# Project Proposal

## Raspberry Pi Kernel

## Summary

The Raspberry Pi offers a unique experience for students and developers to learn/design and program with a system on chip. This small system offers many unique experiences for all types of interests.

My proposal is to develop a kernel for the Raspberry Pi. This kernel will be focused on a couple of objectives to support GPIO, a user console and UART. The Raspberry Pi 2 uses an ARM cortex processor with a Broadcom BCM 2836 peripherals controller. The following Broadcom peripherals are supported by the ARM cortex processor: <https://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf>

* Timers
* Interrupt Controller
* GPIO
* USB
* PCM/I2S
* DMA controller
* I2C
* I2c /SPI slave
* SPIO, SPI1, SP2
* PWM
* UART0, UART1

My project will focus on the design and implementation of the kernel interfaces to control the GPIO, interrupts, UART and user console I/O. The UART will be used to create a “headless PI” this will allow for users to connect to the system via serial console. Users will be able to create circuits using the 3.3v GPIO by inputting commands into the serial console. They will also be able to create inputs on the GPIO pins and detect line level on those inputs. When a line level change is detected on a circuit the system will allow for an interrupt handler to change the state of the system. The scope of the project is limited but with good design practices the goal is to be able to expand the kernel to support a greater range of functionality, features and purpose.

## I/O descriptions and examples

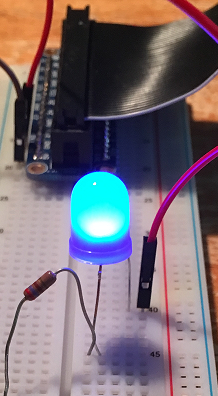
There are four types of input that will be covered GPIO output, GPIO input and User input. The specific examples of I/O usage are described below.

**Example 1: User Input**

User Input: set\_gpio\_16, out /Kernel will sets up GPIO pin number 16 and then sets the line level to 3.3v.

**Example 2: GPIO output**

User has set up GPIO pin to drive 3.3v. This GPIO out would typically drive an LED or some other small electrical device.



**Example 3: GPIO Input**

User inputs into (set\_gpio16, in) (lvl\_dtct\_16) gpio 16 will then be set to an input line. The typical use case for this will a pull down switch. When the switch is engaged the line will then be pulled down. The lvl\_dtct\_16 will cause an interrupt to be sent.

**Example 4: User interface Output**

When an interrupt has been received as in example 3 the console will display a message that an interrupt has been received.

## Requirements

* 1. Definite Requirements
     1. User shall be able to send input through serial console cable to system
     2. System shall manage the Memory Mapped IO registers
     3. Users shall be able to send GPIO commands to the system
        1. This shall include initialization
        2. Setting Line level High
        3. Setting Line level Low
        4. Managing line level detection
  2. Nice-to-do
     1. Semaphore and or Spin Lock for asynchronous usage of GPIO pins
     2. Higher function interrupt mechanism
  3. Neither of the above
     1. Basic process control
     2. Memory management
     3. Implement ANSI C specification in order to become self-hosted.

## Properties

* 1. Sufficient

The design should meet the basic requirements in order to provide meaningful use to the user

* 1. Modularity

The kernel design needs to be modular so that it can be easily expandable to provide other layers and functionality.

* 1. Robustness

The kernel needs to be robust in order to provide sufficient protection of the hardware

* 1. Efficiency

The kernel design must provide efficiency so that functionality can be accessed and executed in a reasonable amount of time.

* 1. Cohesion

Cohesion must also be considered in this way it will make it easier to maintain and expand the kernel.

## Packages

* 1. UART – Package for UART
  2. GPIO - Package for GPIO
  3. MMIO – Base Package to initialize the hardware
  4. Console – Package supporting console IO
  5. Kernel – Package for initialization
  6. Rejected Package Design:

Kernel- Kernel initialization package

MMIO – Base Package to support the hardware initialization

I/O – Package to support IO functionality

The above package design was rejected because it did not support cohesion in the IO package. This would require that all IO functionality would be implemented under one package. However, the I/O is controlled through different features of the kernel. The decision was made to create separate packages for those features.