

PUMPKIN™

INCORPORATED

[www.pumpkininc.com](http://www.pumpkininc.com)

# Hardware and Software Design of an RTOS-based MSP430-based Picosatellite

Andrew E. Kalman, Ph.D.

Slide 1



**MSP430**  
Advanced Technical Conference

4th  
IV  
Annual

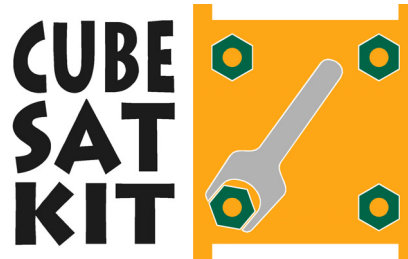
# Introduction

- Andrew E. Kalman
  - President and CTO, Pumpkin, Inc.

- Author of



- Creator of the



- 20+ years of embedded systems design and programming experience
- Contact: [aek@pumpkininc.com](mailto:aek@pumpkininc.com)

# Outline

- Overview: Seminar Goals
- Part I: An Overview of Picosatellites
- Part II: The CubeSat Picosatellite Standard
- Part III: Product Requirements for a Picosatellite Kit
- Part IV: The CubeSat Kit's Structure & Layout
- Part V: Why Pumpkin chose TI's MSP430 for the CubeSat Kit
- Part VI: Hardware Architecture of the CubeSat Kit's MSP430-centric FM430 Flight Module
- Part VII: Rapid MSP430 Software Development using the Salvo RTOS
- Part VIII: Summary

# Overview

- This seminar will present an in-depth look at Pumpkin's experience with the MSP430 as it was applied to our CubeSat Kit, an off-the-shelf build-it-yourself picosatellite kit. Of particular interest to ATC attendees:
  - Why we chose the MSP430 for an aerospace application.
  - How the MSP430 fits into an expandable bus-driven architecture.
  - How to integrate the MSP430 into a +3.3V/+5V power system.
  - How to design *zero-power* peripheral interfaces to the MSP430.
  - Advantages of the available MSP430 development toolsets.
  - The power, interface and peripheral chips we used with the MSP430.
  - Protecting the MSP430 from overcurrent / overvoltage.
  - How an RTOS simplifies MSP430 programming.
  - Software solutions to common problems in MSP430 applications.

# Part I: Picosatellites

- Development, debugging & functional testing: typical lab environment.
- Pre-delivery and pre-launch:
  - Temperature, vacuum and shake tests.
  - Integration into launch vehicle.
  - Picosatellite may remain in storage for months on end waiting for launch.
- Launch & deployment: high g-forces (10g or more).
- Operation in space: vacuum, wide temperature range (-20° to +60° C), solar radiation, and remoteness.
- End of mission: deorbit and burn up in earth's atmosphere.

# Picosatellites (cont'd)

- Picosatellite components:
  - Structure.
  - Command & Data Handling (C&DH), with high-frequency transceiver and antenna(s).
  - Communications (COM).
  - Electrical Power System (EPS).
  - Attitude Determination & Control System (AD&CS).
  - Payload.
  - Software, Software, Software.
  
- Picosatellites are often launched in groups from dedicated launchers as *secondary* payloads on a rocket.

# Part II: CubeSats

- The CubeSat is a 10x10x10cm, 1kg public picosatellite design specification proposed by Stanford and Cal Poly San Luis Obispo universities.
- Low-earth orbit (LEO) CubeSat missions have typical lifespans of 3-9 months.
- Cost to complete a CubeSat mission (inception to launch to operation to end-of-life) ranges from <\$100,000 to \$1,500,000.
- Working from a standard promotes rapid development and idea sharing
- Picosatellites are already a hot topic in aerospace. Worldwide interest is focused on CubeSats in particular, partly because they are becoming a *de facto standard*.



# Part III: Kit Requirements

- This is not the \$5,000,000-and-up satellite market!
- High percentage of potential end-users represented by educational organizations (universities and high schools).
- Timelines allow less than 24 months from mission inception to launch.
- Must be able to accommodate a wide variety of user payloads and high levels of mission complexity.
- CubeSats are by nature low-power due to the available solar radiation (<1W for typical 10x10x10cm “1U” cube).
- Scalable, expandable and non-obsolescent architecture.
- Multiple power supplies necessary for flexibility.



# Kit Requirements (cont'd)

- A COTS CubeSat kit should:
  - Implement as many of the CubeSat standard's specifications as possible, thereby shortening the design & development timeline.
  - Integrate as much functionality into as small a space as possible, so as to leave plenty of room for the CubeSat's payload and other mission-specific hardware.
  - Be flexible, with a design that can accommodate a wide range of customer requirements and configurations.
  - Be expandable and modular, to accept other COTS hardware and software.
  - Provide a software development platform that's easy to use and debug.
  - Have a clearly-defined future upgrade path.
  - Include high-value-added features (hardware and software) to differentiate itself from the competition and "home-grown" CubeSats.
  - Be affordable.

# Part IV: Structure & Layout

- Mechanical structure is constrained by exterior dimensions, defined access area and other requirements in the CubeSat specification. We chose a sheet-metal design for strength, light weight, maximal internal volume and design flexibility.
- A module bussing scheme patterned after the PC/104 standard was chosen for interconnecting modules and connector reliability.
- *All* of the MSP430's I/O pins are on the electrical bus, as well as GND/+3.3V/+5V, +5V\_USB, power switching, power control, modem control signals, and a few other signals.
- ½U, 1U, 1½U, 2U and 3U sizes differ only by a single part – the chassis walls assembly.

# Structure & Layout (cont'd)

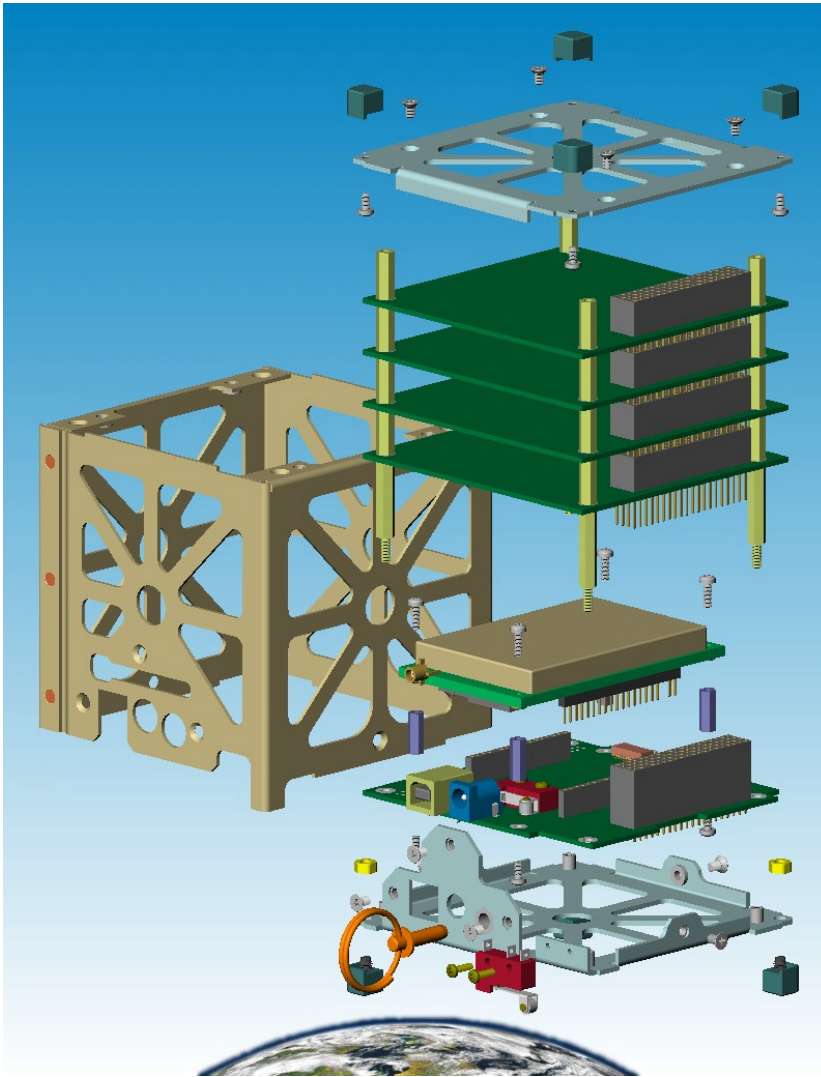


Figure 1: Exploded View of a 1U MSP430-based CubeSat Kit.

Clockwise from center left: Chassis Walls Assembly, Cover Plate, User Module Stack, High-Frequency Transceiver, FM430 Flight Module, Base Plate.

MSP430 resides on FM430 Flight Module.

