WHITE PAPER

The Many Ways of Programming an ARM® Cortex®-M Microcontroller

Joseph Yiu May-2013

Abstract

Besides the C and C++ programming languages which most software developers use, various programming methods and languages are available for microcontroller programming. For example, the ARM® Cortex®-M microcontrollers can be programmed in Java™, Arduino™, high level graphical programming languages, and other language abstractions. This paper introduces various development environments, interesting features and other aspects such as interoperability with the ARM CMSIS device driver libraries. We will also examine how some of these new technologies help us to address some of the new advanced application areas like M2M, modeling based software development, as well as how some of these solutions enable new users to start learning microcontroller programming.

1 - Introduction

In the last few years we have seen an increasing number of microcontroller vendors offering 32-bit microcontrollers based on ARM® Cortex®-M processors. With more and more embedded system developers starting to use 32-bit microcontrollers for their projects, we are also seeing

i) Many programming language previously only available for desktop computer being ported to ARM microcontrollers

ii) New programming language and development suites developed for ARM microcontrollers.

This momentum is driven by a number of factors:

- Technology

- Ecosystem
- Applications
- Design Cycles

1.1 Technology

A large part of the changes happen only in the last few years because limitations in legacy architectures made it difficult or less efficient to use high level language to program. For example, the memory size is often limited and the performance might not be sufficient.

With modern microcontrollers based on ARM Cortex-M processors, you can see that many of them can run at over 100MHz and the performance (e.g. CoreMark/MHz) is higher than desktop processors you used 10 years ago. The ARM processors are also very C friendly and designed with OS support in mind. So there is no issue with using any high level language which might requires high performance or need to have OS supports.

1.2 Ecosystem

The open nature of the ARM architecture allows software tool vendors to develop new tools for a wide range of microcontroller products from different vendors. If the same development was focused on other architectures, their products could only reach a small market.

1.3 Applications

The wider adoption of ARM processors in the microcontroller industry provides an opportunity for various programming tools vendors to create diverse application development environments, and many of these can be significant to particular embedded segments. For example, beside traditional development suites, we also see software development solution focus on:

- M2M and Internet of Things (IoT) solutions (e.g. Java™)
- Scientific and Mathematic applications (e.g. MathWorks® Inc MatLab®/SimuLink®, National Instruments™ LabVIEW™)
- Education and hobbyists market (e.g. Arduino™, mbed)
- Control and FSM designs (e.g. IAR VisualSTATE®, Verum® ASD:Suite®)
- Alternate programming languages (e.g. Ada, Pascal, Basic)

1.4 Design Cycles

As time to market is getting more important, more designers prefer to use high level language to help reducing the time to spend on creating low level task code or porting codes. For example, use of Java language might help software developers to create and test applications before the hardware is available, use of Arduino and mbed enables designer to create working prototype quicker and easier.

The rest of this paper gives a high level overview of some of these development suites and insight of various software integration aspects.

2 - Java

Java™ technology provides numerous advantages such as software investment capitalization for developers and prototyping capabilities to speedup concepts specification. It is also a hot topic due to its potential in Internet of Things (IoT) and Machines to Machines (M2M) applications. Some of you might not be aware that there are multiple Java development environments available from different vendors: Oracle® (www.oracle.com/javame) and IS2T (www.is2t.com/en/products-microej-sdk-std.php**)** are two of the examples.

2.1 - Java ME Embedded - Oracle

First let's start from the Java ME (Micro Edition) Embedded from Oracle®. A key target of the Oracle Java on ARM® architecture is IoT and M2M applications. Using the Java ME Software Development Kits (SDK), you can develop Java applications with Eclipse and Netbeans (netbeans.org) IDE (Integrated Development Environment).

The Java ME provides a software development that is identical to the Java SE Embedded (Standard Edition, Embedded). But when the application is built, you select which target platform to use (e.g. Micro Edition, Standard Edition). When Java ME is select, the Byte Code object (.class) generated is optimized with small memory foot print. The byte code object file is then loaded and executed on top of the Java VM. If you use an API not available in Java ME, the IDE will inform you that the particular class is not available.

For example, you can have a microcontroller running the Java ME Embedded, and Java applications can be stored on a SD card, or on chip. When the system is started, the Java ME Embedded can read a configuration file (a text file) and identify the Java application to load and executed.

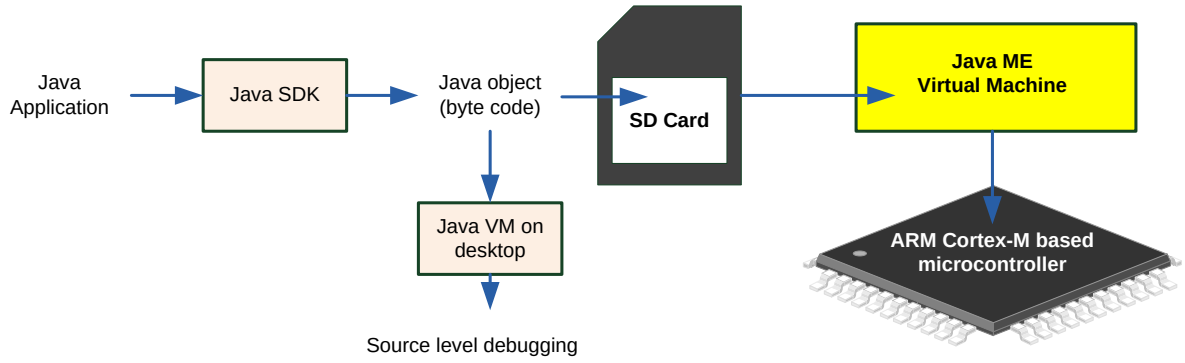


Figure 1 : Design flow using Java ME Embedded

When comparing Java ME to Java SE, some of the features on Java SE are currently not available on Java ME. For example, currently the GUI library component, or JavaFX, is not available on Java ME.

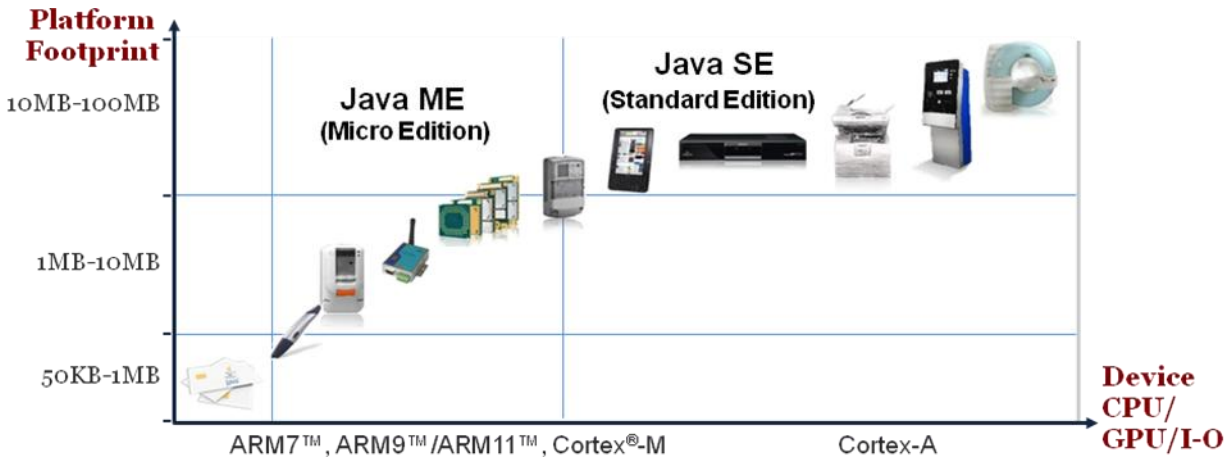


Figure 2 : Different flavors of Java platform for ARM processors

Java objects execute in a sandboxed environment, typically called Java Virtual Machine. With Java ME, currently you need typically 256KB of RAM and 1 to 1.5MB of program ROM to host the application and the JVM. Due to the nature of JVM, the applications running inside JVM cannot guarantee real time behavior. However, since the JVM itself runs on top of a RTOS inside an embedded system, additional tasks with real time requirements can run as other application threads and communicate with Java applications inside JVM though an event communication interface.

