Lab 4

Platform Architect (I)

v2 TA: 謝明倫
v1 TA: 陳宥融

Objectives – the following 2 w:

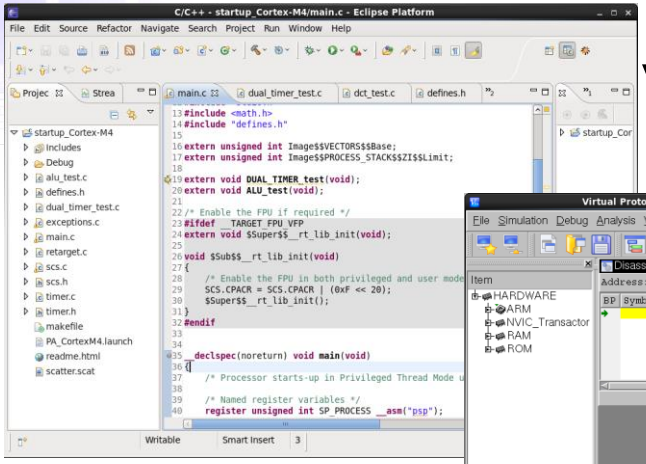
- Doing hardware-software co-simulation under ESL environment
- Be familiar with Platform Architect
  - SystemC transaction-level system build up
  - Packing HDL module (with AHB wrapper)
  - SystemC & RTL HDL co-simulation

Objectives – today:

- SystemC transaction-level system build up
  - What is Platform Architect ?
  - Build up our system
  - Transaction-level system model software simulation / debug

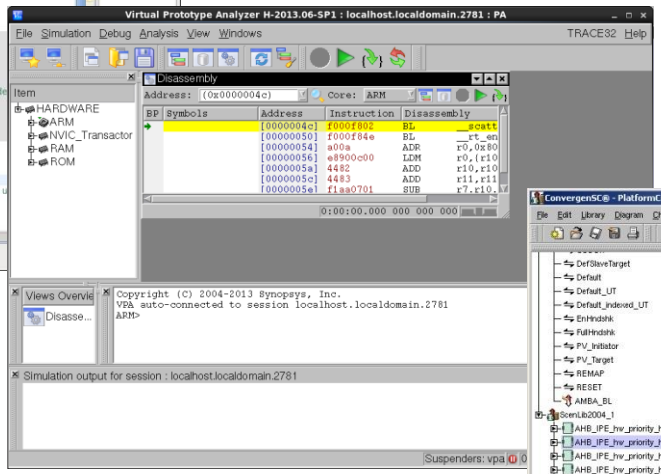
Platform Architect consists of

DS-5
(SystemC IDE)



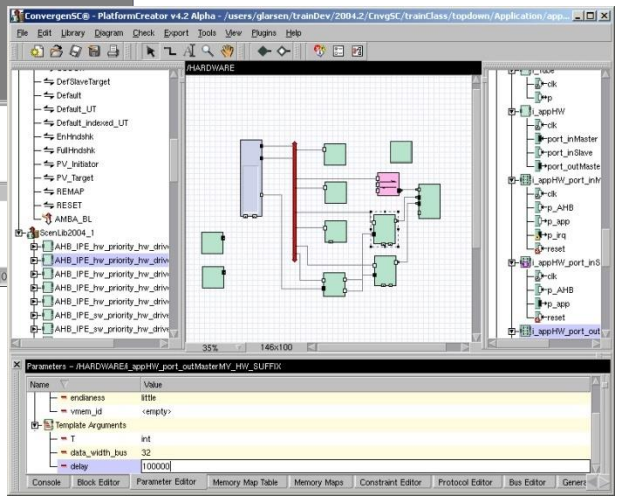
Simulation and Debugging

Virtual Prototype Analyzer
(Analysis API)



Analysis

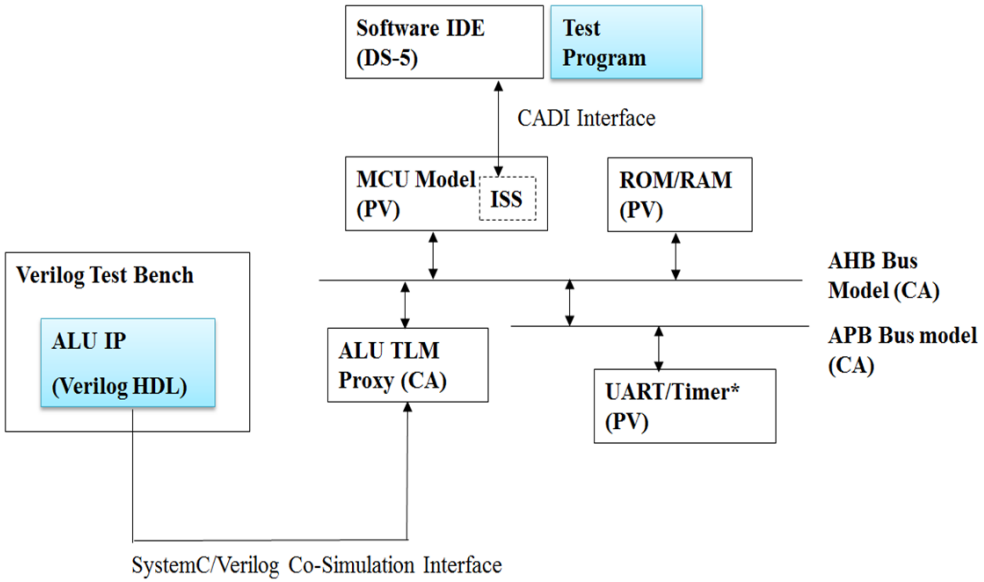
System Designer
(Platform Creater)



