SPI on RTU

(plus other fun features & peripherals)

Jerry Chen
Richard Lin
Allen Tang

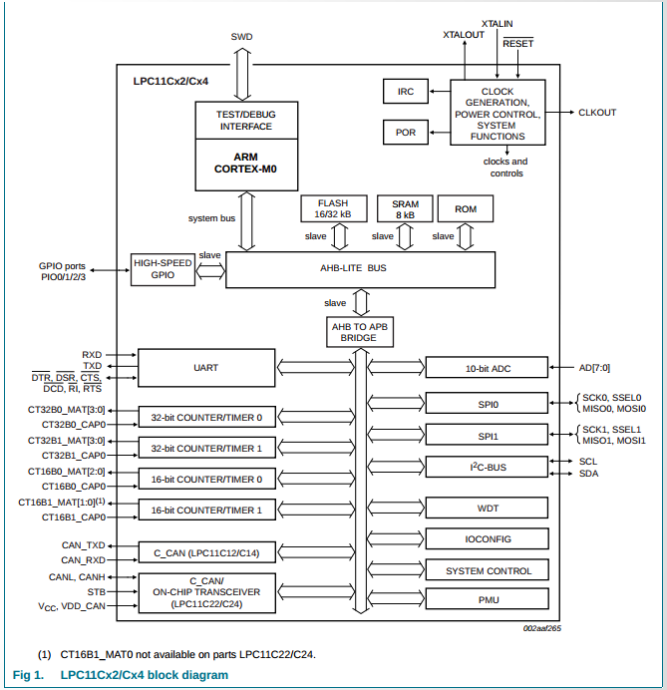
Intro: Embedded Architecture

Microcontrollers:

- typically CPU core
  - optimized for speed, not precision timing
- hardware peripherals
  - when you need timing predictability

Peripherals:

- timing demands
  - many protocols have strict requirements
- SPI, I^{2}C, CAN, ...



(1) CT16B1_MAT0 not available on parts LPC11C22/C24.

Fig 1. LPC11Cx2/Cx4 block diagram

microcontroller layout

http://www.nxp.com/documents/data_sheet/LPC11CX2_CX4.pdf

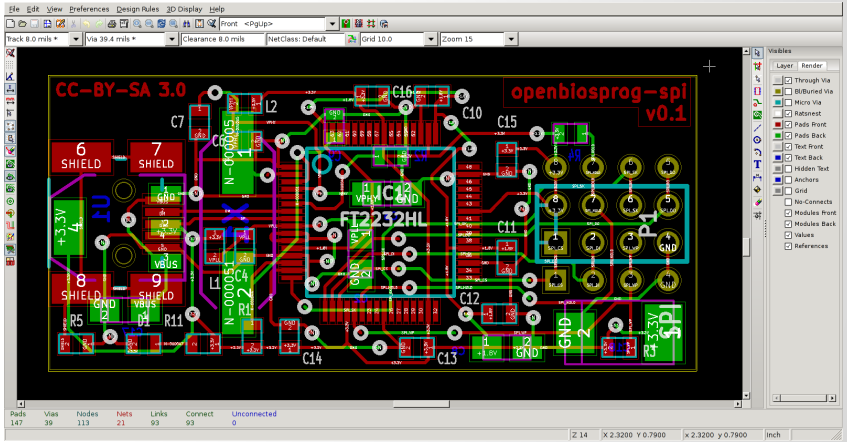
Motivation: Peripherals

Problems

- non-reconfigurable
  - only what the chip manufacturer gives
- static, fixed pinning
  - makes PCB routing a terrible bloody mess