# Part V: Choice of MSP430

- The MSP430's CubeSat Kit-friendly features include:
  - -40° to +85° C operating temperature range.
  - Relatively large code space (up to 60K) enables a high level of systems integration, thereby reducing processor count.
  - 16-bit architecture and highly orthogonal and C-friendly architecture makes for efficient high-level programming.
  - Good mix of on-board peripherals. Plenty of I/O.
  - JTAG & Flash accelerate the design & debug cycle.
  - Ultra-low-power design, dual clock sources and fast startup from sleep help minimize power consumption.
  - Well-organized prioritized interrupt vectors and interrupt control.
  - Wide range of tools (assemblers, compilers, RTOS, IDEs, JTAG debuggers, programmers, development kits, etc.) offered.
  - Ready parts availability.

# Choice of MSP430 (cont'd)

- The CubeSat Kit's MSP430 Flight MCU is responsible for:
  - Doing supervisory tasks, e.g. monitoring on-board temperatures, voltages and currents, etc.
  - Managing bidirectional communications with the mission's ground station(s) (COM).
  - Acting autonomously and/or through commands from the ground station(s) or other satellites (C&DH).
  - Handling data storage and retrieval, either on-chip (in RAM or Flash) or off-chip (mass storage).
  - Configuring its I/O to interface to multiple peripherals that concurrently share the same MSP430 port pins and peripherals.
  - Depending on system complexity, EPS and AD&C functions may be implemented either in the MSP430 or in additional mission-specific CPUs and  $\mu$ C's that communicate with the MSP430.

# Part VI: FM430 Hardware

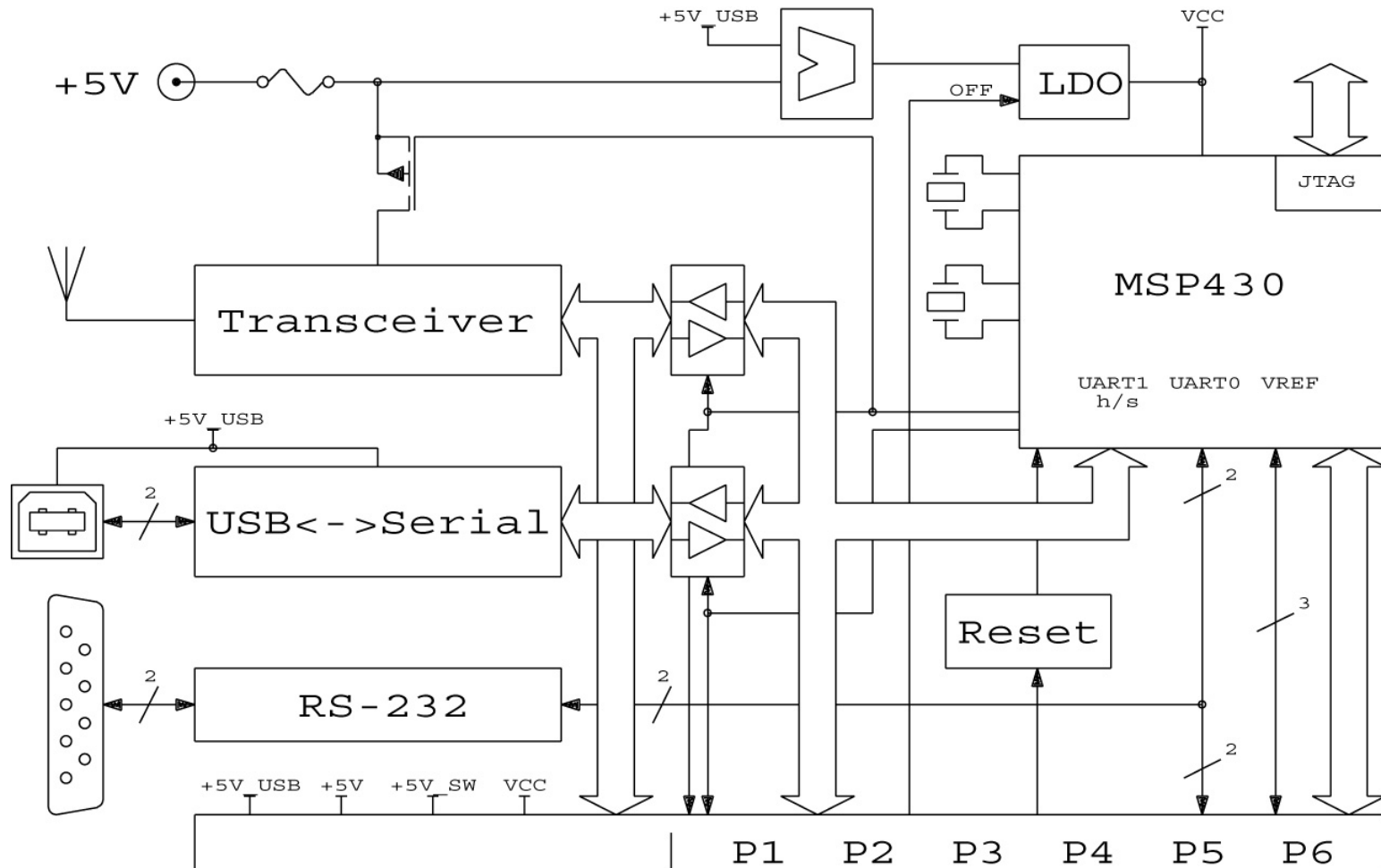


Figure 2: FM430 Flight Module Block Diagram



# FM430 Hardware (cont'd)

- Power system requirements:
  - Many high-power ( $\geq 1\text{W}$ ) wireless transceivers operate with 5V supplies. The CubeSat Kit also accepts user payloads in the form of 5V-only PC/104 modules. Therefore this must be a mixed-supply system, supporting both 5V and the MSP430's 3.3V.
  - The CubeSat Kit provides USB connectivity as a means of communicating with ground personnel when fully assembled and / or immediately prior to launch. USB can provide 5V at 500mA max, and can also be used to charge on-board batteries.
  - As only 1W average power from the sun is available to charge solar cells on a 1U CubeSat, the power consumption of the MSP430 and all support circuitry must be minimized so that the FM430 can run 24x7 in its supervisory roles while batteries are being charged. Transmitting alone requires 4-5W of continuous power and therefore can only be done for a few minutes of every orbit.





# FM430 Hardware (cont'd)

- Power system solution:
  - 5V available system-wide on CubeSat Kit Bus<sup>™</sup>.
  - Due to concerns over switching noise, system complexity and FM430 protection, TI's TPS769xx family of LDO linear ultralow-power regulators was chosen to supply 3.3V to the CubeSat Kit's Flight Module from the +5V system bus:
    - ♦ 17 $\mu$ A max quiescent current @ 100mA output.
    - ♦ Overcurrent protection.
    - ♦ 1 $\mu$ A sleep current via  $-EN$  (on CubeSat Kit Bus).
    - ♦  $-40^{\circ}$  to  $+125^{\circ}$  C operating range.
  - The CubeSat Kit Bus also provides a total of 4 terminals (C, NO and NC) from 10A Launch and Remove-Before-Flight Switches to facilitate a wiring-free main power switching scheme.

# FM430 Hardware (cont'd)

- ...
- These regulators require careful attention to choice of output capacitor for stabilization. Yet their stability is quite tolerant of changes in output capacitor ESR that will occur in space due to temperature fluctuations. Additionally, unused 3.3V power is available system-wide via the CubeSat Kit Bus. *This LDO is the largest single power consumer on the Flight Module* when the FM430 is sleeping. If the Flight Module is the only consumer of 3.3V power, overall system inefficiencies due to the 5V↔3.3V downconversion are essentially irrelevant.
  - Because the Flight Module runs at 3.3V, and system power is supplied at 5V, we have the opportunity to feed the 3.3V LDO from a multitude of 5V sources, via a simple Schottky diode drop. Therefore the Flight Module is powered from system 5V or USB 5V, whichever is present (Figure 3).

# FM430 Hardware (cont'd)

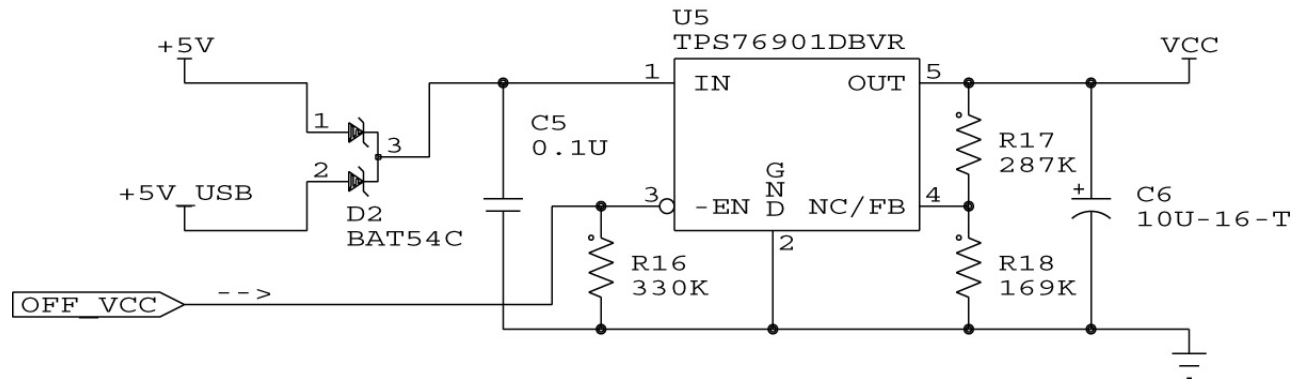


Figure 3: Multiple 5V Sources Feeding Ultralow-power 3.3V LDO Linear Regulator

# FM430 Hardware (cont'd)

- Mixed 3.3V/5V design requirements:
  - Flight Module components and their operating voltages:
    - ◆ 3.3V: MSP430, reset supervisor & serial-to-USB converter (serial side)
    - ◆ 5V: 2.4GHz spread-spectrum transceiver & serial-to-USB converter (USB side)
  - The MSP430 drives either the transceiver, the serial-to-USB converter or the CubeSat Kit Bus through USART1. Because of the high quiescent power consumption of the transceiver (1W) and serial-to-USB converter (2mW) compared to the MSP430, these devices should only be powered when connectivity is required. Therefore a *zero-power, isolating* and (for the transceiver) *level-shifting* interface must be employed.
  - Power-up / power-down of 3.3V and 5V sides may occur separately at any time. MSP430 must be *protected from overcurrents* on its I/O pins from output drivers. Transceiver and serial-to-USB converter must not be powered through input pins via MSP430 outputs.

