VLSI System Design
Part VI : Advanced Topics

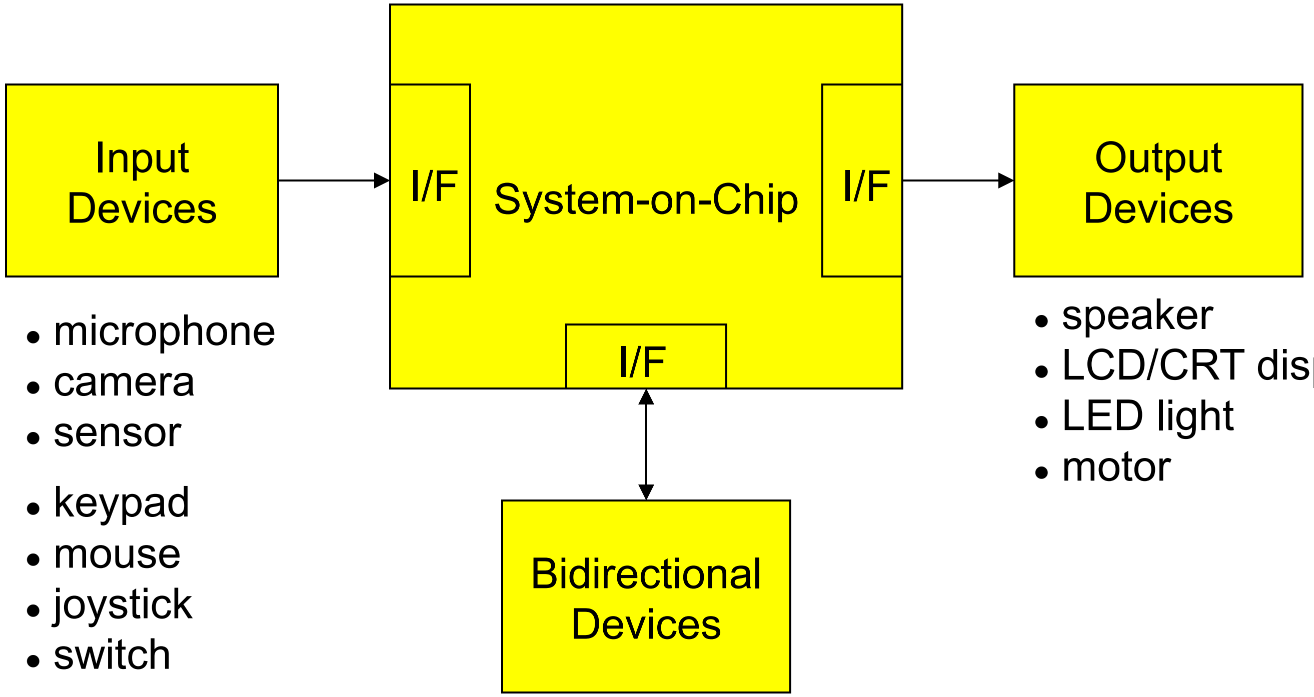
Lecturer : Tsuyoshi Isshiki

Dept. Information and Communications Engineering,

Tokyo Institute of Technology

isshiki@ict.e.titech.ac.jp

Today’s VLSI : System-on-Chip (SoC)



- microphone
- camera
- sensor
- keypad
- mouse
- joystick
- switch

- speaker
- LCD/CRT display
- LED light
- motor

- peripheral bus (IEEE1394, USB, RS232C, PCI, SCSI, AGP, ISA, ATA, …)
- storage (SRAM, DRAM, FLASH-ROM, disk drive)
- network (Modem, Ethernet, wireless)

