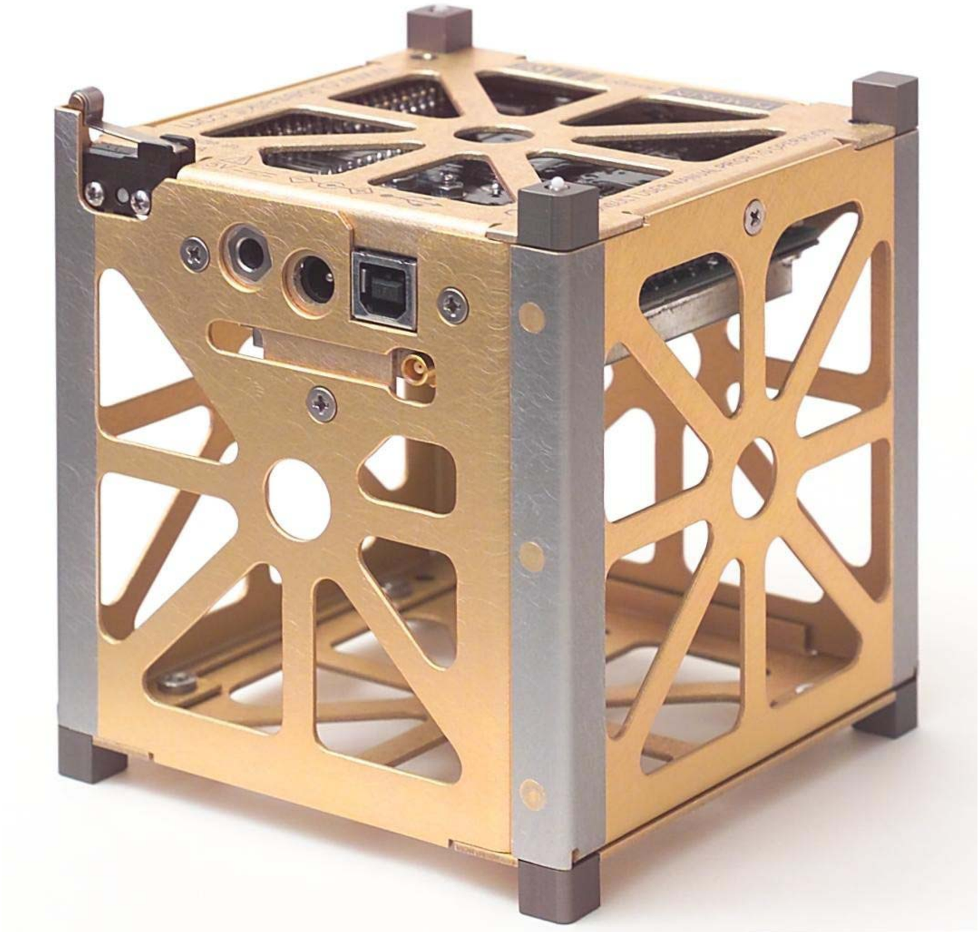


Making the Most of Limited I/O

(in your CubeSat and everywhere else)

Andrew E. Kalman, Ph.D.

Introduction

- Andrew E. Kalman
  - President and CTO, Pumpkin, Inc.
  - Author of Salvo — The RTOS that runs in tiny places.™
  - Creator of the CUBE SAT KIT
  - 20+ years of embedded systems design and programming experience.
  - Contact: aek@pumpkininc.com

Outline

- Overview: Presentation Goals
- Part I: Sharing On-Chip Peripherals
- Part II: Simultaneous Input & Output
- Part III: GPIO Pin Personalities
- Part IV: Efficient GPIO Pin Decoding
- Part V: Summary

