C++ Object-Oriented RTL Modeling for System-Level Synthesis/Verification on the C2RTL Framework

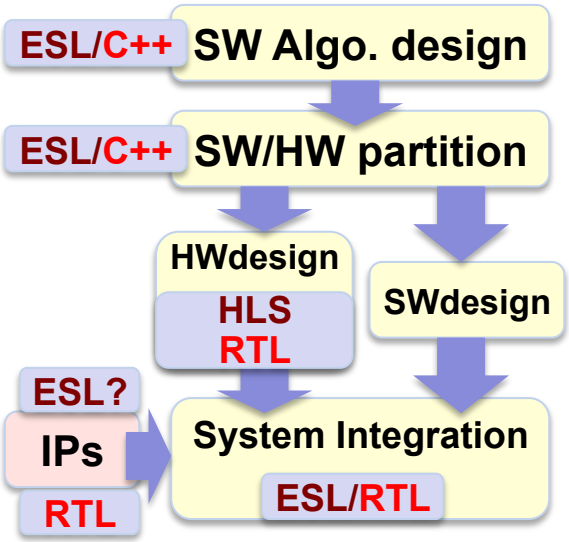
Tsuyoshi Isshiki

Global Scientific Information and Computing Center
Dept. of Communications and Computer Engineering
Tokyo Institute of Technology

MPSoC '17
July 6th, 2017

Design Challenges in Complex SoCs

- **Ever-growing design complexity, shorter time-to-market**
  - SW complexity, custom-HW complexity, large # of third-party IPs
- **HLS/ESL : IP-level RTL synthesis, SW development virtual platform**
  - System integration of synthesized IPs are still done on the ***RTL model***
  - Not all third-party IPs come with ESL model . . .
  - *"ESL Hand-off : Fact or EDA Fiction?"* (DAC '08 panel)
- **System integration : enormous verification cost**
  - Critical design faults may only be caught during system-level RTL (cycle-accurate) simulations



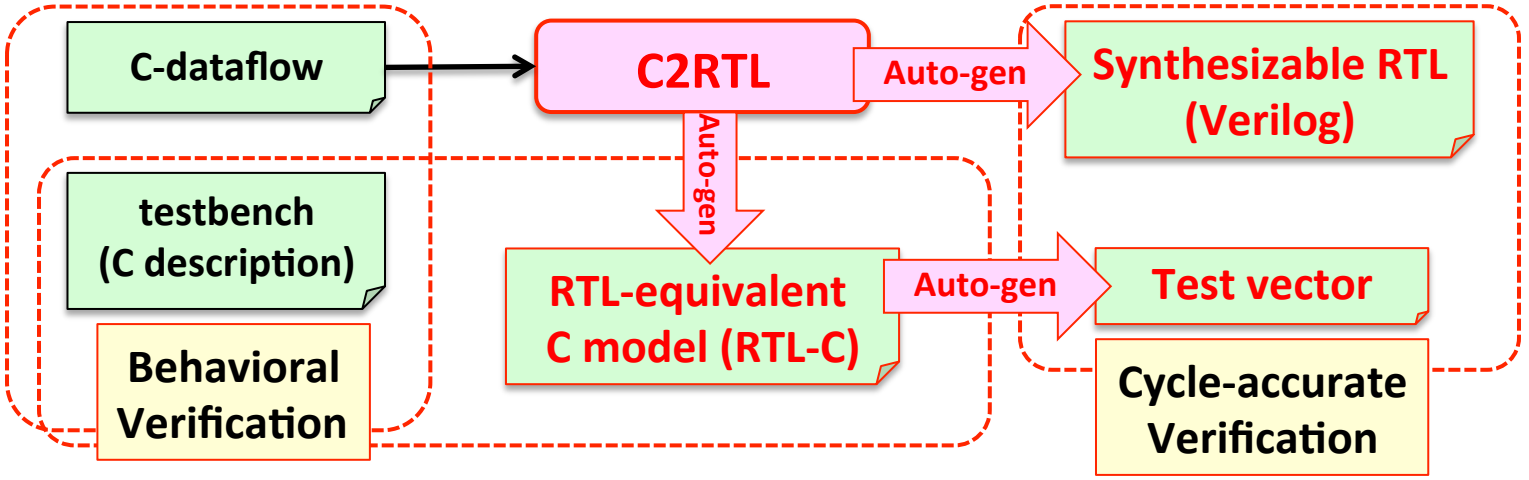
System-level RTL verification is still the significant bottleneck in today's complex SoC designs

C2RTL Design Framework [MPSoC '15/'16]

- **C-dataflow model : directly describe RTL behavior in C**
  - HW attributes (bit-width/register/memory) with *GCC-attributes*
  - No language extensions, no built-in classes : *easy & intuitive coding*
  - Single-cycle behavior of the total system : *the only design constraint*

  → *C-dataflow model = RTL behavior reference model & RTL itself !!*

  → *JPEG Encoder ASIP: area/throughput competitive with commercial IPs*
- **C2RTL outputs : RTL-C (cycle-accurate C), RTL (Verilog, testbench)**
  - Fast RTL verification on RTL-C using the same original C testbench
  - RTL test vector generation (for comprehensive RTL testing)
  - RTL debugging using C-debugger!! (ex. VisualStudio)