# FM430 Hardware (cont'd)

- Mixed 3.3V/5V design challenges:
  - Any scheme that doesn't keep signals at the CMOS rails will consume large (relative to MSP430) quiescent power.
  - Resistive level shifters (e.g. for 3.3V↔5V) either waste power or don't provide sufficient isolation.
  - There are many integrated bidirectional level-shifting solutions on the market today. But few of them offer level translation and power-off isolation with *zero current draw*.
  - Popular FET-based (bus) switches have simple, symmetric flow-through architectures that are well-suited to voltage translation and bus isolation. However, they require charge pumps to increase the voltage at each switch's gate so that  $V_{GS}$  is adequate to turn on the switch. This adds 10 $\mu$ A-2mA of current draw, typically on a *per-level-translated-pin* basis.

# FM430 Hardware (cont'd)

- Mixed 3.3V/5V design solutions:
  - Assuming that individual control outputs idle high, they can be level-shifted using discrete MOSFETs and resistors at zero power (Figure 4).
  - To translate 3.3V up to 5V, we use half of an AHCT244 powered at 5V (on demand). The  $-\text{OE}$  control signal must also be level-shifted to 5V for minimum power consumption (Figure 5).
  - To translate and isolate 5V down to 3.3V, we use half of an LVC244A powered at 3.3V (permanently). The  $-\text{OE}$  control signal must also pulled to the chip's VCC for minimum power consumption and power-on / power-off isolation (Figure 6).