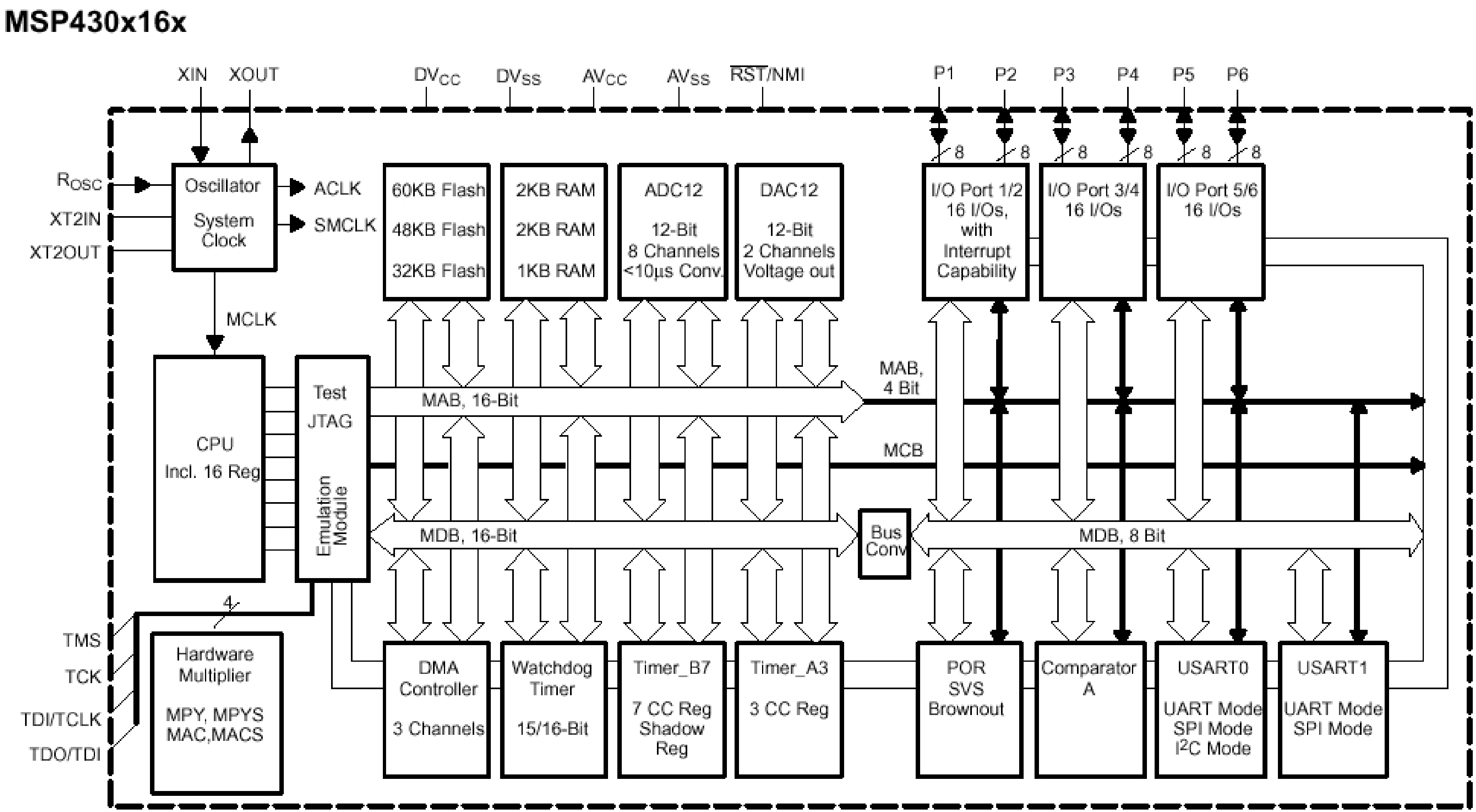
Overview

- This presentation is targeted at CubeSat developers who are either in the hardware and software design stages of their CubeSat mission or are looking at ways to add new functionality to an existing CubeSat design.
- We discuss how to achieve module-level peripheral sharing with a minimum of extra circuitry and under efficient software control.
- We examine individual GPIO pins to see how they can perform more than just a single function.
- We present circuit-design techniques that allow GPIO pins to function simultaneously as both inputs and outputs.
- Examples of how these ideas are implemented in the CubeSat Kit are also presented. *The emphasis is always on using as few GPIO pins as possible.*

Part I: Sharing On-Chip Peripherals

- Modern MCUs are chock-full of useful peripherals:
  - e.g. TI's MSP430:
    - GPIO
    - Clock Module
    - Timer_A3
    - Timer_B7
    - USART (with UART, SPI and I2C modules)
    - ADC12
    - DAC12
    - DMA Controller
    - etc.
- In order to be most efficient (viz. mass, power, cost) let's use a given peripheral in *multiple operating modes* instead of using multiple peripherals in single operating modes.

