CLASSPERT

MAKING EMBEDDED SYSTEMS BY ELECIA WHITE

FINAL PROJECT REPORT

FLIGHT CONTROL SYSTEM ADAPTER

Peter Griffin

TABLE OF CONTENTS

1. INTRODUCTION 3
2. APPLICATION DESCRIPTION 3
3. HARDWARE DESCRIPTION 3
4. SOFTWARE DESCRIPTION 6
   4.1. Software licensing 9
   4.2. Software versioning control 9
5. BUILDING INSTRUCTIONS 9
   5.1. Building the hardware 9
   5.2. Building the software 9
   5.3. Testing and debugging 9
      5.3.1. Hardware tools 9
      5.3.2. Software tools 10
6. FUTURE DEVELOPMENTS 10
   6.1. Preparing for production 10
   6.2. Extending the idea 10
7. GRADING 11
   7.1. Project Criteria 11
   7.2. Self-assessment 12

1. INTRODUCTION

While radio control (RC) systems for RC aircraft and drones are common, there are not yet good way to use other control systems with most off-the-shelf flight computers. Manned drones and personal flight systems would greatly benefit from being able to adapt high precision wired control systems (such as those commonly used in earth mover equipment and cranes) to work with a variety of flight computers. Existing control systems for these craft are often difficult or impossible to apply to new designs because they are invariably customized to each system. To address this gap, I have created a system for rapidly and accurately converting the analog signals from control systems (such as those found here) to the 16 channel SBUS or Inverted SBUS protocol commonly used on existing flight computers.

2. APPLICATION DESCRIPTION

Some of the important components include an ADC that can handle up to eight channels at 12-bit resolution or higher. At least three major sets of hardware are required: something to adjust the voltage levels of the analog signals, something to log errors, and an indicator LED and/or piezo buzzer for cueing humans to review the data log when errors are detected. Additionally, the ability to invert the signal if needed, either through software or with external hardware. An overview diagram of the components can be seen below.

3. HARDWARE DESCRIPTION

The system is based on a MK20DX256VLH7 microcontroller that is included in the Teensy 3.5 Development Board from PJRC. This microcontroller features 512 KB of flash memory, 256 KB of RAM, a clock speed of 120 MHz, 12-bit ADC, 17 total timers including 4 16-bit timers, among other common peripherals such as UART, I^2C, SPI, USB, DMA, etc.