# FM430 Hardware (cont'd)

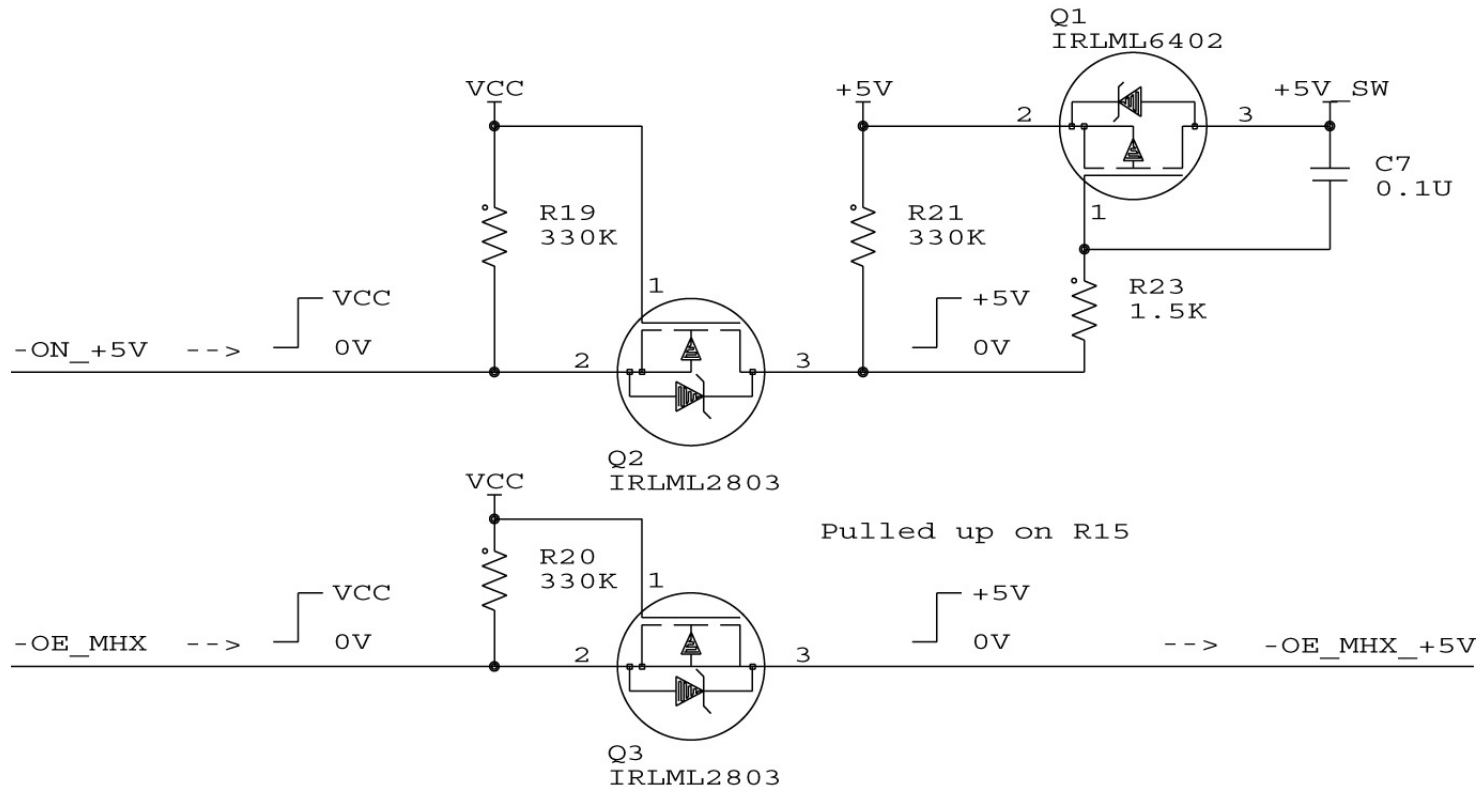


Figure 4: Level-shifting for Individual Unidirectional Control Signals



# FM430 Hardware (cont'd)

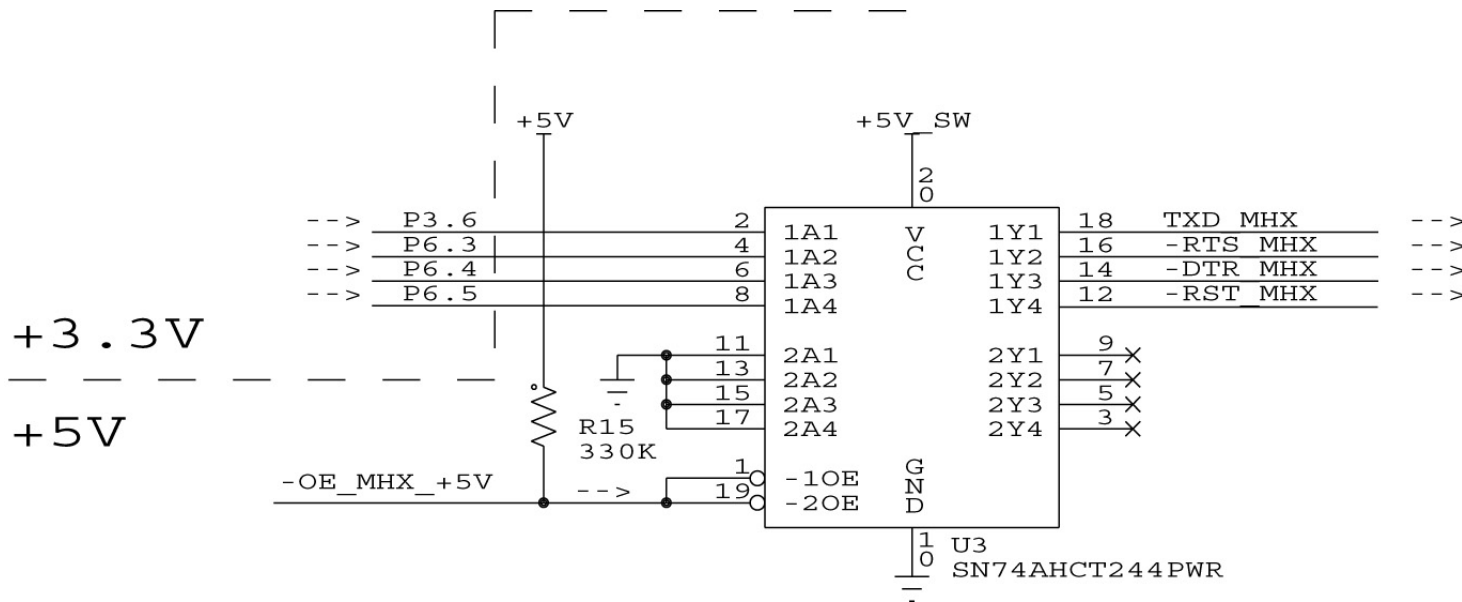


Figure 5: Level-shifting up to +5V using AHCT Logic

# FM430 Hardware (cont'd)

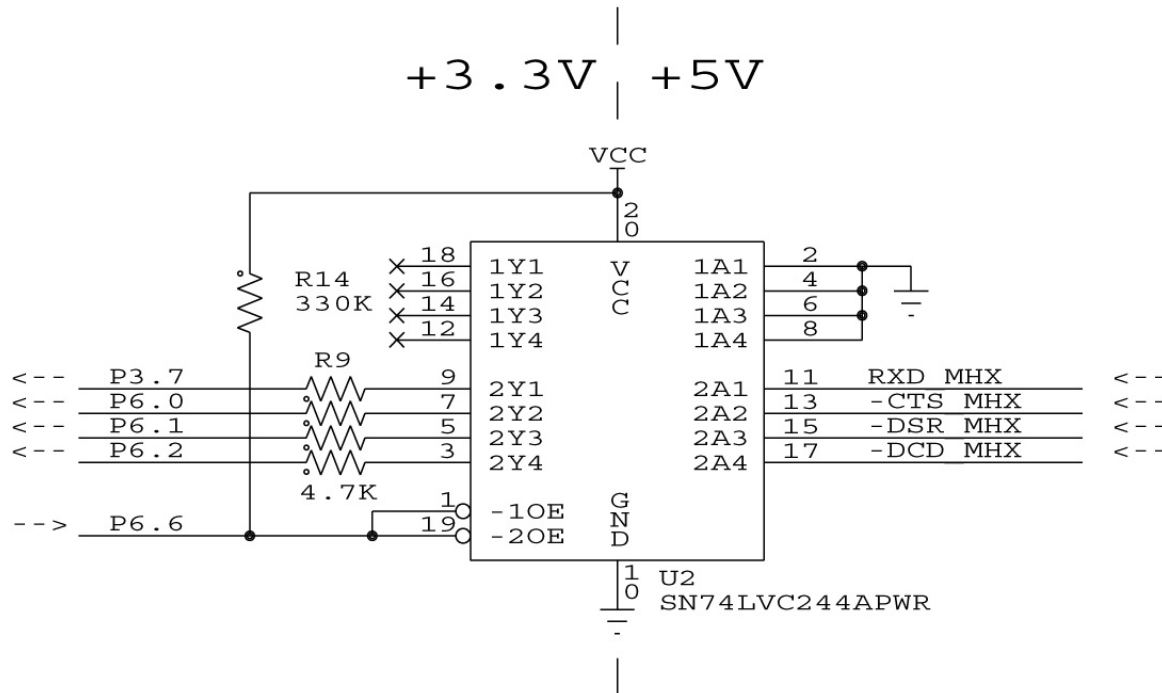


Figure 6: Level-shifting down to +3.3V and Isolating using LVC Logic

# FM430 Hardware (cont'd)

- Isolated bus-powered serial-to-USB interface requirements:
  - The USB interface must power the FM430 Flight Module over USB when a USB cable is connected.
  - The Flight Module must be powered from either the system 5V or the USB interface automatically.
  - When unpowered, the serial-to-USB converter must be *completely isolated* from the MSP430, and consume *no power*.
  - Programmable support for USB VID, PID, etc.
  - Customizable USB drivers for multiple platforms (Win, Mac OS, Linux, etc.).



# FM430 Hardware (cont'd)

- Isolated bus-powered serial-to-USB interface solution:
  - FTDI's FT232BM with external configuration EEPROM and Win/Mac OS/Linux drivers.
  - 5V from USB bus drives 3.3V LDO for Flight Module.
  - Bus-powered design provides 3.3V to power LVC244A isolation circuitry via FT232BM's 3V3\_OUT pin.
  - Programmable support for USB VID, PID, etc.
  - When USB cable is removed, LVC244A automatically isolates MSP430 from FT232BM through its partial power-down mode (Figure 7).

