ECE 448
Lecture 16

Bare Metal System Software Development

Required Reading

- *P. Chu, FPGA Prototyping by VHDL Examples*

  Chapter 9, Bare Metal System Software Development

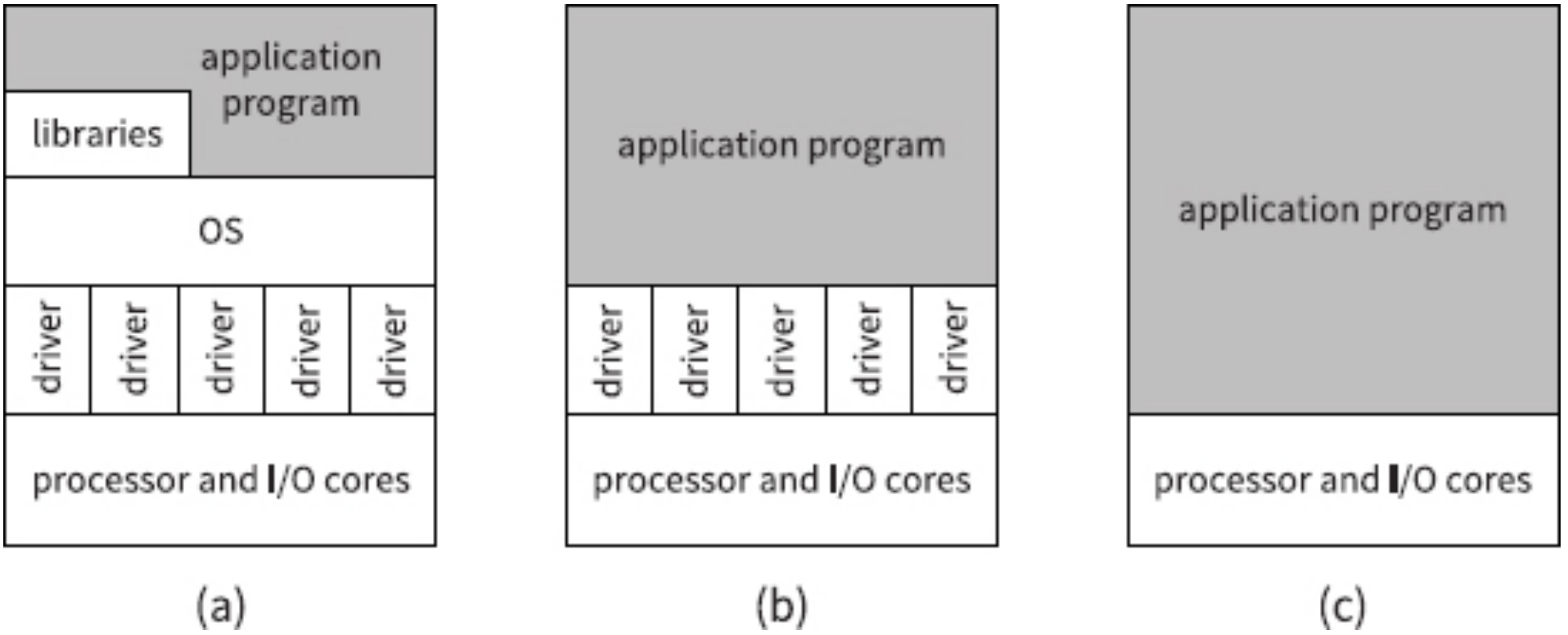
- *Source Code of Examples*

http://academic.csuohio.edu/chu_p/rtl/fpga_mcs_vhdl.html

- *Basys 3 FPGA Board Reference Manual*

  7. VGA Port

