Proliferation in Code Generators for Microcontrollers

May 01, 2016 By justin bauer (http://mcuhq.com/user/1)

The past 7 years have seen a general trend in microcontroller vendors in supplying graphical user interfaces for configuring their microcontroller's peripherals. No longer does one need to stare at a 300+ page datasheet in order to figure out what registers to write, pins to setup, and oscillators/timers to configure before you actually see a simple 10KHz PWM signal out to drive an H-bridge so your motor can finally turn. The amount of time required to understand the steps in order to achieve even the most basic functionality is daunting for beginners.

Starting from a blank `main.c` usually leads beginners in googling "How do I do xxxxx with a PICxxxxx". This then leads them to a few forum posts or someone's blog where an extremely outdated microcontroller such as the PIC16F88A is used to perform a roughly similar process (Side note: Asking a general question like "How I setup the PWM to..." on the Microchip forum will yield answers telling you to read the datasheet anyways. Users on the TI and AVR forums are more forgiving). This then leads to other people using this old code and its outdated drivers and it self-perpetuates. Common drivers for I2C, SPI, ADC, PWM, I/O, etc are hard for beginners to write and understand. For professionals, they understand the technology and just need the time to write and debug the driver. Either way, people end up with their own rolled I2C driver and such just to communicate with a temp sensor. I remember in 2011 doing this with my CoWorkers at my job. There was always someone that had a driver to do XXXX with a PICXXXXX. I'd end up using it and finding issues and self-patching it and then calling it my own months later after several iterations and then giving it out to other people. This worked as was OK, but kinda tiresome and prone to error. Plus migrating the driver to work on different architectures was a major pain and the code was littered with a bunch of `#ifdef` hell everywhere to accommodate them all. The custom drivers are of course mandatory and convenient to have when working within the constraints of your business, but the ease-of-use of a code generator can massively reduce the headache and time to market.

Below is an outline to what I see as the Pros and Cons of using code generation tools for microcontrollers:

Pros

- Save on initial engineering time for proof of concept design
- Helps hobbyist get their widget working with minimal effort
- New parts can be readily adopted by easily migrating auto-generated drivers
- Less bugs
  - For the most part, except be wary of **new** features from the code generator or code for a new micro. The first release of Microchip's Code Configurator (http://www.microchip.com/mplab/mplab-code-configurator) for example would send out a null character if you called the UART intitilation routine. This was caused by the programmers mistakenly clearing the `TXREG` in the auto-generated code which initiated a write to the UART.
- Easier to read
  - At least for other people other than the original creator.

Cons

- Cohesiveness and multitasking hardware
  - Using the code configurator is great if the project is straight-forward and contains enough of

the hardware peripherals that you can dedicate one module for every task. This probably won't be the case in a production setting where your hardware is pushed to its limits and the `TIMR2` for example is multitasking for different features. Maybe you don't have enough PWM pins and you have to steer them or create some in software. Sometimes you have a lot of interrupts with in-depth code that needs to sift through the various interrupt sectors. This entails scattering your code in various sections of the auto-generated, `/*Place your handler here*/`. I prefer that the APIs use callbacks so that you can separate and source control the application code. Some APIs provide callbacks such as Microchip's Timer peripherals, but not all as seen in TI's Code Composer Studio (I could be wrong though).

- People don't read the datasheet
    - Yea, "RTFM" comes to mind here. Things will go wrong, even when the auto-generated code is right. Reading a simple ADC measurement could yield poor results if the embedded engineer didn't understand the significance of the sample timing. Jason Sachs has a good article writeup (https://www.embeddedrelated.com/showarticle/110.php) that dives into this issue for those interested.
- Inefficient memory usage
    - A good compiler should not generate code for unused auto-generated code, however sometimes the code will contain irrelevant initialization routines for pins or modules that you are not using. I don't think this is a large con, but I wanted to mention it.

