Hardware Engineering Summer 2014 Internship

Spirent Communications

Kevin Hu

August 7, 2014

SPIRENT Communications

Inspired Innovation

Inspired Innovation

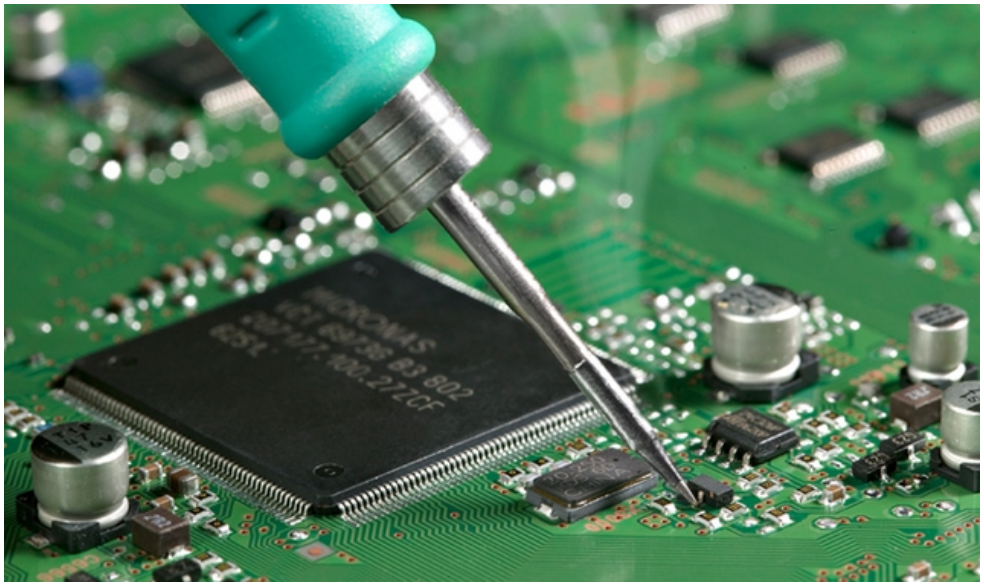
Proprietary & Confidential—Page 1

Overview

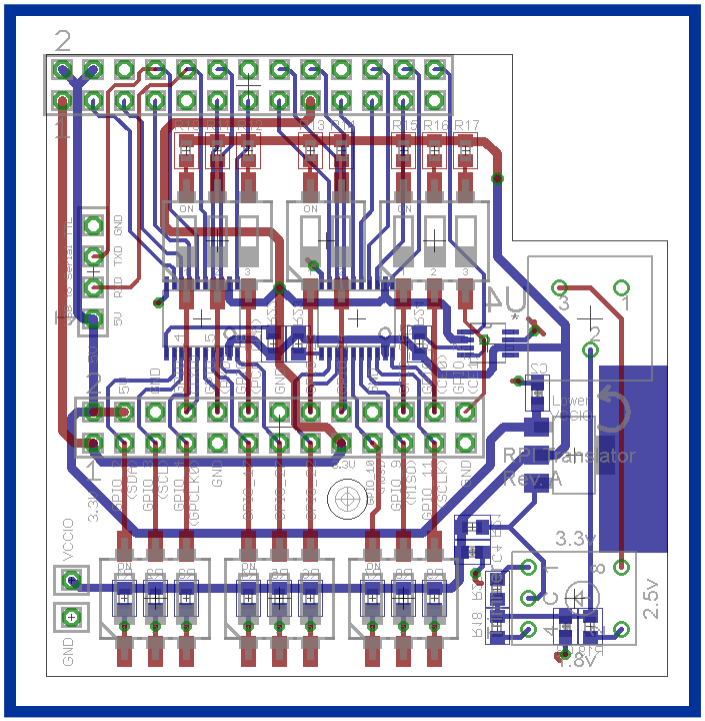
1. Chassis Development
2. Voltage Translator Circuit
3. Raspberry Pi Read/Write Benchmarking
4. I2C Protocol
5. Signal Capture
6. Signal Decode
7. Kernel Programming
8. Outstanding Issues
9. Conclusion