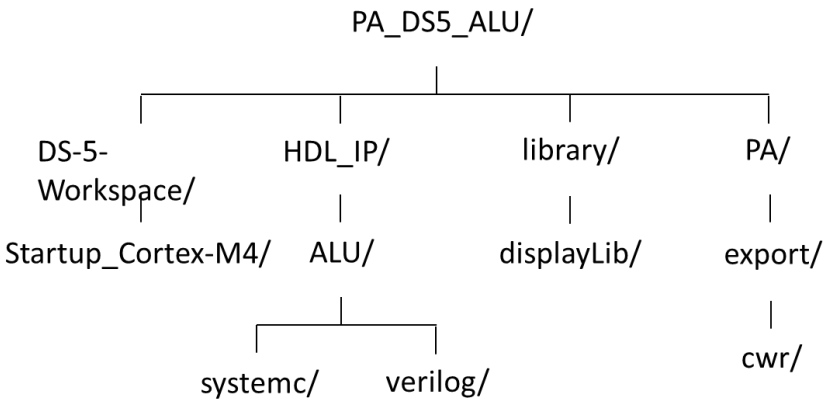
Platform Assembly and Configuration

Top view

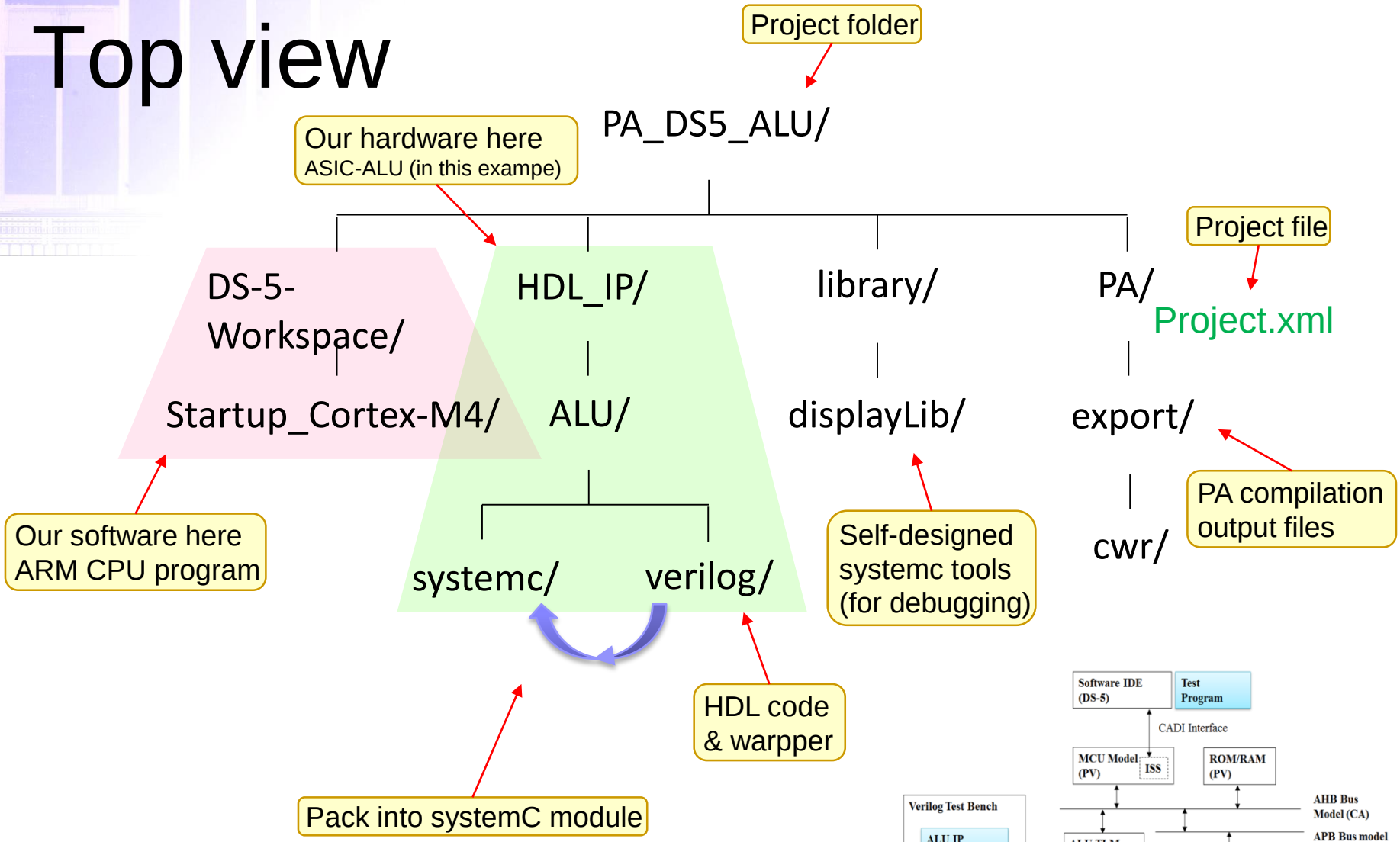
- Target architecture



- Files



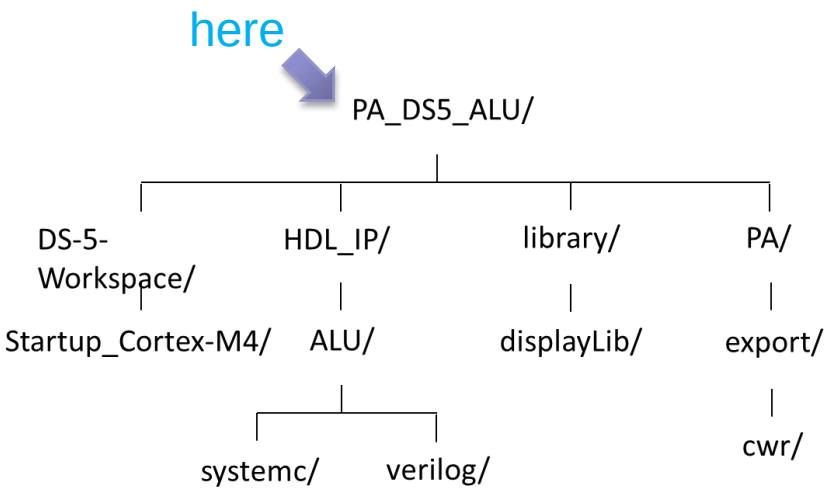
Top view


Project folder
PA_DS5_ALU/
Our hardware here
ASIC-ALU (in this exampe)
Project file
DS-5-
Workspace/
HDL_IP/
library/
PA/
Project.xml
Startup_Cortex-M4/
ALU/
displayLib/
export/
Our software here
ARM CPU program
Self-designed
systemc tools
(for debugging)
PA compilation
output files
cwr/
systemc/
verilog/
HDL code
& warpper
Pack into systemC module
Software IDE
(DS-5)
Test
Program
CADI Interface
MCU Model
(PV)
ISS
ROM/RAM
(PV)
AHB Bus
Model (CA)
Verilog Test Bench
ALU IP
(Verilog HDL)
ALU TLM
Proxy (CA)
APB Bus model
(CA)
UART/Timer*
(PV)
SystemC/Verilog Co-Simulation Interface

