RISC-V Lab

Ex6: Specification

Content

Project building blocks: Nexsys Video

- HDMI video in/out
- audio stereo in/out
- 100/1000M Ethernet
- USB 2 (uart, parallel/SPI, PS/2)
- sdcard, 8 LED, 8 DIP
- OLED display 128x32
- connectors for extension PCBs
  - 4x PMOD = 4*8 IOs (3.3V)
  - FMC LPC (lots of IOs)
- ... (see "Design Reference")

Project building blocks: HW

- Interfaces: I2C, SPI, PCM Highway, I2S (audio)
- MIDI (Keyboard, Sound equipment)
- IC: ADCs / DACs, H drivers, Irda ...
- HDMI camera (in lab: 1920x1080, 60 fps)
- RGB cameras & displays
- **modules from Aliexpress, ebay**
- historic: disk drives, PS2 keyboard & mouse, ...
- mechanics: modell servos (PWM), stepper/DC motors, ...

Project building blocks: IP cores

- OpenTitan: TL-UL peripherals
- Opencores.org: Interfaces (USB, Ethernet), Arithmetic units, 8/16bit CPUs, ...
- Pulp Platform: AXI & logarithmic interconnects, Peripherals, ...
- RISC-V CPUs: OpenHW, T-Head Semi (AliB), Chips Alliance (WD), (picorv32, FEMTORV32)
- LiteX (Python!), SpinalHDL(VexRiscV)
- Special: Spiral FFT, fpganes,
- SW: DOOM, ...

Make sure the **testbench of the IP works** *before* planing its use!

Project Ideas

- class A ("really interesting")
  - require HW / SW codesign
    - real time requirements => HW
    - high computational throughput => HW
    - high complexity, flexibility => SW
  - not possible with µC
  - (beat PC - difficult!)
- class B: use throughput of FPGA
- class C: build (smart, DMA) µC peripherals

Project Ideas: A

- **Real Time Ethernet - process Ethernet frames on the fly**
  e.g. Ethercat like network, man in the middle attack
- **Multi core system: standard CPUs or specialized cores**
  ray tracer, particle simulator, fractals, neural networks...
- **Game/demo: Graphics card (+ sound)**
  triangle shader (3D pipeline!), 2D: fill, line, circle, sprites
- **Real time video processing**
  object tracking, edge extraction,... frame or line buffered
- **Low latency audio processing**
  time (massively parallel FIR) or frequency domain (FFT)

Project Ideas: A

- **Software defined radio / Modulated data transmission**
  100MS/s DAC PCB / data via laser pointer
- **malloc() in parallel HW**
- **Gameboy Advanced*
  video & sound, sw loader, ...
- **Rotary display**
  string of 32 leds on custom PCB + motor + slip rings
- **Laser Beamer (very difficult mechanics!)**
  laser printer motors & mirrors (good) / stepper motors (bad)

Project Ideas: new A

- multi core real time processing (graphics card, audio, network?)
- (P4?) switch - build FMC card with e.g. 3x ETH
- minimal 3d pipeline (must know algorithms before)
- real time use of DDR3 (e.g. as video memory)
- integrate LARGE core (T-Head C910, Rocket, ...)
- SW defined multi phase DCDC converter
- RVLAB
  - switch to open source DDR3 controller
  - port to different FPGA/PCB (e.g. Tang Nano 20k)