Part I: Sharing On-Chip Peripherals



Part I: (cont'd)

- Sometimes a multi-mode peripheral's GPIO pins are uniquely dedicated to a particular mode, and sometimes they are shared amongst different operating modes.
  - E.g. TI's MSP430's USART0 peripheral:

UART	P3.4 P3.5	Tx Rx	Output Input
SPI	P3.3 P3.2 P3.1	SCK MISO MOSI	Output Input Output
I2C	P3.3 P3.1	SCL SDA	Output Input & Output

Part I: (cont'd)

- Basic thinking:
  - *"I'll use peripheral X connected to device A in mode 1, peripheral Y connected to device B in mode 2, and add another MCU to connect to device C in mode 3. How hard can it be to add another processor?"*
- Intermediate thinking:
  - *"I'll use peripheral X connected to device A in mode 1 and to device B in mode 2, and bit-bang device C in mode 3 via unused GPIO pins with some software I found on the web. Software peripherals are **so** cool."*
- Advanced thinking:
  - *"I'll use peripheral X connected to device A in mode 1 and to device B in mode 2, and by adding a small circuit and using one more GPIO pin I can even connect peripheral X to device C in mode 3. This is kinda fun!"*

Part I: (cont'd)

- In this example, can we use all three modules of a single peripheral in our design? Yes!
  - The UART's pins are separate from SPI and I2C, so a serial device (inherently point-to-point) can be permanently attached to the UART module (**`P3.5`** & **`P3.4`**).
  - SPI devices can be attached to the SPI module (**`P3.3`**, **`P3.2`** & **`P3.1`**). Each device requires its own chip select signal via a separate GPIO pin.
  - I2C devices can be attached to the I2C module (**`P3.3`** & **`P3.1`**).
- To be able to use all three modules, our connected devices need to meet a few requirements:
  - The MCU must *initiate and terminate* all communications.
  - Each device's *communications must not be corrupted* by the actions of the module or the other devices.

Part I: (cont'd)

- Example triple-mode application:
  - UART: still camera
  - SPI: magnetometer
  - I2C: ADC (solar panel currents)
- Simple application:
  - Initialize USART0's UART module for camera, request image data from camera and receive it
  - Initialize USART0's SPI module for magnetometer, select magnetometer, get flux data, deselect magnetometer
  - Initialize USART0's I2C module for ADC, get data
  - repeat
- The faster you can communicate with a connected device, the quicker you can switch between devices and between modules.

Part I: (cont'd)

- Gotchas:
  - The (re-)initialization of each module is necessary, especially if the source of the previous initialization is unknown or unpredictable. Module initialization usually requires several registers to be configured. *Better to re-write the registers completely* than assume certain bits are (still) configured as you need them.
  - With interrupt-driven code, *it's essential to ensure that communications are complete between the module and a connected device before you change to a different module.* E.g. if talking at 1200bps (N,8,1) to a serial device with 32-byte packets via the UART module, you cannot switch to the SPI or I2C modules until a minimum of 267ms (i.e. 1/4s – a very long time for an MCU!) after the last byte is enqueued for sending.
  - Devices that require the MCU's constant attention and responsiveness (like an RS-232 modem which may deliver incoming characters at any time) need dedicated peripherals.

Part I: (cont'd)

- Software Issues:
  - A strictly sequential ordering of device access makes for very simple code:

```
main ()
{
  do
  {
    … other code …
    UARTReadCamera();
    SPIReadMagnetometer();
    I2CReadSPCurrents();
    I2CReadTemps();
    … other code …
  } while (TRUE);
}
```

because the re-initialization of the various modules is clearly laid out. But it does not accommodate *different sampling rates* very well.