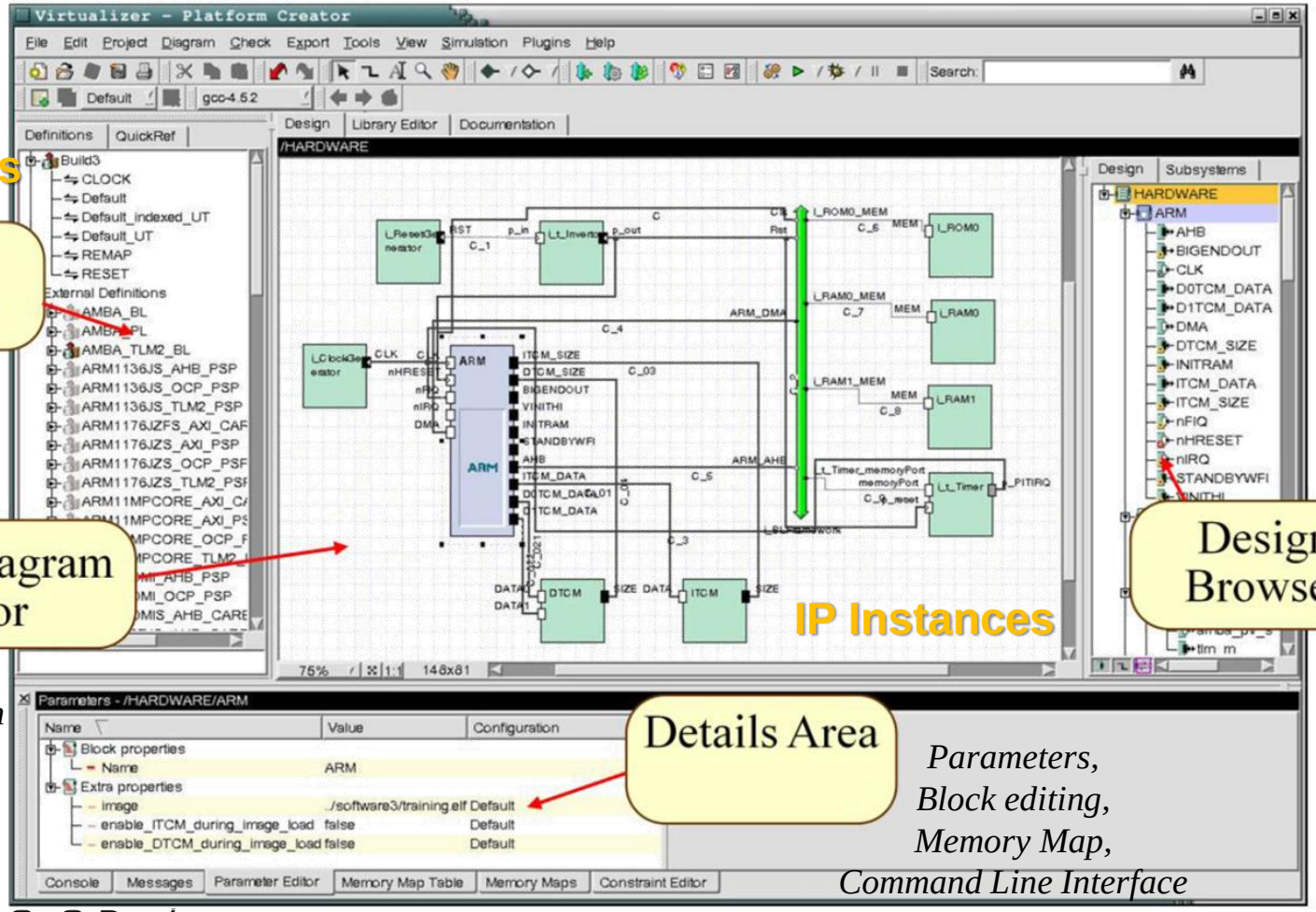
Using Platform Architect

- source pa_setup.csh

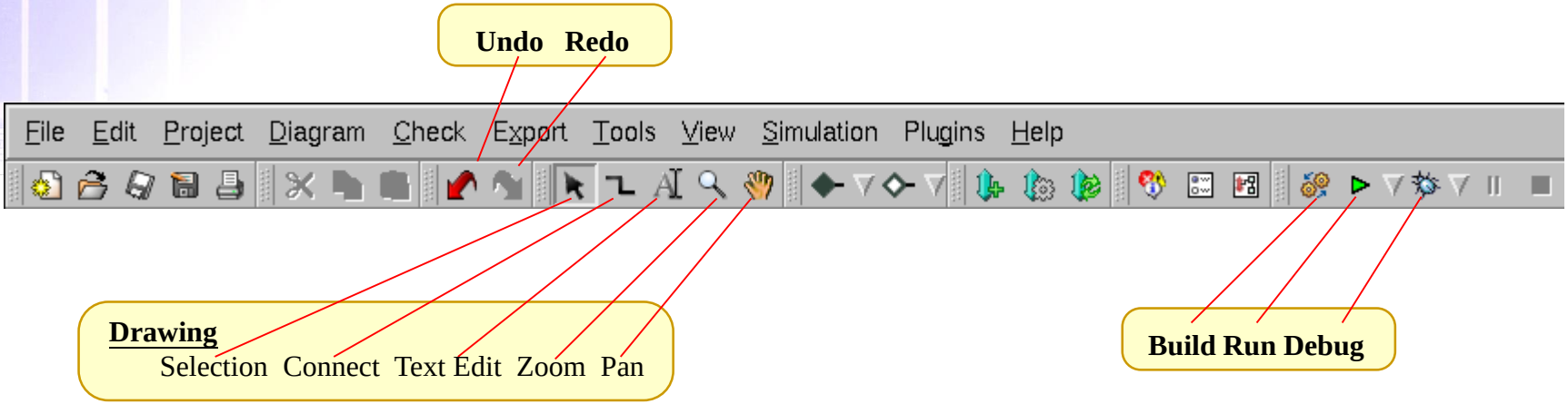
- source run.csh ← *To turn on tools for the project*
  - To start Platform Creator:
    - % pct &
  - To start VPA
    - % vpa &