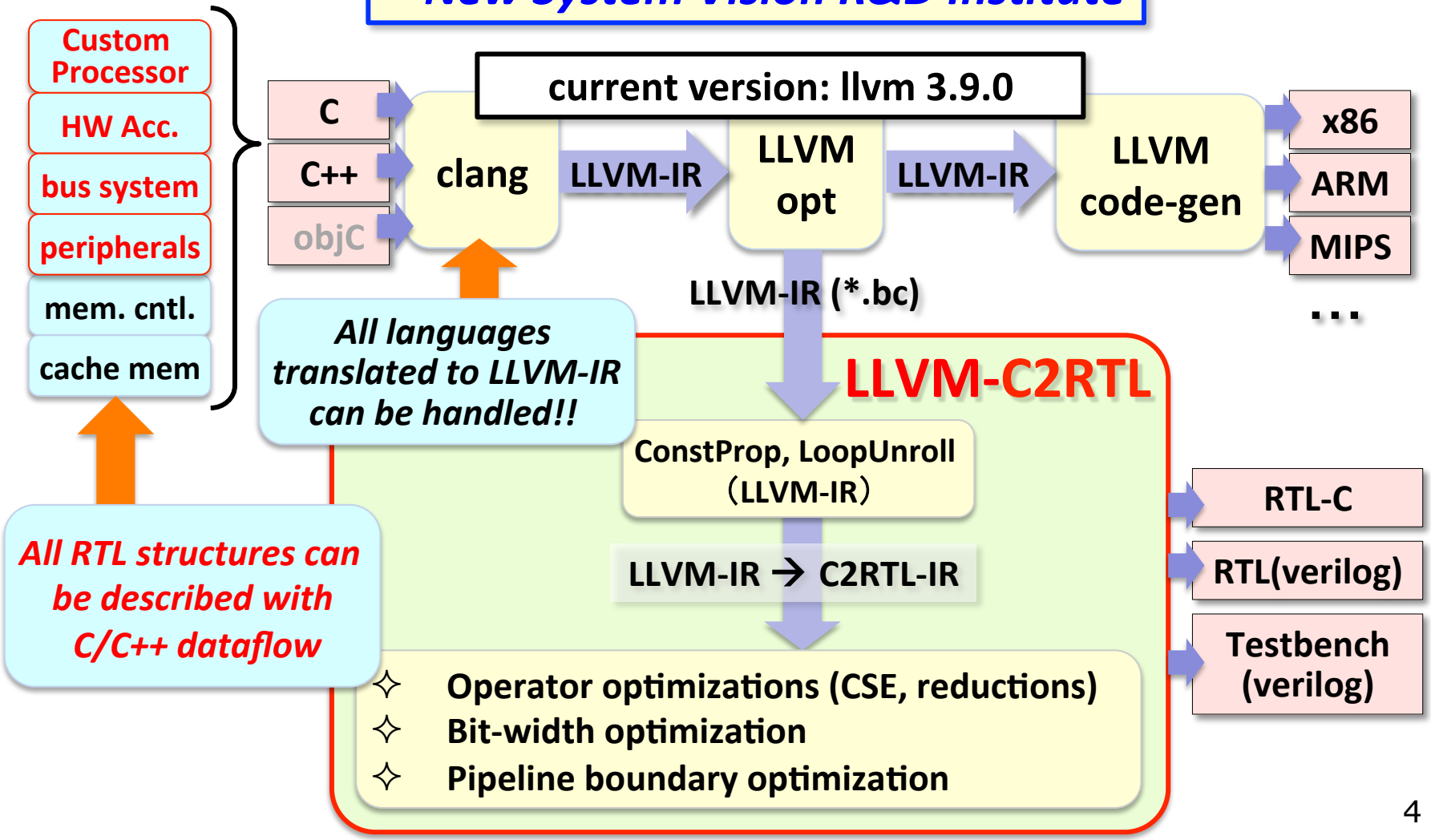


New System Vision
Research and Development Institute

LLVM-based C2RTL Framework

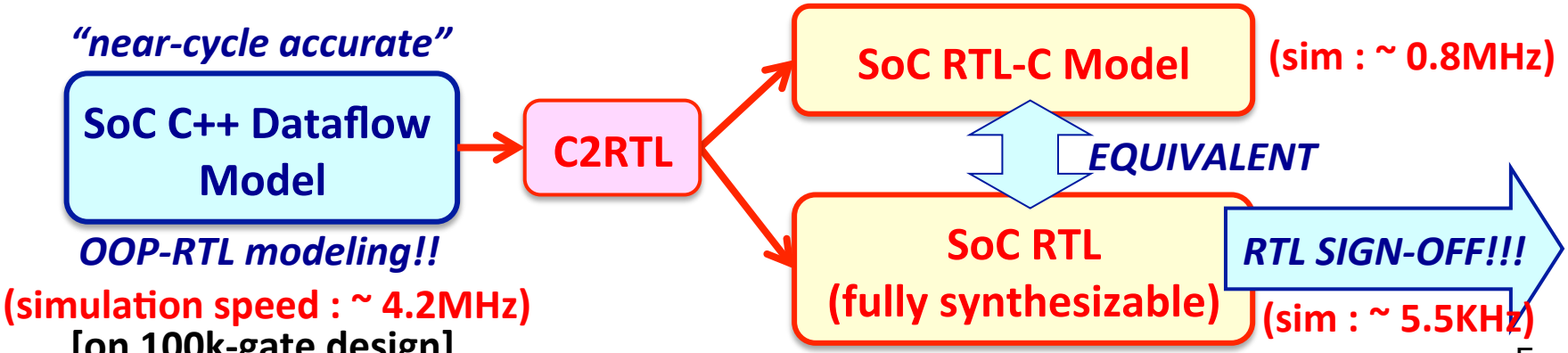
(To be distributed by *NSV Foundation)