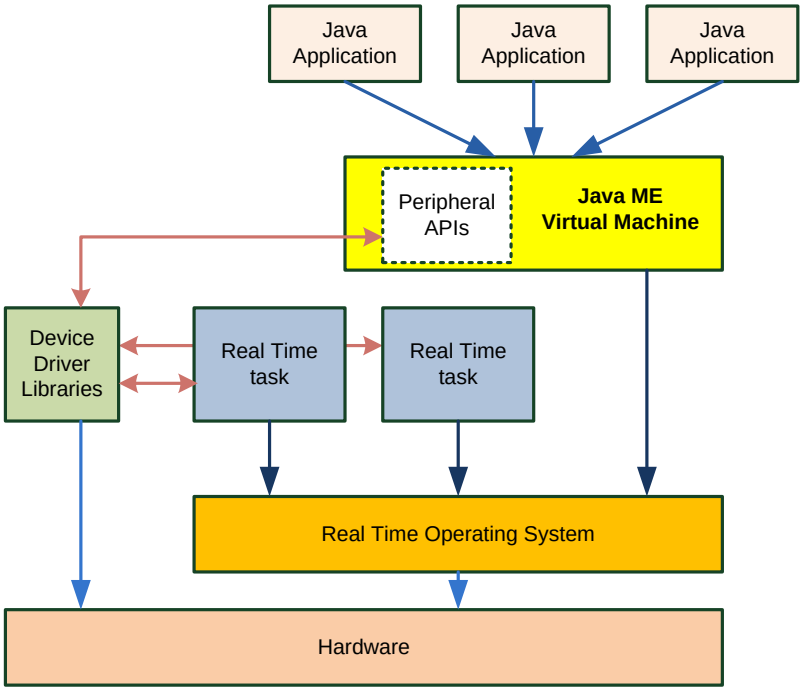


Figure 3 : Software architecture of combining Java applications with real time applications

Source level debugging is available on :

- desktop environment,
- simulated environment, or

- on actual hardware.

The Java execution environment contains an optional debug agent (running on target hardware), which can communicated with the debugger via a built-in Device Manager (running on debug host). The communication channel can be UART, TCP/IP, USB, etc.

However, currently there is no integrated solution for source level debugging of the complete system (Java applications + real time applications) on hardware.

A key value of using Java ME is to enable a consistent environment for Internet of Things (IoT) applications where Java codes developed can be used from small microcontroller systems, to internet gateway or home server, to large database server. For example, data are often collected with small embedded system and then passed on to server applications. On the other hand, some of these embedded systems can also subscribe information from servers. Java technology enables the software developer to integrate these information transfer functionalities easily.

The consistency between Java ME and Java SE make it possible to develop Java applications that work across different embedded platforms.

The Java ME also provides TCP/IP communication APIs that allows Java applications to open TCP/IP sockets and communicate with other devices. In addition, you can potentially add other 3rd party communication stacks like Bluetooth. Optional APIs are available for server/client communications, allowing small microcontroller system to talk to Oracle database servers seamlessly.

The nature of Java sandboxed environment can help provide better security, but at the same time can be inconvenient for control tasks. Java ME has added peripheral APIs to make this easier, but unlike traditional C programming, the peripheral APIs are fairly high level and might not able to support many device specific features. In those cases you might want to separate some of these I/O control tasks and run them as application threads alongside with the JVM in an RTOS environment.

2.2 – MicroEJ, STM32Java – IS2T

For some embedded system designers, the microcontrollers they use only have several hundred KB of flash and RAM and this limitation makes the current version of Oracle Java ME virtual machine unsuitable. To solve this problem, another company called IS2T has a Java product called MicroEJ® which uses a different approach. The MicroEJ is also available as part of the STM32Java SDK product from STMicroelectronics (www.stm32java.com).

Instead of loading the byte code object directly to the virtual machine, the Java object is first optimized and preprocessed and linked off board before loading on to the microcontroller.

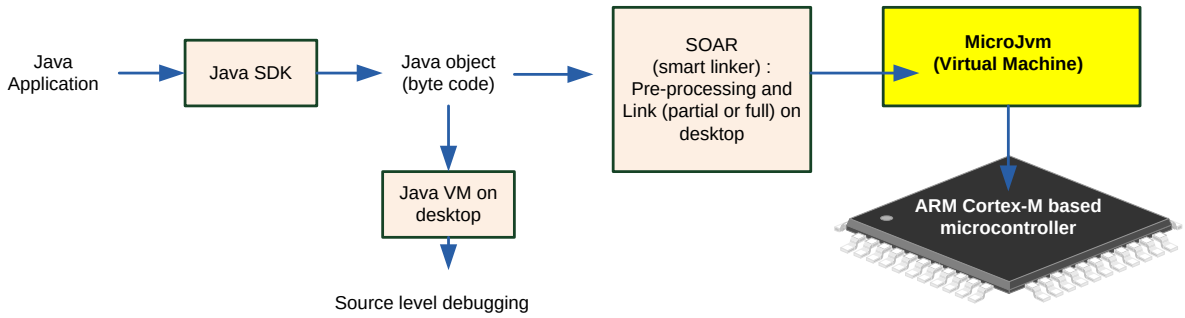


Figure 4 : Design flow using IS2T MicroEJ or STM32Java

This arrangement has several advantages:

- Quicker start up time. For example, on ARM Cortex®-M3/M4 processor-based microcontrollers running at 120MHz, it only takes 2ms to boot up.
- Smaller memory requirement. The MicroJvm® virtual machine can fit into a microcontroller as minimum memory requirements are 28KB of flash memory and 1KB of SRAM (not including application code). Even with an additional rich GUI Human Machine Interface (HMI) library, it only needs 90KB to 140KB of flash overall.
- Better optimization because the preprocessing and linking are carried out on a personal computer, which is more powerful then the microcontroller.

The potential drawback is that the microcontroller cannot dynamically download at run-time Java byte code objects, which is not an issue for majority of embedded applications.

The MicroJvm virtual machine in MicroEJ confronts all requirements for Java VM (VM engine, error check, security, memory optimizer (i.e. garbage collection).

