Embedded C#:

Will it Beat Out C and Python in the Embedded World?

Gabriel Stroe

Lawrence Technological University

Introduction

Embedded systems and IoT devices have become a staple in our world and can be seen in just about every piece of technology there is. When considering how these systems are made one generally assumes that a small microprocessor running some compiled C language is used. But, with the plethora of languages available today is there any change that those systems can be created with something other than C. With the rise of the Raspberry Pi, Python has certainly made its way into the market. Although, what about C#? It is a common language used in many software development products. It is rather easy to learn and develop with, so can it be used with embedded applications? This is what is to be explored in this paper: with the current languages used in the embedded development world, does C# have any room to mark out its own territory and use in this market?

Embedded Systems Today

Today, the embedded world is dominated by C/C++ for traditional style embedded development and Python with the rise in popularity of the Raspberry Pi. C/C++ is so popular that it is estimated that 95% of embedded system applications are written in C/C++ (Mazzie, Montelisciani, Baldi, Fantoni, 2015). The vast majority of business applications use C or C++ with several different compilers out there depending on the microprocessor. For example, there is Keil or the GNU GCC toolchain for arm processors, and there also is Microchips XC compilers for their Peripheral Interface Controllers (PICs). The reason why C is so common in embedded applications is because it has very little overhead compared to other modern day languages. Its runtime and memory usage is very efficient (Herity, 2015).

Now even today, C is predominantly used over C++. This is most likely due to two reasons. The first is that most embedded software engineers in the industry today were only taught C and assembly during their college career, and the second being that there is a fair amount of misconceptions in the embedded world about C++ (Herity, 2015). The general misconception is the C++ has large overhead costs that require more clock cycles for the same compiled code and that it also requires more memory to store (Herity, 2015). This used to be true 15 years ago when C++ compilers made very poorly optimized machine code. Although, today this argument doesn’t really have a leg to stand on as C++ is so close to C it can be seen as an expanded version of C with extra features, many of which are very useful for embedded applications like classes, overloading methods, and namespaces (Herity, 2015). It is idea of an expanded C language that when discussed for embedded systems that C and C++ are ussally group up as one set of languages. Most compliers for embedded systems generally compile for both languages (Herity, 2015). Now, generally overhead created by C++ comes from a few features (such as virtual classes) creating reference tables that cost more memory and CPU cycles to go through. This is seen as a rather minimal impact today with current compilers (Herity, 2015). Another reason for C++’s lack of use in the embedded world stems from a lack of knowledge by the majority of users out there. This can change if as new embedded developers, who are more familiar C++, join the embedded development workforce. In addition, with the cost of faster microprocessors with more memory has dropped significantly in recent years (Herity, 2015). This really reduces the impact of any overhead that C++ might add and really opens the door for its use.

Ever since the Raspberry Pi was released the use of python in embedded applications has gone significantly, especially for the hobbyist community. Python has actually become the most popular introductory language to be taught in Computer Science Programs today (Radcliffe, 2016). This is because of its simplicity. Python is very easy to learn which allows more and more people to developing with it and with the low cost of a Raspberry Pi or an Ardunio, many hobbyists are developing applications on this platform and are now starting to migrate into the industry (Radcliffe, 2016). This is really starting to erode at the C/C++’s dominance in the embedded world. Just two years after its release, at least 8 start ups companies created products with the Raspberry Pi as its main microcontroller and most of them use Python as their primary language (Marinos, 2017).

There is a big downside to using Python for embedded system applications. That is, due to it being an interpreted language running on a virtual machine, its runtime is significantly slower. Virtualizing introduces a lot of overhead even for processors that are designed for virtualization (Radcliffe, 2016). To combat this some developers are developing low level extensions like Cython that speeds up mathematical operations (Radcliffe, 2016). In addition, Just-in-time Compilers (JIT) are being created to compile some of the python code to machine code. This runs in parallel with the python interpreter to help speed its decision making especially when traveling in a loop (Radcliffe, 2016).