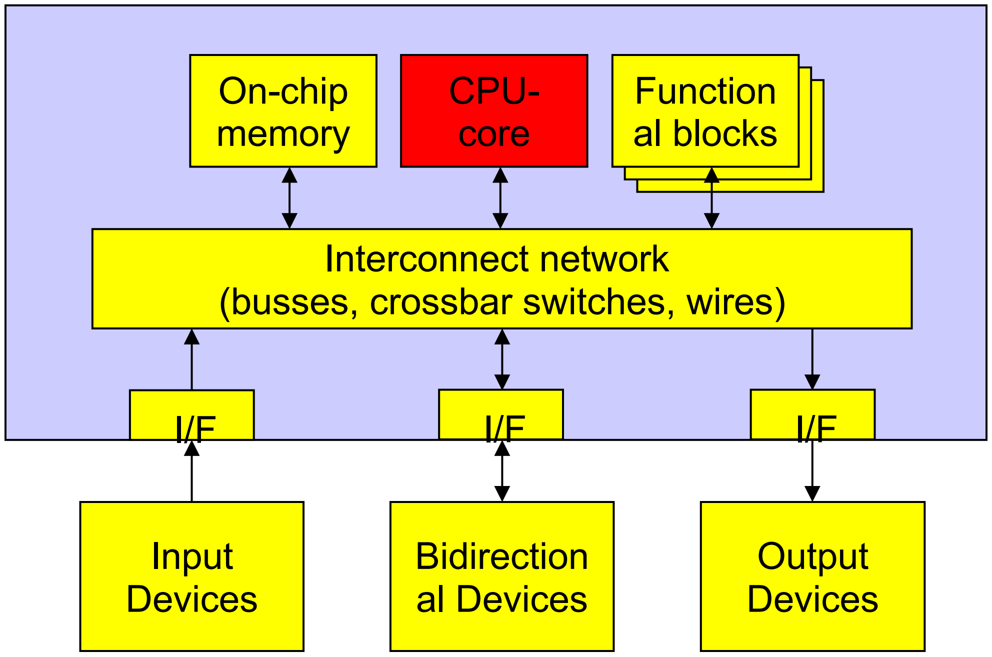
“Design Crisis”

- ➢ While semiconductor process technology is advancing very rapidly, design productivity is not catching up → productivity gap is widening (design crisis)
- ➢ Possible solutions to increase design productivity
  - High-level syntheis → describe at algorithm-level
    - limited to VLIW-type architecture which can execute single control-flow (instruction-level parallelism)
  - Design reuse → IP-based designs
    - a limited number of successful IPs (CPU-cores, bus systems)
  - Unified language from system specification to RTL description → consistent description environment → detect design errors at early design phase

CPU-Core Embedded System (1)

A) Advantages of embedding CPU-cores in VLSI

- Many high-performance low-power CPU cores available as IPs (good compilers and debuggers are also available for those CPUs)
- Implementing large portion of the system in software programs makes the system more flexible to design changes (upgrades, dug-fix)



CPU-Core Embedded System (2)

B) Design paradigm : *Hardware/Software Codesign*

- Design the hardware and software concurrently
  - Hardware : CPU-cores, peripheral devices, custom hardware (for time-critical processes)
  - Software : program for the CPU-core (for non-time-critical processes)
- Find any inconsistencies in the system specifications at the early design phase, and make any necessary changes to the design.
- *Language issues*
- Synthesis issues (HW/SW partitioning, Application-Specific Instruction Processor (ASIP) synthesis)
- Simulation environment (HW/SW cosimulation)

Language Issues

- In order to construct a systematic design environment for SoC, development of a powerful design description language is essential.
  - Capture both hardware and software designs
  - Easy to write, easy to read
  - Support different levels of abstraction (system specification → algorithm description → RTL)
- Many C-based description languages have been proposed.

C-Based Design Languages (1)

A) Why C-language ?

- Very popular
- Variety of good compilers and debuggers
- A lot of designs are first written in C (design reuse)
- Easy to debug (less # of codes, sequential execution is easier to understand than concurrent execution in HDLs)
- Fast simulation
- Easy to describe both hardware and software

C-Based Design Languages (2)

B) What does C-language lack?

- Hardware behavior description :
  - Concurrency, pipelining
  - Synchronization
  - Detail timing
- Hardware structure description:
  - Modules, module instantiation, interconnection
  - Hierachical structure
  - Bit-size precise data types
- ➢ These features are built in either as language extensions or class libraries in the new C-based languages.

C-Based Design Languages (3)

C) Examples

- Consortium organized (many companies involved in the development of design environment)
  - SystemC
  - SpecC
- Dependent to specific design environment :
  - BDL(NEC), Bach-C(Sharp), Handel-C(Celoxica)
  - HardwareC(Stanford) : *can describe and synthesis multiple control-flow designs*
- HDL extension :
  - SystemVerilog