Part I: (cont'd)

- Software Issues:
  - A more complete approach uses a scheduler or RTOS:

```
void TaskReadCamera ( void )
{ for (;;) { AcquireUSART(); InitUART(); ReadCamera();
  OSDelay(ONE_SEC);   ReleaseUSART(); } }

void TaskReadMagnetometer ( void )
{ for (;;) { AcquireUSART(); InitSPI();  ReadMagnetometer();
  OSDelay(HALF_SEC);  ReleaseUSART(); } }

void TaskReadSPCurrents ( void )
{ for (;;) { AcquireUSART(); InitI2C();  ReadSPCurrents();
  OSDelay(TENTH_SEC); ReleaseUSART(); } }

void TaskReadTemps ( void )
{ for (;;) { AcquireUSART(); InitI2C();  ReadTemps();
  OSDelay(FIVE_SEC);  ReleaseUSART(); } }
```

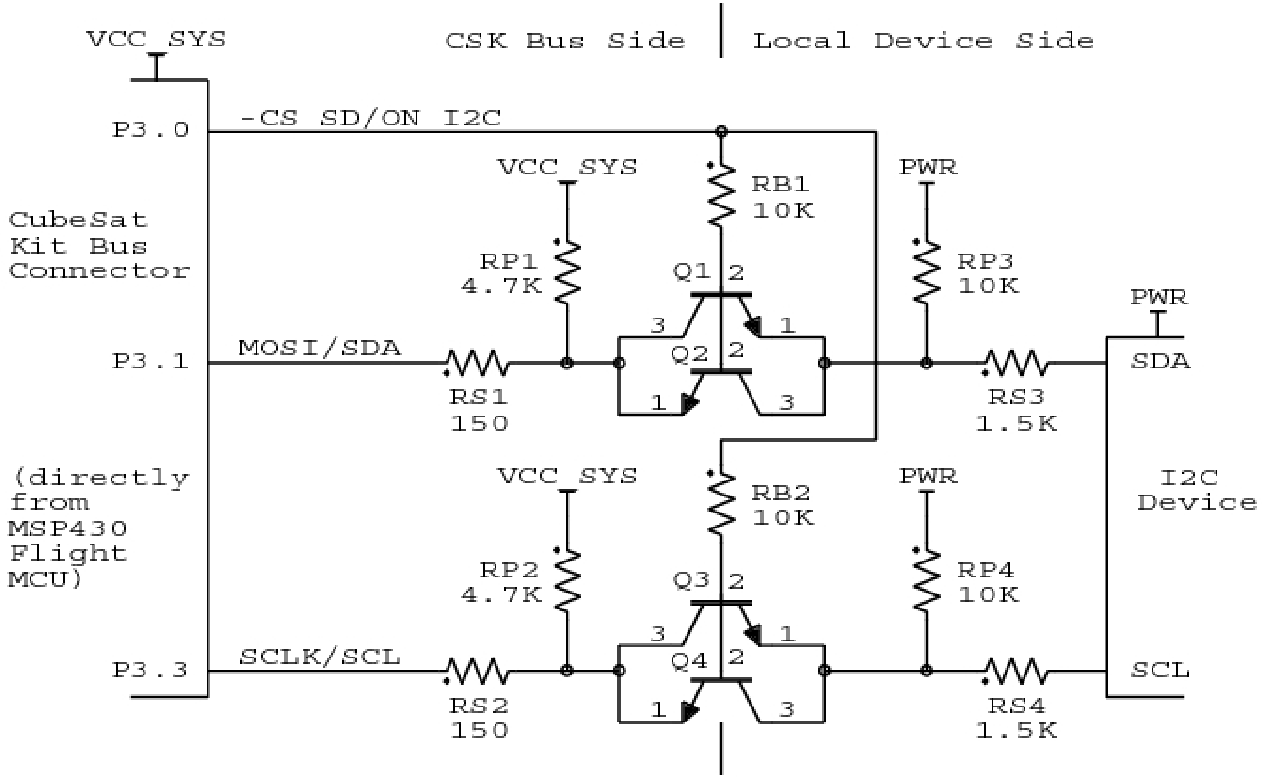
  - This allows processes to share the modules at arbitrary rates. Faster sampling is automatically interleaved with slower sampling. Task priorities further simplify scheduling.