C# in Embedded Systems Today

So will C# beat its competitors with C/C++ and Python? Well generally, when programmers think of embedded system languages today, C# is one of the last ones they think of if they even consider it at all. C# today is generally considering a strictly windows language with focus on client side GUIs and web based API thanks to ASP.net. This is changing though, with the recent release of .net core, C# is going multiplatform and has considerable reduced some of its overhead, largely because with .net core standard library collection is more modular then with the .net framework (De la Torre, 2016). Also, Microsoft in 2007 released the .net Micro Framework which can be compiled on ARM based processor with at least 256 KB of general flash memory and 64 KB or RAM (Miller, 2015). It uses a more efficient version of Microsoft’s Common Language Runtime to run on these small platforms. (CLR) (Miller, 2015). By and large, development with the .net micro framework is rather small and is mostly done by hobbyist in already familiar to the .net platform looking for to get into the embedded world (Miller, 2015). Another method for using C# on an embedded system is to use it with Windows 10 IOT running on a Raspberry PI or Arduino. This is actually done using the .net core libraries. It seems to be the most commonly used method for running C# in an embedded system today, although it is still relatively niche when compared to python and especially C/C++ (Dobric, 2015).

There are quite a few downsides that have been really holding C# back for becoming more common in embedded systems. One of these downsides is .net’s CLR is that it isn’t actually a compiler. It is an interpreter in its current iteration. Although, a JIT compiler with the CLR is currently in development (Dobric, 2015). This means that C# has the same issues that python has in the embedded world. It is just so much slower then C or even C++. Also because since everything in C# is an object, even when a compiler is developed, it is very likely that C# will have the same issues that C++ used to have before some more recent optimizations. C++ Compilers used to have very poorly optimized machine code when compiled which was generally linked with its class feature (Herity, 2015). C# will likely have this issue have this but with every instantiated object. Furthermore, the memory overhead of C# is fairly large. A benchmark from Computer Language Benchmarks shows that while C# with .net core is only 33% slower in terms of computation time, it is about 2000% larger when it comes to memory usage. That is a massive jump. Lastly the fact the .net has an active garbage collector running really decreases C#’s performance and likely hood to be considered for an embedded application. Although as with is done with Java in some embedded systems the garbage collector can be removed or reduced to a very small point as in what is down with the .net micro framework (Miller, 2015).

The Future of C#

So with all those negatives, where does C# actually go from here? Does it really even have a chance at competing with C/C++ or Python. C and C++ already has 95% of the market and are already very good at meeting the small overhead requirement that many embedded systems require. Also, Python is the most common language for a new student to learn in introductory computer science courses. So what front could C# attack from to grab any share of the market from these two competitors. Well there are actually three that really stand out: the dominance that the .net framework has in the industry as a whole, the way the technology is moving in terms of hardware, and the much faster development time.

When looking at the computer science industry today, C# was 4th most popular language according to Glassdoor (Bouwkamp, 2016). While falling behind SQL, Java, and JavaScript it is about twice as popular as C++ and Python. Also, SQL, Java and JavaScript don’t really have any development being made today for embedded applications (largely because SQL and JavaScript really aren’t designed for it). This really shows how poplar the .net framework has become in recent years. For general applications with the inclusion of ASP.net, it really has become an industry standard for development. In addition, the industry is moving towards Internet of Things (IoT) devices (Greenough, 2016). This shift will cause many developers to move to these IoT platforms and instead of learning a new language or trying to develop with something they are not familiar with, it be much easier for developers to use the language they are already familiar. This really starts to create that demand for C# in the embedded system world, and with a market for it comes the incentive for Microsoft to really start providing support.

Now the major argument against C# is its overhead. This is a fairly strong point, its memory cost is very high and even if Microsoft develops a compiler that can compile C# for several microcontrollers it is design in how every object is reference would most likely lead to a less efficient then compiled C++ machine code. That being said IoT is moving towards the all in one System on a Chip (SoC) type of platforms which include devices like the Raspberry Pi and Arduino. PICs are still used but generally their entire feature set is already programed from the manufacture. C# already runs fairly well in these systems using .net Core. Furthermore, the high overhead is becoming less and less of an issue as processing speed and memory density increase significantly without increasing the price. For example, the First Raspberry Pi released for $30 with a single core 700MHz processor with 512MB of RAM in 2012 and then just 4 years later in 2016 they released the Raspberry Pi 3B for $30 with a 4 core 1.2 GHz processor with 1 GB of RAM (Wright, 2016). This trend is continuing with the applications for these devices generally staying the same. This makes room for C#’s overhead costs and really minimize its impact.