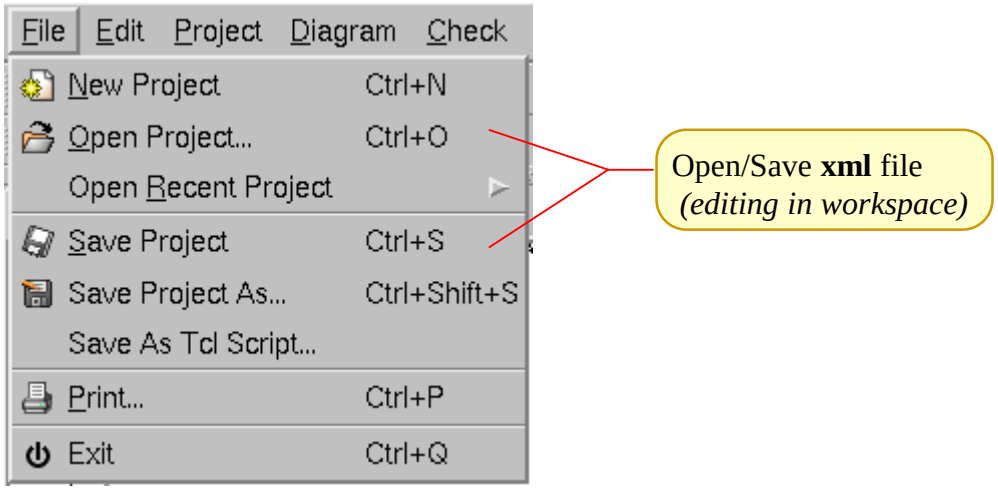


PA – main environment: Platform Creator

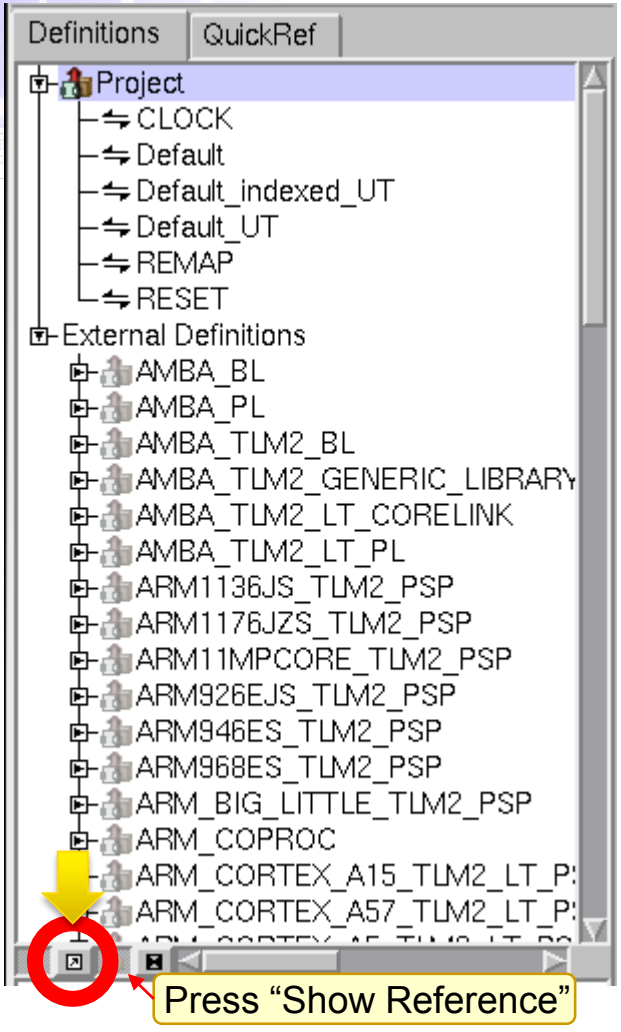


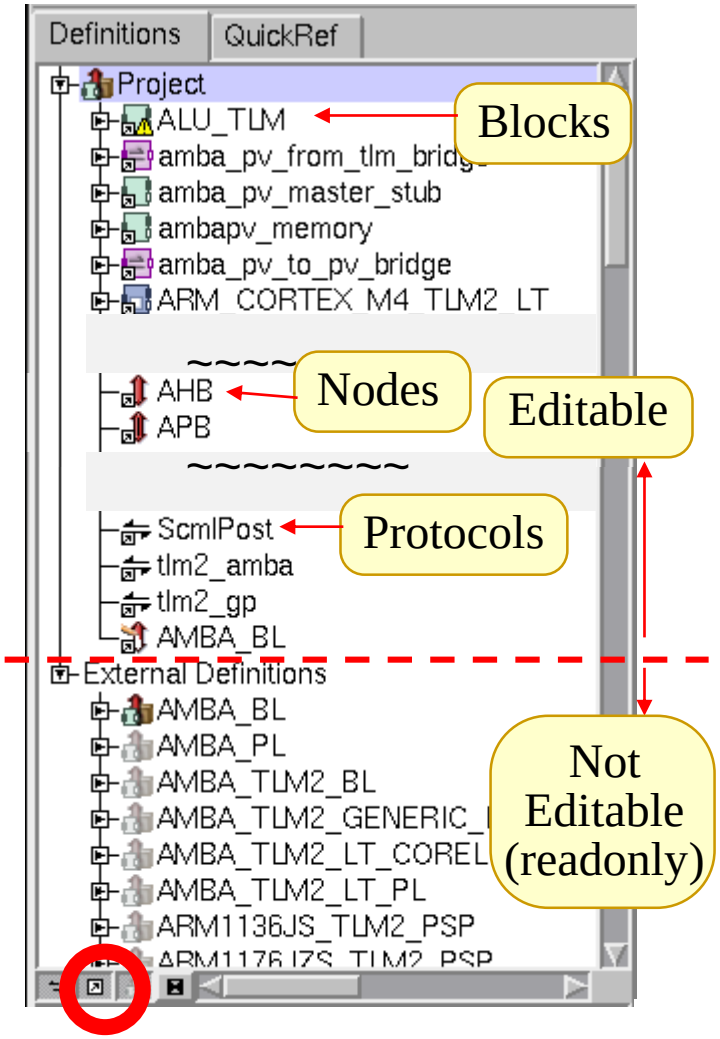
Menus and the toolbar





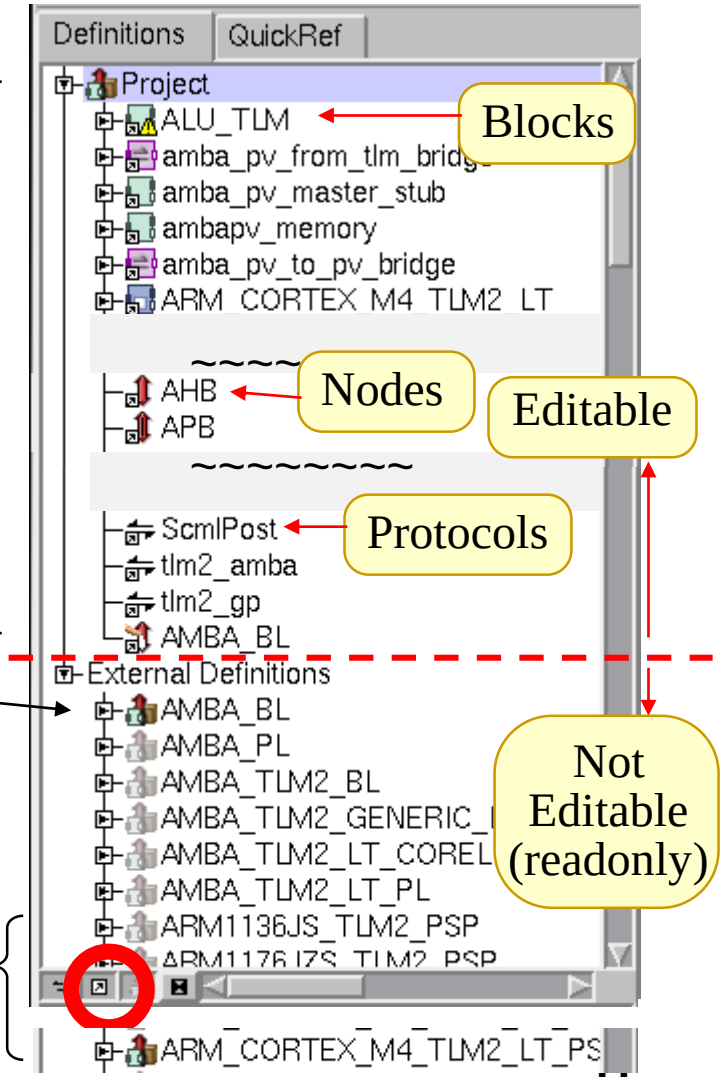
A closer look at model libraries





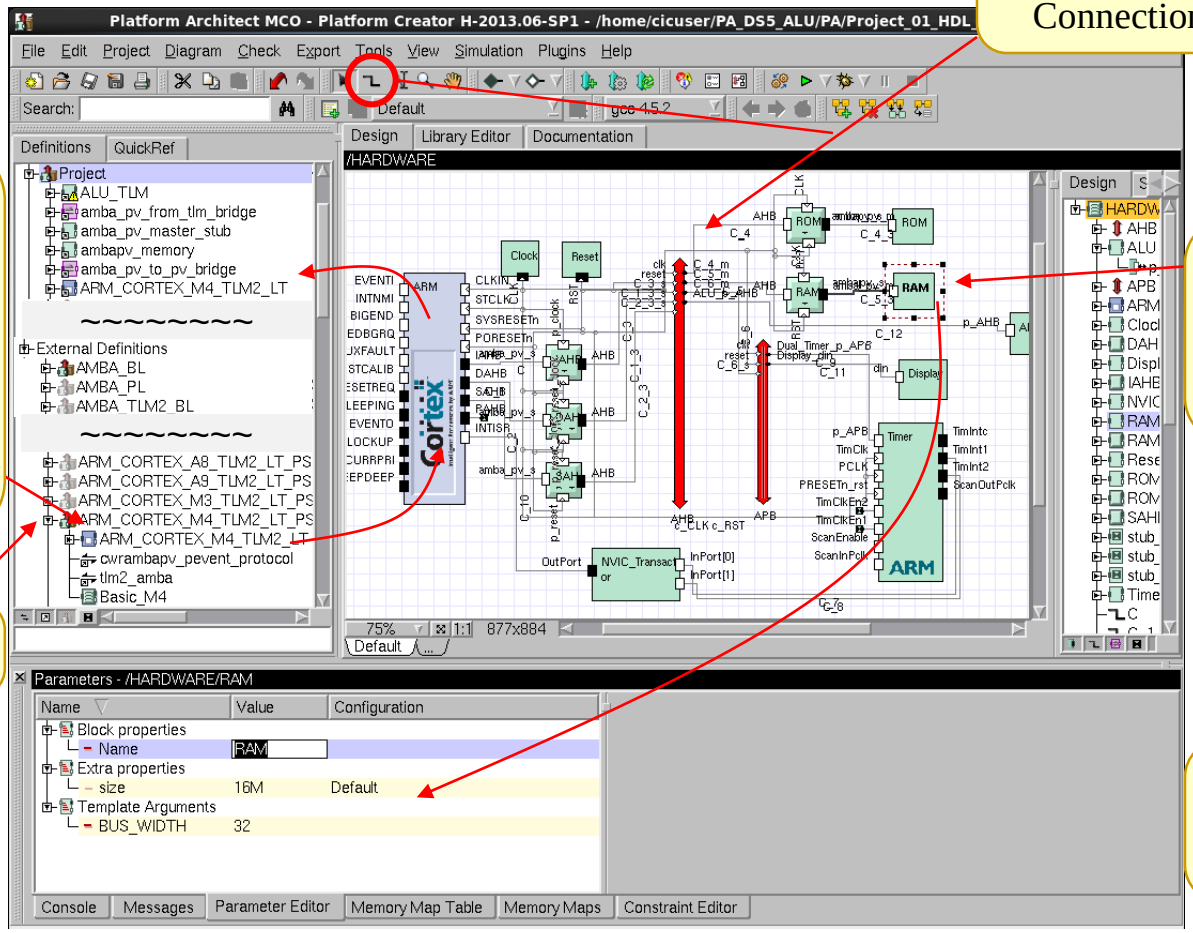
A closer look at model libraries

- System Library
  - Your system workspace, the system under construction
  - Displays the system blocks, nodes, and protocols of your design