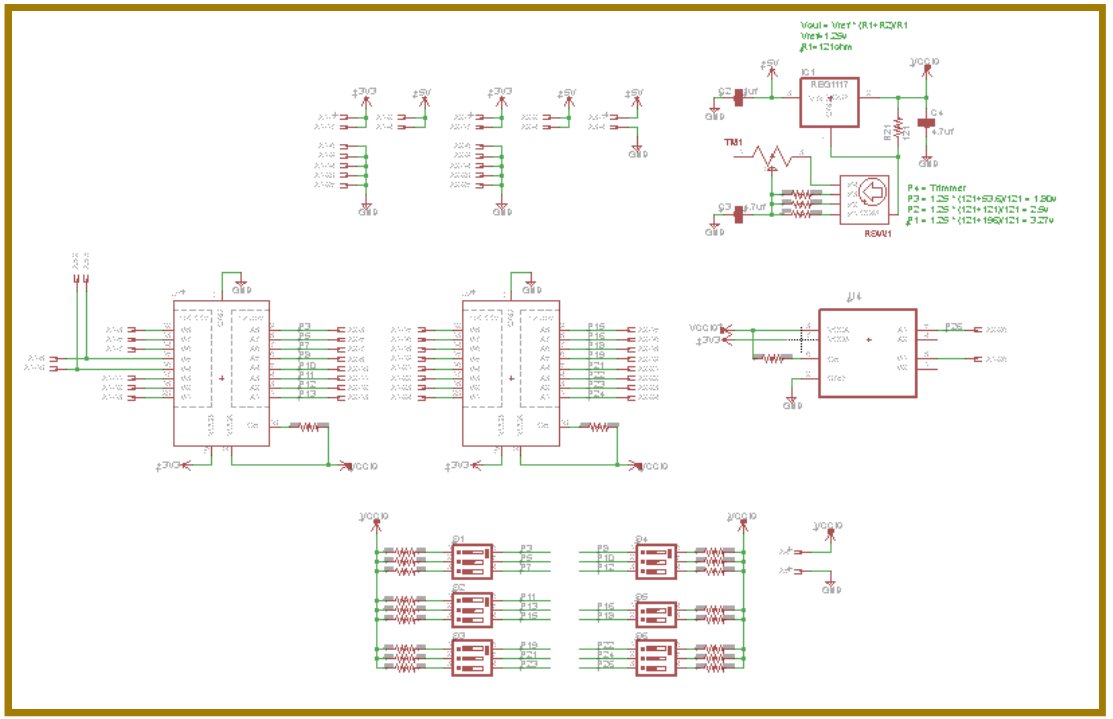
Chassis Development

- Week 1
- Basic Schematic Diagram Reading
- Soldering skills



Voltage Translator Circuit

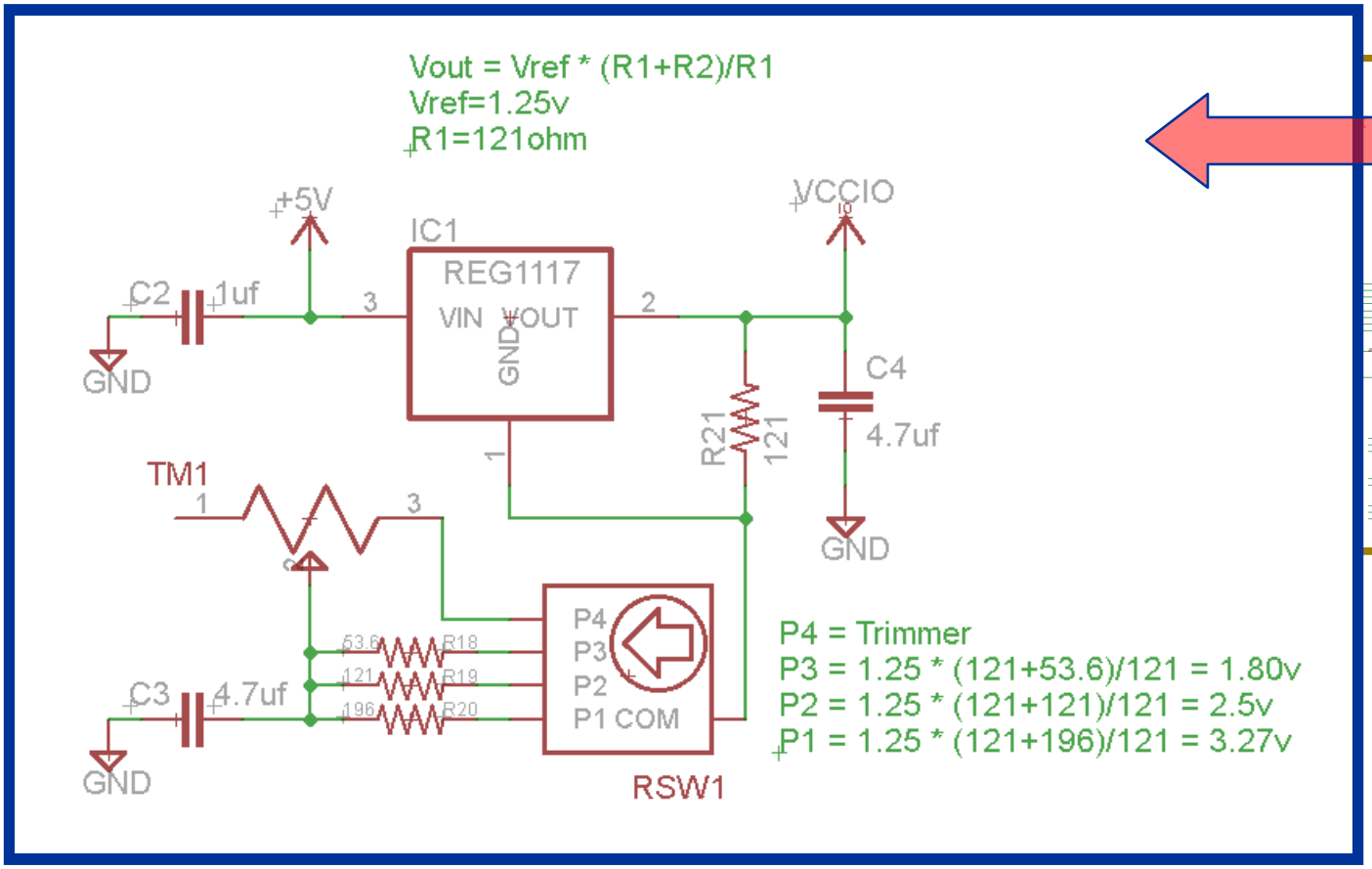


