


# CS486C – Senior Capstone Design in Computer Science

## Project Description

<b>Project Title:</b> Lightweight Command and Control Solution for Embedded Products	
<b>Sponsor Information:</b>  Mission Systems	<b>Randy Derr</b> , Staff DPME General Dynamics Mission Systems Randy.Derr@gd-ms.com  General Dynamics

### Project Overview:

The Internet of Things (IoT) has ballooned into a major sector of the high-tech economy over the last decade, as the cost of small, low-power Wifi chipsets has dropped and Wifi-based broadband has become essentially universally available in modern households. IoT refers to the connection of thousands of everyday devices – ranging from cars to refrigerators to light switches to security cameras and many more – to Wifi broadband, making them remotely accessible/configurable. All of these devices, of course, communicate over Wifi, which is ultimately a radio signal; this has made digital radio technologies a vital foundational technology for supporting the information revolution. Since the early 90s, software defined radio (SDR) has been around to help detect available spectrum and switched frequencies when needed. This will become increasingly important as more and more IoT devices come online; 20.4 billion by 2020, according to Gartner. Add to that the growth of military and commercial drones, wearables, and medical equipment, and you can see why some analysts predict SDR will be worth \$29 billion by 2021.

### Challenges in Embedded Systems

Embedded systems/products, such as IoT, must run on very small embedded chipsets, and therefore must maintain a minimal footprint; small footprint is defined as having limited memory, CPU (power), small form factor, etc. There are also other limiting factors to also consider such as Power (W) utilization, duration of runtime and the overall cost of the product. With these physical limitations of the processing hardware, any applications that run on this system must also conform to these limits, making clever, efficient systems design a central design requirement: An IoT application must run with minimal RAM usage, FLASH (image size), and PROCESSES (threads). Software Defined Radio (SDR) is a perfect example of such an embedded system, and is what our team at General Dynamics is tasked with perfecting. For an SDR, it is crucial that any auxilliary applications (e.g., processes that support external control or monitoring of the SDR) run within a minimal software footprint in order to devote a maximum of the limited hardware/processing resources on the embedded system to the main SDR code/processes to perform its primary function as a radio (transmitter or receiver). At the same time, there is a need to have some sort of external, real-time access to the device to monitor/configure its performance during deployment. In short, what is needed is a clever solution that provides an easily-accessible, preferably GUI-based monitoring and control infrastructure for deployed SDR devices, but that maintains a minimal footprint on the devices themselves.

The goal of this project is to explore and prototype such a solution, and is presented by the Radio Team at General Dynamics, whose job it is to develop SDR solutions for our customers. Our team is involved in a wide variety of development activities spanning the hardware/software spectrum: we develop novel hardware, but also work to develop new software running on a variety of legacy platforms; our team needs to understand how to develop software and firmware on both modern and legacy processors, and is always looking for unique ways to modernize existing software.

### The Problem: A lightweight infrastructure for SDR command and control

When SDR devices are deployed, there must be some way to monitor their performance and (re)configure their behaviors; this is what we call “a command and control (C2) interface for the SDR”. Our current solution to this challenge was designed in 2008 based on Adobe® Flash®, which was later “updated” into an HTML5 based solution. The problem is that this solution is based on outdated technologies, had been band-aided together over

the years to keep it going, and was never that efficient to begin with. In the Adobe® Flash® version, the system was based on communication via platform generated (we call the SDR device the “platform”) XML files: A simple browser-based GUI would query the platform and new XML files were generated, e.g., to encapsulate status/control messages; for every type of status, action, command, etc. XML files were created and exchanged. This method of communication was ported to HTML5 using the same PHP script, but wrapping the Adobe® Flash® with JS instead.

As one can imagine, this is not a very efficient way to communicate. A high-frequency status message, e.g. one that is monitored, say, once per second, requires a constant stream of XML files to be generated. In addition, as the HTML layout gets more completed (more information on the screen), the platform needs to constantly create new and more complex XML files. In short, this situation is not congruent to the initial statement made above, where the user-interface application should be efficient and run in a small footprint.