- Currently Open IP Libraries
  - AMBA Bus Library
    - Nodes
    - Bridges
    - Protocols
  - Processor Library



Typical usage - drag and drop

3. **Connect** elements with Connection Tool



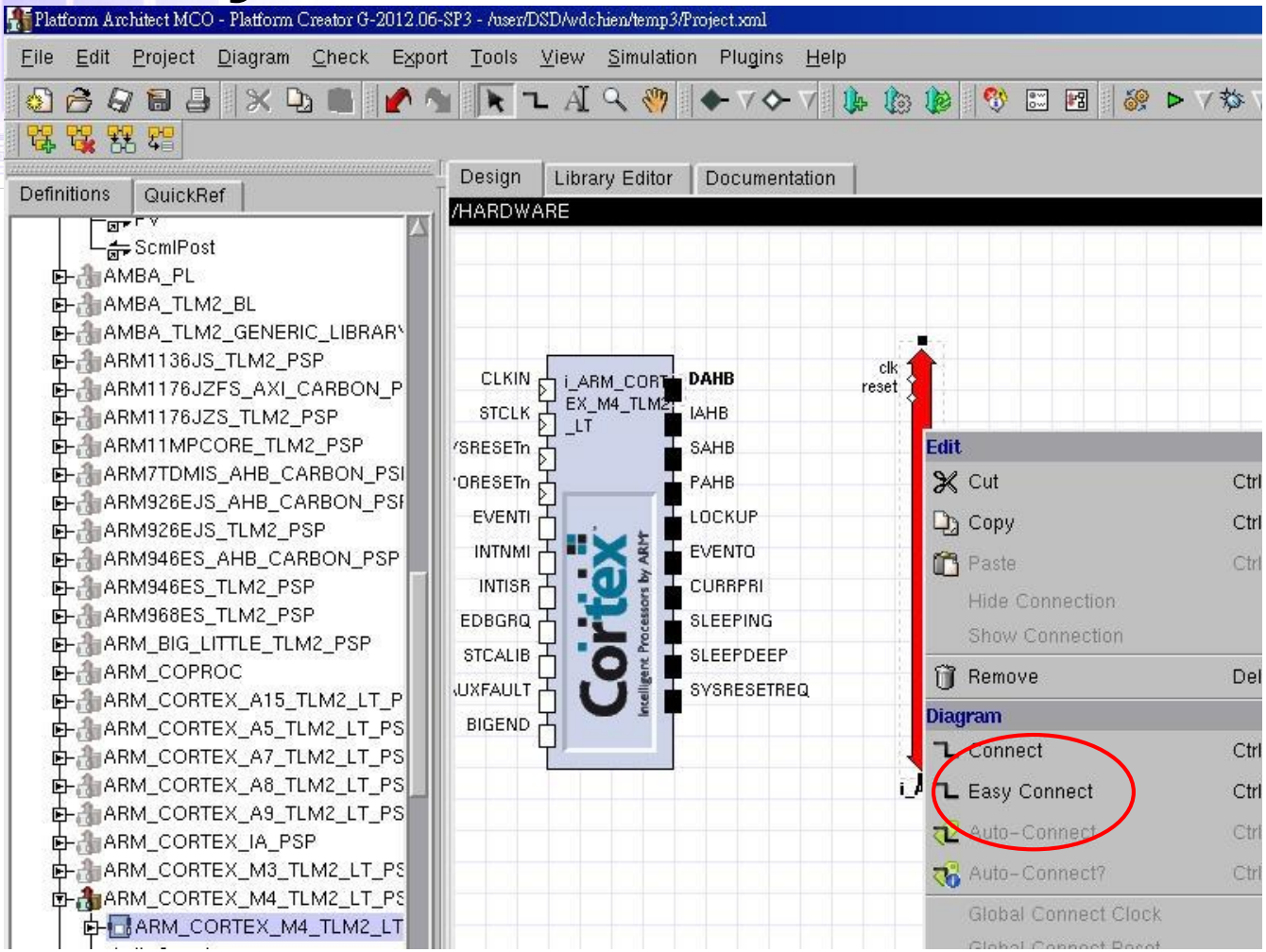
2. **Drag** and **drop** blocks and bus nodes from the IP libraries into the Design Editor window. The block appears in your Workspace

1. **Open** IP libraries

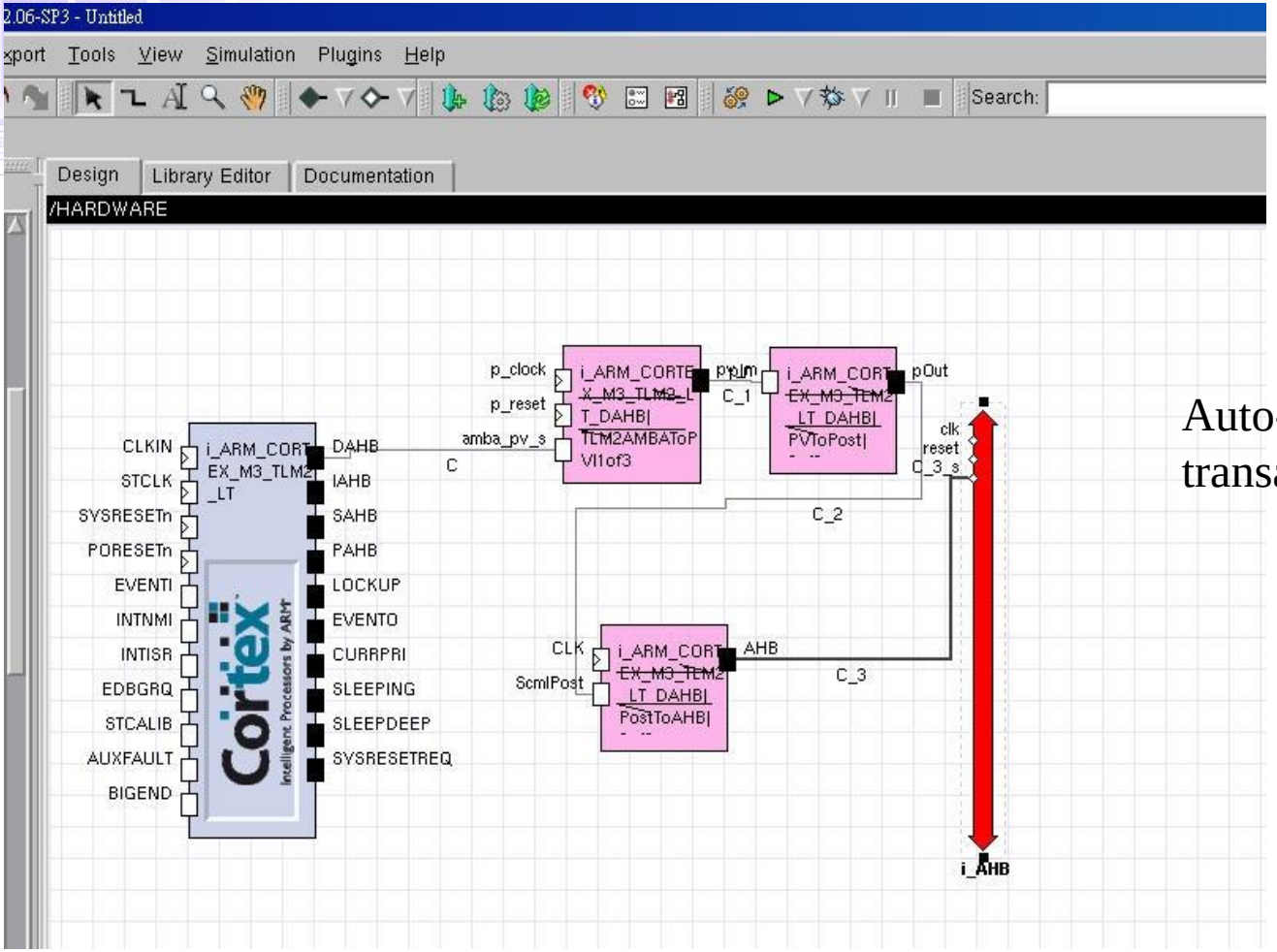
4. **Select** blocks and nodes to **set** memory map and other parameters