*** New System Vision R&D Institute**

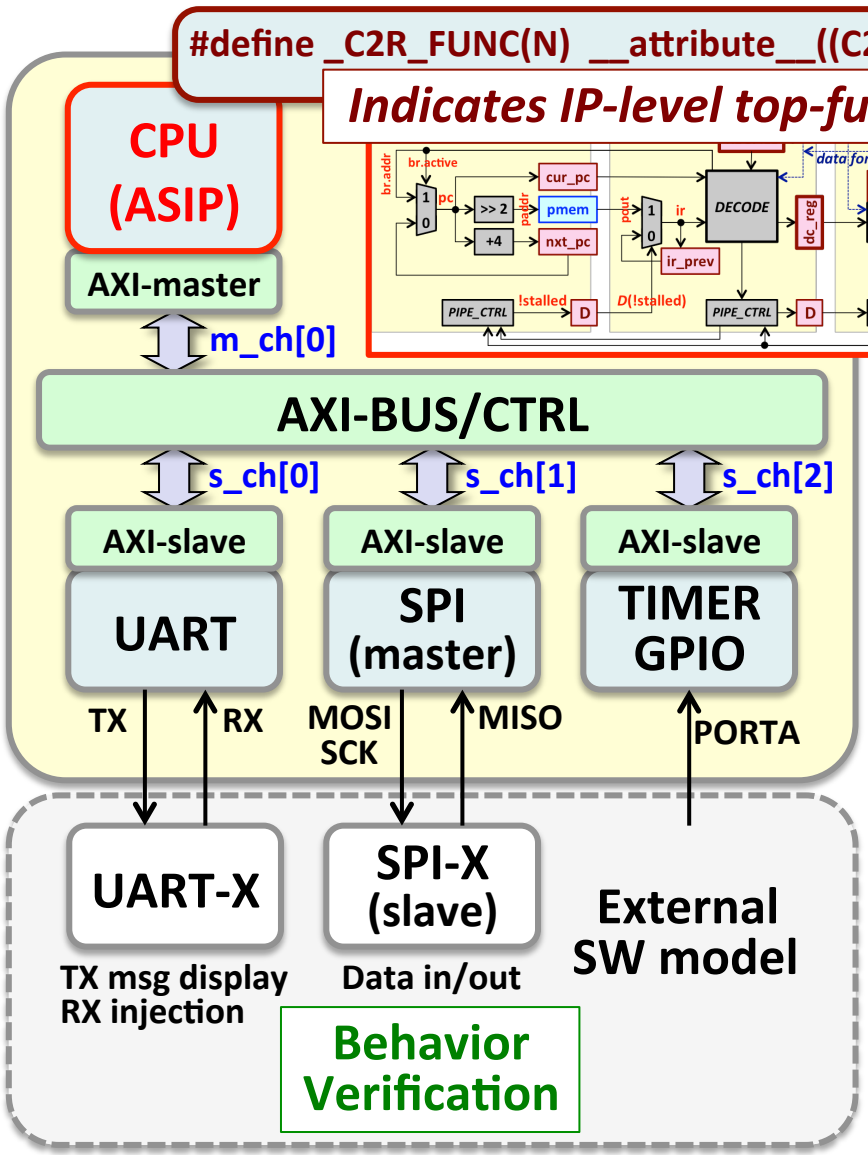


New Features on LLVM-C2RTL

- **C++ support : *object-oriented RTL modeling***
  - Class methods, inheritance, virtual functions
  - Template classes, template functions
  - → Greatly facilitates design reuse and platform optimization
- **C++ system-level integration**
  - Component (IP) level : pipelined-RTL from C++ dataflow model
    - ❖CPUs, peripherals, interconnects (busses, NoCs)
    - →RTL (verilog), RTL-C (same APIs as original C++ model)
  - System level : component instantiations and connections
    - →Subsystem-level, SoC-level, system-level + environment model