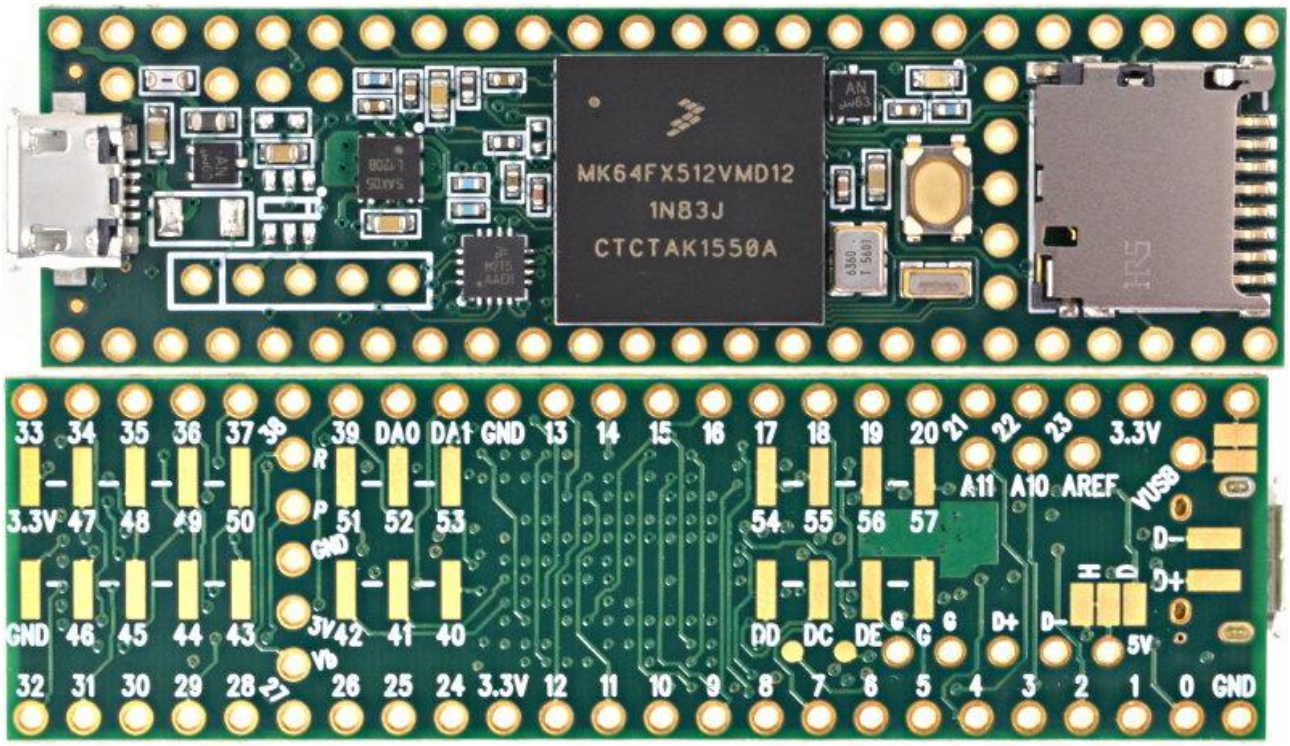


Figure 3-A – Teensy 3.5 Development Board (Top and bottom)

The Teensy also features an LED and a micro-SD card slot that are used as part of the prototype's interface.

The control interface consists of two SG Series joysticks made by Ruffy Controls. These use Hall effect sensors to very precisely measure each axis and each axis includes a backup sensor. The particular pair I'm using require a 5V input and outputs between 1V and 4V to indicate position for each sensor. The left joystick has a single button on the top while the right joystick has a pair of buttons on the top and a grip button (to be used as a dead man switch).

Figure 3-B – SG Series Joysticks (from Ruffy Controls)

For analog signal processing, I'm using a set of 8 voltage dividers and pair of LM324N op amps (in a voltage follower configuration) to rescale the analog output from the 1V-4V range to the 0.75V-3V range. The parts are powered from the 5V supply rail.