The code generators do a fine job of abstracting the register writes which can get you up and running very quickly. You can't go wrong with a code generator as long as you ALSO understand the relevant sections of the datasheet. The devil is always in the details.

Why the shift to peripheral code generators

Code generation itself is not new for microcontrollers. Tools have existed such as MathWorks Simulink blocks and "UML to C State Machines" have been in existence for a while. Sinelabore (http://www.sinelabore.com/doku.php) in one such example of a UML to state machine program that can be invoked directly into MPLABX. I've use it in the past for state machine generation and used it to simulate my program. Here is my short github project (https://github.com/mcuhq/SinelaboreRT_UDP_CS) about it. I've outlined some ideas as to why there was a push in this direction:

1. Time to market reduction is a good marketing sell
    - The less time one has to mucker with boilerplate code to simply get a generic serial bootloader working is a GREAT step in the right direction (AN1310 (http://ww1.microchip.com/downloads/en/AppNotes/01310a.pdf)) anyone?
2. It worked for Arduino
    - You'd be hard pressed not to see an Arduino nowadays at any tech gathering whether it be at an elementary school or graduate level program. They make things dead simple and let you focus on the rest of your project. `#include everythingINeed.h`
3. Growing Complexity with Internet Connectivity
    - Implementing a TCP/IP or USB stack is tough, even for the most seasoned people. The recent push for the "Internet of Things" mandates that the microcontroller will be hooked up to the internet. If you aren't using an off-the-shelf IoT enabled device to connect to your micro such as the HC-06 or popular ESP8226, you will be stuck implementing it on the metal which will take time. Having a code generator that can generate the peripheral code for you

helps immensely.

Current Peripheral Code Generators

Here is a list of popular peripheral code generators that I have used in the past for several microcontrollers

1. Microchip - MPLAB Code Configurator (http://www.microchip.com/mplab/mplab-code-configurator)
    - Somewhat new to the game. Integrated nicely inside of their Netbeans fork called MPLABX. Mostly supports their 8-bit product line, but has plans to expand to the rest of their product families. The recent acquisition of Atmel in April 2016 may see the generator ported to the AVR 32-bit family....who knows.
2. Cypress - PSoC Creator (http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide)
    - I love coding on the PSoC5LP series microcontrollers just so I can use their easy-to-use Universal Digital Blocks (UDB). Simply drop these onto your design and optionally connect them in hardware. Double-clicking on a block will bring up the configuration details and a PDF on how to use the auto-generated API. Pretty slick, the downside is that they are expensive microcontrollers.
3. Texas Instruments - Grace (http://www.ti.com/tool/grace)
    - Stand-alone program or integrates within Code Composer Studio. Pretty similiar to MCC, but has a nice overview page that details what modules are being used and includes more text on the background usage. It also segments the configuration out to different levels such as beginner or power user.
4. Renesas - Code Generator Plug-in (http://am.renesas.com/products/tools/coding_tools/coding_assistance/cg_p/index.jsp)
    - I've only slightly used my YRPBRL78G13 board. The code generator is similar to the above already mentioned. It integrates into their Eclipse-based IDE
5. Infineon - DAVE (http://www.infineon.com/cms/en/product/microcontroller/development-tools-software-and-kits/tricore-tm-development-tools-software-and-kits/dave-tm-for-the-infineon-tricore-tm-microcontroller-family/channel.html?channel=ff80808112ab681d0112ab6b5b2807eb)
    - I have not personally used any of Infineon's products yet, but DAVE (Digital Application Virtual Engineer) appears to be the code generator for their 8,16, and 32-bit line.
6. STMicro - STM32CubeMX (http://www2.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html?icmp=pf259242_pron_pr_jun2014&sc=stm32cube-pr6)
    - A sleek interface that is stand-alone or Eclipse plug-in. Offers power consumption details which I haven't seen before in any of the other generators.