SystemC Example (1)

clocked process

```
class ProcessA : sc_sync {
public:
    const sc_signal <int> & in;
    const sc_signal <bool> & in_ready;
    sc_signal <int> & out;
    sc_signal <bool> & out_ready;
   ProcessA(const char *name, sc_clock_edge &CLK,
        const sc_signal<int> & a,
        sc_signal<bool> & a_rdy,
        sc_signal<int> & b, sc_signal<bool> &
        b_rdy) : sc_sync (name, CLK), in(a),
        in_ready(a_rdy), out(b),
        out_ready(b_rdy){ }
    void entry();
};
```

input port

output port

class constructor

behavior description

SystemC Example (2)

```
void ProcessA::entry(){
    while(true){
        if(out_ready.read()== true){
                wait(); // wait 1 clock cycle
                out_ready.write(false);
        }
        wait_until(in_ready.delayed()== true);
        int v = in.read();
        out.write(v + out.read());
        out_ready.write(true);
    }
};
```

event for other processes

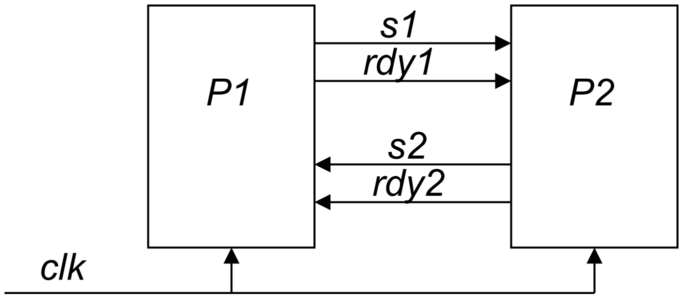
event trigger (synchronization)

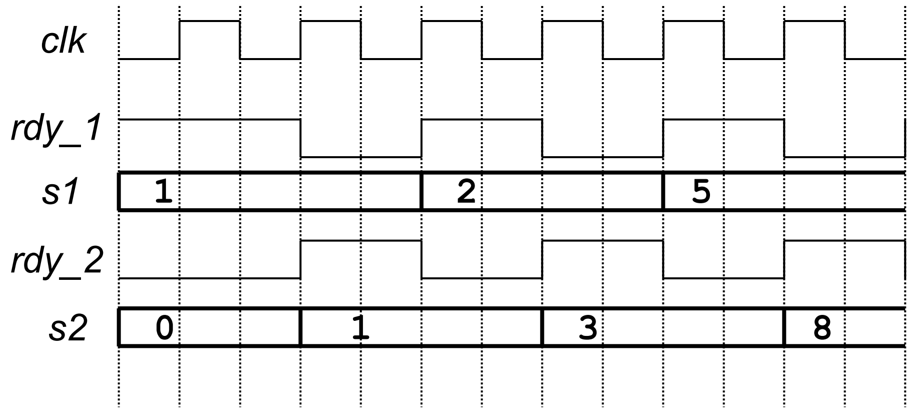
internal behavior

All of "entry()" functions are multi-threaded in the program and executed concurrently.

SystemC Example (3)

```
                          int sc_main(){
                              sc_signal<int> s1("sig1"),s2("sig2");
signal declaration          { sc_signal<bool> rdy1("rdy1"),rdy2("rdy2");
                              sc_clock clk("CLK",20,0.5,0.0);
process declaration         { ProcessA p1("P1",clk.pos(),s1,rdy1,s2,rdy2);
                              ProcessA p2("P2",clk.pos(),s2,rdy2,s1,rdy1);
signal initialization       { s1.write(1); s2.write(0);
                              rdy1.write(true); rdy2.write(false);
simulation start            { sc_start(10000); //run for 10000 time units
                              return 0;
                          };
```