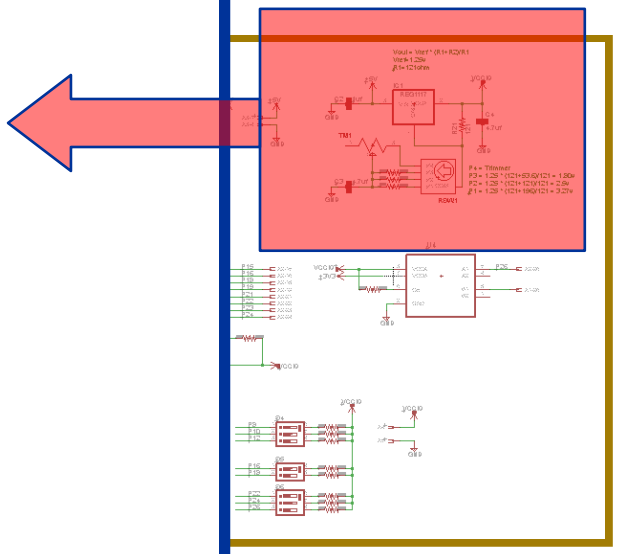


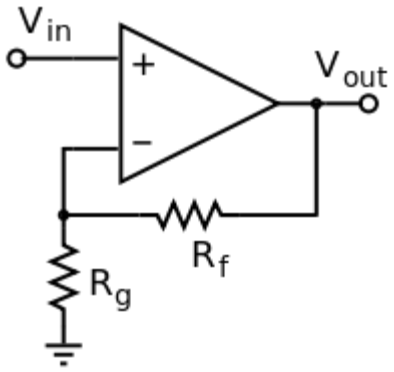
- Week 2
  - Schematic Diagram, Circuit Analysis
  - Debugging with zero assumptions