For application developers, the MicroEJ SDK product is a quick way to develop applications with feature-rich GUI (graphic user interface). The MicroEJ product contains various choices of Java Platform (JPF). Java Platform (JPF) includes the root components as well as additional packages: MicroJvm virtual machine, standard libraries such as B-ON + CLDC (core embedded Java API), MicroUI™ (embedded user interface), MWT (Micro Widget Toolkit, an embedded widgets framework), NLS (embedded national support), runtime PNG image decoder, and graphical tools for the design of fonts, front panels and story boards.

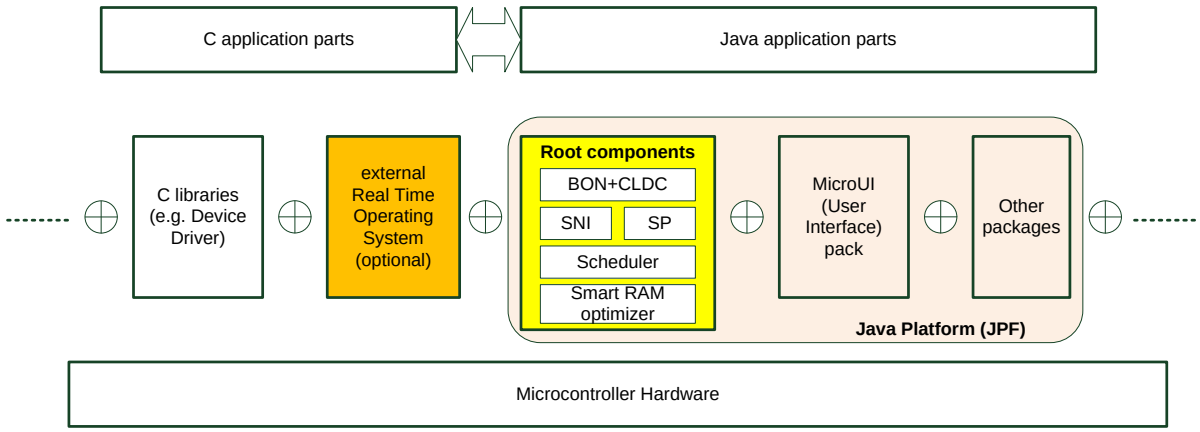


Figure 5: Components in Java platform (JPF)

Java applications can access to device driver library functions (usually in C) through SNI (ESR012, Simple native Interface) or SP (Shield Plug). C code can also call Java methods. For example, an example for Java application to access a C function is as followed:

```
package com.corp.examples;
public class Hello {
  public static void main(String[] args){
   int i = printHelloNbTimes(s);
  }
  public static native int printHelloNbTimes(int times);
}
```

And the C function is declared with SNI:

```
#include <sni.h>
#include <stdio.h>

jint Java_com_corp_examples_Hello_printHelloNbTimes(jint times){
  while (--times){
    printf("Hello world!\n");
  }
  return 0 ;
}
```

Similarly, C code can also call Java object:

```
#include <sni.h>
void main(){
   SNI_callJavaVoidMethod(Java_com_corp_examples_Hello_printHelloNbTimes, 3);
  }
```

And the Java object being called is defined as:

```
package comp.corp.examples;
```

```
public class Hello {
  public static void printHelloNbTimes(int times){
    while (--times >= 0) {
      System.out.printfln("Hello world!");
    }
  }
}
```

The MicroJvm virtual machine contains its own task scheduler called green thread. It needs a timer hardware for periodic interrupt generation. This allows the MicroJvm virtual machine to run on its own, and can also be run within a single task with a third party RTOS.

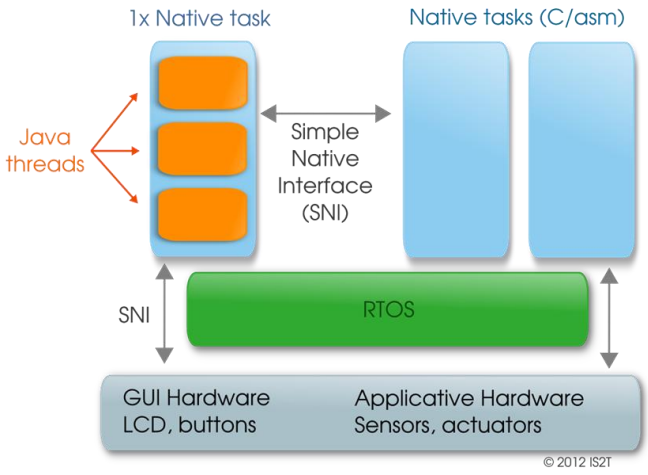


Figure 6: Software architecture in mixed Java and native code (e.g. C, assembly) applications

One of the key attractions of MicroEJ is the graphic library support. With the Object Oriented nature of Java language and the automatic memory management, Java is excellent for handling of dynamic GUI components in GUI designs. MicroEJ has a library pack called UI (User Interface) which provides user interface components and allow developers to create GUI in short time. The graphic library supports various operation modes to enable the applications to work in different types of microcontroller platforms (i.e. different types of graphic buffer types).

The MicroEJ development include an Eclipse based IDE, and contains a simulator for designing Java applications on desktop. The simulator has an Hardware In the Loop API to extend it with real I/O, allowing simulations to run with true inputs. It also includes a Front Panel Designer tool that permits the design of a virtual device.

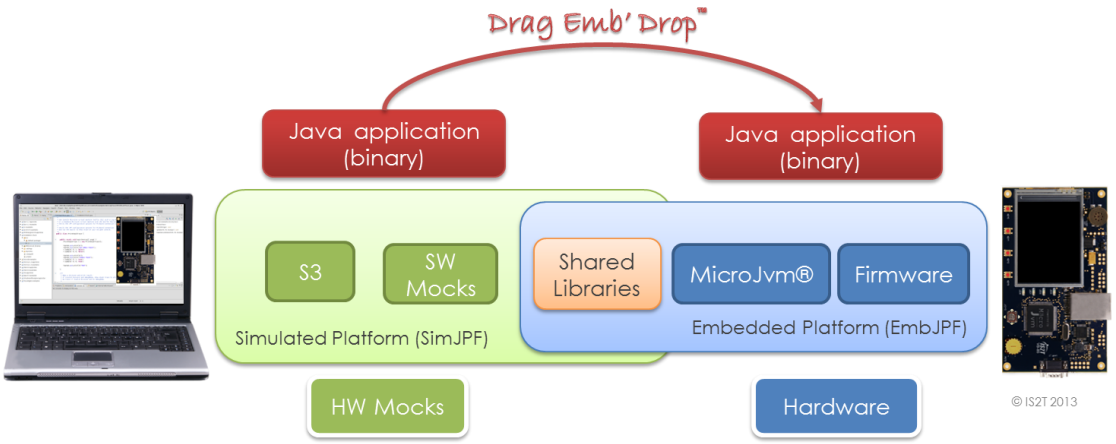


Figure 7: Transfer of software development environment

3 - Scientific and Mathematic applications

3.1 – MATLAB®/Simulink®

Recently (February-2013) MathWorks® Inc announced their work with ARM and STMicroelectronics to enable MATLAB and Simulink design support on ARM Cortex-M processor-based microcontrollers. At the time of writing the Embedded Coder support for ARM Cortex-M series processors has not been officially released, but a demonstration was shown in the Embedded World Conference in Nuremberg, Germany.