A Simple SoC Modeling



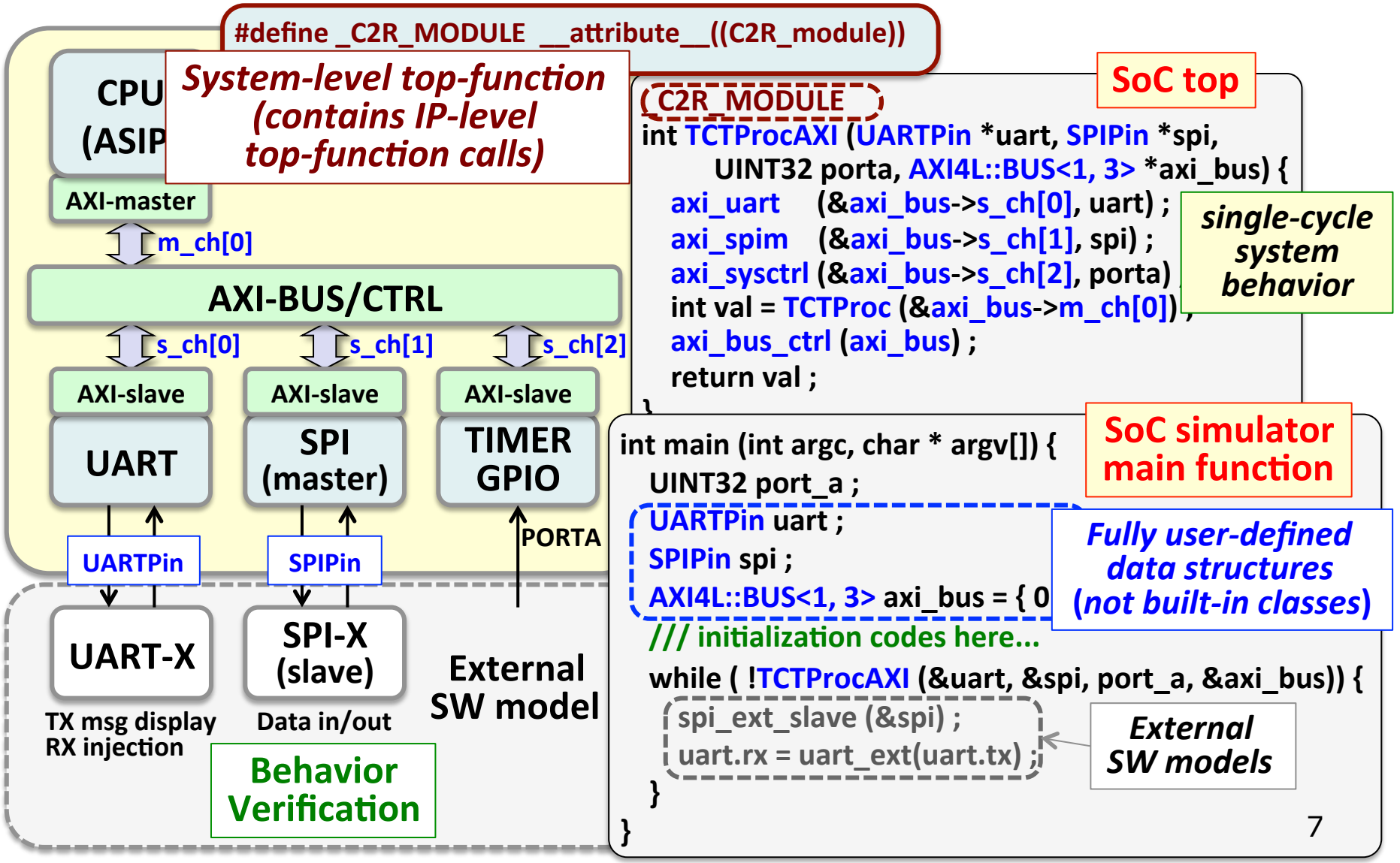

```
_C2R_FUNC(4)
int TCTProc (AXI4L::CH *axi) {
    fetch ();
    decode ();
    execute (axi);
    writeback ();
    return (cpu.halted == 1);
}
```

CPU top (4 pipe-stages)

single-cycle behavior model

- **SoC component**
  - **CPU** (AXI-master) : 4-stage RISC
  - **UART** (AXI-slave)
  - **SPI-master** (AXI-slave)
  - **TIMER/GPIO** (AXI-slave)
  - **AXI-BUS/CTRL** (AXI interconnect)
- **External SW models**
  - UART-X : msg display/injection
  - SPI-slave : data input/output (*verification against SW application outputs*)

SoC-RTL Model + SoC Simulation Model



AXI BUS Resource Model

AXI Channels

```
struct AXI4L {
  /////////// AXI-channels ///////////
  struct CH {
    struct ADDR {   /// read-addr, write-addr
      struct MA { UINT32 addr ; BIT valid ; } m ;
      struct SL   { BIT ready ; } s ;
    }  raddr , waddr ;
    struct RDAT {   /// read-data
      struct MA { BIT ready ; } m ;
      struct SL   { UINT32 data ; UINT2 resp ; BIT valid ; } s ;
    }  rdat ;
    struct WDAT {  /// write-data
      struct MA { UINT32 data ; UINT4 strobe ; BIT valid ; } m ;
      struct SL   { BIT ready ; } s ;
    }  wdat ;
    struct WRES {   /// write-resp
      struct MA { BIT ready ; } m ;
      struct SL   { UINT2 resp ; BIT valid ; } s ;
    }  wres ;
    UINT32 intr ;   /// interrupts (not AXI)
  } _T(direct_signal) ; /// all signals are unlatched wires
  /// continued···
```

```
#define _BW(N)  __attribute__((C2R_bit_width(N)))
#define _T(N)     __attribute__((C2R_type(N)))
typedef uint8_t   BIT _BW(1), UINT2 _BW(2), UINT4 _BW(4);
```

Bit-width attributes, signal-type attribute

```
///struct AXI4L {
 template <int MC, int SC> struct BUS
 {  CH  m_ch[MC], s_ch[SC];  };
```

AXI BUS (MC masters, SC slaves)

AXI-BUS_CTRL top

```
_C2R_FUNC(1)
void axi_bus_ctrl (AXI4L::BUS<1, 3> *axi_bus) {
   static AXI4L::CTRL<1, 3> axi_ctrl;
   axi_ctrl.connectChannel (axi_bus);
}
```

Class methods (not shown here) are provided for setting/clearing channel signals

AXI BUS Controller (Interconnect)