Software Hierarchy



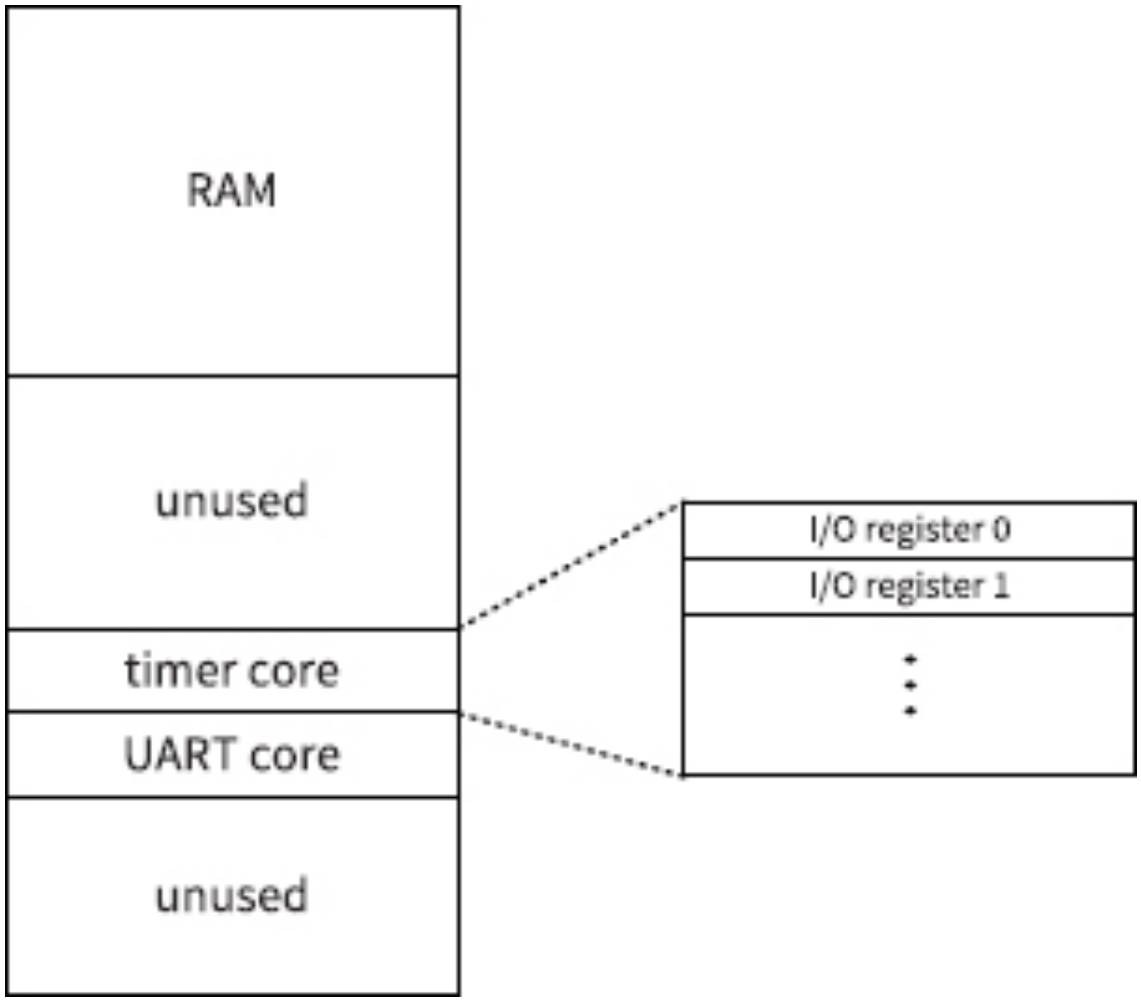
Desktop-like System

Bare Metal Systems

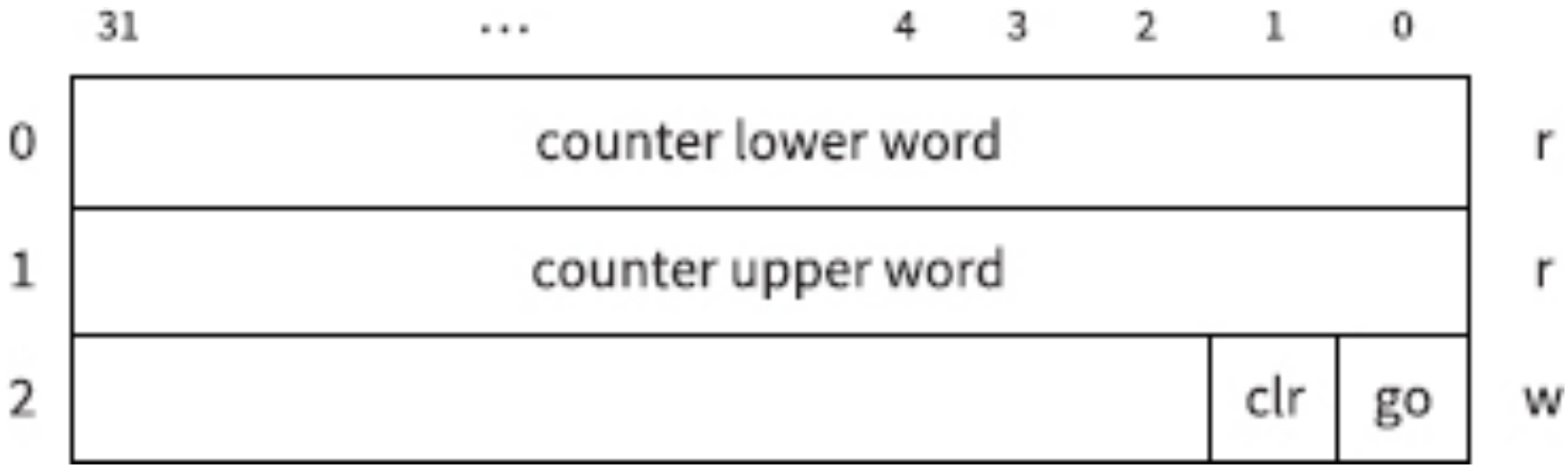
Basic Embedded Program Architecture

```
main(){
   sys_init();
   while(1){
      task_1();
      task_2();
      ...
      task_n();
   }
}
```

Address Map of a Simple System



I/O Register Map of a Timer Core



I/O Address Map of the FPro System

- The subsystem provides 64 slots to connect up to 64 (i.e., 2^6) I/O cores
- Each I/O core is allocated with 32 (i.e., 2^5) registers
- Each register is 32 bits wide (i.e., a word)
- The subsystem requires a memory space of 2^{11} (i.e., $2^6 * 2^5$) words or 2^{13} bytes.
- The 11-bit word address for the MMIO subsystem appears as