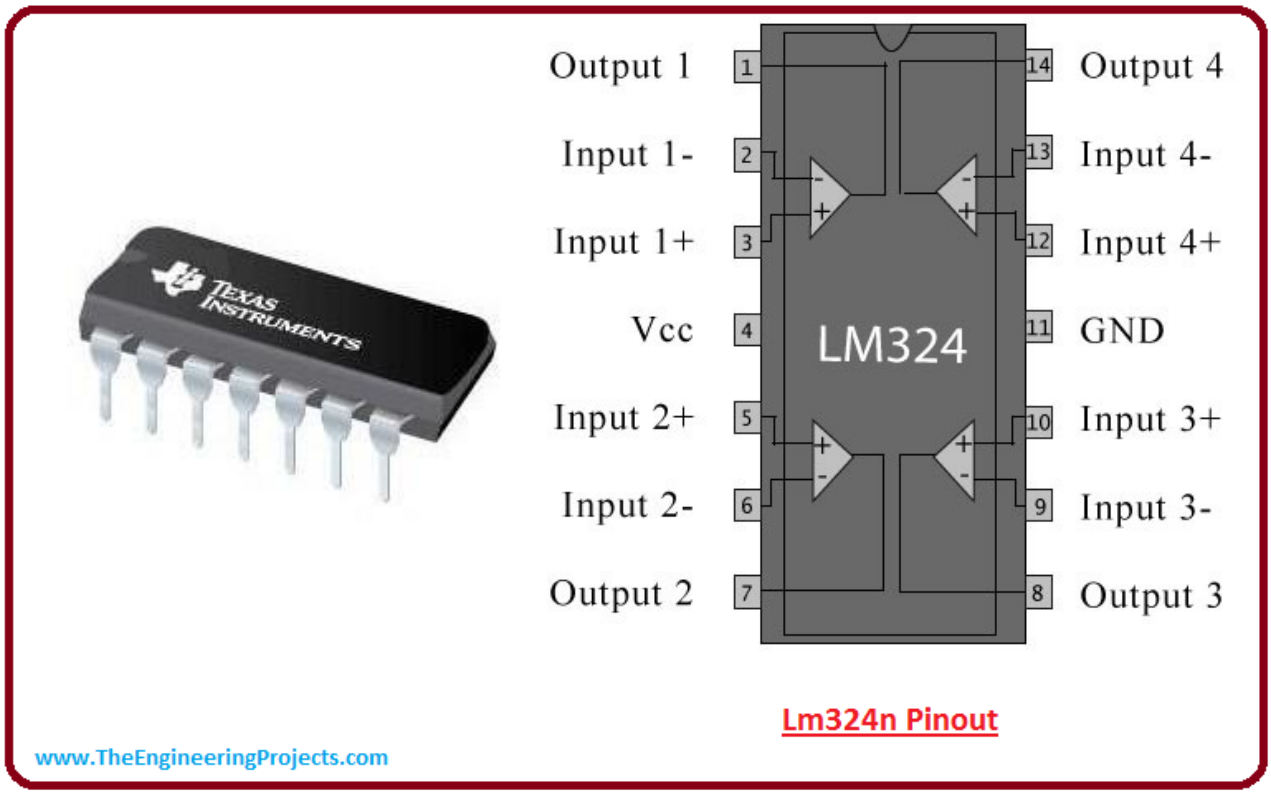


Figure 3-C – LM324N (from TI)

A hybrid hardware-software block diagram is presented next, detailing the parts and their basic connections:

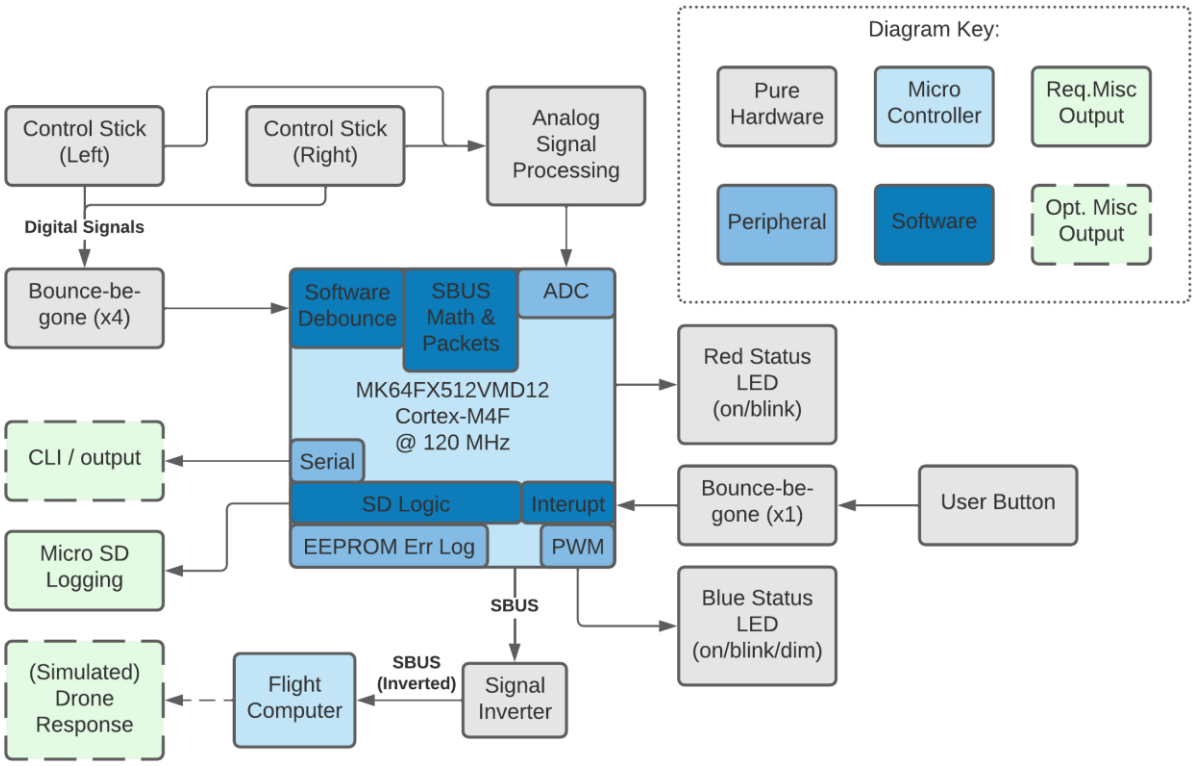


Figure 3-D – Hybrid block diagram

4. SOFTWARE DESCRIPTION

Since the project would constitute a safety-critical system, the primary software goals are reliability and robustness. Given the limited development time, this likely comes at the cost of flexibility. A brief summary of each software component is described below, followed by an overview of how the pieces work together:

ADC:

There are eight sensors (two for each axis on each of the joysticks) that are captured by the internal ADC. This raw reading is digitally processed to handle potential errors and rescale the output.