AXI bus-controller

```
///struct AXI4L {
struct MasterStatus { UINT8 slaveID; BIT active; } ;
template <int MC, int SC> struct CTRL {
   MasterStatus RStat[MC]  _T(state), WStat[MC] _T(s
   void connectChannels (BUS<MC, SC> *bus) {
      bus -> resetAllChannelSinks () ;
      bus -> connectInterrupts () ;
      for (i = 0; i < MC; ++i) { connectRCh (i, bus); }
      for (i = 0; i < MC; ++i) { connectWCh (i, bus); }
};
```

AXI-BUS_CTRL top

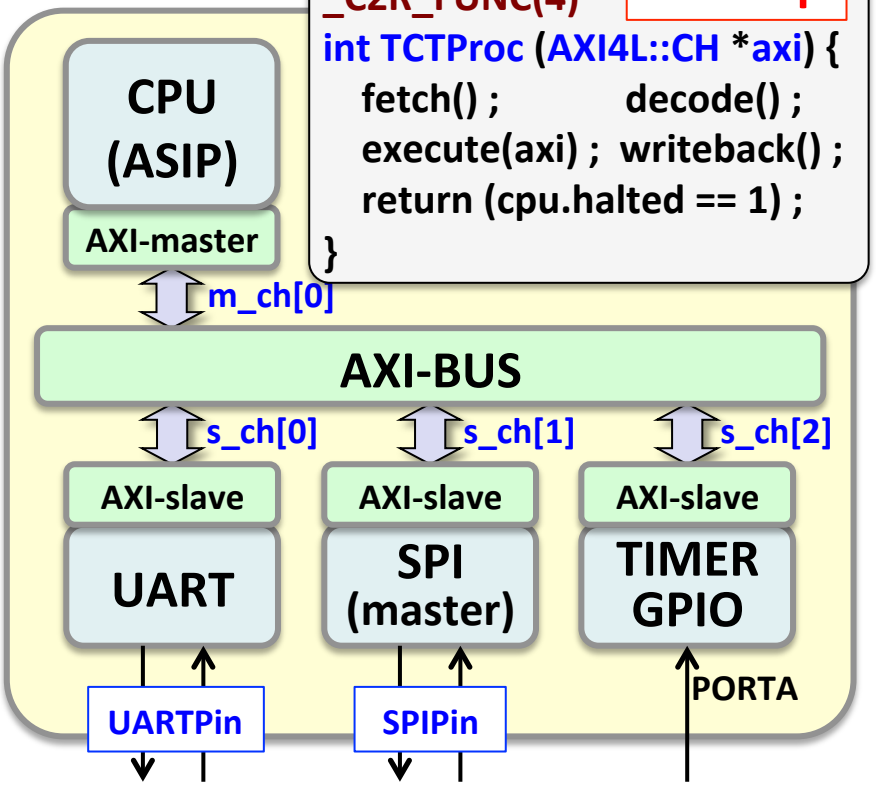
```
_C2R_FUNC(1)
void axi_bus_ctrl (AXI4L::BUS<1, 3> *axi_bus) {
   static AXI4L::CTRL<1, 3> axi_ctrl;
   axi_ctrl.connectChannels (axi_bus);
}
```

```
void connectRCh (int midx, BUS<MC, SC> *bus) {
   int sidx;
   if (updateRStat (midx, bus, &sidx))
      connectRCh (midx, sidx, bus);
} /// no arbiter yet…, works only for MC = 1
```

```
BIT updateRStat(int midx, BUS<MC, SC> *bus, int *sID) {
   *sID = RStat[midx].slaveID;  /// Rstat[midx] : master channel status (registers)
   BIT active = RStat[midx].active;      /// active → wire
   CH &mc = bus->m_ch[midx];
   if (!active && mc.raddr.m.valid) {   /// activate master[midx] on current cycle
      *sID = decodeAddr (mc.raddr.m.addr) ;
      RStat[midx].set(*sID, 1) ;  /// update Rstat[midx] registers
      active = 1 ;
   }
   else if (active && mc.rdat.m.ready && bus->RDATvalid(*sID)
      RStat[midx].reset();   /// release master[midx] on NEXT cycle
   return active; }
```

Expresses full details of RTL behavior with high abstraction C++ statements

IP-Level Top Functions



CPU top

```
_C2R_FUNC(4)
int TCTProc (AXI4L::CH *axi) {
   fetch() ;          decode() ;
   execute(axi) ;  writeback() ;
   return (cpu.halted == 1) ;
}
```

```
axi_uart    (&axi_bus->s_ch[0], uart) ;
axi_spim    (&axi_bus->s_ch[1], spi) ;
axi_sysctrl (&axi_bus->s_ch[2], porta) ;
int val = TCTProc (&axi_bus->m_ch[0]) ;
axi_bus_ctrl (axi_bus) ;
```

Each IP is separately RTL-synthesized

AXI-UART top

```
_C2R_FUNC(1)
void axi_uart (AXI4L::CH *axi, UARTPin *uart) {
   static AXI_UART axi_uart ;
   axi_uart.uart.set_pin (uart) ;
   axi_uart.fsm (axi) ;
}
```