Part I: (cont'd)

- By using multiple modules of a single on-chip peripheral, you:
  - Use fewer GPIO pins, thus freeing them for other purposes.
  - Use fewer on-chip peripherals, thus freeing them for other purposes.
  - Potentially run at lower power (less activity inside MCU).
  - Design with less mass, power and cost for better efficiency.
- CubeSat Kit example:
  - UART, SPI and I2C devices are all accessed through USART0.
  - 5 peripheral pins are required, plus one additional GPIO pin to prevent I2C corruption while SPI clock and data are on shared pins **`SCK`**/**`SCL`** and **`MOSI`**/**`SDA`**.
  - USART1 is dedicated to USB (pre-launch) or transceiver (pre- and post-launch). This allows the CubeSat Kit to listen on USART1 100% of the time for incoming commands and data. This also facilitates a "direct pipe" between mass storage (SD card on USART0:SPI) and transceiver (on USART1:UART).

Part I: (cont'd)

- CubeSat Kit example: I2C isolator allows concurrent use of SPI and I2C modules in USART0 peripheral at a cost of one additional GPIO pin:

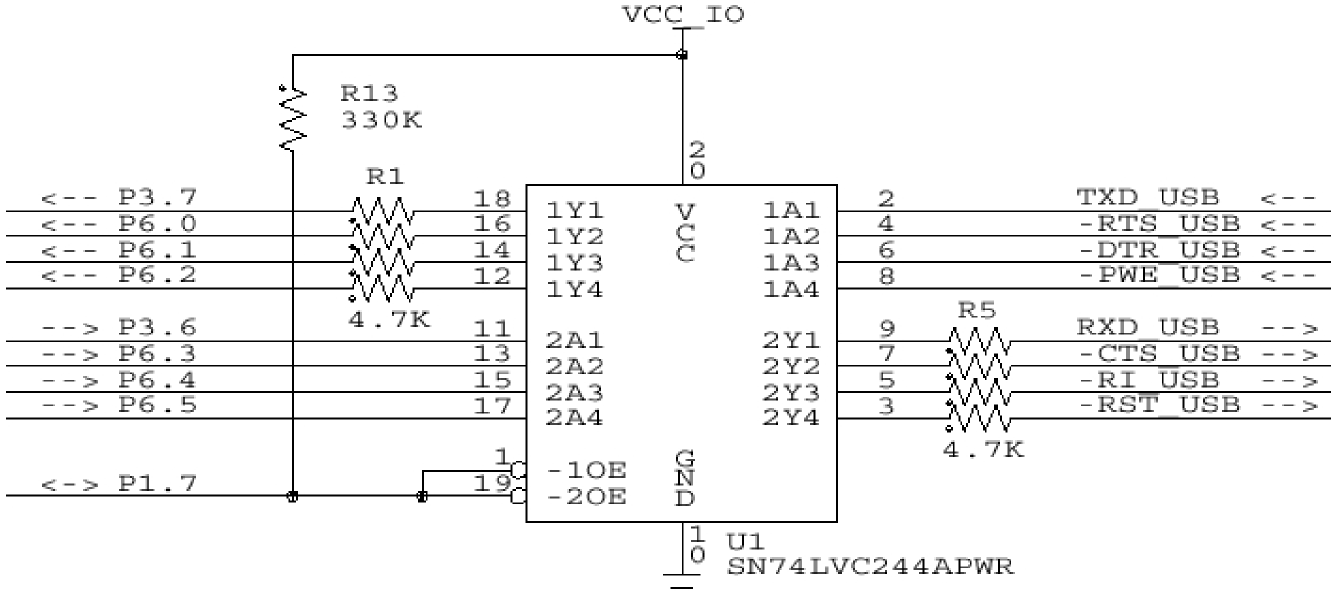


Part II: Simultaneous Input & Output

- Basic thinking:
  - *"Pin A will spend its life as a digital output, and I'll dedicate pin B as a digital input to sense condition X."*
- Advanced thinking:
  - *"By careful design and good programming, I can sense an important bit of information (X) on pin A as an input even though I normally use that pin as an output."*

Part II: (cont'd)

- CubeSat Kit example: A single GPIO pin connects USART1 to USB, *and detects if USB is present*.