# FM430 Hardware (cont'd)

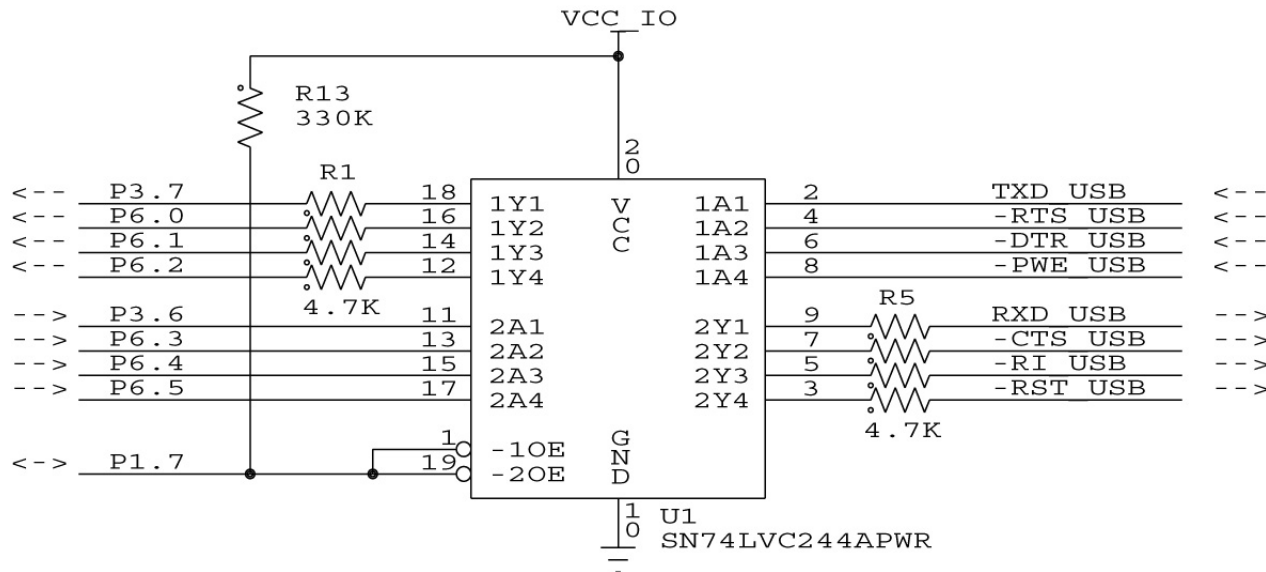


Figure 7: 3.3V Isolation Driven by Presence of External Power VCC\_IO

# FM430 Hardware (cont'd)

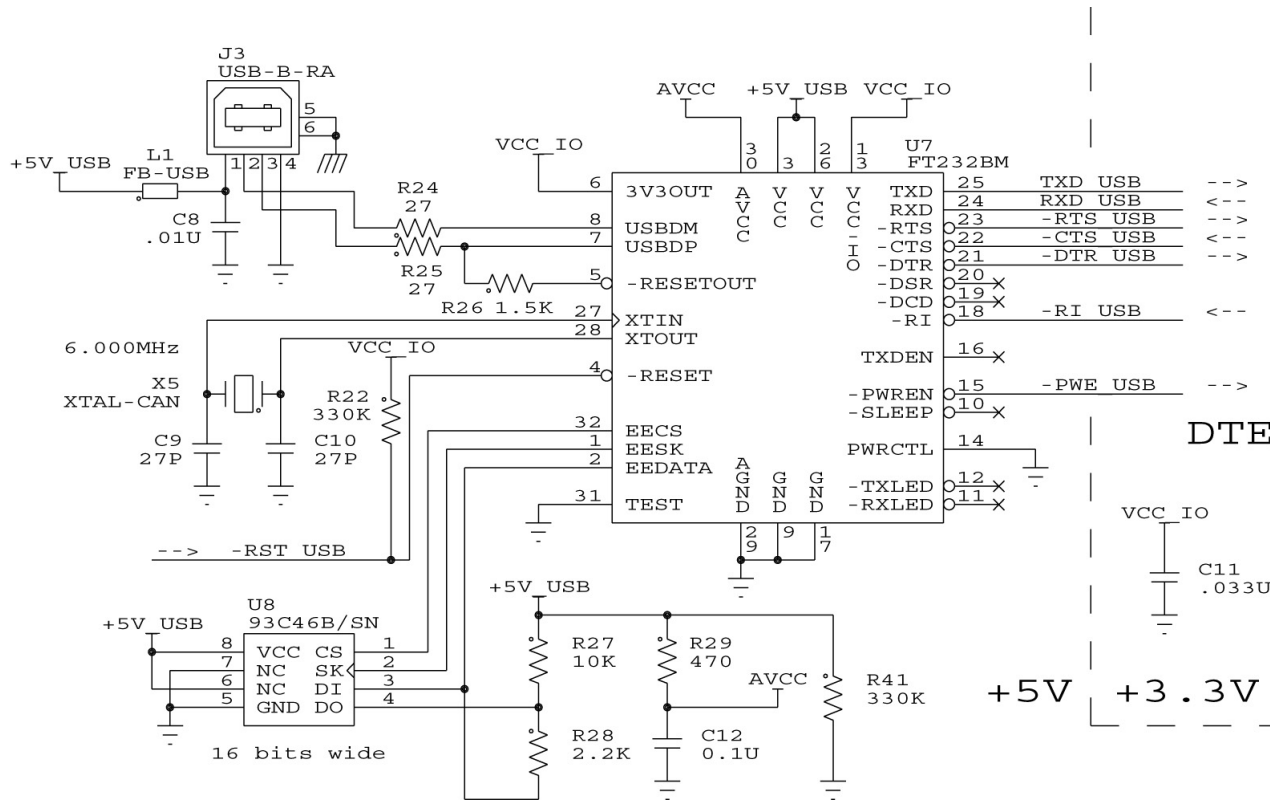


Figure 8: Bus-powered Serial-to-USB Converter for 3.3V Interface

# FM430 Hardware (cont'd)

- Protecting the MSP430's I/O:
  - Absolute maximum over / undervoltage:  $\pm 0.3V$
  - Overvoltage specification can be violated if MSP430 is OFF and driving circuitry is ON. In-line resistor will protect MSP430 by dropping voltage and limiting current.
  - Absolute maximum diode current at any pin:  $\pm 2mA$
  - Overcurrent specification can be violated if MSP430 I/O is configured as an output and connected to another output. This could happen due to a programming error if I/O pin is used as both input and output, depending on overall system configuration. In-line resistor will protect MSP430 by limiting current.
  - Figure 6 and Figure 7 illustrate this technique.  $4.7k\Omega$  series limiting resistors chosen for adequate margin under all possible operating conditions (including user error).





# FM430 Hardware (cont'd)

- Reset supervisor:
  - TI's TPS3838 series nanopower supervisory circuit.
  - Supply current < 300nA.
  - -40° to +85° C operating range.
  - Open-drain output is required for reliable JTAG debugging.
  - -MR input on CubeSat Kit Bus to allow external MSP430 reset.

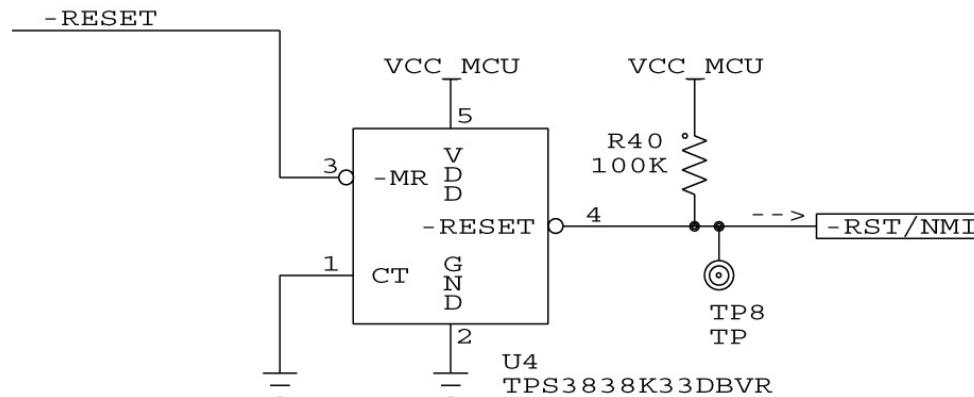


Figure 9: Nanopower Reset Supervisor with Open-drain Output

# FM430 Hardware (cont'd)

- Multitasking the MSP430's I/O pins:
  - By isolating the transceiver and serial-to-USB converter from the MSP430's bus, we can configure the directions of the eight associated pins for multiple purposes: RS-232 Tx/Rx + DCE handshaking (transceiver enabled), RS-232 Tx/Rx + DTE handshaking (serial-to-USB converter enabled), and user-defined (neither enabled).
  - With individual I/O pins dedicated to external peripheral enables, multiple external peripherals can be driven from a single internal peripheral. E.g. external camera, SD/MMC card and I2C bus can all be driven from USART0 (but only one at a time).

# FM430 Hardware (cont'd)

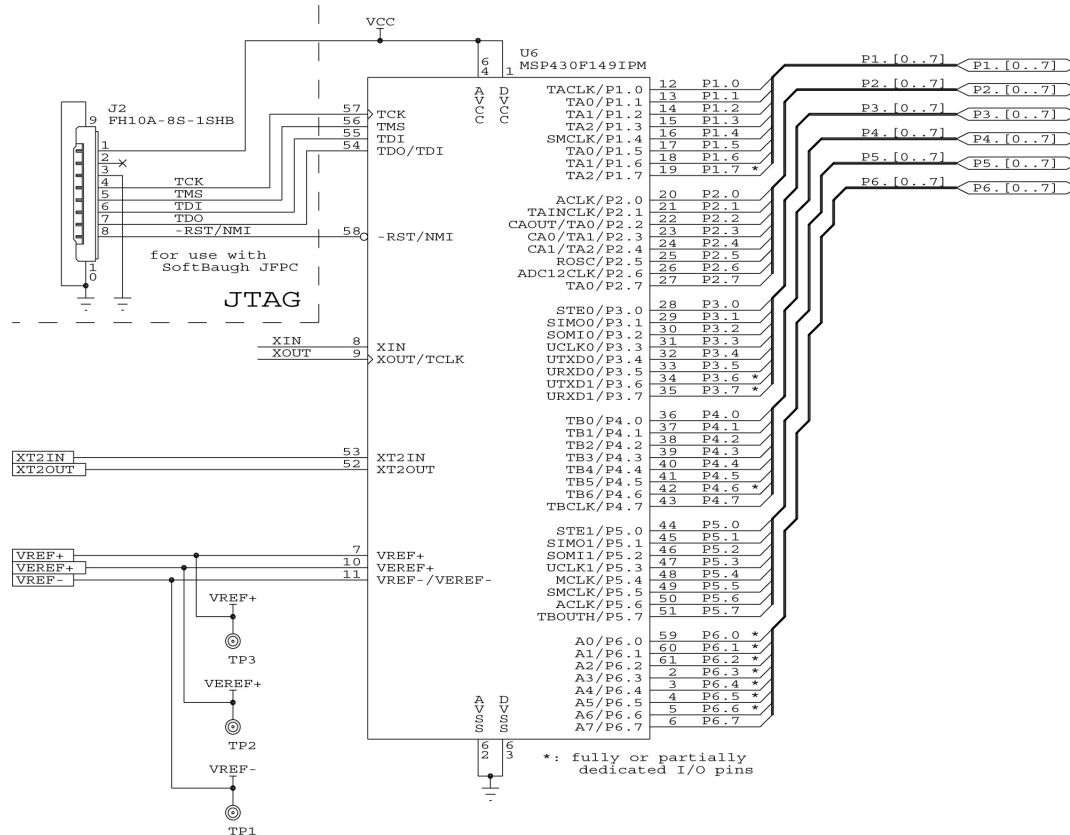


Figure 10: Dedicated I/O Pin Usage. Pin Modes Dictated by Configuration

# FM430 Hardware (cont'd)

- FM430 design results:
  - Accepts 64-pin MSP430's, including 'F149, 'F169, 'F1611, 'F1612.
  - System-wide power consumption of <250μW asleep, <10mW operational, functional over -40° to +85° C operating range.
  - Implemented on a single 90x96mm six-layer PCB.
  - MSP430's entire I/O pinout (and more) is available on two 40-pin PC/104-type stacking connectors.
  - Mini-JTAG connector present for programming and debugging.
  - Having entire I/O space on CubeSat Kit Bus connector facilitates the design of additional functionality (e.g. SD/MMC card socket for mass memory) as future on-board enhancements to the original (Rev A) design.