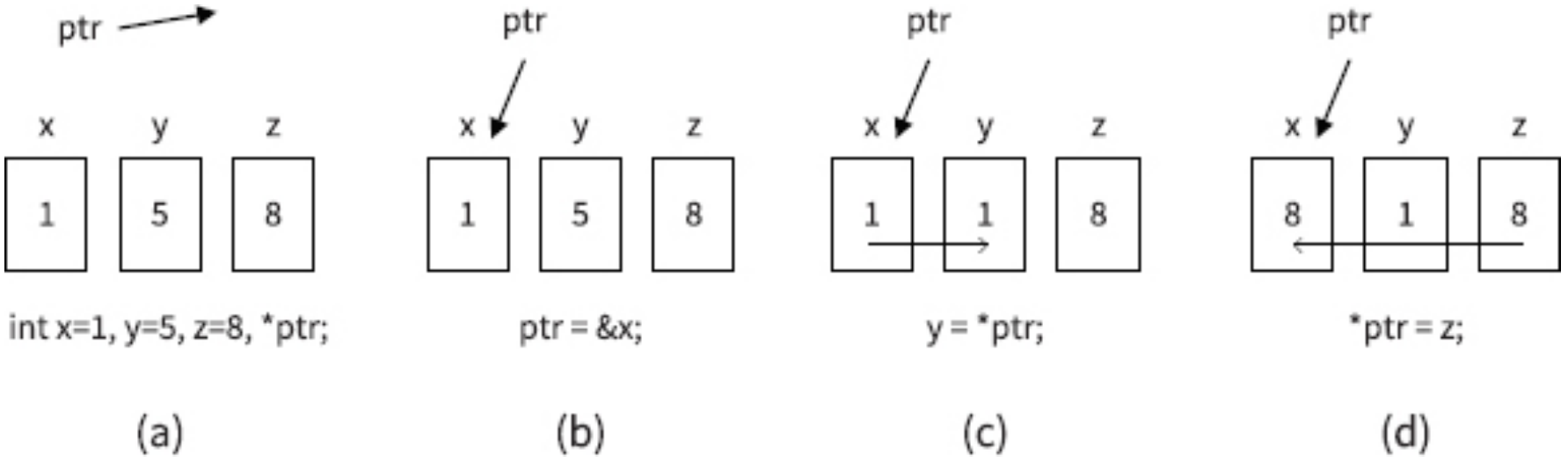
  sss_sssr_rrrr

  in which ssssss is the slot number and rrrrr is the register offset. When combined with the video subsystem, its 22-bit word address becomes 00_0000_0000_0sss_sssr_rrrr
- However, most I/O cores will not use all 32 registers and will not always need 32 data bits

C Pointers

```
int x=1, y=5, z=8, *ptr;

ptr = &x;      // ptr gets symbolic address of x
y = *ptr;      // content of y gets content pointed by ptr
*ptr = z;      // content pointed by ptr gets content of z
```



C Pointer to I/O Register

```
int sw;
int pattern=0x0055;

sw = *(0xc0000180);
*(0xc0000100) = pattern;
```

Robust I/O Register Access

- The address assignment of the FPro system is fixed for the slots in the MMIO subsystem and various modules in the video subsystem.
- Only two parts may change:
  - Bridge base address
  - Slot assignment in the MMIO subsystem
- Instead of using hard literals, we express the information using symbolic constants and record them in
  - chu_io_map.h : a C/C++ header file used for software development
  - chu_io_map.vhd : a VHDL package declaration to be used in conjunction with hardware development

Robust I/O Register Access (cont.)

- The processor treats the MMIO and video subsystems as a single I/O module and communicates with the module via the bridge
- The bridge base address is the starting address of the module and is assigned when a system is created
- For the MicroBlaze MCS configuration, it is a fixed value of 0xc0000000
- For an IP core attached to the MMIO subsystem, its base address can be calculated using the assigned slot number. Recall that each slot contains 32 words (128 bytes). The base address of slot n is

$$\text{Bridge\_base\_address} + n * 32 * 4$$

Slot assignment of the vanilla FPro system

```
#define S0_SYS_TIMER   0      // slot 0
#define S1_UART1       1      // slot 1
#define S2_LED         2      // slot 2
#define S3_SW          3      // slot 3
```

Slot and constant definitions in chu_io_map.h

```
#ifndef _CHU_IO_MAP_INCLUDED
#define _CHU_IO_MAP_INCLUDED

#ifdef __cplusplus
extern "C" {
#endif

#define SYS_CLK_FREQ 100

//io base address for microBlaze MCS
#define BRIDGE_BASE 0xc0000000

// slot module definition
#define S0_SYS_TIMER  0
#define S1_UART1      1
#define S2_LED        2
#define S3_SW         3
// ... additional slot definitions for cores in Parts III and IV

#ifdef __cplusplus
} // extern "C"
#endif

#endif  // _CHU_IO_MAP_INCLUDED
```

inttypes.h

- C has many predefined data types, such as short, int, and long. The width (i.e., number of bits) of each data type is left to the compiler and implementation.
- While interacting with low-level device activities, it is often important to know the exact width and format of registers and data.
- To facilitate this, C provides a header file, inttypes.h, which explicitly specifies the width and format of each data type.
- It is good practice to use these data types for low-level coding.