MATLAB and Simulink are commonly used for mathematical computations. MATLAB is normally text based, and Simulink provides a graphical development environment that allows you to define computations in block diagrams.

The Embedded Coder generates C code from MATLAB or Simulink designs. The code generated will support the CMSIS-DSP library and will be optimized for the Cortex-M series processors. The design will also contains built in support for peripherals on STM32F4 (Cortex-M4) microcontrollers.

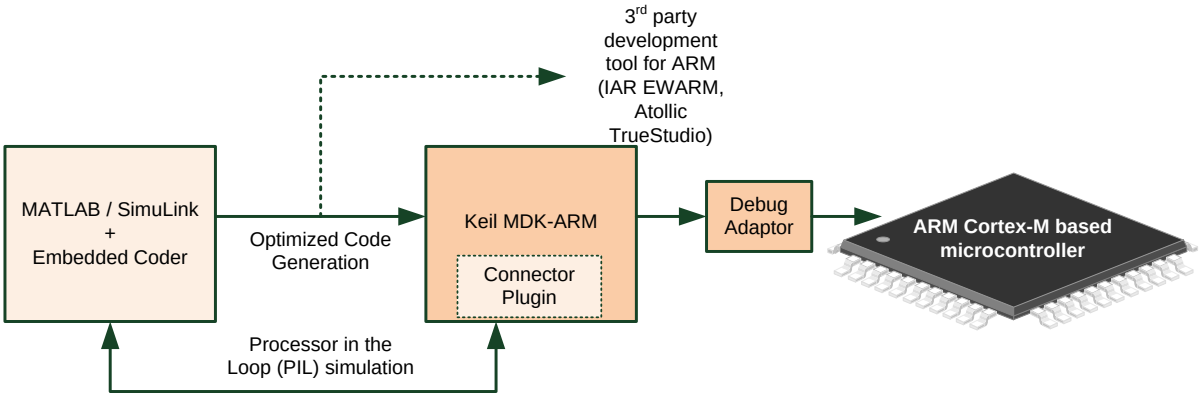


Figure 8: Microcontroller software development flow using Simulink with traditional tools

The model based software development allows quicker development time. For example, a motor control system can by developed by first creating a model of the control system in Simulink:

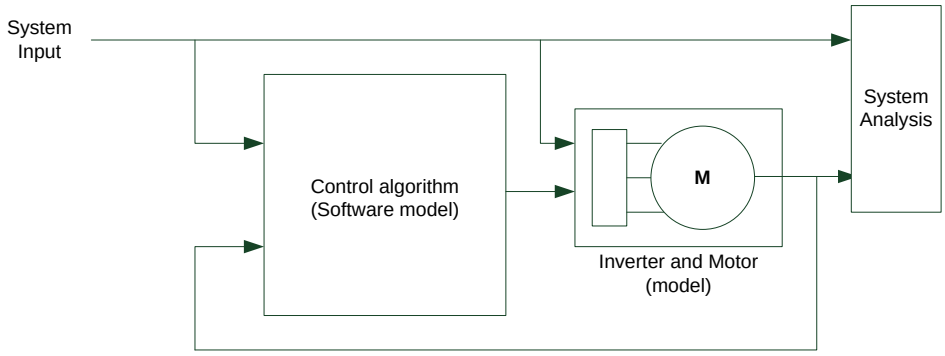


Figure 9: Example control application – a motor control system – simulated in Simulink

The model can be tested using simulation inside Simulink. Once the design is working, the user can then use Embedded Coder to generate the C code, and test the design in hardware using Processor-In-the-Loop (PIL) configuration.

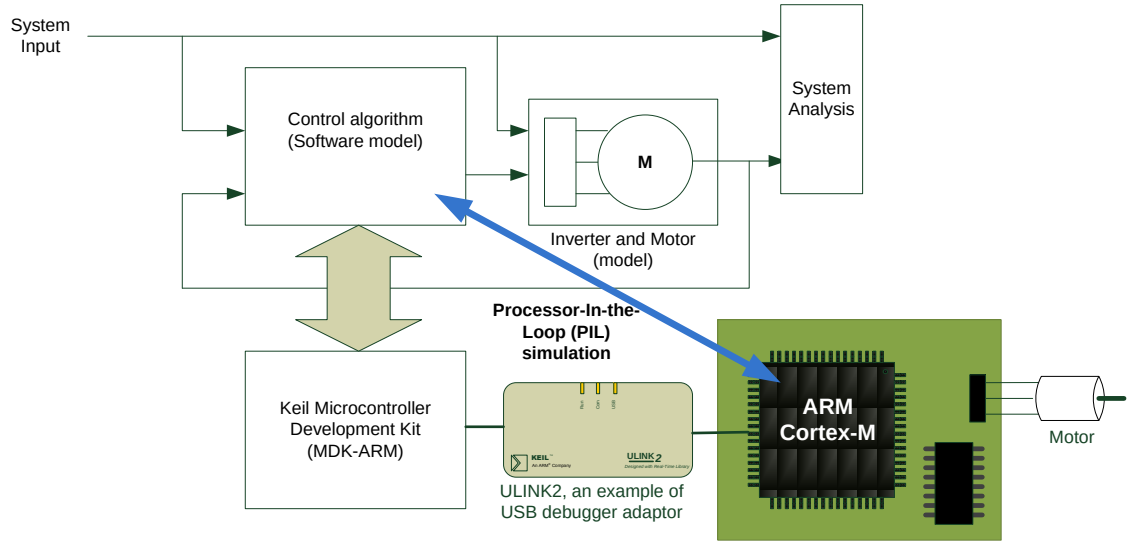


Figure 10: Running example control application in Simulink with Processor-in-the-Loop (PIL)

Finally, you can add peripheral control code to your project, generate the complete project and test the whole design fully on target hardware.

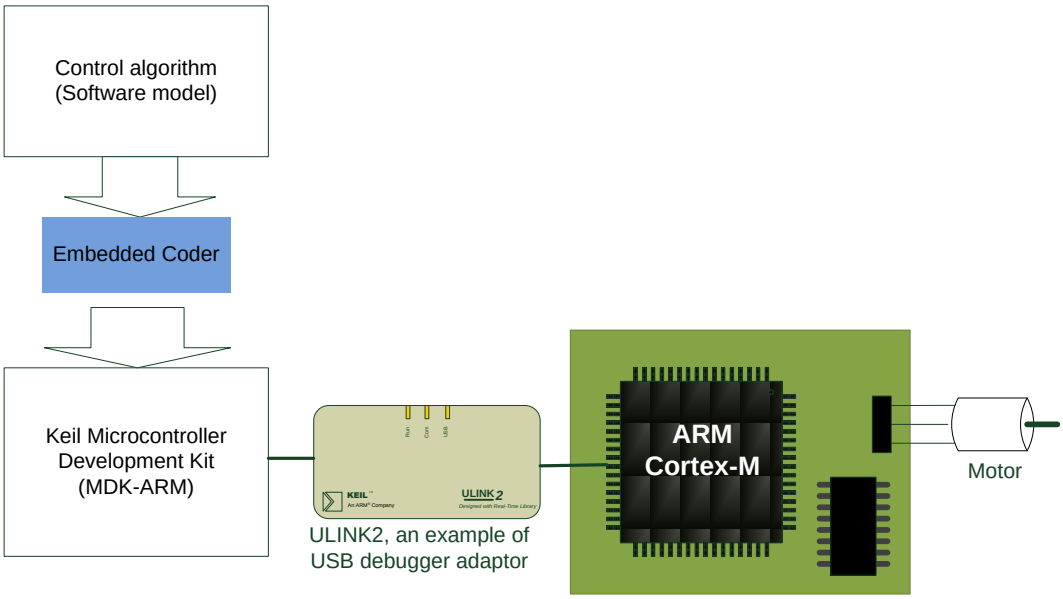


Figure 11: Transfer of control application developed in Simulink to hardware platform

The ability to test the design using PIL configuration allows quicker development. Since the control algorithm code is generated automatically, it avoids the possibility of human errors in code porting.