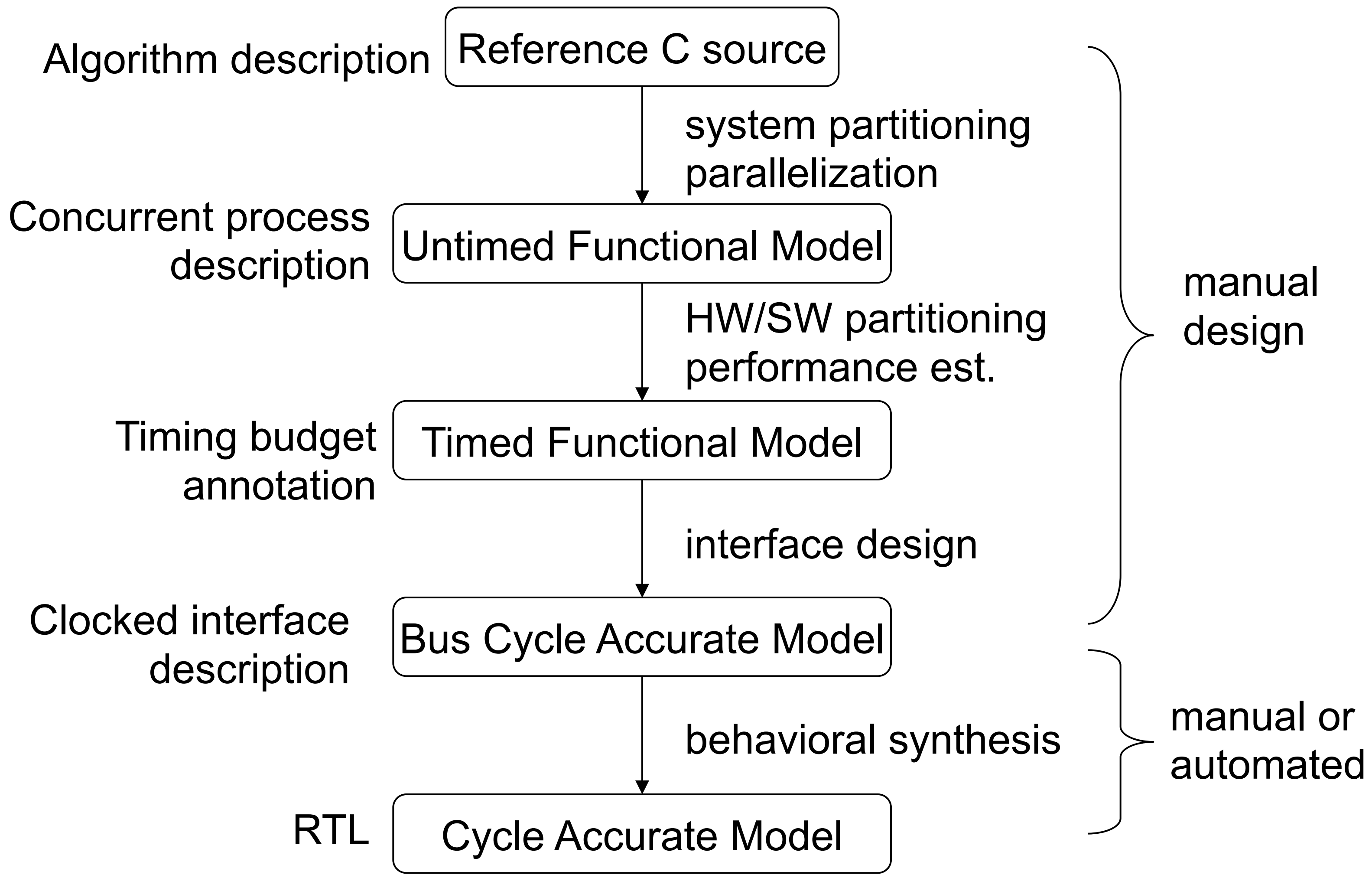




SystemC Design Flow

- Untimed Functional Model (UTF)
  - Collection of concurrent processes communicating through buffered channels
- Timed Functional Model (TF)
  - Timing information (computation latency) added to UTF to estimate timing behavior
- Bus-Cycle Accurate Model (BCA)
  - Communications modeled in RTL
  - Internal computations still use TF
- Cycle Accurate Model (CA)
  - Functionally equivalent to RTL

SystemC Design Flow



C-Based Design Languages (4)

D) Current and future directions

- Main focus of C-based languages is how to model concurrent hardware behavior on sequential language (such as C).
  - SystemC uses multithreaded execution for each process.
- Final goal is to develop a unified language which can be used at all levels of abstraction
  - System specification
  - Algorithm description
  - RTL structural description

High-Level Synthesis and C-Based Design Methodology

- For many C-based design methodologies, RTL description is still written by hand
  - RTL description is written in C-based language, and is “translated” (not “synthesized”) into HDL description for logic synthesis
  - *Not many designers are still convinced that high-level synthesis is mature enough to use.*
- Some C-based design methodologies, however, utilizes high-level synthesis aggressively (BDL, Bach-C)
  - Each independent process is synthesized into customized VLIW processor
  - Partitioning the design into multiple processes (each implemented with VLIW processors), and implementing interconnections of processes is still mainly done by hand.
  - Already applied to a number of real LSIs.