Data Types in inttypes.h

- int8_t: signed 8-bit integer
- uint8_t: unsigned 8-bit integer
- int16_t: signed 16-bit integer
- uint16_t: unsigned 16-bit integer
- int32_t: signed 32-bit integer
- uint32_t: unsigned 32-bit integer
- int64_t: signed 64-bit integer
- uint64_t: unsigned 64-bit integer

I/O Macros in chu_io_rw.h

```
#ifndef _CHU_IO_RW_H_INCLUDED
#define _CHU_IO_RW_H_INCLUDED

#include <inttypes.h>    // to use unitN_t type
#ifdef __cplusplus
extern "C" {
#endif

#define io_read(base_addr, offset) \
   (*(volatile uint32_t *)((base_addr) + 4*(offset)))

#define io_write(base_addr, offset, data) \
   (*(volatile uint32_t *)((base_addr) + 4*(offset)) = (data))

#define get_slot_addr(mmio_base, slot) \
   ((uint32_t)((mmio_base) + (slot)*32*4))

#ifdef __cplusplus
} // extern "C"
#endif

#endif /* _CHU_IO_RW_H_INCLUDED */
```

GpoCore class definition in gpio_core.h

```
class GpoCore {
   /* register map */
   enum {
      DATA_REG = 0 //data register
   };
public:
   GpoCore(uint32_t core_base_addr);          //constructor
   ~GpoCore();                                //destructor; not used
   /* methods */
   void write(uint32_t data);                 //write a 32-bit word
   void write(int bit_value, int bit_pos); //write 1 bit
private:
   uint32_t base_addr;
   uint32_t wr_data;                          //same as GPO core data reg
};
```

GpoCore class implementation in gpio_core.cpp

```
GpoCore::GpoCore(uint32_t core_base_addr) {
   base_addr = core_base_addr;
   wr_data = 0;
}

GpoCore::~GpoCore() {}

void GpoCore::write(uint32_t data) {
   wr_data = data;
   io_write(base_addr, DATA_REG, wr_data);
}

void GpoCore::write(int bit_value, int bit_pos) {
   bit_write(wr_data, bit_pos, bit_value);
   io_write(base_addr, DATA_REG, wr_data);
}
```

GpiCore class definition in gpio_core.h

```
class GpiCore {
    /* register map */
    enum {
        DATA_REG = 0 //data register
    };
public:
    GpiCore(uint32_t core_base_addr);  //constructor
    ~GpiCore();                        //destructor; not used
    /* methods */
    uint32_t read();                   //read a 32-bit word
    int read(int bit_pos);             //read 1 bit
private:
    uint32_t base_addr;
};
```

GpiCore class implementation in gpio_core.cpp

```
GpiCore::GpiCore(uint32_t core_base_addr) {
    base_addr = core_base_addr;
}

GpiCore::~GpiCore() { }

uint32_t GpiCore::read() {
    return (io_read(base_addr, DATA_REG));
}

int GpiCore::read(int bit_pos) {
    uint32_t rd_data = io_read(base_addr, DATA_REG);
    return ((int) bit_read(rd_data, bit_pos));
}
```

TimerCore class definition in timer_core.h

```
class TimerCore {
   /* register map */
   enum {
      COUNTER_LOWER_REG = 0,   //lower 32 bits of counter
      COUNTER_UPPER_REG = 1,   //upper 16 bits of counter
      CTRL_REG = 2             //control register
   };
   /* masks */
   enum {
      GO_FIELD  = 0x00000001, //bit 0 of ctrl_reg; enable
      CLR_FIELD = 0x00000002  //bit 1 of ctrl_reg; clear
   };
public:
   TimerCore(uint32_t core_base_addr);  //constructor
   ~TimerCore();                        //destructor; not used
   /* methods */
   void pause();                        //pause counter
   void go();                           //resume counter
   void clear();                        //clear the counter to 0
   uint64_t read_tick();                //retrieve # clocks elapsed
   uint64_t read_time();                //read time elapsed (in microsecond)
   void sleep(uint64_t us);             //idle for us microseconds
private:
   uint32_t base_addr;
   uint32_t ctrl;                       // current state of ctrl_reg
};
```