3.2 – LabVIEW™

LabVIEW™ is a product from National Instruments™ which is popular in scientific research organization as it has a rich library of data processing software components. It is a graphical programming environment which works on personal computers as well as microcontrollers. The LabVIEW graphical programming language offers all of the features you expect in any programming language such as looping, conditional execution, and the handling of different data types. The main difference in working with LabVIEW is that you implement the design of the program in diagrams. For example, you can represent a simple loop to compute the sum of 1 to 10 by the For Loop shown in Figure 12

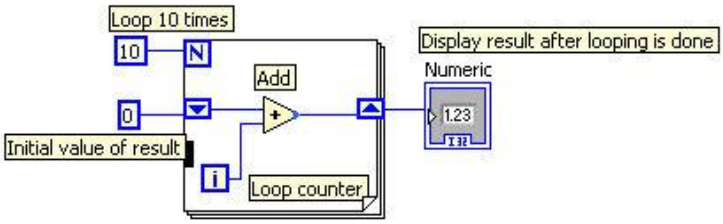


Figure 12 : A simple for loop to add 1 to 10 in LabVIEW programming

The LabVIEW programming environment provides a comprehensive library of functions including functions for digital signal processing (e.g. filter and spectral analysis) , mathematic, array/matrix processing, etc. These ready-to-use components allow application software to be developed without an in-depth knowledge of programming or algorithms. For complex applications, you can design the software into a hierarchy of modules called *virtual instruments* (VIs) and subVIs. For

example, the figure below show a LabVIEW subVI on the right which find the largest variables from four input variables, and this subVI is used by another VI.

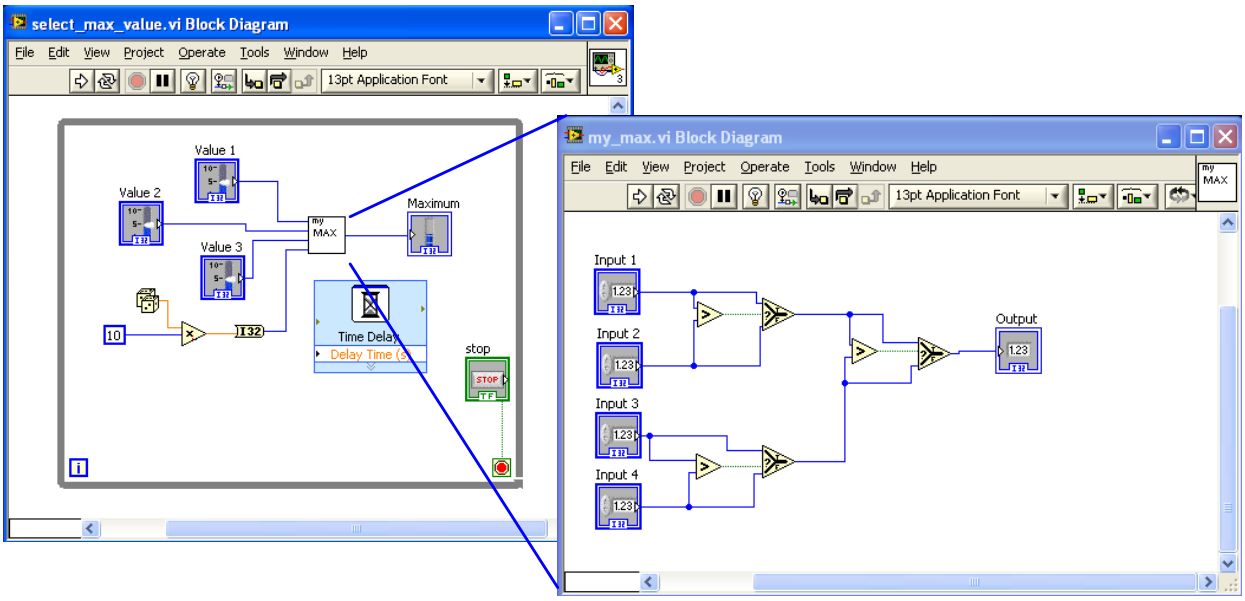


Figure 13: Hierarchical software design in LabVIEW

The LabVIEW C Code Generator takes LabVIEW graphical code and generates procedural C code. So you can add the generated code in projects for traditional programming environments such as Keil® MDK-ARM, IAR Embedded Workbench, etc. During program development, you can first test the LabVIEW code by running it inside the LabVIEW environment on personal computer.

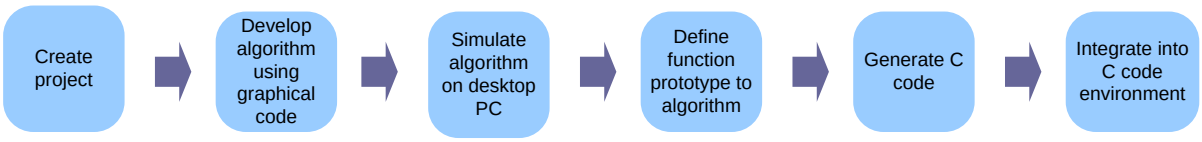


Figure 14: Typical development flow for using LabVIEW C code generator

4 - Education and hobbyists

4.1 – Arduino™

Arduino™ (www. arduino.cc) is a popular platform for engineers, hobbyists and education organizations. It is based on Atmel® AVR® and ARM processor-based microcontrollers and contains many different hardware platforms. Examples of Arduino include the Arduino Due (based on Atmel ARM Cortex-M3 processor-based SAM3X8E,), and Arduino Uno (based on Atmel's AVR ATmega328 microcontroller).

Arduino enthusiasts have recently developed (or thought of developing) a number of designs based on the Arduino platform including:

- A guitar-hero flame thrower: http://arduino.cc/blog/2012/04/18/arduino-flamethrower-guitarhero-rockstar/
- A lawn-mowing robot: http://arduino.cc/forum/index.php?topic=2557.0
- Auto lacers (shoe lace that lace by itself), flame-throwing latern and more: http://www.instructables.com/id/20-Unbelievable-Arduino-Projects/

And, most recently with the launch of the Arduino Due, which is based on the Atmel SAM3X8E board with an ARM Cortex-M3 processor, we can expect to see a number of other cool, new projects.