Project Ideas: B

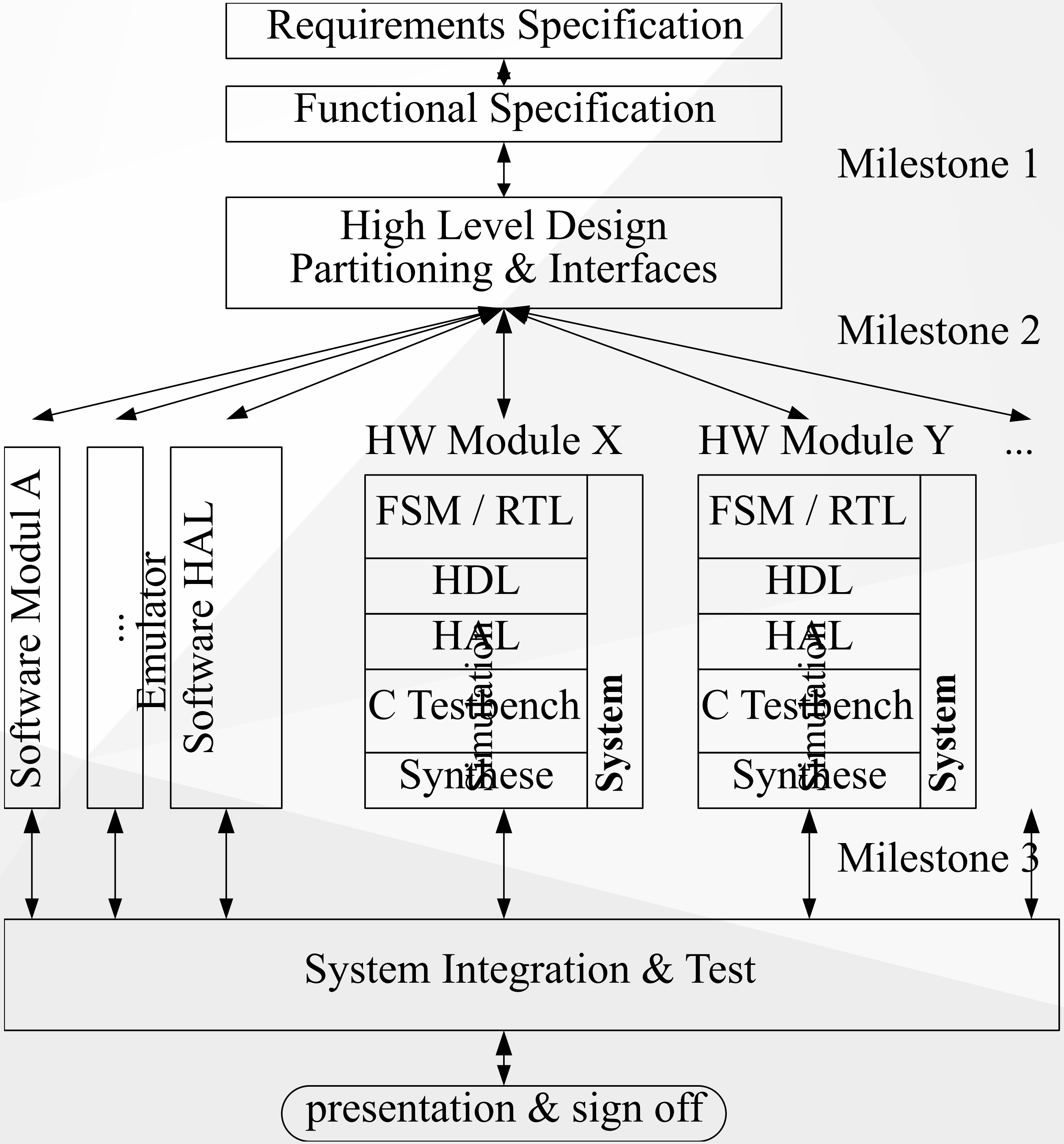
- Encryper / Decrypter (DES / AES / ChaChan/ TRIT) + IO (can be A)
- Logic analyzer/ mixed signal oscilloscope
- Multi axis robot control ("Spider" walking with 18 servos)

Project Ideas: C

- SD card defragger (mostly SW)
- (Stepper) motor control, force feedback joystick
- Theremin
- weather station with LED matrix display

Not all projects are sufficient for 4 persons !

Project Flow



Partitioning

Criteria

- functionality: manage complexity „divide and conquer“
- performance: latency and throughput
- timing, resource sharing, ...

Main issue: complexity => main principle: **Orthogonality**

- single, clearly defined task per module
- **independence** of all other modules

Test: How many parts need to be changed if functionality X is added or the environment changes ?

Interfaces

- simple, easy to understand (optimal: state less)
- hide implementation

Applied to HW design

- TL-UL is standardized and visible => use for communication between independent peripherals
- preference
  1. CPU<->peri or peri<->RAM (DMA)
  2. peri to peri over "register bus" (ex1.) or TL-UL
  3. proprietary connection between modules

- CPU is always master!
  At any time the CPU can set a module into a defined states (e.g.off)
- consistent register structure
  - across register bits, registers within a module and modules
  - sequence of the (bits, registers)
  - names (= semantics of the registers and bits)
  - right aligned, zero padded, ...
  - leave place for extensions (between register bits, registers and modules)

- internal module register readable for debug (e.g. state registers). May not be used during normal operation. Ban from normal HAL, if possible make them only visible in a debug mode.
- advanced (real SoC)
    - individual gated clk for every module
    - individual nreset for every module

C Traps & Pitfalls

```
// find the bugs (at least one error per paragraph):

y = x/*p;        /* p points at the divisor */

struct {
  int age; char *name;
} limits[] = {
  001, "baby",
  012, "teenager",
  100, "grandfather"
};

y = x<<4 + y ;    /* y = x*2^4 + y */

i = 0 ;
while (i < n)
  y[i++] = x[i] ;
```

C Traps & Pitfalls

```c
if (n<2)
  return
    longrec.date = x[0] ;
    longrec.time = x[1] ;


if (x = 0) // wrong
  if (0 == y)
    error();
  else {
    z = x / y;
  }
```

C Traps & Pitfalls

```c
#define abs(x) x>0?x:-x
y = abs(a)-1 // wrong
y = abs(a-b) // wrong

#define abs(x) (((x)>=0)?(x):-(x))
y = abs(x[i++]) // still wrong

#define assert(e) if(!(e)) assert_error(__FILE__,__LINE__)
if (x > 0 && y > 0)
  assert(x > y);
else
  assert(y > x);
```

from: Andrew Koenig: C Traps und Pitfalls, Addison-Wesley
The book is recommended reading !