TimerCore class implementation in timer_core.cpp

```
TimerCore::TimerCore(uint32_t core_base_addr) {
    base_addr = core_base_addr;
    ctrl = 0x01;
    io_write(base_addr, CTRL_REG, ctrl);  // enable the timer
}

TimerCore::~TimerCore() { }

void TimerCore::pause() {
    // reset enable bit to 0
    ctrl = ctrl & ~GO_FIELD;
    io_write(base_addr, CTRL_REG, ctrl);
}

void TimerCore::go() {
    // set enable bit to 1
    ctrl = ctrl | GO_FIELD;
    io_write(base_addr, CTRL_REG, ctrl);
}

void TimerCore::clear() {
    uint32_t wdata;

    // write clear_bit to generate a 1-clock pulse
    // clear bit does not affect ctrl
    wdata = ctrl | CLR_FIELD;
    io_write(base_addr, CTRL_REG, wdata);
}

uint64_t TimerCore::read_tick() {
    uint64_t upper, lower;
```

FPro utility routine declarations and macros in chu_init.h

```
// library
#include "chu_io_rw.h"
#include "chu_io_map.h"
#include "timer_core.h"
#include "uart_core.h"

// make uart visible by other code
extern UartCore uart;

// define timer and uart slots
#define TIMER_SLOT S0_SYS_TIMER  // slot 0
#define UART_SLOT  S1_UART1      // slot 1
```

FPro utility routine declarations and macros in chu_init.h

```
// timing functions
unsigned long now_us();
unsigned long now_ms();
void sleep_us(unsigned long int t);
void sleep_ms(unsigned long int t);

// define debug function
void debug_off();
void debug_on(const char *str, int n1, int n2);

#ifndef _DEBUG
#define debug(str, n1, n2) debug_off()
#endif

#ifdef _DEBUG
#define debug(str, n1, n2) debug_on((str), (n1), (n2))
#endif

// low-level bit-manipulation macros
#define bit_set(data, n) ((data) |= (1UL << (n)))
#define bit_clear(data, n) ((data) &= ~(1UL << (n)))
#define bit_toggle(data, n) ((data) ^= (1UL << (n)))
#define bit_read(data, n) (((data) >> (n)) & 0x01)
#define bit_write(data, n, bitvalue) \\
        (bitvalue ? bit_set(data, n) : bit_clear(data, n))
#define bit(n) (1UL << (n))

#endif
```

FPro utility routine implementation in chu_init.cpp

```
TimerCore _sys_timer(get_slot_addr(BRIDGE_BASE, TIMER_SLOT));
UartCore  uart(get_slot_addr(BRIDGE_BASE, UART_SLOT));

unsigned long now_us() {
   return ((unsigned long) _sys_timer.read_time());
}

unsigned long now_ms() {
   return ((unsigned long) _sys_timer.read_time() / 1000);
}

void sleep_us(unsigned long int t) {
   _sys_timer.sleep(uint64_t(t));
}

void sleep_ms(unsigned long int t) {
   _sys_timer.sleep(uint64_t(1000 * t));
}

void debug_on(const char *str, int n1, int n2) {
   uart.disp("debug: ");
   uart.disp(str);
   uart.disp(n1);
   uart.disp("(0x");
   uart.disp(n1, 16);
   uart.disp(") / ");
   uart.disp(n2);
   uart.disp("(0x");
   uart.disp(n2, 16);
   uart.disp(") \n\r");
}

void debug_off() {
}
```

Vanilla FPro test program in main_vanilla_test.cpp

```
#define _DEBUG

#include "chu_init.h"
#include "gpio_cores.h"

void timer_check(GpoCore *led_p) {
   int i;

   for (i = 0; i < 5; i++) {
      led_p->write(0xffff);
      sleep_ms(500);
      led_p->write(0x0000);
      sleep_ms(500);
      debug("timer check - (loop #)/now: ", i, now_ms());
   }
}

void led_check(GpoCore *led_p, int n) {
   int i;

   for (i = 0; i < n; i++) {
      led_p->write(1, i);
      sleep_ms(200);
      led_p->write(0, i);
      sleep_ms(200);
   }
}

void sw_check(GpoCore *led_p, GpiCore *sw_p) {
   int i, s;
```