# FM430 Hardware (cont'd)

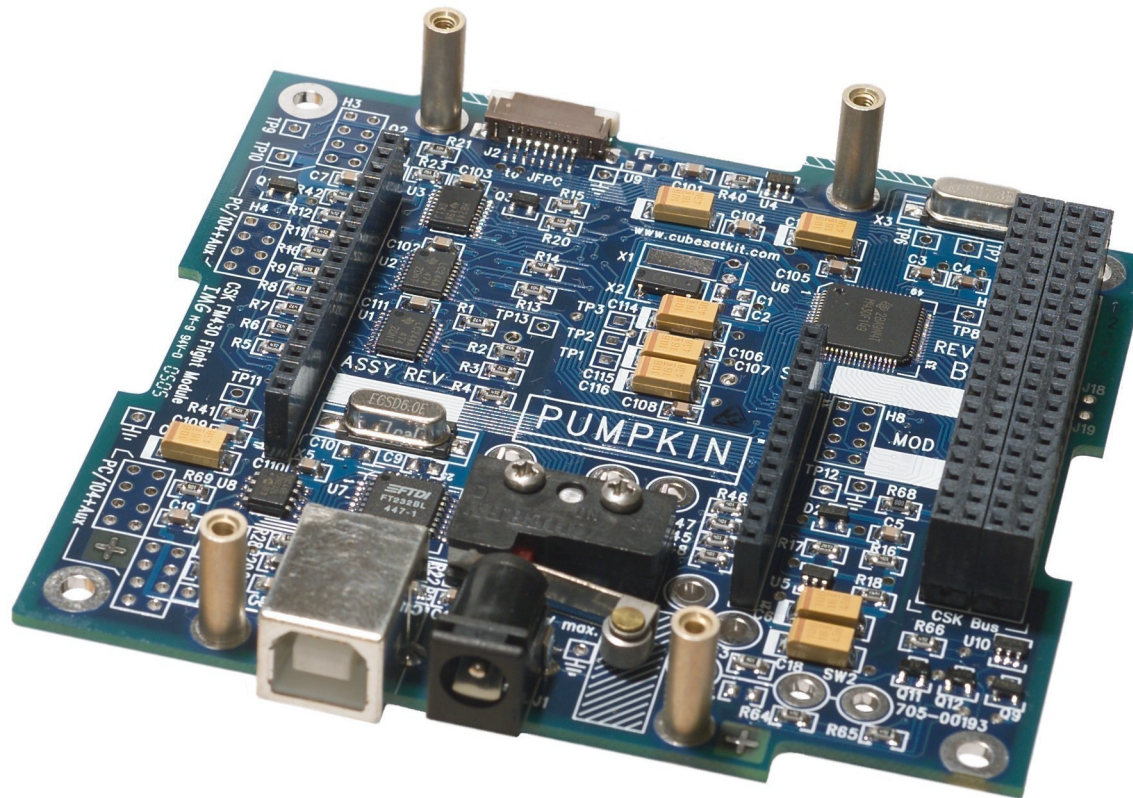


Figure 11: FM430 Flight Module Rev B

# Part VII: FM430 Software

- Integrated C development environment:
  - Isolates end-users (often novices) from having to learn assembly language ... ignorance is bliss.
  - Applications written in C have acceptable speed, power and size results for vast majority of applications.
  - Plenty of C toolsets & IDEs available for the MSP430:
    - ◆ Archelon / Quadravox AQ430
    - ◆ GCC mspgcc
    - ◆ HI-TECH C for MSP430
    - ◆ IAR Embedded Workbench for MSP430
    - ◆ ImageCraft ICC430
    - ◆ Rowley CrossWorks for MSP430
    - ◆ TI Code Composer





# FM430 Software (cont'd)



[www.pumpkininc.com](http://www.pumpkininc.com)

- Pumpkin chose an RTOS-based approach to the CubeSat Kit's system software because:
  - We felt it necessary to provide a formal paradigm to handle the interactions between the various sub-applications that would run on the CubeSat Kit, thereby managing complexity.
  - We wanted to extract the maximum performance out of the MSP430 – concurrency is a must.
  - We needed modularity so that users could pick and choose which software components they needed for their particular application, and to be able to selectively develop and debug individual application components.
  - Our experience has shown both novice and advanced programmers quickly appreciate the benefits of standardized multitasking and RTOS services.
  - Programmers can communicate the actions of their tasks well to the hardware designers because of the clearly defined linear program flow.

Slide 36





# FM430 Software (cont'd)



[www.pumpkininc.com](http://www.pumpkininc.com)

- Pumpkin's Salvo RTOS was chosen because:
  - Running your own RTOS under real-world, real-product conditions in end-user hands can only benefit all end-users of the RTOS.
  - Easier to customers to learn than a fully preemptive RTOS, no debugging headaches. All coding done in C.
  - Enables higher ISR and overall system performance (cycles / CPU watt) because of a variety of unique design features in Salvo.
  - Full feature set includes priority-based multitasking, delay services, multiple event types (semaphores, binary semaphores, messages, message queues and event flags), waiting with timeouts, support for low-power modes, complete configurability.
  - But perhaps most importantly, *Salvo's RAM footprint is tiny, much smaller than any other RTOS*. This allows us to run memory-hungry applications (e.g. image capture, IP-based web server, etc.) without requiring external memory or a larger Flight MCU.

Slide 37



# FM430 Software (cont'd)

*We wanted a clean, lean and reliable MSP430 software application that could simultaneously manage the various internal and external MSP430 peripherals throughout the picosatellite in a clearly understandable way.*

*Our customers' experience has reinforced the correctness of our decision to go with an approach based on the Salvo RTOS as the foundation for the CubeSat Kit's software.*

# FM430 Software (cont'd)

- CubeSat Kit software requirements:
  - USART0: Manage shared access to SD/MMC card, I2C & user.
  - USART1: Manage isolation & interface to USB, transceiver & user.
  - USB: Detect when USB I/F is present. Acquire & release USB interface.
  - Transceiver: Tx / Rx when requested. Acquire & release transceiver interface, including the control of transceiver power when Tx'ing / Rx'ing.
  - P6: Sample at a variety of rates via A/D inputs. Handshake control to USB / transceiver interface.
  - User processes: TCP/IP & web server over SLIP, 1200 baud Tx/Rx software USART, camera image capture, magnetometer data capture, command processor, EPS control and monitoring, mission supervisor, etc.
  - Power Consumption: Sleep whenever no activity is warranted.

# FM430 Software (cont'd)

- Task to read ambient temperature

```
unsigned long int DegC; // global, accessible to all
...
void TaskMeasureAmbientTemp( void )
{
    unsigned int ADCresult;

    /* setup ADC12 to read ch 10, etc. */
    ADC12CTL0 = ADC12ON+REFON+REF2_5V+SHT0_6;
    ADC12CTL1 = SHP;
    ADC12MCTL0 = INCH_10+SREF_1;

    /* wait 10ms for reference startup */
    OS_Delay(1);

    /* enable conversions */
    ADC12CTL0 |= ENC;

    for (;;)
    {
        ADC12CTL0 |= ADC12SC;           /* start conversion */
        OS_Delay(200);                 /* wait 2s          */
        ADCresult = ADC12MEM0;         /* read result     */
        DegC = (((long)ADCresult-1615)*704)/4095; /* calc. DegC      */
    }
}
```

# FM430 Software (cont'd)

- TaskMeasureAmbientTemp() attributes:
  - Runs independently of others, i.e. loosely-coupled.
  - Runs at a low priority. Ambient temp sensing is not a high-priority issue in this system. OK if other, higher-priority tasks prevent it from running immediately after its 2s delay expires.
  - Uses minimal run-time resources. During the 2s period between successive reads of ADC12MEM0, *no* CPU cycles are expended on TaskMeasureAmbientTemp(), and other tasks are free to run.
  - No inter-task communications, because it runs alone, accessing global variables.
- TaskMeasureAmbientTemp() additional features:
  - Salvo's ability to context-switch at any place in the task allows other tasks to run while TaskMeasureAmbientTemp() is waiting for 10ms delay during ADC12 initialization.

# FM430 Software (cont'd)

- Task to detect if USB is connected:

```
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 */
    }
}
```

# FM430 Software (cont'd)

- TaskDetectUSB() attributes:
  - Runs independently of others, i.e. loosely-coupled.
  - Runs at a moderate priority. System should detect USB connections quickly.
  - Uses minimal run-time resources. During the 250ms period between testing for USB presence, *no* CPU cycles are expended on TaskDetectUSB(), and other tasks are free to run.
  - A binary semaphore is used to control access to a shared resource, the USB / transceiver interface.
- TaskDetectUSB() additional features:
  - TaskDetectUSB() will be "held off" until the USB / transceiver interface is available. If the interface is not available (i.e. another task is using it), TaskDetectUSB() will acquire it when the interface is released *and* TaskDetectUSB() is the highest-priority task waiting to use the interface.



# FM430 Software (cont'd)

- Task to transmit data via transceiver:

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

        /* turn transceiver on, wait 1s while it resets */
        Enable_5V_to_MHX();
        OS_Delay(100);

        FM430PutstrTx1("ATA\r"); /* enter command mode, wait 1s */
        OS_Delay(100);

        /* send some data, wait 1s, turn transceiver off */
        sprintf(strTmp, "TalkMHX:" " Ambient temp is %d C", DegC);
        FM430Msg1(strTmp);
        OS_Delay(100);
        Disable_5V_to_MHX();

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

# FM430 Software (cont'd)

- TaskTalkMHX() attributes:
  - Runs independently of others, i.e. loosely-coupled.
  - Runs at a moderate priority.
  - Uses minimal run-time resources. During the 3s period that the task waits for delays to expire, *no* CPU cycles are expended on TaskTalkMHX(), and other tasks are free to run.
  - A binary semaphore is used to control access to a shared resource, the USB / transceiver interface.
- TaskTalkMHX() additional features:
  - Like TaskDetectUSB(), TaskTalkMHX() must acquire the USB / transceiver interface before proceeding, etc.
  - Once TaskTalkMHX() has acquired the USB / transceiver interface, all other tasks wishing to use the interface must wait (i.e. "are blocked") until TaskTalkMHX() releases the interface at the end of transmission.