All this said, why would the industry move from what it already knows in C or even C++ to C#. Even if the overhead costs are not as impactful due to current hardware, why not use C just because has already being done and is known to work. Well, the simply answer to that is development time. The reason why high level languages like C# and Python are so popular is because they increase development time significantly. This is an issue the embedded system world today and the introduction of languages like C# or even Python will really speed that process up, saving companies hundreds of hours on development time and costs; and with its familiarity to the C language the current embedded developers have, C# going to be far more appealing to Python.

Conclusion

So will C# take over the embedded systems world and become the replacement for the ever so dominate C language. Well, no it most likely won’t take over the Market completely. C’s roots in the embedded systems are just too deep for the for seeable future. Especially when it comes to PICs or real time systems that really need that high speed performance it will be likely that C will still be predominantly used. That being said, the industry is moving away from this type of development and going towards the SoC or Arduino type of product as the main controller. It is here where C# really stands a chance in the market.

C# already has strong roots in software development and the lines between software development and embedded development are blurring as IoT devices become predominant in the industry. And, as the hardware becomes faster with higher capacities of memory, the overhead concerns of C# really start to fade. Lastly C# is just so much faster to develop with then C#. The extra features it presents to the programmer allow for very quick development when compare to the rather lack luster in features C.

All of this does rest in the hands of the general market though and on Microsoft. Without the desire to use it in the industry, Microsoft is unlikely to support this type of development. The future looks promising though, as with .net core already going multi-platform and its use with boards such as the Raspberry PI is fairly common. Still, it will be sometime before change starts to happen. It is likely that C# will never be more dominate then C for true low level development, as more developers who use the .net framework go into IoT development, C# is likely to grow and become a household name for embedded systems development.

References

Bouwkamp, K. (2016, July 25). The 9 Most In-Demand Programming Languages of 2016. Retrieved April 22, 2017, from http://www.codingdojo.com/blog/9-most-in-demand-programming-languages-of-2016/

Computer Language Benchmarks. (n.d.). Retrieved April 22, 2017, from https://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=csharpcore&lang2=gpp

De la Torre, C. (2016, June 27). Cesar de la Torre [Microsoft] – BLOG Cesar de la Torre [Microsoft] – BLOG. Retrieved April 22, 2017, from https://blogs.msdn.microsoft.com/cesardelatorre/2016/06/27/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/

Dobric, D. (2015, September 22). How to run .NET Core Application on Pi2 - Win10 IoT Core? Retrieved April 22, 2017, from http://developers.de/blogs/damir\_dobric/archive/2015/09/22/how-to-run-net-core-application-on-pi2-win10-iot-core.aspx

Greenough, J. (2016, July 18). How the 'Internet of Things' will impact consumers, businesses, and governments in 2016 and beyond. Retrieved April 22, 2017, from http://www.businessinsider.com/how-the-internet-of-things-market-will-grow-2014-10

Herity, D. (2015, February 17). Modern C in embedded systems – Part 1: Myth and Reality. Retrieved April 17, 2017, from http://www.embedded.com/design/programming-languages-and-tools/4438660/Modern-C--in-embedded-systems---Part-1--Myth-and-Reality

Marinos, A. (2017, January 27). Two years of Pi: Units sold and Raspberry startups! Retrieved April 22, 2017, from https://resin.io/blog/two-years-of-pi-units-sold-and-raspberry-startups/

Mazzei, D., Montelisciani, G., Baldi, G., & Fantoni, G. (2015). Changing the programming paradigm for the embedded in the IoT domain [Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on]. New York, NY: IEEE. doi:10.1109/WF-IoT.2015.7389059

Miller, C. (2015, March). The Microsoft .NET Framework in Embedded Applications. Retrieved April 22, 2017, from https://msdn.microsoft.com/en-us/magazine/dn913183.aspx

Radcliffe, T. (2016, August 29). Python vs. C/C in embedded systems. Retrieved April 19, 2017, from https://opensource.com/life/16/8/python-vs-cc-embedded-systems

Wright, S. (2016, March 23). Raspberry Pi 1, 2, 3 compared. See the differences. Retrieved April 22, 2017, from https://www.stewright.me/2016/03/raspberry-pi-1-2-3-compared/