PWM:

The PWM is generated by the analogWrite(…) function in device servicing mode. This signal drives the blue LED to allow for a third status to be indicated by dimming the LED (see state machine diagram).

SDIO:

The Teensy 3.5 board has an onboard micro SD card slot that supports access via SDIO. The recommended method for managing SD card access is the SdFat library and is used to facilitate copying data from the EEPROM. Note, this process is not completely implemented at the time of submission.

Serial (UART):

Serial output is used to send packets using the SBUS protocol to the flight computer. A separate serial connection is also used to print debug information to a connected terminal if available. In device servicing mode, the terminal can also be used to issue commands. Note, this last feature is not completely implemented at the time of submission.

Button interrupt:

The user button triggers one of two different ISRs depending on whether the device is in operation mode or device servicing mode. Software debounce is unnecessary because the interrupt is idempotent and only sets flags cueing the main loop to change state if it has not already.

- Operation Mode – Turns off the warning LEDs and sets blink flag to off.
- Device Servicing Mode – Cueing the device to retry copying data to the micro SD card, or cueing the device to begin clearing the EEPROM depending on current state.

Interesting algorithm:

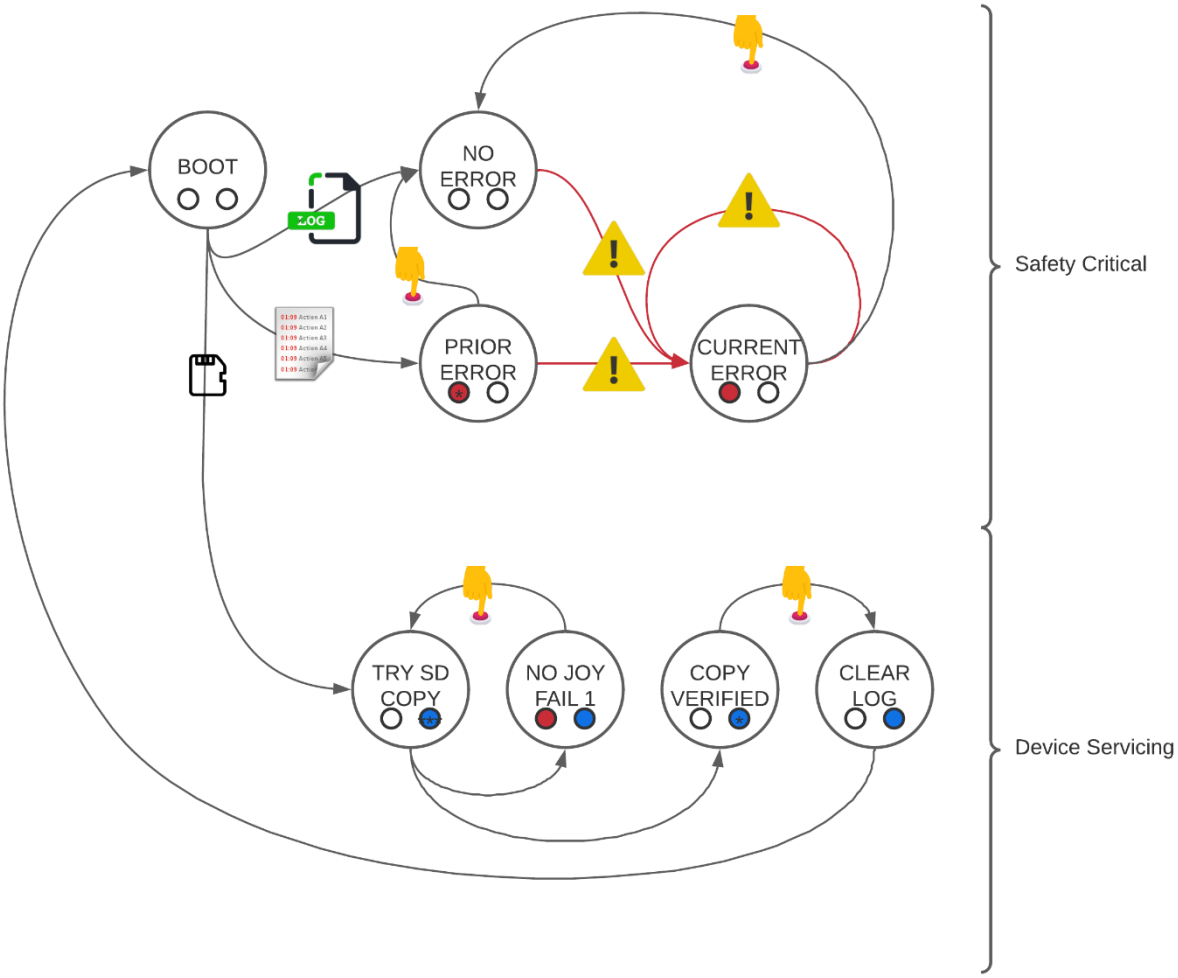
Converting the raw inputs from the ADC into the outputs expected by the flight computer requires a few steps.