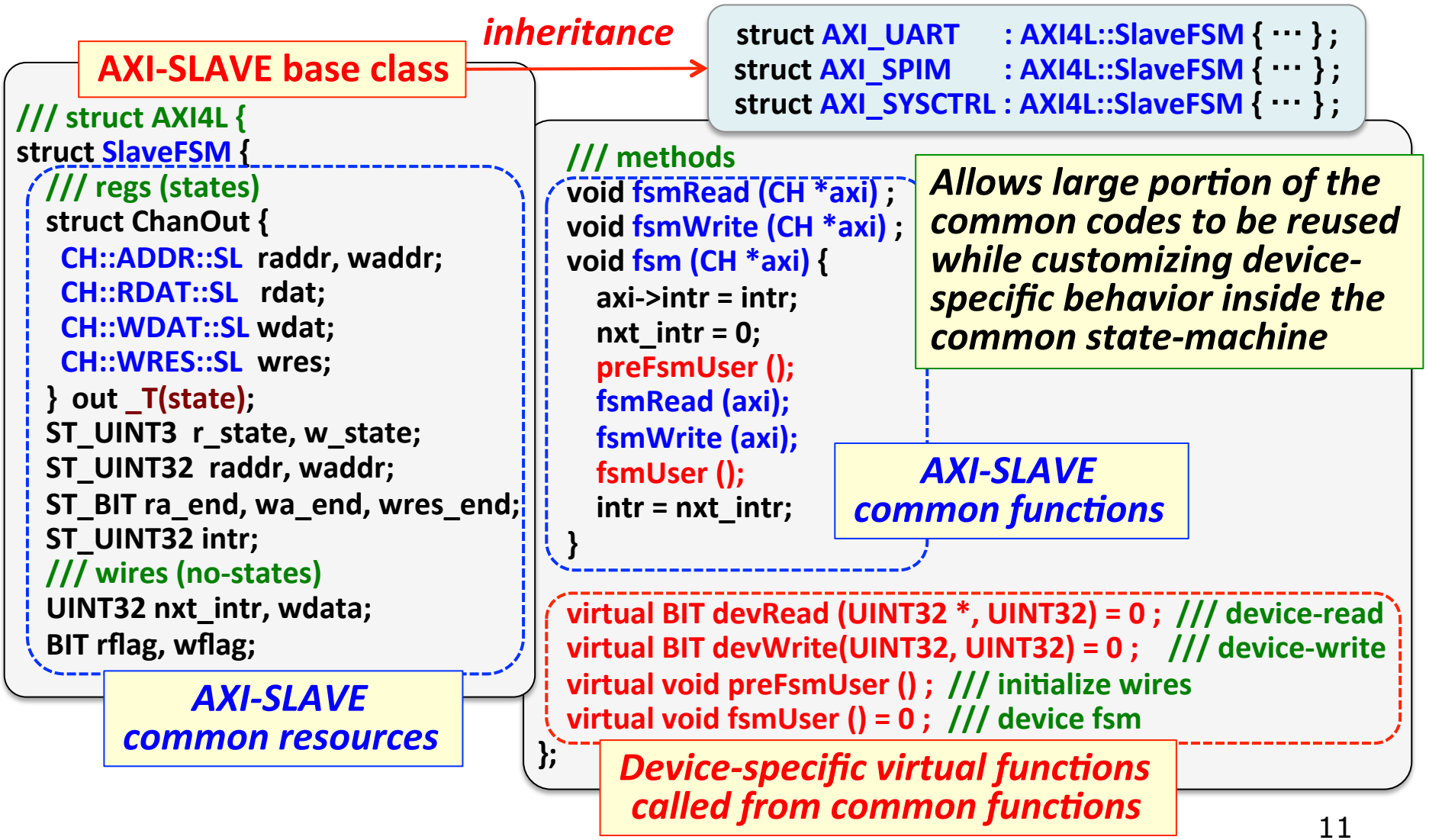
AXI-SPI top

```
_C2R_FUNC(1)
void axi_spim (AXI4L::CH *axi, SPIPin *spi) {
   static AXI_SPIM axi_spim ;
   axi_spim.spim.set_pin (spi) ;
   axi_spim.fsm (axi) ;
}
```

AXI-SYSCTRL top (TIMER/GPIO)

```
_C2R_FUNC(1)
void axi_sysctrl (AXI4L::CH *axi, UINT32 porta) {
   static AXI_SYSCTRL axi_sysctrl ;
   axi_sysctrl.sysctrl.set_port_a (porta) ;
   axi_sysctrl.fsm (axi) ;
}
```

Object-Oriented RTL Design Reuse!!



Device-Specific Reads Inside FSM

```
void AXI4L::SlaveFSM::fsmRead (CH *axi) {
  /// drive AXI-Slave-Output signals
  axi->setSLRead (out.raddr, out.rdat);
  switch ( r_state ) {
  case R_Init :
    if (axi->raddr.m.valid) {
      raddr = axi->raddr.m.addr;
      out.raddr.ready = 1;
      ra_end = 0; r_state = R_Addr;
    } break;
  case R_Addr : {
    out.raddr.ready = 0;
    if (axi->raddr.m.valid == 0) { ra_end = 1; }
    UINT32 d = out.rdat.data;
    if (out.rdat.valid || devRead(&d, raddr)){
      out.rdat.set(d, RSP_OK, 1); /// drive rdat
      if (axi->rdat.m.ready) { r_state = R_End; }
    } break; }
  case R_End : /// make sure master released
    out.rdat.reset(); /// release rdat
    if (ra_end || axi->raddr.m.valid == 0) {
      nxt_intr |= 1; r_state = R_Init; /// raise interrupt
    } break;
  } }
```