5. **Export** design for simulation build

Easy connect

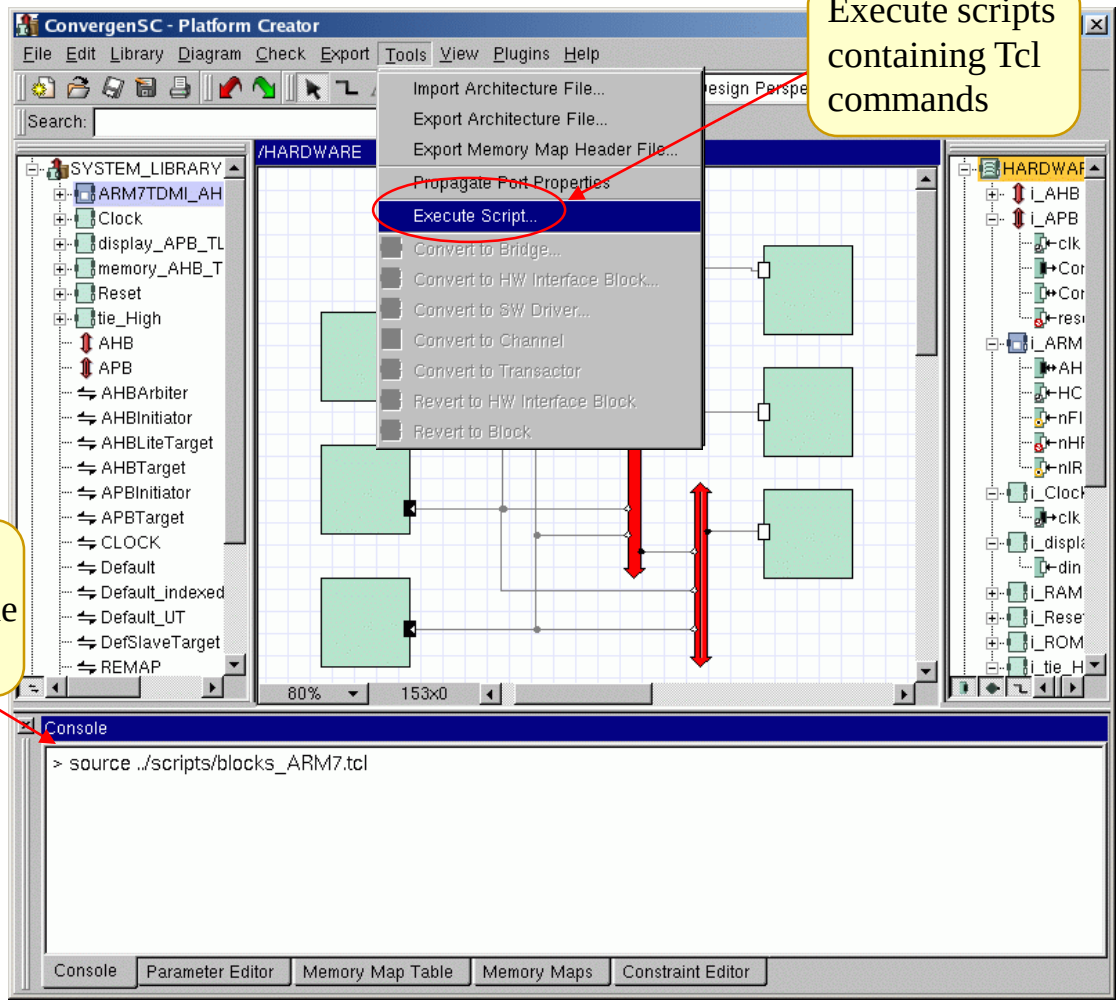


Easy connect



Auto-links the corresponding transactor(s) for you

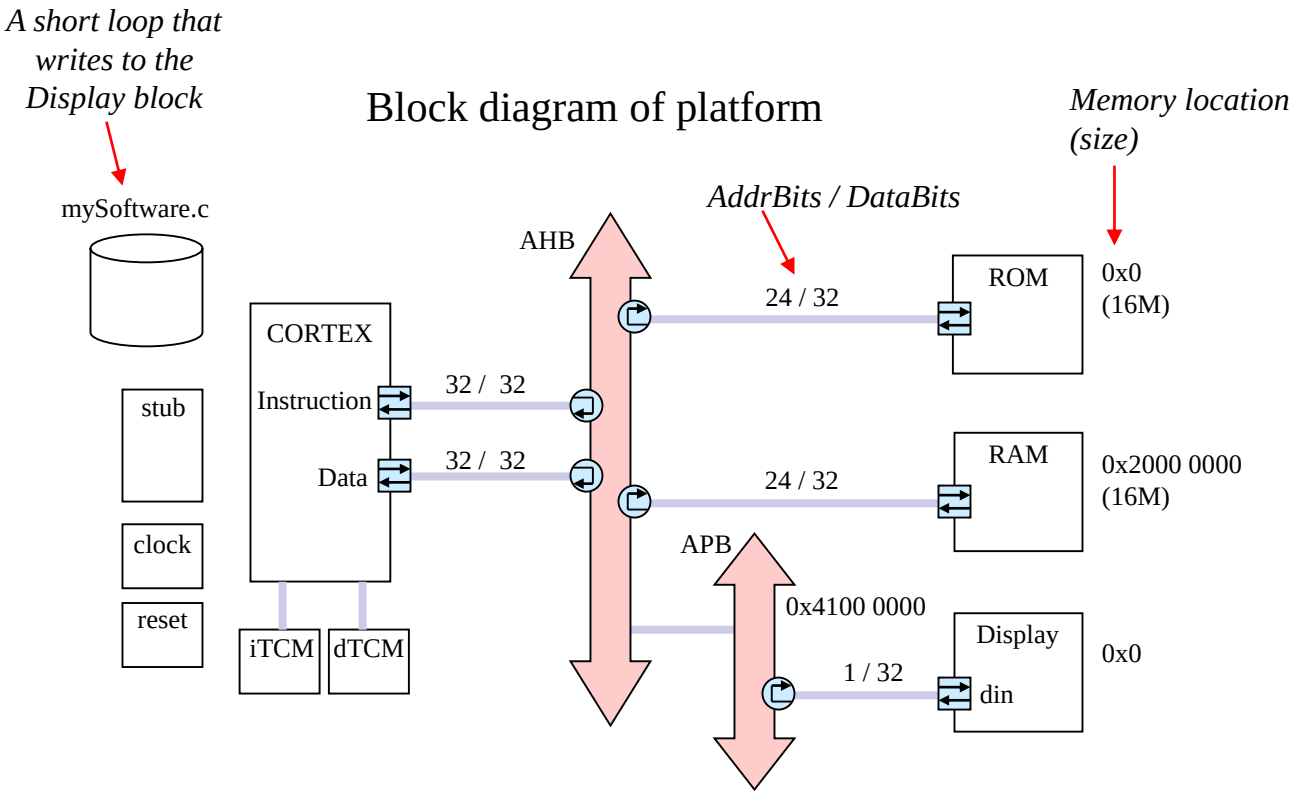
A second typical usage is that of scripting. All GUI commands have matching Tcl commands