Final Thoughts

Time is money and whatever can save engineering time is a good tool to use. Nearly all microcontroller vendors ship their IDE with a code generator. I'd use them for boilerplate sections and avoid them when trying to fit code into a small memory footprint or when writing assembly.

Comments

- You can use BBCodes and Emoticons.

2 comments Most Recent

Leave a comment...

() **opal** 4 months ago

Hello Justin,

Thanks for of putting this list together. It has been very useful in outlining the trend.
Would be nice if you keep it up-to-date/fresh enough in future.

Two thoughts, though:
1. Interesting enough that these are the generators offered by the manufacturers of the hardware. As if the niche is too small for the thirdparty players.
2. These generators deal mostly with setting-up the hardware. How about the tunnable software blocks like interpreters or protocols?

Regards

Reply

justin bauer opal 2 months ago

Hello Opal,

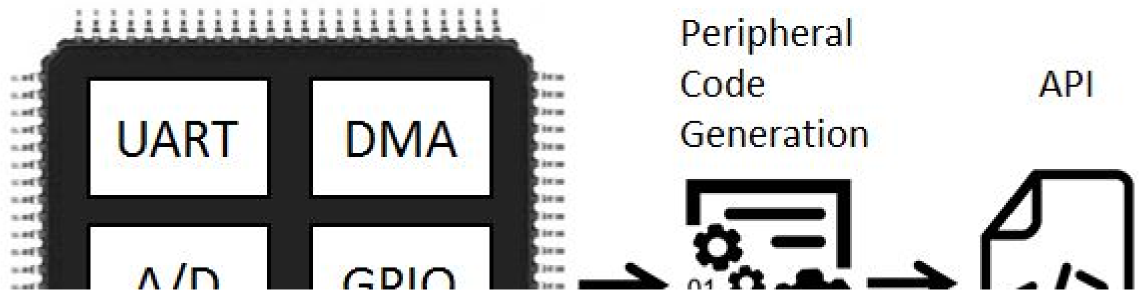
Apologies for the late reply.

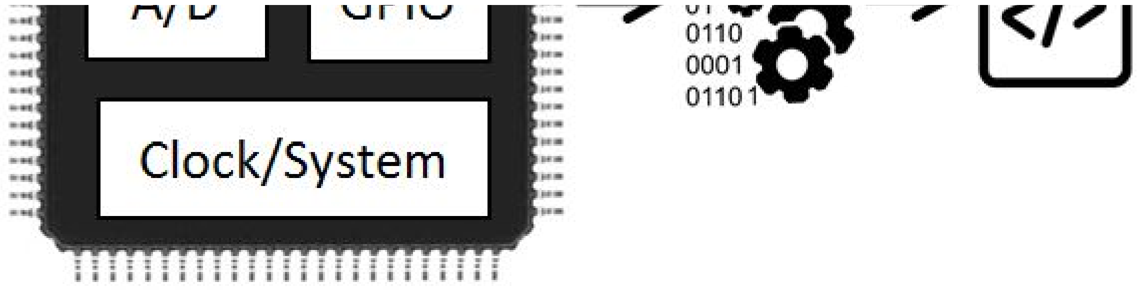
I'll try to keep this list up to date as new software is developed.

1. I don't know of any 3rd party code generators. It seems that most of the tools originate from the original manufacturer, which makes sense since they can easily add support for new modules. It will be harder for a 3rd party to identify the small intricate details between micros that make the functionality different, let alone keep up with the errata documents between hardware revisions.

2. I know that MPLABX does provide integration for MathWorks Simulink where you can configure loop control blocks for things like doing Sensorless Field Oriented Control for an AC Induction Motor. @see http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=SW007023 (http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=SW007023)

Reply





Author: justin bauer (http://mcuhq.com/user/1)
Created: May 01, 2016
Updated: May 03, 2016
Comments: 2

Related

Qt 5.6 Released for Long Term Support (http://mcuhq.com/3/qt-56-released-for-long-term-support)