SPIRENT Communications Inspired Innovation

Voltage Translator Circuit







SPIRENT Communications

Inspired Innovation

Raspberry Pi Read/Write Benchmarking

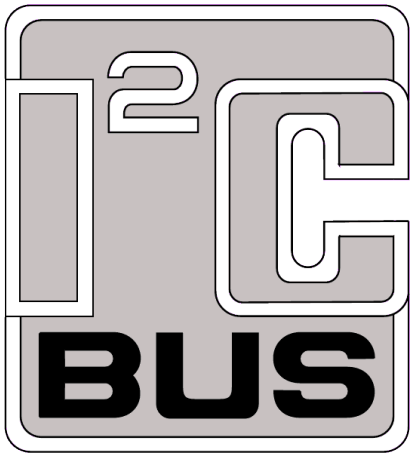
- Weeks 2-3
- How fast can the Raspberry Pi toggle?
- Writing: Simple assignment to pin
- Reading: Read pin, store to memory

Raspberry Pi Read/Write Benchmarking

- Libraries: Python RPI GPIO Library, BCM2835 C Library
  - Perusing software libraries for unfamiliar hardware
  - Python Extension Programming: C library in Python
- Code Efficiency

I2C Protocol

- Week 4
- Understand I2C Protocol, used in many embedded systems
- Sending all necessary info in one 2-wire bus
- Consider all possible cases



Signal Capture

- Weeks 4-6
- Design and coding in C
- Code Efficiency
- Assembly Language
- Getopt

```
// Read input pin
uint8_t bcm2835_gpio_lev(uint8_t pin)
{
    volatile uint32_t* paddr = bcm2835_gpio + BCM2835_GPLEV0/4 + pin/32;
    uint8_t shift = pin % 32;
    uint32_t value = bcm2835_peri_read(paddr);
    return (value & (1 << shift)) ? HIGH : LOW;
}
```

```
uint32_t bcm2835_peri_read(volatile uint32_t* paddr)
{
    if (debug)
    {
        printf("bcm2835_peri_read  paddr       ", (unsigned) paddr);
    return 0;
    }
    else
    {
    // Make sure we dont return the _last_ read which might get lost
    // if subsequent code changes to a different peripheral
    uint32_t ret = *paddr;
    *paddr; // Read without assigneing to an unused variable
    return ret;
    }
}

// read from peripheral without the read barrier
uint32_t bcm2835_peri_read_nb(volatile uint32_t* paddr)
{
    if (debug)
    {
    printf("bcm2835_peri_read_nb  paddr       ", (unsigned) paddr);
    return 0;
    }
    else
    {
    return *paddr;
    }
}
```

```
void compressed(){
    volatile uint32_t * paddr = bcm2835_gpio + BCM2835_GPLEV0/4 + pinArray[0].gpioPins/32;
    uint8_t temp;   // to compare, print only when diff

    rawtime = time(NULL) - (time_t)(3600*10);   // 3600s/hr * (UTC-10) hrs for HI time
    t = clock();

    valueArray[j++] = (uint8_t) *paddr;     // j = 0, j++ for first iteration of [j-1] in loop

    // 10.5MHz
    for(i=1;;i++){      // infinite loop, wait for ctrl+c
        temp = (uint8_t) *paddr;  // inlined peri_read_nb() from bcm2835.h
        if(valueArray[j-1] != temp){
            valueArray[j] = temp;
            j++;
        }
    }
}


void nonCompressed(){
    volatile uint32_t * paddr = bcm2835_gpio + BCM2835_GPLEV0/4 + pinArray[0].gpioPins/32

    rawtime = time(NULL) - (time_t)(3600*10);   // 3600s/hr * (UTC-10) hrs for HI time
    t = clock();

    // 9MHz
    for(i=0;;i++)           // infinite loop, wait for ctrl+c
        valueArray[i] = (uint8_t) *paddr;
}
```