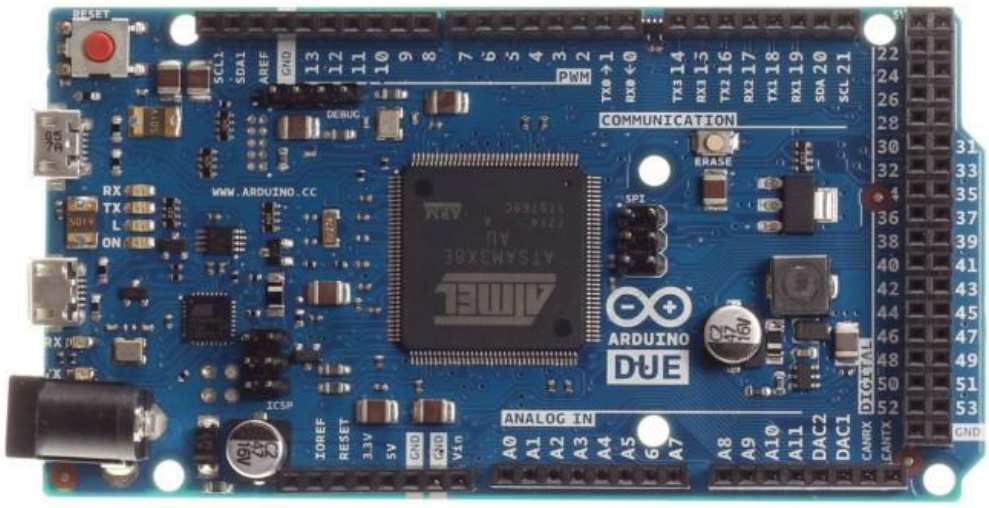


Figure 15: Arduino Due

The Arduino IDE and the boards are designed to be very easy to use, making it suitable for students and hobbyists to learn embedded programming. In addition, the Java base IDE can be use with Windows, Linux and MacOS systems.

Figure 16: Arduino IDE can be used on Windows, MacOS and Linux.

Different from traditional source development approaches, Arduino contains a large software library for various peripheral control functions. These functions are at high level and users do not have to spend a long time to learn how to program the peripherals. One particular interesting nature of the Arduino APIs is that most of the APIs are hardware platform independent. So you can use the same program on different Arduino boards.

An Arduino program is called a sketch and it contains two parts:

`void setup() – This is executed in the beginning of the program execution.`

`void loop() – This is executed repeatedly after setup().`

During programming, users insert their own application code into these two functions. During compilation, the IDE adds an additional software framework for the system and adds the underlying code for the peripheral API calls. The source code for the APIs are available so it is possible to customize the code if necessary.

Since the Arduino is open source, developers can prototype a system quickly using Arduino, and convert their projects to use other development suites if necessary. For example, you might need to do this when integrating your design with other middleware, or restructure your application code to optimize for low power. As a result, the Arduino platform is more than just an education tool, as it

can be very useful for product prototyping. The Arduino users community is also a strong reason for using Arduino, as there are plenty of example projects available (users contributions). And if a user found a problem, there is a very high chance that other experience users can provide assistance.

The Arduino boards are very affordable (starting from about US$25).

4.2 – mbed

The mbed platform is also targeted at education, hobbyists and rapid prototyping market. The mbed SDK also contains a rich set of peripheral APIs, making it very easy and quick for beginners to create feature rich applications. Currently the mbed boards features ARM Cortex-M3 (NXP LPC1768), Cortex-M0 (NXP LPC11U24) and Cortex-M0+ (Freescale™ KL25Z) microcontrollers, and are at an affordable price range. The mbed board is designed with a 40-pin DIP form factor so that it can be connected to other development boards easily. It has a mini USB connection at the top.

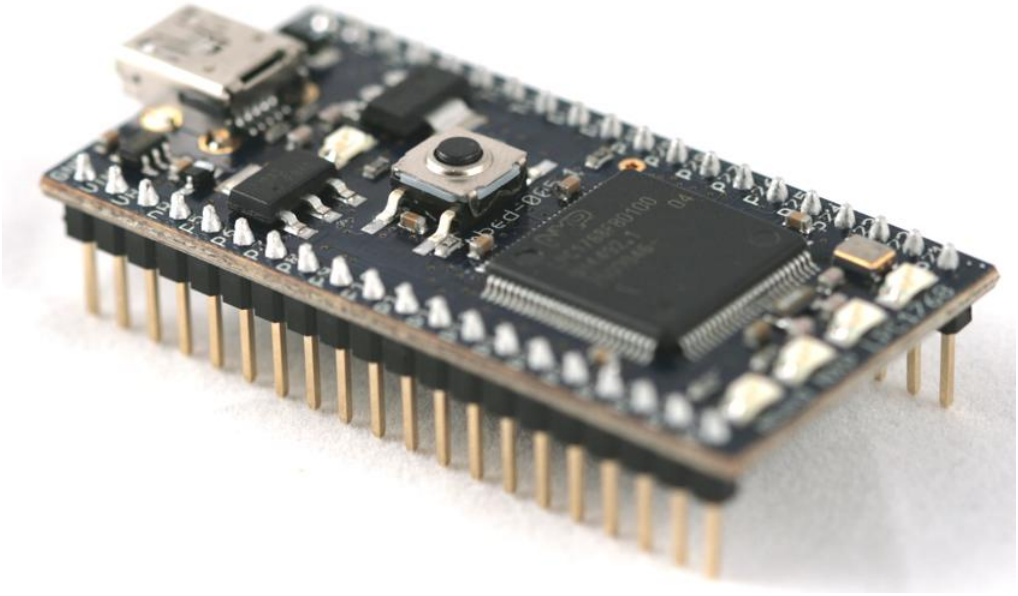


Figure 17:A mbed board with NXP LPC1768 microcontroller

One unique nature of the mbed development platform is that the development tool is web based and is accessed via a web browser. So users can use the mbed IDE on any computer (Windows, Linux, MacOS, etc) and do not have to install any software.

Once you have developed a project and compiled the design successfully, you can download the binary image to your PC. When the mbed board is plugged in to the USB port, it appears as a USB mass storage device (like a flash drive) and you can copy the program image to it. Then you can push a button on the board to reset it, and it will start the program execution.