### **The envisioned system: Project details**

The aim of this project is to explore a complete revision of the C2 system for SDRs. We would want to keep the browser-based GUI concept for the front end, as browsers provide a nice universal GUI platform that is inherently internet-accessible. Although the Capstone team will develop such details, we envision a simple web application in which SDR devices can be registered, monitored, and configured. Some key features of the system will include:

- Develop an appropriate on-board database for SDR devices. A key design challenge will be developing an efficient backend database cornerstone for the system; if data and logs can be stored in an efficient flexible manner, this can serve as the basis of a simple extensible solution. As IoT and SDR technologies mature, new applications will be made available on embedded platforms and a flexible DB core will support such evolution. Thus, an early design task will be to evaluate small footprint Database Management Systems (DBMS) for embedded systems to determine the best fit; some of these might include ITTIA, McObject, HamsterDB and others.
- Support all of the major command, control and status functions required on the SDR, including:
  - Status of radio (Transmit, Receive, File Status, etc). This is pulled from the RF Core and not on the application processor. This means that there is additional backend query that is needed
  - Command of radio. To start or stop the radio from transmitting or receiving. Again, this is control to the RF Core and not on the application processor
  - Retrieve logs from SDR.
- Provide a web-based GUI solution to all (sysadmin) end-users to easily monitor and manage a large deployed network of SDR devices. A bare bones solution will provide a very simple interface that just lists all active SDRs; a more complete solution will flesh this out into a truly usable GUI, which might include features like:
  - A user management module to support authenticated access, role-based user permissions (i.e. monitor-only vs edit/delete, etc.)
  - A power dashboard that allows users to overview all deployed SDR devices and their current status at a glance.
  - Ability to graphically view/analyze performance of any device in more detail, e.g., a graphical tool based on logged data that can graph various performance parameters in zoomable timeframes
  - An alerts mechanism that allows SDR admins to set “alerts” that fire when certain events are detected in monitored SDR devices.
- The solution should run on a simple platform (we recommend a Raspberry Pi for easy availability), which will serve as a stand-in for a real SDR system.
- Must provide performance profiling tools for the chosen platform (e.g. the Pi) that clearly show the footprint (memory, processor, etc.) of the on-board elements, i.e., the on-board process the front-end GUI is communicating with, the on-board DB, and any other elements on the device). This is vital to see how well the solution meets our goal of minimizing the on-device footprint of the C2 interface.

### **Stretch Goals**

- The solution outlined above is based on a simulated SDR device (i.e., the Raspberry Pi). If time remains in the closing phases of the project, it would be ideal to test and verify the performance of the solution on

several other embedded platforms. Some possible candidates include BeagleBone Black and BeagleBoardXM

The software solution that is developed will be the first step in integrating into our existing baselines. This will allow our SDR to mature and use modern embedded techniques to run in current and future products.

### **Knowledge, skills, and expertise required for this project:**

- Knowledge of mobile application programming frameworks, with particular emphasis on cross-platform frameworks like Ionic and React Native.
- Knowledge of modern Web2.0 programming techniques required to develop the administrative web app
- Knowledge of back-end server and database technologies, with emphasis on configuration and deployment of cloud-based server resources.

### **Equipment Requirements:**

- Laptop with Virtual box (open source), running some Linux variant (Ubuntu is recommended)
- Raspberry Pi will be provided

### **Software and other Deliverables:**

- The software applications as described above, deployed and tested successfully with real data. Must include a complete and clear User Manual for configuring and operating the software.
- A strong as-built report detailing the design and implementation of the product in a complete, clear and professional manner. This document should provide a strong basis for future development of the product.
- Complete professionally-documented codebase, delivered both as a repository in GitHub, BitBucket, or some other version control repository; and as a physical archive on a USB drive.