  - **`P1.7`** configured as an output: USB is connected to the MCU when **`P1.7`** is **`LOW`**, and isolated and disconnected from the MCU when **`P1.7`** is **`HIGH`**.
  - **`P1.7`** configured as an input: If **`P1.7`** is **`HIGH`**, then USB is present since USB drives **`VCC_IO`**. If **`P1.7`** is **`LOW`**, USB must not be present (because **`VCC_IO`** is **`LOW`**). Either way, *USB remains isolated and disconnected from the MCU* while being probed.

Part II: (cont'd)

- CubeSat Kit example (cont'd):

```
void TaskDetectUSB ( void )
      {
        for (;;)
        {
          /* proceed if USB/MHX I/F is not in use */
          OS_WaitBinSem(BINSEM_USB_MHX_AVAIL_P, OSNO_TIMEOUT);
          OpenUSBMHXIF(USB);

          if ( !FM430status.USBpresent && (P1IN & BIT7) )
          {
            FM430status.USBpresent = 1;
            FM430Msg0("DetectUSB: USB connected.");
          }
          else if ( FM430status.USBpresent && !(P1IN & BIT7) )
          {
            FM430status.USBpresent = 0;
            FM430Msg0("DetectUSB: USB disconnected.");
          }

          CloseUSBMHXIF(USB); /* release USB/MHX I/F */
          OSSignalBinSem(BINSEM_USB_MHX_AVAIL_P);

          OS_Delay(25); /* come back in 25 ticks */
        }
      }
```

Part II: (cont'd)

- CubeSat Kit example (cont'd): This single-pin driver and sensor fulfills several functions:
  - As an output, it independently controls the USB's access to a shared on-chip resource (USART1).
  - As an input, it allows us to sense whether USB is present or not, *even when USART1 is connected to something else.* The choice of the LVC244A buffer with I_{off} mode is critical to this function, and illustrates how *hardware choices are important*. Detection of USB present requires just a simple task that runs occasionally, in concert with all other processes that use USART1.
  - And it does it all with just a single GPIO pin!

Part II: (cont'd)

- Other examples: Multifunction analog/digital pins normally configured as digital output pins can also function as analog input pins if:
  - the driving source (output) of the analog signal can be (over-)driven by the digital output without damage, and
  - if the normal recipient of the digital output can be disabled while the analog signal is being sampled.

```
void ReadAnalogIns ( void )
{
  DisableP6IF();              /* disable digital devices on P6      */
  ConfigureP6AnalogIns();     /* P6 pins become analog inputs       */
  ReadP6AnalogIns();          /* get 'em!                           */
  ConfigureP6DigitalOuts();   /* P6 pins go back to digital outputs */
  EnableP6IF();               /* enable digital devices in P6       */
}
```

Part III: GPIO Pin Personalities

- Basic thinking:
  - *"I have two serial devices I need to connect to my MCU, one DTE and one DCE. Their handshaking signals are exactly opposite, so I need one set of GPIO for the DTE and one set for the DCE."*
- Advanced thinking:
  - *"I'll use the same GPIO pins for the DTE and DCE handshaking, and will (re-)configure the pins in software based on whether the MCU is communicating with the DTE or DCE device."*

Part III: (cont'd)

- CubeSat Kit Example: Hardware handshaking with USB (DTE) and transceiver (DCE) using the same pins:
  - *Here, the signals flow in the same directions, but represent complementary functions based on the RS-232 device's type (DTE or DCE). The application's UART code must take this into account and sense or drive the handshaking pins appropriately.*

Pin Name	from/to DTE	from/to DCE
P6.0	← **-RTS**	← **-CTS**
P6.3	**-CTS** →	**-RTS** →

Part IV: Efficient GPIO Pin Decoding

- Basic thinking:
  - *"I have been using individual GPIO pins to control major subsystems (e.g. power to power-hungry devices)."*
- Advanced thinking:
  - *"Why not use the combination of these individual pins to add functionality without consuming more GPIO pins?"*

Part IV: (cont'd)