Signal Capture

- Manual Trace of I2C Instructions
- Si5338 Board, i2cset / i2cget
- Single Processor, Single Threaded

Signal Decode

- Design and code in Python
  - Easier user interface
  - Speed is not imperative
- PEP 8 - Style Guide for Python Code
- Understanding of I2C Protocol
- Consider all possible cases

```
def decode(valueArray):
    state = IDLE

    # Start decoding data transfer 1s and 0s
    for i in valueArray:
        # IDLE, wait for START
        if state == IDLE:
            if idle(i,standbyValue) == True:
                state = SLAVE_ADDRESS
            continue

        # SLAVE ADDRESS
        elif state == SLAVE_ADDRESS:
            if slave_address(i) == True:
                state = READ_WRITE
            continue

        # READ / WRITE
        elif state == READ_WRITE:
            if read_write(i) == True:
                state = ACK_NACK
            continue

        # ACK / NACK: After Read/Write or Data
        # If NACK: DATA_BYTE state determines if Stop/Restart follows
        # If ACK: DATA_BYTE state determines if ClockStretch/Stop/Restart/Data
        elif state == ACK_NACK:
            if ack_nack(i) == True:
                state = DATA_BYTE
            continue
```

```
    # DATA: Includes possible ClockStretch/Stop/Restart
    # Immediately follows ACK
    elif state == DATA_BYTE:
        state = check_clock_stop_restart(i)
        if read_byte(i,state) == True:  # If decoded end of a byte
            state = ACK_NACK
        # Do not 'continue'; run through STOP and RESTART checks first

    # STOP: Detected in DATA_BYTE state.
    # Not ELIF since we do not want to read another line value
    if state == STOP:
        write_instruction(state)
        clear()
        state = IDLE
        continue

    # RESTART: Detected in DATA_BYTE state.
    # Not ELIF since we do not want to read another line value
    if state == RESTART:
        write_instruction(state)
        clear()
        state = SLAVE_ADDRESS

# Waveform expected to end in STOP, returning IDLE state
if state != IDLE:
    errorFlag()
```

S0x70WACK0x26ACK0x01ACKP
S0x70WACK0x27ACK0x07ACKP
S0x43RACKP
S0x70WACK0x29ACK0x5eACKP
S0x70WACK0x2aACK0x27ACKP
S0x15RACK0x00ACKP
S0x70WACK0x2cACK0x00ACKP
S0x70WACK0x2dACK0x00ACKP
S0x70WACK0x2eACK0x00ACKP
S0x70WACK0x2fACK0x14ACKP
S0x70WACK0x30ACK0x33ACKP
S0x70WACK0x31ACK0x05ACKP
S0x70WACK0x32ACKP
S0x70WACK0x33ACK0x07ACKP
S0x70WACK0x34ACK0x10ACKP
S0x1aRACK0x00ACKP
S0x70WACK0x36ACK0x7bACKP
S0x70WACK0x37ACK0x00ACKP
S0x70WACK0x38ACK0x00ACKP

Signal Decode

- Logic Analyzer
- *Understanding the Linux Kernel*
  - Process Scheduling
  - ‘chrt’ to reassign priority

Kernel Programming

- Interface between user space and hardware
- C Programming
  - Zero tolerance for runtime error (must reboot each time)
- Building Kernel Tree for module inserts

Kernel Programming

- CONFIG PREEMPT RT Patch: https://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT_Patch
- *Linux Device Drivers*
  - HelloWorld, Memory, Char Device, Proc Entry, sysfs

Outstanding Issues

- Signal Capture still has varying high latencies even using kernel
- sysfs implementation
- Signal Decode may have an edge case error

Conclusion

- Ask the right questions AND consider all possibilities for answers
  - Software: Algorithm, Implementation, Optimization, Assembly, Memory, OS
  - Hardware: Physical Memory, Wiring, Soldering, Processor Specs
- Edge Cases

Thank you!

SPIRENT Communications

Inspired Innovation

Inspired Innovation

Proprietary & Confidential—Page 23