Common Read Channel FSM

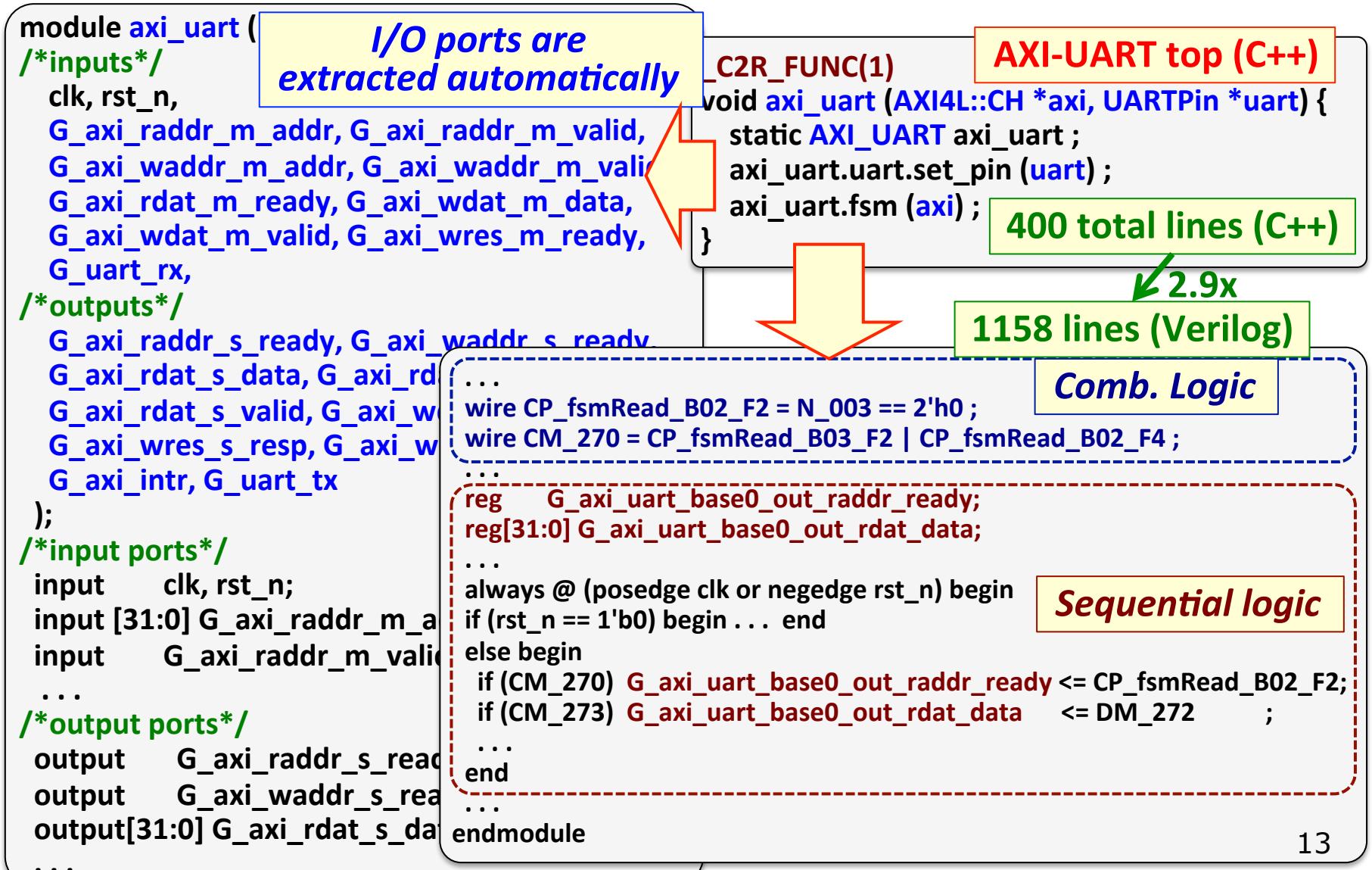
```
BIT AXI_UART::devRead (UINT32 *d, UINT32 a) {
  if (a & 0x7) { *d = uart.read_status(); return 1; }
  else if (uart.activated && uart.rx.buf.not_empty) {
    *d = uart.rx.buf.buf[uart.rx.buf.rp];
    rflag = 1; return 1;
  }
  else { return 0; }
}
```

```
BIT AXI_SPIM::devRead (UINT32 *d, UINT32 a) {
  if (a & 0x6) { *d = spim.read_status(); return 1; }
  else if (spim.activated) {
    *d = spim.do_rx (IO_ReadData, (a & 1));
    rflag = 1; return !spim.tx_rx.rx_stalled;
  }
  else { return 0; }
}
```

```
BIT AXI_SYSCTRL::devRead (UINT32 *d, UINT32 a) {
  *d = sysctrl.reg[(a >> 2) & 0xf];
  return 1;
}
```

Device-specific read operations

Generated RTL(Verilog) Model (AXI-UART)