Platform-based Design

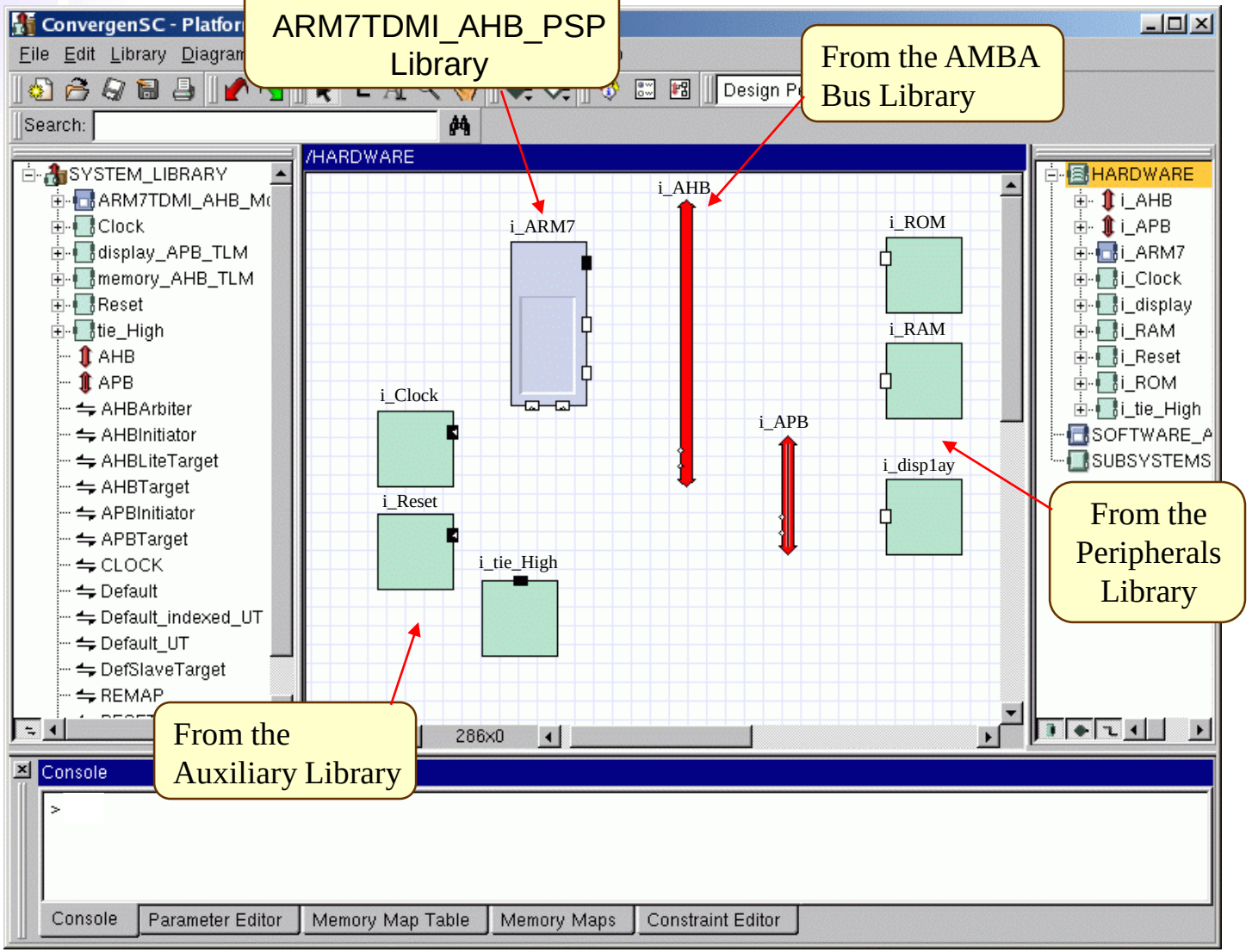
- Architecture build-up (PC)
  - 1. Place Blocks
  - 2. Complete Connections
  - 3. Create Memory Map
  - 4. Set Parameters
- Software programs writing (DS-5)
  - 1. Make Application Software
  - 2. Build image (.axf)
- Run simulation & debug

To illustrate platform-based design, we will build a small system. This is the block diagram