- CubeSat Kit Example: Careful use of decoder pins enabled additional functionality without using any new pins:
  - Initially, **`–ON_SD`** enabled power to the SD card, and **`–CS_SD`** functioned as the SD card's SPI chip select pin.
  - The I2C isolation buffer needed a driver pin. Since I2C and SPI are on the same peripheral, could the **`–ON_SD`** and **`–CS_SD`** signals be used to decode the 3+ individual states to support SPI and I2C operation?
  - The solution presented itself as the **`–CS_SD/ON_I2C`** signal, one that selects the SD card when **`LOW`** and enables I2C communications when **`HIGH`**. The complementary nature of the two signals works to our advantage. Piggybacking the **`ON_I2C`** signal on **`–ON_SD`** instead would have been an unfortunate choice …
  - The net result is 4 states which fully address the needs of SPI or I2C decoding, and minimizing power consumption, while using only two GPIO pins that had already been allocated to other purposes.

Part IV: (cont'd)

-ON_SD → / -CS_SD/ON_I2C ↓	LOW (0V)	HIGH (+3.3V)
LOW (0V)	SD card ON & selected, other SPI devices deselected, I^2C disabled – use SPI to access SD card	SD card OFF & selected, I^2C disabled – use SPI to access other SPI devices via other SPI devices' –CS lines
HIGH (+3.3V)	SD card ON & deselected, other SPI devices deselected, I^2C enabled – use I2C to access any I2C device	SD card OFF & deselected, other SPI devices deselected, I^2C enabled – use I2C to access any I2C device

Truth Table for SPI / I^2C selection when multiple SPI Devices are Present

Part V: Summary

- Careful planning in the design stage and good coding can result in reduced GPIO pin count requirements, leading to other efficiencies.
- Any time a connection to an external device is not needed 100% of the available time, consider sharing its interface (external GPIO pins and internal peripherals) with other devices. You may save resources in doing so.
- Reducing GPIO pin requirements can allow you to do more with less.
- Signal polarities often affect how efficiently you can use single GPIO pins for multiple purposes.

Part V: (cont'd)

- The CubeSat Kit implements:
  - Two UARTs
  - One I2C chain to unlimited I2C devices
  - One SPI chain to on-board SD and off-board SPI devices
  - Detection and control of the USB interface
  - Control of an I2C isolating interface
  - Control of the transceiver interface
  - Control of power to the transceiver
  - Control of power to the SD card
  - Control of power to a RTC chip
  - Chip select of the SD card

with just 12 of the MSP430's GPIO pins and two peripherals, leaving 36 of 48 available GPIO pins and the rest of the peripherals free to the user.