Generated RTL-C (Cycle-Accurate) Model

Identical API as the original C++ Top Function

Simulation state container (reg/memories)

```
void axi_uart_RTL_Core (AXI4L::CH* axi, UARTPin* uart_pin, ST_axi_uart *G)
{ //// input wrapper : C++ struct member inputs → RTL-signals
  U32B W_G_axi_raddr_m_addr  = (*axi).raddr.m.addr ;
  U32B W_G_axi_raddr_m_valid = (*axi).raddr.m.valid ;
  U32B W_G_axi_waddr_m_addr  = (*axi).waddr.m.addr
  U32B W_G_axi_waddr_m_valid = (*axi).waddr.m.valid
  U32B W_G_axi_rdat_m_ready  = (*axi).rdat.m.ready
. . .
  //// combinational logic :
. . .
  U32B CP_fsmRead_B02_F2    = (N_003 == 0x0);
  U32B CM_270       = (CP_fsmRead_B03_F2 | CP_fsmRead_B02_F4);
. . .
  //// sequential logic :
. . .
  if (CM_270)  G->axi_uart.base0.out.raddr.ready = CP_fsmRead_B02_F2;
  if (CM_273)  G->axi_uart.base0.out.rdat.data  = DM_272        ;
. . .
  //// output wrapper : C++ struct member outputs ← RTL-signals
  (*axi).raddr.s.ready = W_G_axi_raddr_s_ready;
  (*axi).waddr.s.ready = W_G_axi_waddr_s_ready;
. . .
}
```

AXI-UART top (C++)

```
_C2R_FUNC(1)
void axi_uart (AXI4L::CH *axi, UARTPin *uart) {
   static AXI_UART axi_uart ;
   axi_uart.uart.set_pin (uart_pin) ;
   axi_uart.fsm (axi) ;
}
```

External SW models can connect to these RTL-C (cycle-accurate) models

Generated SoC-Top RTL(Verilog) Model

```
module TCTProcAXI  ( clk, rst_n, . . .  ) ;
 /* ports, local wires */ . . . .

 axi_uart M0 (
 /*inputs*/
  .clk(clk), .rst_n(rst_n),
  .G_axi_raddr_m_addr   (G_axi_bus_s_ch_0_radd
  .G_axi_raddr_m_valid  (G_axi_bus_s_ch_0_radd
  .G_axi_waddr_m_addr (G_axi_bus_s_ch_0_wadd
  .G_axi_waddr_m_valid (G_axi_bus_s_ch_0_wadd
  .G_axi_rdat_m_ready   (G_axi_bus_s_ch_0_rdat_m_ready ),
  . . . .
 /*outputs*/
  .G_axi_raddr_s_ready   (G_axi_bus_s_ch_0_raddr_s_ready),
  .G_axi_waddr_s_ready  (G_axi_bus_s_ch_0_waddr_s_ready),
  . . . .
 );
 axi_spim      M1 ( . . . . . );
 axi_sysctrl   M2 ( . . . . . );
 TCTProc       M3 ( . . . . . );
 axi_bus_ctrl  M4 ( . . . . . );

endmodule
```

IP-module instantiation

SoC top

```
_C2R_MODULE
int TCTProcAXI (UARTPin *uart, SPIPin *spi,
      UINT32 porta, AXI4L::BUS<1, 3> *axi_bus) {
   axi_uart    (&axi_bus->s_ch[0], uart) ;
   axi_spim    (&axi_bus->s_ch[1], spi) ;
   axi_sysctrl (&axi_bus->s_ch[2], porta) ;
   int val = TCTProc (&axi_bus->m_ch[0]) ;
   axi_bus_ctrl (axi_bus) ;
   return val ;
}
```

9 lines (C++)

35.6x

320 lines (Verilog)

C++/Verilog Model Sizes & Simulation Speed

IP-block	# C++ lines (*.cpp, *.h)	# RTL-C lines (*.cpp, *.h)	# Verilog lines (w/o testbench)	gates
TCTProc (AXI-master)	1,595	3,882	4,127	78,771 gates
UART (AXI-slave)	402	949	1,158	4,845 gates
SPI (AXI-slave)	166	887	852	3,207 gates
SYSCTRL (AXI-slave)	78	666	539	6,977 gates
AXI-BUS/CTRL	165	802	359	1,451 gates
SoC top	9	516	320	0 gates
TOTAL	2,415	7,702	7,355	95,251 gates

Simulation length : 4,216,354 cycles (C++), 4,148,919 cycles (RTL)		
Simulation model	Simulation time	Simulation speed
C++ model (original)	0.981 sec	4,298,016 cycles/sec
RTL-C (no vector dump)	5.227 sec	793,748 cycles/sec
RTL-C (with vector dump)	15.199 sec	272,973 cycles/sec
Verilog (icarus-verilog)	751.517 sec	5,520 cycles/sec

Summary

- **LLVM-C2RTL C++ support : *object-oriented RTL modeling***
  - **Methods, inheritance, virtual functions, templates**
  - **→ Greatly facilitates design reuse and platform optimization**
  - **→ Planned to be distributed through *the NSV FOUNDATION***
- **C++ system-level integration**
  - **Component (IP) level : pipelined-RTL from C++ dataflow model**
  - **System level : component instantiations and connections**
  - **→ *Verification model / RTL model from a single C++ source!!***
- **C++ system-level verification**
  - **C++ dataflow model : near-cycle-accurate (no pipeline modeling)**
  - **RTL-C model : cycle-accurate (RTL test vector generation), 5x slower than C++ dataflow model, 140x faster than Verilog**
  - **RTL (Verilog) : fully synthesizable (IP-level modules, SoC top module)**

Thank You for Your Attention!

Tsuyoshi Isshiki

isshiki@ict.e.titech.ac.jp

Global Scientific Information and Computing Center

Dept. Communications and Computer Engineering

Tokyo Institute of Technology