Ideals

- reconfigurable
- arbitrary functions
- predictable timing



suboptimal board layouts

http://randomprojects.org/wiki/File:Openbiosprog-spi-pcb-kicad-0.1.png

RTU: Basics

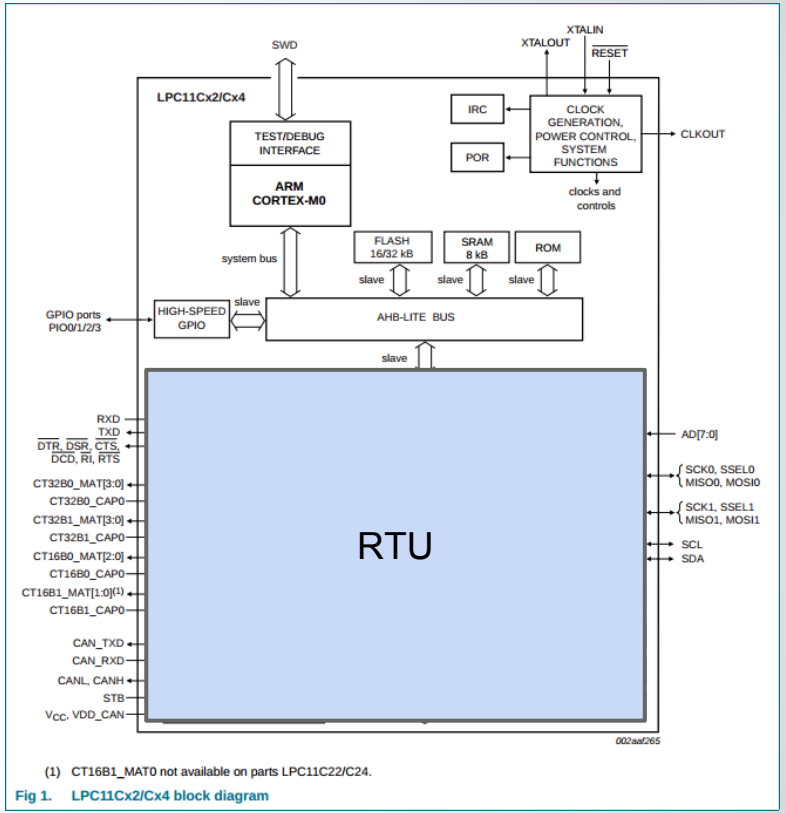
RTU: Real Time Unit

CPU with timing predictability

- cycle-predictable timing possible
- low-level, bare metal programming

Host processor and coprocessor

- Fast host processor running application software
- RTU coprocessor doing timing-critical work



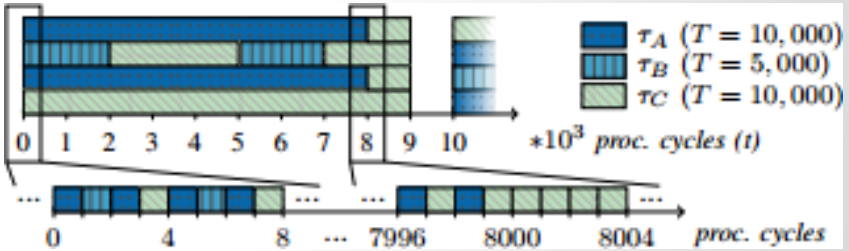
RTU architecture diagram

http://www.nxp.com/documents/data_sheet/LPC11CX2_CX4.pdf

RTU: Introduction to FlexPRET

FlexPRET

- multithreaded research processor
- RISC-V ISA
- trade-off timing predictability and CPU utilization



FlexPRET (Zimmer et. al) Overview
http://www.bromans.com/publ/zimmer-et-al-2014-flexpret.pdf

RTU: Software Primitives

GPIO Operations

how to interact with the real world

- **gpo_write**: write into the GPIO array
- **gpo_set**, **gpio_clr** masked write into the GPIO array
- **gpi_read**