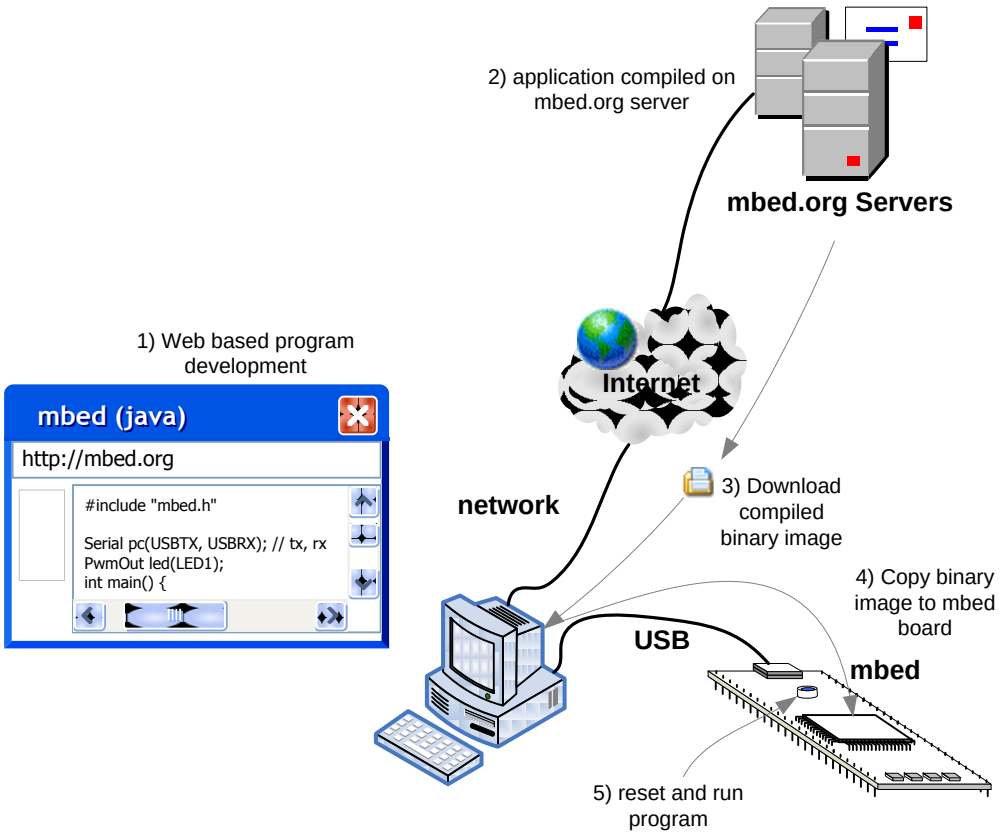


Figure 18:Design flow with mbed platform

Starting from mbed SDK 2.0, the mbed platform is open source and you can export the project to traditional development suite like Keil® Microcontroller Development Kit (MDK-ARM) directly. For complex designs with multiple application tasks, mbed platform have built-in RTOS support using CMSIS-RTOS APIs.

5 – Control and FSM designs

In many embedded applications, especially in control applications, you can view the system operations as a Finite State Machine (FSM). There is a wide range of software development suites and control applications including FSM designs. Here we only cover a couple of these.

5.1 – IAR VisualSTATE from IAR Systems

The IAR VisualSTATE® product allows you to create Finite State Machine designs from a graphic design environment. You can define the states available, the events that can cause state transitions and the corresponding operations for each state.

The VisualSTATE design environment is based on UML (Unified Modeling Language). State transitions are triggered by “events”. For example, in the following diagram a simple FSM for radio control unit is defined with 3 states. In the descriptions of each state transition, text on the left of “/” describes the event(s) which triggers the transition and the text on the right of “/”describes the action(s) to be taken at the state transition.

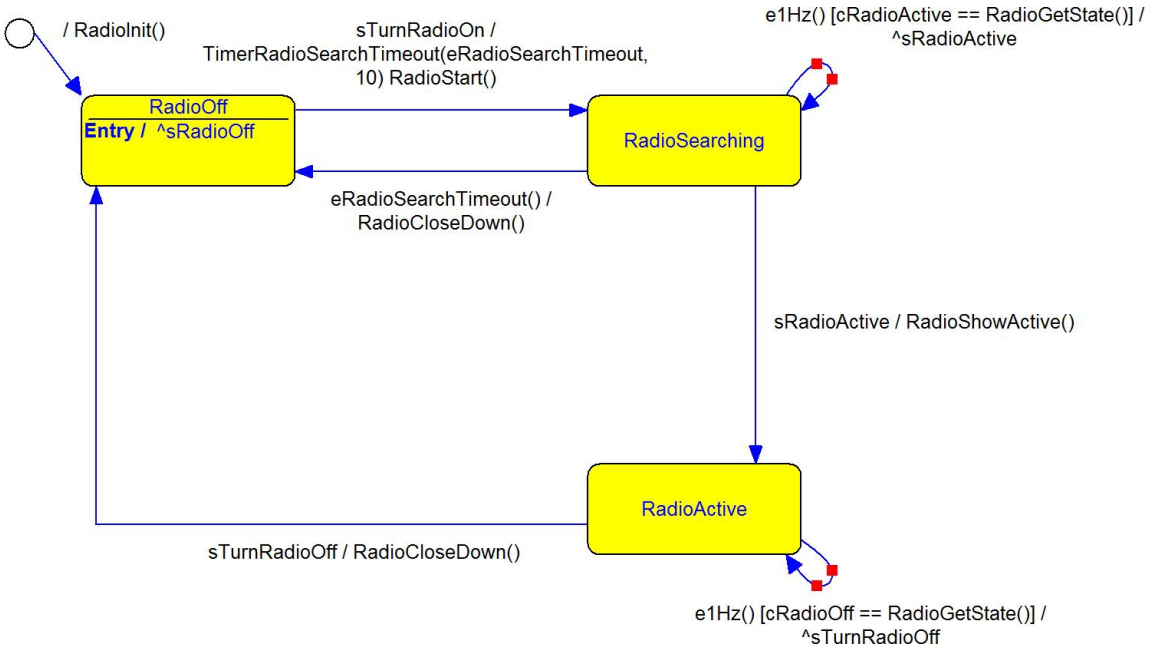


Figure 19: A simple state machine design in IAR VisualSTATE

VisualSTATE allows you to define internal events (called signals, indicated by "^" prefix) that can be emitted as part of the action side of a transition. Such signal events are processed before any new events from the environment. Guard conditions can be specified as part of the trigger definition and must be true for the transition to fire. The guard conditions are defined using Boolean expressions which has a C like syntax.

The VisualSTATE allows you to create the state machine and define the state machine operations easily. After the state machine is designed, VisualSTATE can generate C/C++ codes which can then be integrated into normal project in IAR Embedded Workbench® for ARM (EWARM).

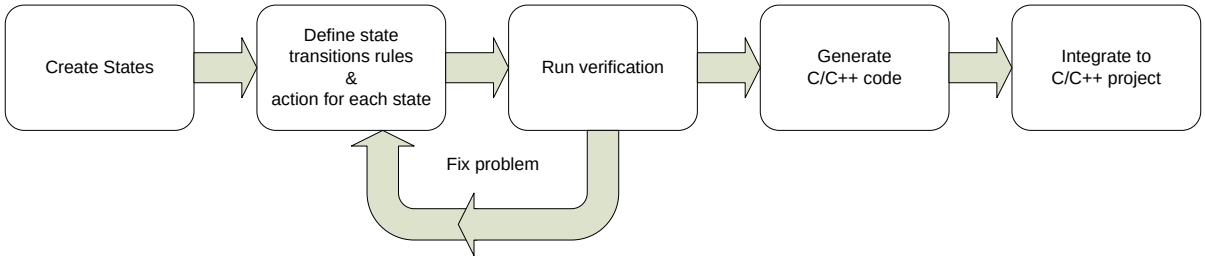


Figure 20: Development flow using IAR Visual State

To help developers, Visual state contains a formal verification tool that can detect potential issues such as:

- Deadlock (local or system level)
- Unreachable state or state combination
- Conflicting transitions
- Systems with ambiguous behavior

The transition sequence that triggers these errors can be annotated by the tool for debugging.

Since the IAR VisualSTATE is tightly coupled with IAR EWARM, it can be debugged with the C Spy debugger with hardware target system connected. The generated FSM code can also run as a thread in an RTOS environment.