# FM430 Software (cont'd)

- Sleep whenever there are no eligible tasks:

```
void OSIdlingHook (void)
{
    __low_power_mode_1();
}
```

- OSIdlingHook() is called only when no tasks are eligible to run. Therefore it's the ideal place to sleep the processor, *until an event* (i.e an internal or external interrupt) *occurs*.
- Exit LPM after each interrupt that calls an RTOS service:

```
#pragma vector=TIMERAO_VECTOR
__interrupt void Timer_A (void)
{
    CCR0 += 10000;
    OSTimer();
    __low_power_mode_off_on_exit();
}
```

- Don't re-enter LPM until Salvo's scheduler has processed event(s). ISRs that are independent of Salvo can resume LPM on exit.

# FM430 Software (cont'd)

- Initialize, create tasks and events, begin multitasking:

```
void main (void)
{
    /* user & Salvo init */
    Init();
    OSInit();

    /* several interrupts are used */
    __enable_interrupt();

    /* create tasks */
    OSCreateTask(TaskStatusMonitor,    OSTCBP(1),  3);
    OSCreateTask(TaskDetectUSB,        OSTCBP(2),  8);
    OSCreateTask(TaskTalkUSB,          OSTCBP(3),  5);
    OSCreateTask(TaskTalkMHX,          OSTCBP(4),  7);
    OSCreateTask(TaskMeasureAmbientTemp, OSTCBP(5), 11);
    ...
    /* create events */
    OSCreateBinSem(BINSEM_USB_MHX_AVAIL_P, 1);

    /* go */
    for (;;)
    {
        OSSched();
    }
}
```

# FM430 Software (cont'd)

- Extending the application:
  - Use additional binary semaphores and task priorities to manage access to resources.
  - Run additional periodic tasks at multiples of system tick period.
  - Use messages and message queues for intertask communications.
  - Use free-running system timer for timestamps.
  - Handle lost events via wait-with-timeout.
  - Interface to mass memory (e.g. via SD/MMC cards).
  - Run uIP-based web server supporting multiple simultaneous connections over TCP/IP via SLIP.

# FM430 Software (cont'd)

- Example Application:
  - CubeSat Kit demonstration code with 10 tasks to exercise hardware for test, handle USART0 and USART1 comms, detect USB, indicate system status, monitor on-board temperature, issue status monitor messages, and run uIP web server and SLIP link.
- Salvo configured for:
  - Priority-based multitasking, 16-bit delays, binary semaphores, 32-bit system timer, 10 tasks and 2 events.
- Salvo's memory requirements on MSP430F169 for this application:
  - 1890 bytes Flash.
  - 107 bytes RAM.
  - Default of 90 bytes RAM for stack is more than sufficient.
- Observed power profile:
  - Over 97% of the time in LPM.

