Selecting Microcontrols for Amateur Radio Projects

November 2016
Erik T. Heidt, AC8VK
Tonight's talk will focus on selecting and getting started with Microcontrollers (MCU) for your Amateur Radio projects.

While the presentation will start with some basics...

I hope to bring up a number of points that would inspire you to experiment in your own projects.
Dramatic Cost Reduction + Performance Increases

Selecting the Right Starting Point for Your Project

Paths to “finished” Builds or Projects

Expansion Options (Hybrid, Shields, Plates, I2C, etc.)

Getting Started
When I first became interested in MCUs... (early 80s)

• Available “Kits” were either expensive manufacture’s demo boards or required advanced DIY skills.

• Programming required EPROMS
  • EPROMS required that you buy/build a burner (difficult for getting started).
  • EPROMS were UV erased, requiring a EV lamp/eraser (costly).

• Development environments were either...
  • Expensive (1,000s of 1980s $)
  • Primitive (open source movement had started, but wasn’t well understood) (i.e. GCC compiler was not released until 1987...)
  • Most Amateur Radio or DIY electronics projects were hand coded in machine code (again - difficult for getting started).
When I first became interested in MCUs...

• MCU’s were very limited in terms of memory or CPU performance...

“The PIC16C84 was introduced in 1993 and has been hailed as the first PIC microcontroller to feature a serial programming algorithm and EEPROM memory.”
-- wikipeda

FYI – that was **64 BYTES** of EEPROM!

• Getting started costs was over $200 (in 1980’s $).
  Accounting for inflation, etc. this is around $600 in 2016
Today there are almost too many options...

Most options under $50, several under $20

Photos Credit: www.sparkfun.com
Today there are almost too many options...

With so many good options, where should the typical HAM start with?

Most options under $50, several under $20

Photos Credit: www.sparkfun.com
Arduino vs. Raspberry Pi

These are the two platforms with the most "getting started" support.

Arduino (and Variants) | Raspberry Pi
Arduino vs. Raspberry Pi

Also have inexpensive finished build options...

Arduino (and Variants)  Raspberry Pi
## Raspberry Pi vs. Arduino

<table>
<thead>
<tr>
<th>Feature</th>
<th>Arduino (and Variants)</th>
<th>Raspberry Pi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>None</td>
<td>Required</td>
</tr>
<tr>
<td>Development</td>
<td>Cross-Platform</td>
<td>Cross or On</td>
</tr>
<tr>
<td>Clock Speeds (Typical)</td>
<td>10s MHZ</td>
<td>1.2 GHz</td>
</tr>
<tr>
<td>Memory</td>
<td>32-256 Kbytes</td>
<td>1 GB</td>
</tr>
<tr>
<td>Digital IO</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Analog IO (ADC/PWM)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Networking</td>
<td>Bolt-On</td>
<td>Built-in</td>
</tr>
<tr>
<td>Power Requirements (Min.)</td>
<td>175 mW</td>
<td>700 mW</td>
</tr>
<tr>
<td>I/O Current Max</td>
<td>40 mA</td>
<td>10 mA</td>
</tr>
<tr>
<td>Licensing</td>
<td>Open</td>
<td>Not-quite Open</td>
</tr>
</tbody>
</table>
Choose the Arduino When:

- Code requirements "fit" available CPU, memory and operating constraints
- Analog IO requirements are simple
- When open and simple hardware design enables embedding the Arduino into finished design or build
- When the presence of the OS is unnecessary

Arduino (and Variants)

No "operating system"
Choose the Raspberry Pi When ...

- Operating system is an advantage, including:
  - Leveraging device drivers for USB gear out of the box — attach and go
  - On-platform software development (may accelerate development)
  - When an "in field" ability to use keyboard and monitor is advantage
- Analog IO is either not necessary or performed though a GPIO compatible device
- Ethernet available on-device

No on-board Analog I/O
On YouTube …

A few “Gotchas” they don’t tell you about…

Or your favorite internet infotainment site…
On YouTube ...

Code Always Works. No Debugger Needed!
Cross-Platform Debugging Is Hard

The microcontroller-based devices (i.e., Arduino) are cross-compiled:

- Build code and push it to the device.
- Open source tools have very limited or no cross-platform debugging capabilities.
- Breakpoints, single stepping and variable inspection are "old school" instrumentation.
- Commercial development tools are available with these capabilities, but require special connectivity to device being debugged.