Timing Operations

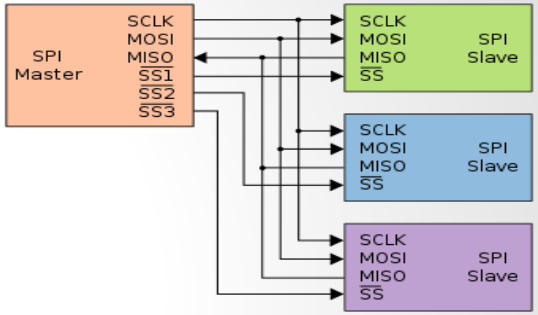
how to time the interactions correctly

- **get_time**
- **delay_until**
- **periodic_delay**: delay until and advance timer

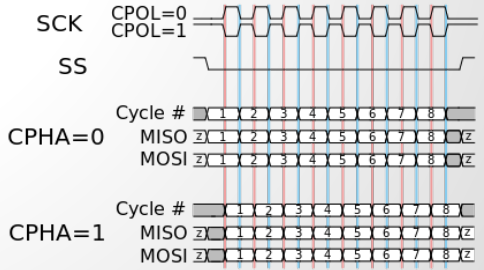
SPI: Introduction

SPI: Serial Peripheral Interface

- talk to external devices: sensors, actuators, etc
- SS / CS pin selects chip
- dual simplex: MISO, MOSI
- synchronous: clocked data
- new bit on one edge, read on other



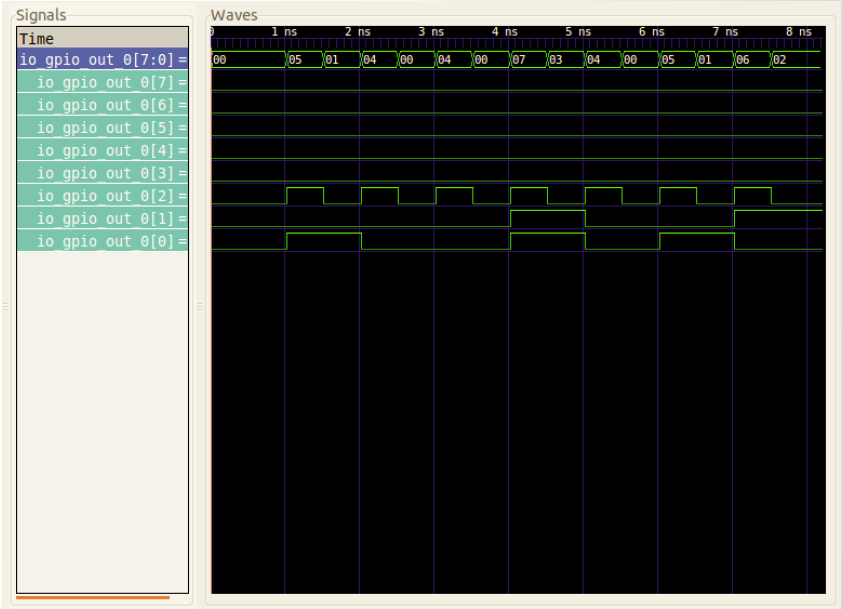
SPI block diagram overview
https://commons.wikimedia.org/wiki/File:SPI_three_slaves.svg



SPI waveform
https://commons.wikimedia.org/wiki/File:SPI_timing_diagram2.svg

SPI: Bitbang Routine

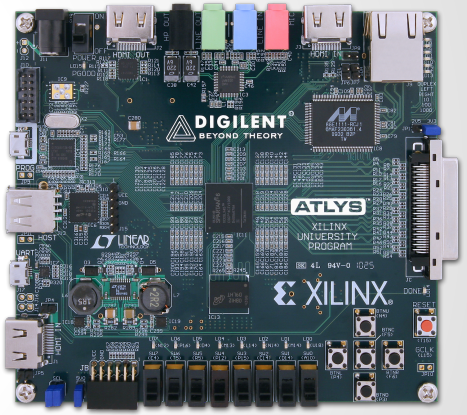
```
uint8_t SPI_transfer(uint8_t write_byte){
  while (bit < 8) {

    … unpack data_bit, pack read_bit …
    to_write = (clk_bit << 2) |
        (read_bit << 1) | data_bit;
    gpo_write(to_write);
    periodic_delay(&clk, PERIOD/2);


    clk_bit = 0;
    to_write = (clk_bit << 2) |
      (read_bit << 1) | data_bit;
    gpo_write(to_write);
    periodic_delay(&clk, PERIOD/2);
    clk_bit = 1;
  }
  return result;
}
```