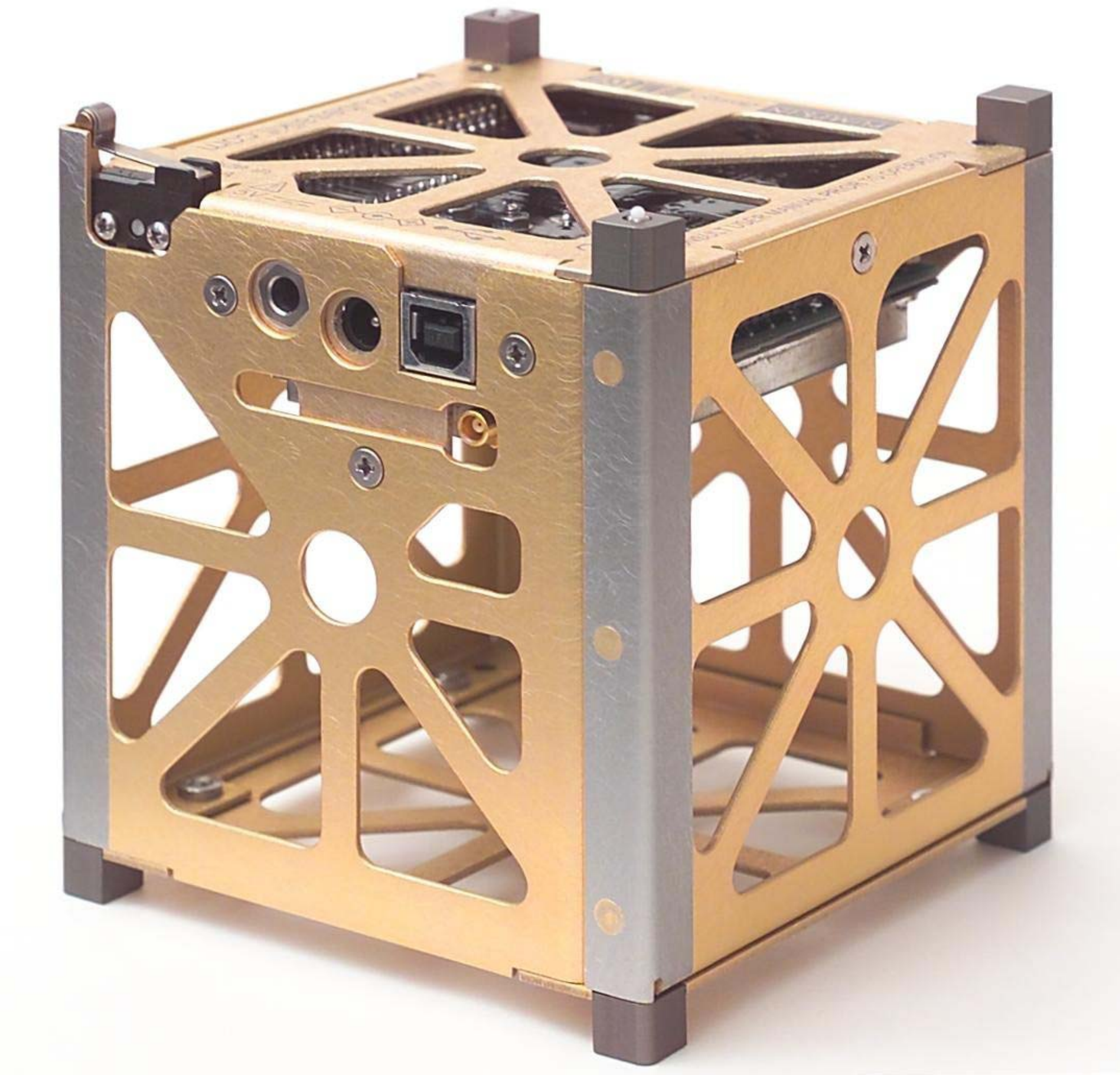
Notice

This presentation is available online in Microsoft® PowerPoint® and Adobe® Acrobat® formats at:

www.pumpkininc.com/content/doc/press/Pumpkin_CubeSatWorkshop2006.ppt

and:

www.pumpkininc.com/content/doc/press/Pumpkin_CubeSatWorkshop2006.pdf



Q&A Session

Thank you for attending this Pumpkin presentation at the Cal Poly CubeSat Workshop 2006!

Notes & References

1. MSP430x16x block diagram from TI's MSP430F1612 datasheets.
2. *CubeSat Kit User Manual*, Pumpkin, Inc. 2005.

Appendix

- Speaker information
  - Dr. Kalman is Pumpkin's president and chief software architect. He entered the embedded programming world in the mid-1980's. After co-founding the successful Silicon Valley high-tech pro-audio startup Euphonix, Inc. he founded Pumpkin with an emphasis on software quality, performance and applicability to a wide range of microcontroller-based applications. He holds two United States patents, is a consulting professor at Stanford University and is invariably involved in a variety of hardware and software projects.

- Acknowledgements
  - Stanford Professor Bob Twiggs' continued support for the CubeSat Kit, and his input on enhancements and suggestions for future CubeSat Kit products, are greatly appreciated.
  - Pumpkin's Salvo and CubeSat Kit customers, whose real-world experience with our products helps us improve and innovate.

- Salvo, CubeSat Kit and CubeSat information
  - More information on the Pumpkin's Salvo RTOS and Pumpkin's CubeSat Kit can be found at http://www.pumpkininc.com/ and http://www.cubesatkit.com/, respectively.
  - More information on the open CubeSat standard and the CubeSat community can be found at http://www.cubesat.info/.

- Copyright notice

First presented at the 3rd Annual CubeSat Workshop at Cal Poly San Luis Obispo, on April 28, 2006.