Recommendations

- Plan ahead and design in "debug" mode(s).
- Consider investment in commercial code and debugging tools. (many are “free”.)
- Consider moving the code to a more powerful platform that is able to support in-system debugging. (i.e., Raspberry Pi)
Parts Don't Explode — Unless There Supposed To...
Kit/MAKER Devices Are Fragile

- Providing a low-cost learning and exploration platform is at odds with electrical hardening. Mistakes that can "smoke" or "brick" devices:
  - Power, ground or overvoltage mistakes.
  - Shorting pins, causing excessive power draws.
  - Drawing too much total current from device.
  - Keep in mind — moderate current thought a small device can cause it to pop into fragments.

Recommendations

- Protect your eyes.
- Prototype on an "hardened" device with protection circuitry or have spares.
- Avoid using power sources capable of delivering excessive voltage or current.
Ways to Kill an Arduino (applies to Raspberry PI and others also...)

- Exceed Total Microcontroller Current
- Shorting I/O Pins to Ground or Each Other
- Apply Overvoltage to I/O Pins
- Apply External Vin Power Backwards
- Apply >5V to the 5V Connector Pin
- Apply >3.3V to the 3.3V Connector Pin
- Short Vin to GND
- Apply 5V External Power with Vin Load
- Apply >13V to Reset Pin

Credit: “10 Ways to Destroy An Arduino”
http://www.rugged-circuits.com/10-ways-to-destroy-an-arduino/

Rugged-Circuit’s blog posting provides protection recommendations to address all these issues.
Daughterboard Approach (bolt-on):
Mount an Arduino (or variant) in your finished project.
Example: K2ZIA/W8TEE Antenna Analyzer

Standalone (build-in):
Take advantage of the low part count for an basic Arduino and include it into the project.
Example: WB7FHC Morseduino
Arduino Standalone Tutorials

- “Arduino - Setting up an Arduino on a breadboard”
  https://www.arduino.cc/en/Main/Standalone
- “How-To: Shrinkify Your Arduino Projects”
  https://www.youtube.com/watch?v=30rPt802n1k
- Use an ATtiny 8 or 14 pin in your Standalone:
  http://highlowtech.org/?p=1695
Raspberry PI | How do you migrate to a finished build or project?

Daughterboard Approach (bolt-on): The R-PI is not Open Hardware and complex (no breadboard / standalones), but the Raspery PI Zero is compact and $6/each.

Standalone (build-in): Requires heavy-EE redesign or use of a commercial System on Module (SOM), such as the i.MX6UL. (Re-code and redesign still required.)
Daughterboard approach to expansion:
• Arduino cards are called “Shields”
• Raspberry PI cards “Plates”

There is a wide range of these available. There is also a large “clone” market. Be prepared for a little tinkering to get these to work.

Examples:
• Prototype spaces
• Relay cards
• Displays
I2C and SPI can be used to add support chips or additional devices using simple interfaces. Supported by both Arduino and Raspberry Pi

Typical options:
- Analog/Digital Controllers
- Digital Frequency Oscillators
- Digital Potentiometers
- LCDs or displays
- Multiplexers
- Real Time Clocks
- Sensors
- Storage (EEPROM, Flash)

Digital Potentiometer being used to control LM567’s target frequency in WB7FHC’s CW decoder
Make: Getting Started with Arduino

Good first read for newcomers.
Check your local library – this is not a “reference”.

SparkFun Inventor's Kit Curriculum
(https://learn.sparkfun.com/resources/39)

Manual and full open course work available for download and use.
Great for self-instruction or teaching others.
**Arduino Projects for Amateur Radio**

By our own... Dr. Jack.

This book is a great next step after you are familiar with Arduino and focused on Amateur Radio example projects.

**Beginning C for Arduino**

By our own... Dr. Jack.

Getting value out of MCU involves programing. This book actually gets into more advanced topics than the other books I reviewed. I would consider it a “beginning and intermediate”.
**Practical C Programming**

While not about Arduino or MCU, this is my favorite introduction to C book.

---

**Design Patterns: Elements of Reusable Object-Oriented Software**

One way to learn how to write better is to review the work of others. This is true for language skills and for code. This book is packed with fantastic code. While not Arduino or MCU oriented, this is great for pressing the Intermediate to Advanced understanding of C++.
Make: Getting Started with Raspberry Pi

Again, great for first steps... but you will outgrow this book quickly.

Next steps with the Raspberry PI are a little murky due to:

1. Need for Linux skills (many basic books are available – check public library)
2. Wide range of programing options (Python, C/C++, Linux command line – name it)
Thanks for your time...

...Questions?