our bitbang SPI

Deploying to FPGA

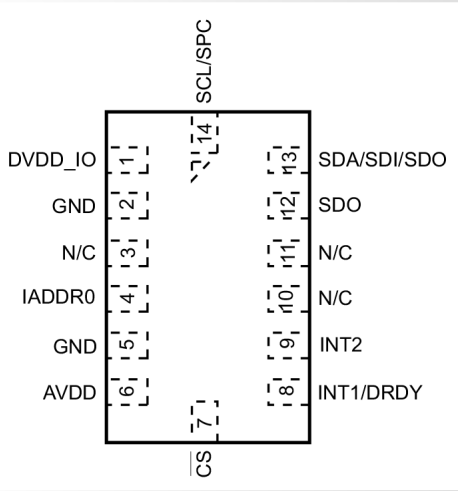
- Successfully deployed FlexPRET to Atlys dev board (Spartan-6 FPGA) and ran C programs
- Connected GPIO to LEDs and digital accelerometer



Digilent Atlys board
www.digilentinc.com

Digital Accelerometer via SPI

- Interfaced with SPI pins of accelerometer chip with bitbang routine
- Read 8-bit values from any axis, normalize it to 1g (64 LSB), and display result on LEDs



Pinout diagram
http://cache.freescale.com/files/sensors/doc/data_sheet/MMA7455L.pdf?fasp=1

Pulse-width Modulation

Another simple protocol!

- Also implement PWM: vary on-time and off-time of fast pulse
- Use bitbanging PWM to toggle (dim) GPIO-attached LEDs

```
volatile uint32_t g_period;
volatile uint32_t g_high_time;

void PWM_generator() {
  uint32_t clk = get_time();
  while (1)
  {
    uint32_t high_time = g_high_time;
    uint32_t low_time = g_period
                        - g_high_time;
    set_pwm_io();
    periodic_delay(&clk, high_time);
    clr_pwm_io();
    periodic_delay(&clk, low_time);
  }
}
```

a simple bitbang PWM

Multithreading PWM and SPI

- Use 2 hardware threads to run SPI (accelerometer) and PWM concurrently
- Allow PWM to read accelerometer data via a shared variable
- Result: accelerometer-controlled LED brightness!

Live Demo

Host Interface

Coprocessor useless without comms, so:

- takes in word-wide command from bus
- 8 bits opcode, 24 bits data

- set_period: sets clock period (ns)
- set_polarity: set CPOL and CPHA
- transfer: does an 8 bit transfer, put the result on the bus

Testing

Need confidence: automated testbench

- Temporal constructs around Chisel tester
- Checks clock jitter and data values
- Uses host interface

```
Cycle 9165, clk 0, MOSI 1
Cycle 9215, clk 1, MOSI 1
Got bit 4 = 1
Cycle 9265, clk 0, MOSI 0
Cycle 9315, clk 1, MOSI 0
Got bit 5 = 0
Cycle 9365, clk 0, MOSI 1
Cycle 9415, clk 1, MOSI 1
Got bit 6 = 1
Cycle 9465, clk 0, MOSI 0
Cycle 9515, clk 1, MOSI 0
Got bit 7 = 0
RAN 9573 CYCLES PASSED
```

Cycle-accurate Timing

Reducing SPI jitter

- Precompute next GPIO byte before periodic_delay
- Consistently place gpo_write after periodic_delay

More accurate PWM

- Enforce minimum time resolution: periodic_delay takes time
- Also precompute waveform and time

The End

Feedback?
Thoughts? Comments?

Thanks for watching!

Special thanks to Michael Zimmer for his work on the FlexPRET processor