion
                        RTS                 ; Return
```

Figure 3. Assembly language program

In the Assembly Language example shown in (Figure 3) it is clear that we can achieve a level of abstraction for viewing the functional modules as objects even if we do not use a full blown object-oriented language. In practice we can put macro definitions and subroutines in include files and separate them from the main program. The programmer may follow the same technique for creating his/her own objects for the code not related with the functional modules.

4.CONCLUSION

Under the pressure of cost factors and demanding consumers we will have to go on writing more capable but compact programs for the microcontrollers with extremely limited resources. In order to make our software more reliable, maintainable and reusable, we can still employ object oriented design and programming techniques even if the compilers for the high level object-oriented languages do not help us in many applications of this kind. Writing in assembly, but using a carefully set environment, simulating the object oriented constructs, seems to be the solution for these specific cases. If we can do that, we will also have the privilege to use object oriented design tools right from the beginning. By this way most of the benefits of the object oriented design and programming may be realized and it is better than doing it conventionally. In this work, it is practiced and seen that this kind of programming environment can be set and used with little overhead. Future efforts may concentrate on the assemblers or simple translators facilitating this kind of programming more.

5. ACKNOWLEDGEMENTS

J2ME, Java and all Java based marks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other Countries. OMG and UML are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries. Dinkumware, is registered trademark of Dinkumware Ltd. Metrowerks and CodeWarrior are trademarks of Metrowerks Corp in the United States and other Countries. All other trademarks are property of respective holders.

REFERENCES

1. B. Selic. “A generic framework for modeling resources with UML”. IEEE Computer Vol 33, page 64, June 2000.

2. Embedded C++ Technical Committee “EC++ web site”. http://www.caravan.net/ec2plus

3. Embedded C++ Technical Committee. “The embedded C++ programming guide lines”. http://www.caravan.net/ec2plus/guide.html.

4. Embedded C++ Technical Committee. “Objectives”. http://www.caravan.net/ec2plus/objectives/ppt/ec2ppt05.html

5.Hammer,D,K. Chaudron,M.R.V. “Component-based software engineering for resource-constraint systems: What are the needs”. In proceedings of, Sixth International Workshop on Object Oriented Real-Time Dependable Systems,2001.

6. “Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices”. White Paper, Sun Microsystems Inc. May 19,2000.http://java.sun.com/products/cldc/wp/KVMwp.pdf

7. Jigorea,R. Manolache,S. Eles,P. Peng,Z. “Modelling of real-time embedded systems in an object-oriented design environment with UML”. In Proceedings of the Third International Symposium on Object-Orineted Real-Time Distributed Computing,2000.

8. Julio L.Medina Pasaje, Michael Gonzalez Harbour, Jose M.Drake. “MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems”. .In Proceedings of the 22nd IEEE Real-Time Systems Symposium,2001.

9. Kuster,J. Stroop,J. “Consistent design of embedded real-time systems with UML-RT”. In Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001.

10. Nilsson, A. Ekman,T. “Deterministic Java in tiny embedded systems”. In Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2001.

11. P.J.Plauger. “Embedded C++ Seminar “at the Embedded Systems Conference, Chicago Ilinois, 1999. http://www.dinkumware.com/embed9710.html

12. Muller, P.O. Stich, C. Zeidler,C. “Components at work: Component technology for embedded systems”. In proceedings of 27th Euromicro Conference,2001.

13. P&E Microcomputer Systems Inc. http://www.pemicro.com