- Bound the input. This prevents unexpected high/low values from creating outputs beyond the allowed range.
- Select the most reasonable sensor reading. Because each axis only has two sensors, blending the readings is still vulnerable to erroneous inputs, but there aren't enough sensors to take the median value. Instead, the system uses the reading from whichever sensor is closer to the midpoint of the range.
- Rescale the output. The range of ADC values must be mapped to the range of values commonly transmitted over SBUS for a radio receiver.

State machine:

The state machine used in this project is diagrammed below. The primary mode of operation is determined on bootup by the presence/absence of a micro SD card. These modes are described below.

- Operation mode. In this mode, the device outputs SBUS control signals over serial while using the red LED to indicate the presence of errors that require user attention.
- Device servicing mode. In this mode, the user can extract data from the EEPROM onto a micro SD card for analysis on a computer.



Summary of student-written and reused code:

This project was based on the student's earlier work creating an Arduino prototype for a similar control system adapter. While the core functionality is similar, the project is now a proper C++ program instead of an .ino sketch and has the following improvements.

- Error Logs are accessible without re-flashing the device
- Removeable micro SD card enables logs to be reviewed on a computer without the need for a USB cable or serial connection (although that works too).
- Many function and variable names have been changed to improve code clarity.
- A software de-bounce library was created based on this hackaday blog post, however, the code is largely the student's own creation as the example code in the blog is not functional.
- Interrupts caused by the user button.

Parts of the code the student did not write themselves include:

- The method of converting outputs into SBUS packets as shown in this blog.
- All Arduino HAL functions
- EEPROM and SdFat libraries and related functions
- Code leveraging the SdFat library and the command line interface was largely adapted from the ExFatLogger.ino example.
- Any PlatformIO environmental code such as the header file with board-specific settings for the Teensy 3.5.

4.1. Software licensing

Most of the Arduino HAL and functions are covered under the GNU General Public License Version 2 as it was simply not feasible to replace the various Arduino libraries that have good documentation and support on the Teensy 3.5, given the time constraints for achieving a functioning prototype. Future work may address this, but this is the current state as of writing. The SdFat library is licensed under the MIT License, details included in comments in the code.

4.2. Software versioning control

Git and Github were the primary version control tools used in this project. It turns out Git is a lot simpler when you don't have to coordinate with others.

5. BUILDING INSTRUCTIONS

5.1. Building the hardware

Bill of materials:

- 5V power supply capable of supplying 200mA continuously (not including any power supply needed for the flight computer)
- Left control stick (see section 3)
- Right control stick (see section 3)
- 2x LM324N op amps
- 8x resistive dividers consisting of:
  - 1/4W, 3000 ohm resistor
  - 1/4W, 12000 ohm resistor
- 5x hardware RC debounce circuits consisting of:
  - 1/4W, 1000 ohm resistor
  - 1/4W, 10000 ohm resistor
  - 0.1 uF ceramic capacitor
- User button (momentary switch)
- Power switch (either SPST or momentary reset button depending on the system this is used with)
- 4x 5mm indicator LEDs (2 green, 1 red, 1 blue)
- 1/4W, 220 ohm resistor (for green LED from 5V rail)
- 1/4W, 100 ohm resistor (for green LED from 3.3V rail)
- 1/4W, 100 ohm resistor (for blue LED from 3.3V pin)
- 1/4W, 120 ohm resistor (for red LED from 3.3V pin)
- Micro SD card extension (if needed to make the slot more accessible)

5.2. Building the software

Code available at: https://github.com/cadet702/Red_Jellies

Code was built with PlatformIO version Core 5.2.4; Home 3.4.1.