# FM430 Software (cont'd)

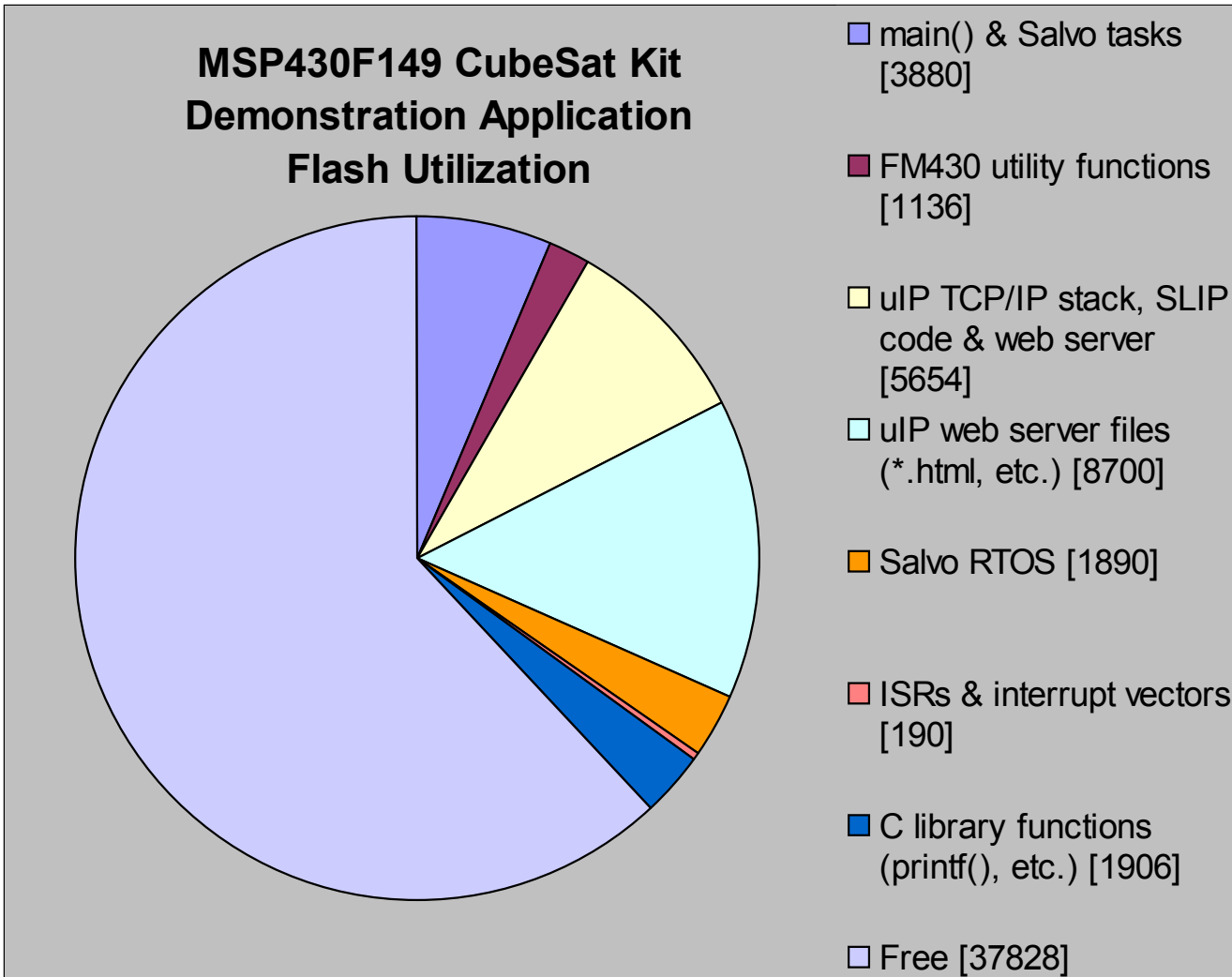


Figure 11:  
Flash Map



# FM430 Software (cont'd)

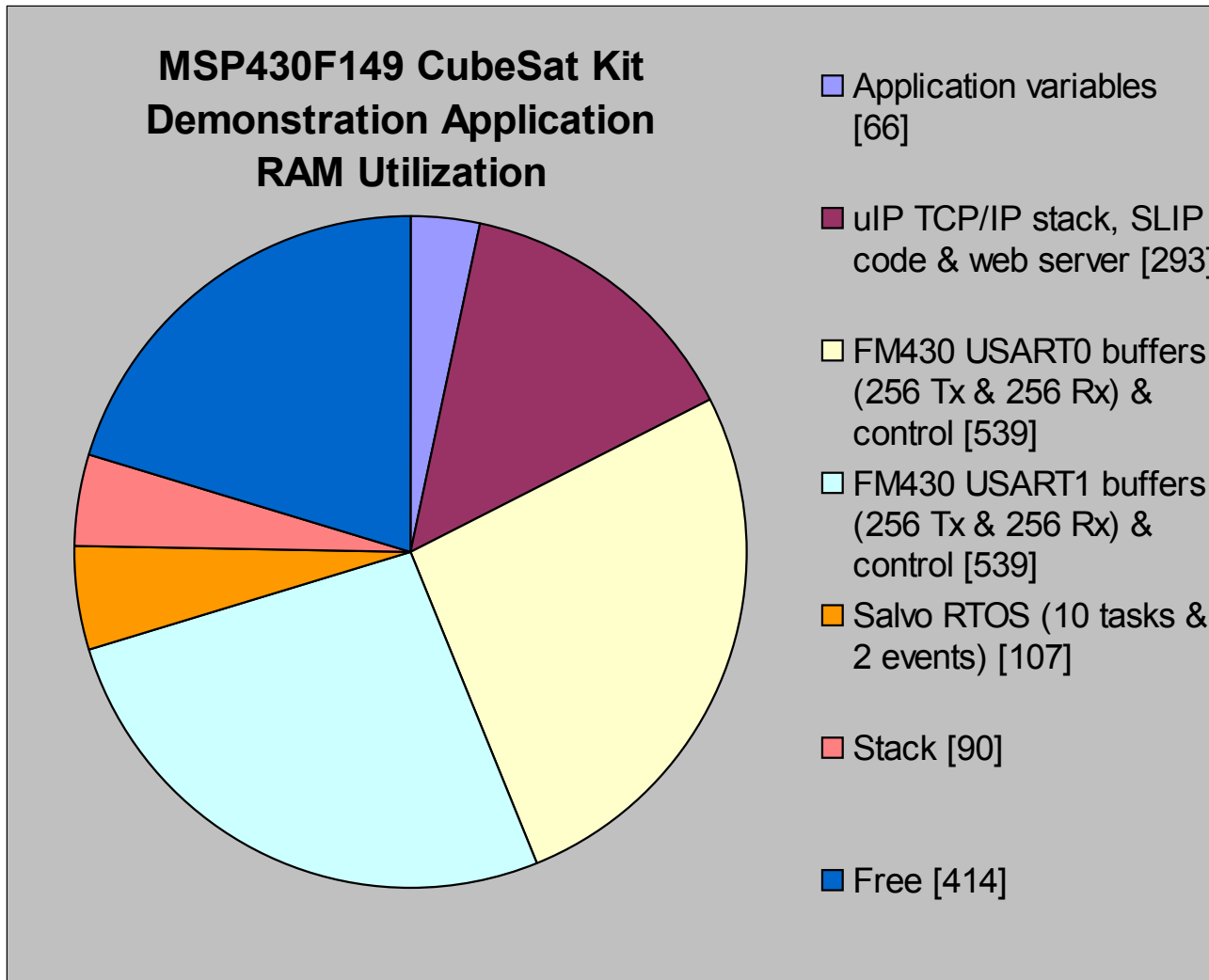


Figure 12:  
RAM Map

# Part VIII: Summary

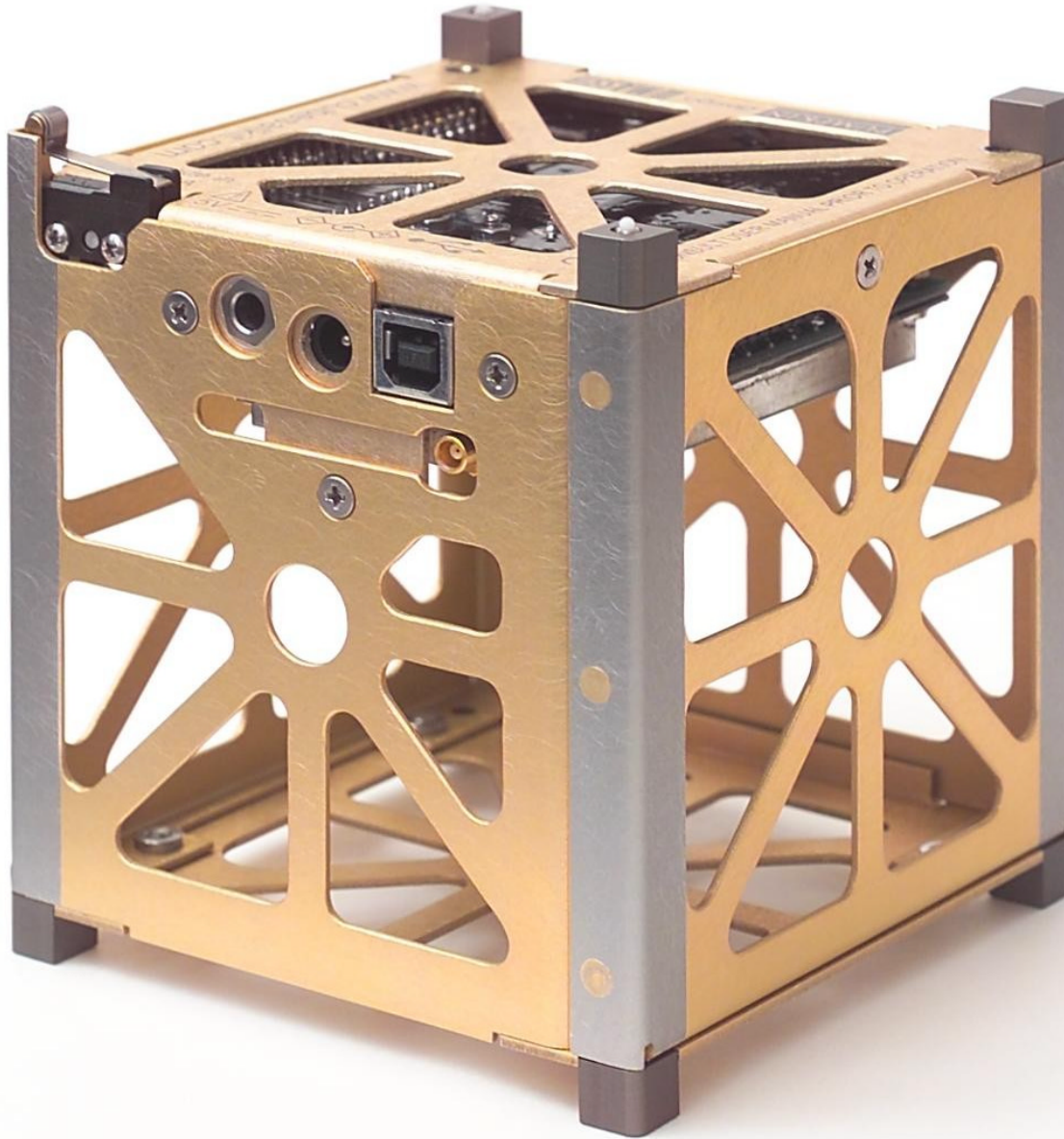
- MSP430 is the right choice for us because:
  - The single-chip MSP430F14x/16x/161x minimizes parts count on the FM430.
  - The MSP430's on-board peripherals (USARTS, ADC, DAC, DMA, timers, interrupts) and multi-purpose I/O enables very high level of application functionality and system integration.
  - Its ultralow-power operation and configurable clock module enable 24x7 MCU operation in a hostile environment.
  - The MSP430's 3.3V operation easily integrated into 5V design.
  - Its C-friendly architecture with 50/55/60K Flash and 10/5/2K RAM, respectively, presents the opportunity for substantial functional software integration, thereby reducing system parts count.



# Summary (cont'd)

...

- There are a wide range of high-quality programming tools available for the MSP430. Programming and debugging the MSP430 over JTAG is *easy*.
- The Salvo RTOS on MSP430 simplifies software development, enables code modularity, and results in a maintainable and expandable high-performance application with low power requirements.
- TI's continuing development of the MSP430 family means that faster and/or larger-memory family members can be seamlessly designed into the CubeSat Kit as future upgrades.



Live Demo

Q&A Session

Thank you for  
attending this  
Pumpkin seminar  
at the ATC 2005!

# Suggested Reading

1. *Interfacing the 3-V MSP430 to 5-V Circuits*, Texas Instruments Application Report SLAA148, October 2002.
2. *SN54AHCT244, SN74AHCT244 Octal Buffers/Drivers with 3-state Outputs*, Texas Instruments Datasheet SCLS288L, July 2003.
3. *SN74LVC244A Octal Buffers/Driver with 3-state Outputs*, Texas Instruments Datasheet SCAS414U, August 2003.
4. *TPS3836E18 / J25 / H30 / L30 / K33, TPS3837E18 / J25 / L30 / K33, TPS3838E18 / J25 / L30 / K33 Nanopower Supervisory Circuits*, Texas Instruments Datasheet SLVS292B, December 2002.
5. *TPS76901, TPS76912, TPS76915, TPS76918, TPS76925, TPS76927, TPS76928, TPS76930, TPS76933, TPS76950 Ultralow-power 100-mA Low-dropout Linear Regulators*, Texas Instruments Datasheet SLVS203E, May 2001.
6. *MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller*, Texas Instruments Datasheet SLAS368D, October 2002.
7. *The I2C-bus specification*, Philips Semiconductors.
8. *Salvo User Manual*, Pumpkin, Inc., 2003.
9. *CubeSat Kit User Manual*, Pumpkin, Inc. 2005.
10. The uIP Embedded TCP/IP Stack, Adam Dunkels, <http://www.sics.se/~adam/uiip/>.



# 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 a successful Silicon Valley high-tech pro-audio startup, 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.

## • 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

- © 2005 Pumpkin, Inc. All rights reserved. Pumpkin and the Pumpkin logo, Salvo and the Salvo logo, The RTOS that runs in tiny places, CubeSat Kit, CubeSat Kit Bus and the CubeSat Kit logo are all trademarks of Pumpkin, Inc. The TI signature, logo and Technology for Innovators are trademarks of Texas Instruments and are used with permission. All other trademarks and logos are the property of their respective owners. No endorsements of or by third parties listed are implied. All specifications subject to change without notice.

First presented at TI's 4<sup>th</sup> annual MSP430 Advanced Technical Conference in Dallas, Texas on November 1-3, 2005.