5.2 – ASD:Suite®, Verum®

For complex safety critical systems with many software components, formal verification of the system is even more important. Verum® addresses such requirement by providing ASD:Suite® (Analytical Software Design Suite), a model based design tool with a highly sophisticated formal verification engine that enables verification of the software system design much earlier in the design cycle, before coding and system level testing. This enables faster software development, as well as assuring high software quality and reliability.

In the IDE of ASD:Suite, you can create and manage software components, and defines the interfaces between different components. You can expand each component and define their behavioural model using a table based editor called Model Navigator.

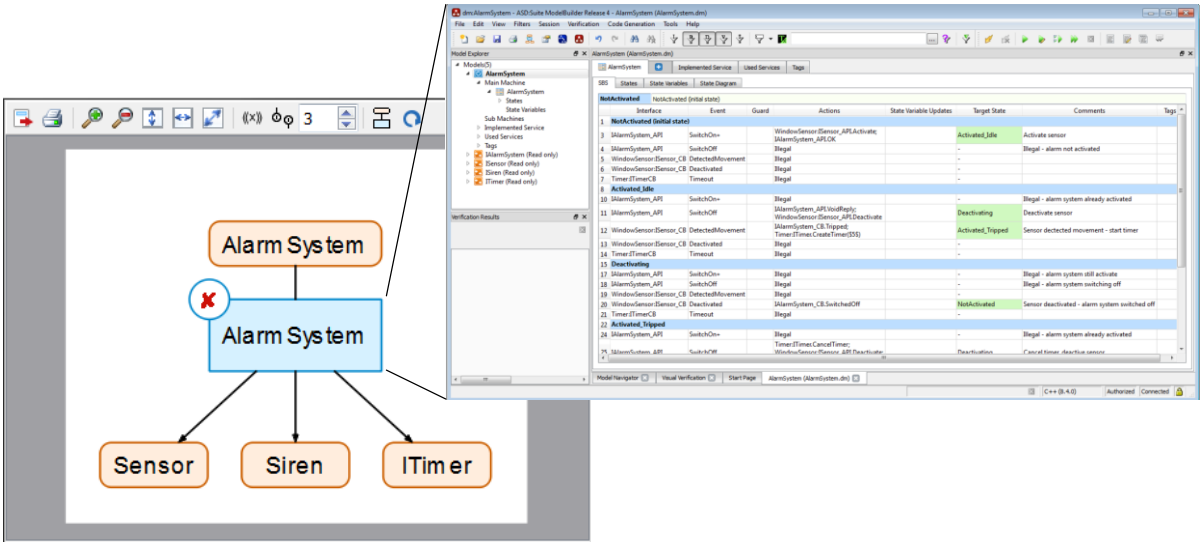
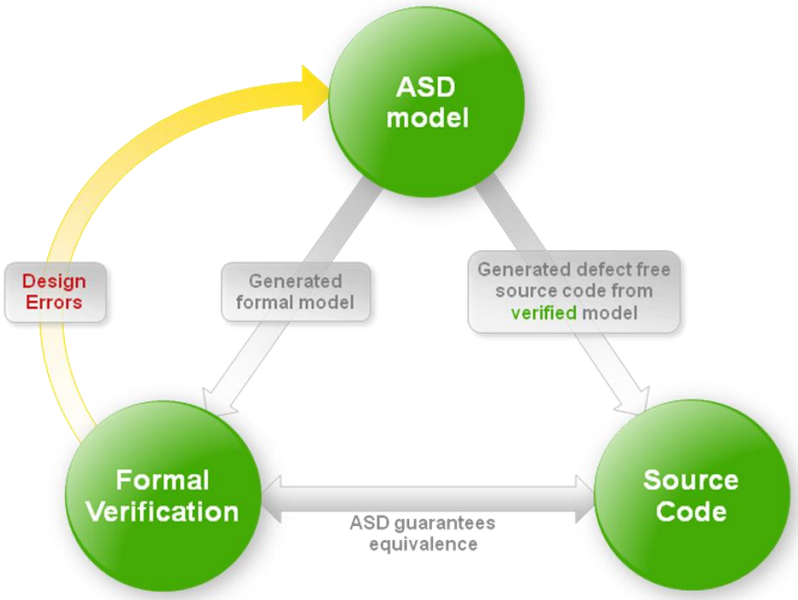


Figure 21: ASD:Suite IDE allows you to create software components and define their details behavior

Once the models are created, you can then run the formal verification engine, which can detect errors such as concurrency issues, dead locks, live locks, unreachable states, illegal behaviors, race conditions, interface mismatches, unhandled error and incomplete behaviors. It can also create trace (annotations) of the conditions which triggers the errors. To fully validate the models, you might need to create model of the external systems which interface with the software components.

Once the models are verified, you can then generate C, C++, C# or even Java code. Since the verified mode have no undefined or ambiguous behavior, and must be fully consistence due to the formal verification flow, ASD:Suite can guarantee that the generated code is fully equivalence to the model defined. By using automatic code generate, human coding errors can be avoided. In addition, a feature called TinyC generator can generate compact code targeted for embedded systems with limited memory foot print, although this limit the design to single thread execution.



Fixure 22: Model based design approach with formal verification support enable design errors to be detected early

The generated code can then be integrated into your embedded project. The generated code can run on various platforms and can integrate with RTOS, as well as on baremetal targets. Also, as the design is language independent, a single model design can be ported to multiple platforms.

6 – Other programming languages

There is no shortage of alternative programming languages. For example, ARM® Cortex®-M processor-based microcontrollers can be programmed with:

- ADA (www.adacore.com, see descriptions below)
- eLua (www.eluaproject.net)
- Pascal (www.mikroe.com/mikropascal/arm/)
- Basic (www.mikroe.com/mikrobasic/arm/)
- Python (http://code.google.com/p/python-on-a-chip/)
- Forth (http://www.mpeforth.com/xc7.htm)

For example, the GNAT Pro Safety Critical development tool kit Ada is now available for use on ARM Cortex-M3 processor-based microcontrollers. There are also additional new programming languages under development, and possibly a long list of other development languages.

7- Conclusions

Modern microcontrollers such as those based on the ARM Cortex-M processors have lead to an increase of alternate programming environments. In addition to providing:

- Higher processing power,
- Comprehensive processor features and
- Unparalleled energy efficiency and flexible memory system,

These ARM processor-based microcontrollers also enabled:

- Efficient RTOS implementations, and also
- Support of various types of high level programming languages.

Some of these new programming methods are targeted at certain user types or application areas. For example, Arduino and mbed are focused on hobbyists and students, while Java development environments could be targeted for Internet of Things (IoT) applications. With the increasing availability of the ARM Cortex-M processor-based microcontrollers, there is a large market space which enables development suites and tools to be developed for specific market. The trend also enables new opportunities for software tools vendors.

Overall, the landscape of the development tools is changing rapidly and embedded software developers are no longer limited to traditional software development methods.

Acknowledgements

Large number of people had helped us in creating this paper: Robin Smith from Oracle, Regis Latawiec from IS2T, Bob Daverveld from Verum, Tom Erkkinen from Mathworks, Agnes Toan from Atmel, Anders Holmberg from IAR Systems.