5.3. Testing and debugging

5.3.1. Hardware tools

Digital Multimeter

Oscilloscope (Analog Discovery Pro 3450)
UART to Virtual COM USB converter

5.3.2. Software tools

Visual Studio Code
PlatformIO Plugin
Mission Planner (drone simulator)

6. FUTURE DEVELOPMENTS

6.1. Preparing for production

This prototype was assembled on a breadboard and encased in a 3D-printed housing. The non-trivial resistances introduced by the breadboard seem to have contributed substantially to the observed instability in analog signal processing. In order to get the project ready for production, a second prototype needs to be created with soldered connections on perfboard to ensure that the noise is within acceptable levels before attempting PCB design.

6.2. Extending the idea

The most recent list of additions and enhancements under consideration can be found on the GitHub project page. At the time of writing, there are still some substantial improvements that can be made purely with software or small hardware changes that could improve the quality of the device substantially. In my opinion, the top three would be: changing the op amp circuit to improve the dynamic range of the analog signals, a handful of power saving changes, and numerous software changes to remove the Arduino licensing limitations (as well as improve clarity, reliability, portability, and efficiency).

Beyond the immediate changes, it is important to make the hardware design compatible with multiple boards to guard against shortage. In particular, the Teensy 4.1 has similar capabilities and looks to be easier to keep in stock. Making the design cross compatible will require carefully selecting pins that serve the same purpose on both boards and finding pin-compatible, low power, rail-to-rail op amps for the analog signal processing since the 4.1 is not 5V tolerant.

7. GRADING

7.1. Project Criteria

CRITERIA	GRADE	COMMENTS
Use a Cortex-M processor	2	MK64FX512VMD12 Cortex-M4F @120 MHz
Have a button that causes an interrupt	3	User button on the body of the device causes an interrupt in both modes of operation.
Use at least three peripherals such as ADC, DAC, PWM LED, Smart LED, LCD, sensor, BLE	2	The following are used: ADC (x8 inputs) PWM LED SDIO (for micro SD card) work in progress Emulated EEPROM (for error logging) Serial (for SBUS output) Serial (for debug and command line) work in progress
Have serial port output	2	SBUS protocol output to flight computer, debugging information is printed to the serial port, and a command line interface is available in debug mode.
Implement an algorithmic piece that makes the system interesting	2	The simple algorithm for selecting the sensor that reports a value closest to the center position qualifies, but is otherwise unimpressive.
Implement a state machine	2	Described and has diagram. Implementation is implicit as opposed to explicit.
Bonus: An analysis of the power used in the system (expected vs actual)	1	Only coarse power analysis performed and no attempts to improve performance.
Bonus: Implementation of firmware update with a description in the report of how it works	1	Nothing beyond the default firmware update process available through PlatformIO.
Bonus: A description of profiling the system to make it run faster	1	No system profiling performed.
Bonus: Having a version control history showing the development process	3	Version history available through GitHub and project board detailing prioritized tasks.
TOTAL		A total of 15 + 6 are available

7.2. Self-assessment

CRITERIA	GRADE	COMMENTS
Project meets minimum project goals	2	Additional hardware and peripherals used. Well documented project. Command line on serial port (work in progress).
Completeness of deliverables	2	Code is readable given the report as a description. Video shows code working. Video demonstrates the project and is explanatory.
Clear intentions and working code	2	The system performs approximately as described in the report and code.
Reusing code	2	Student code was identified.
Originality and scope of goals	3	The student has gone far beyond the requirements to make something novel and awesome.
Bonus: Power analysis, firmware update, or system profiling	1	Only coarse power analysis performed and no attempts to improve performance.
Bonus: Version control was used	3	The log and project board show the project being built and outline future work that has already been identified.
TOTAL		A total of 15 + 6 are available