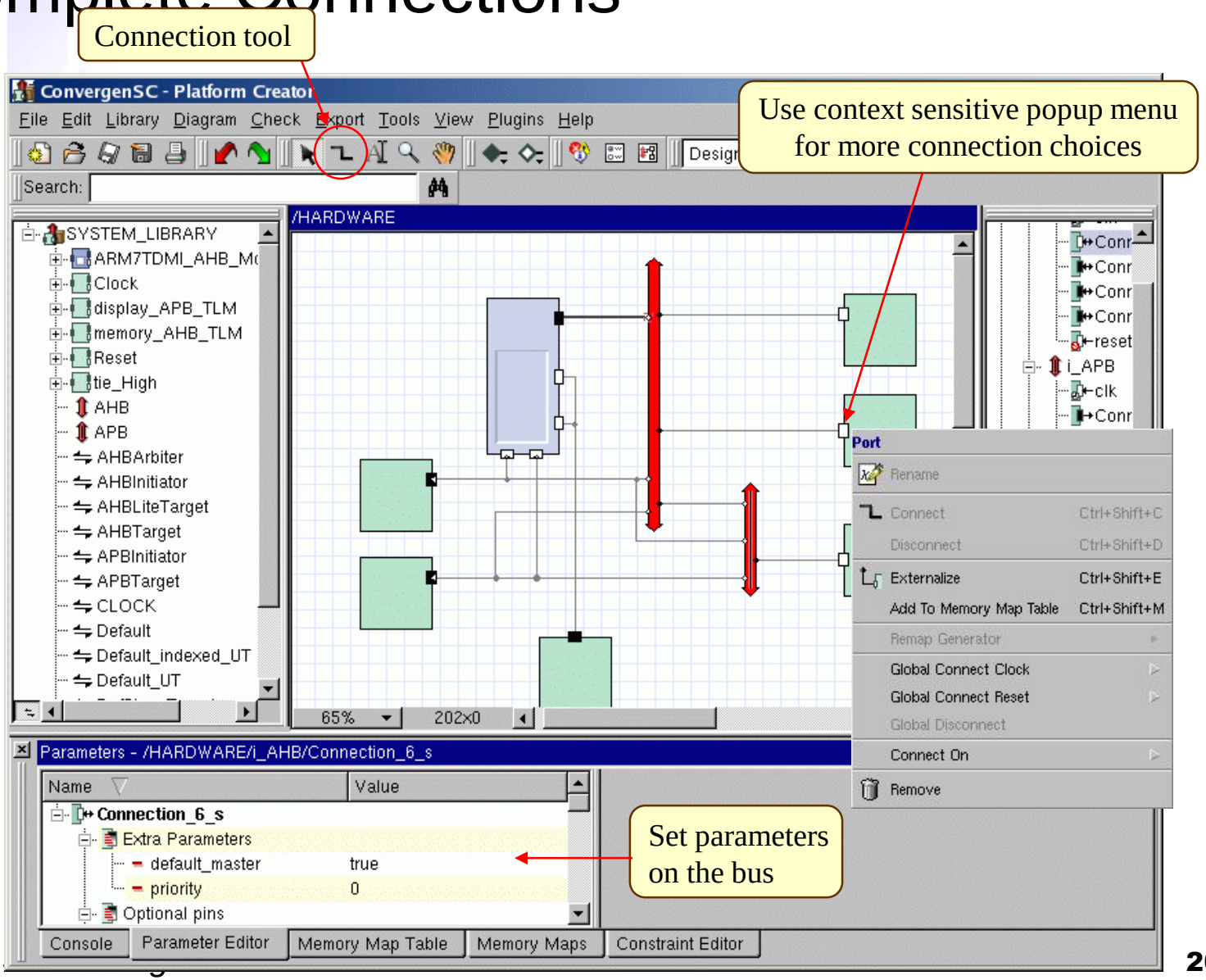


ARCHITECTURE BUILD-UP (PC)

1. Place Blocks and Nodes

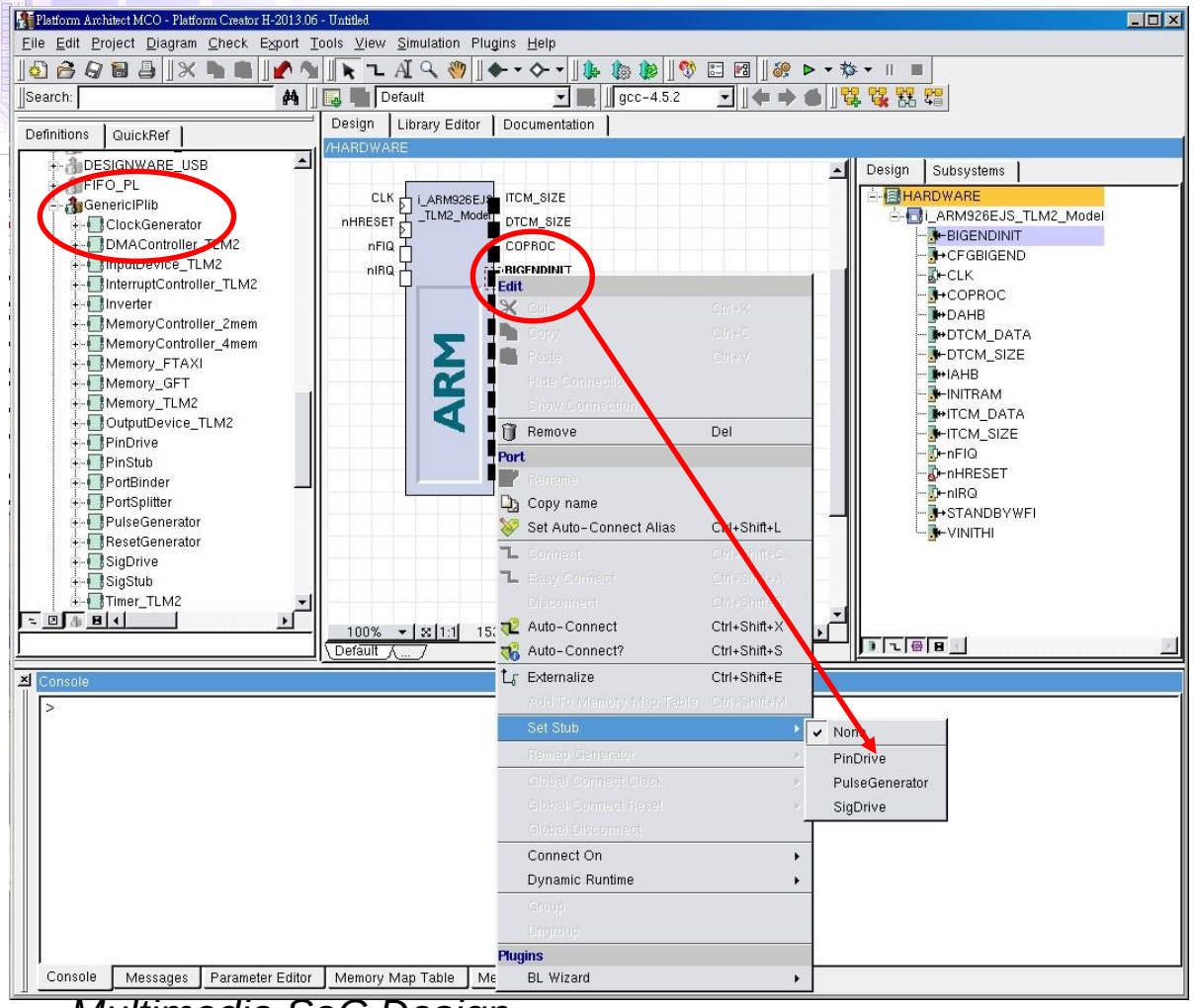


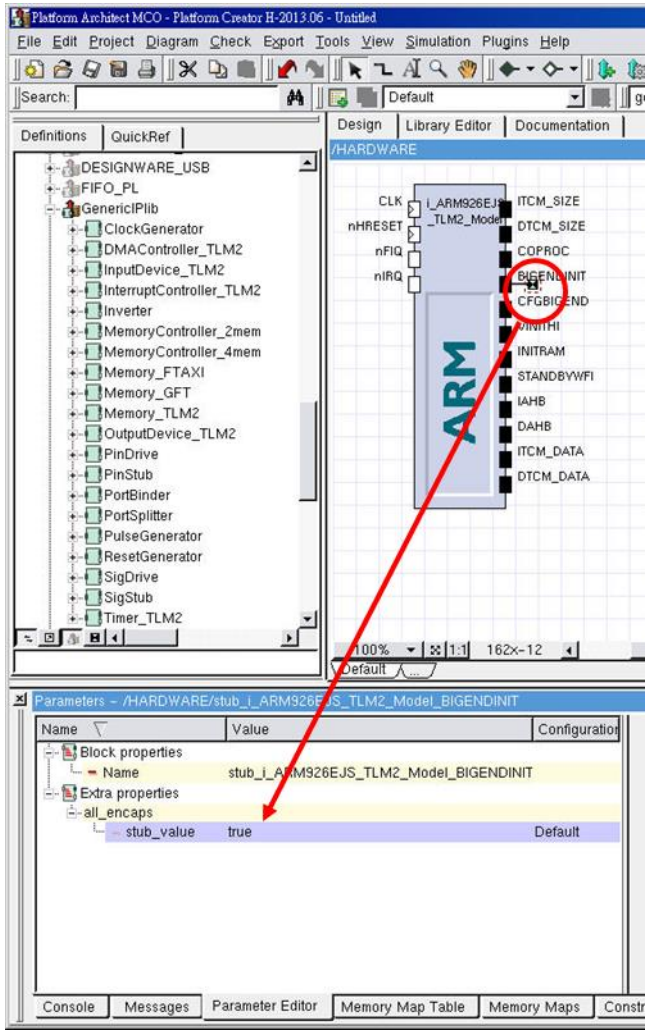
2. Complete Connections



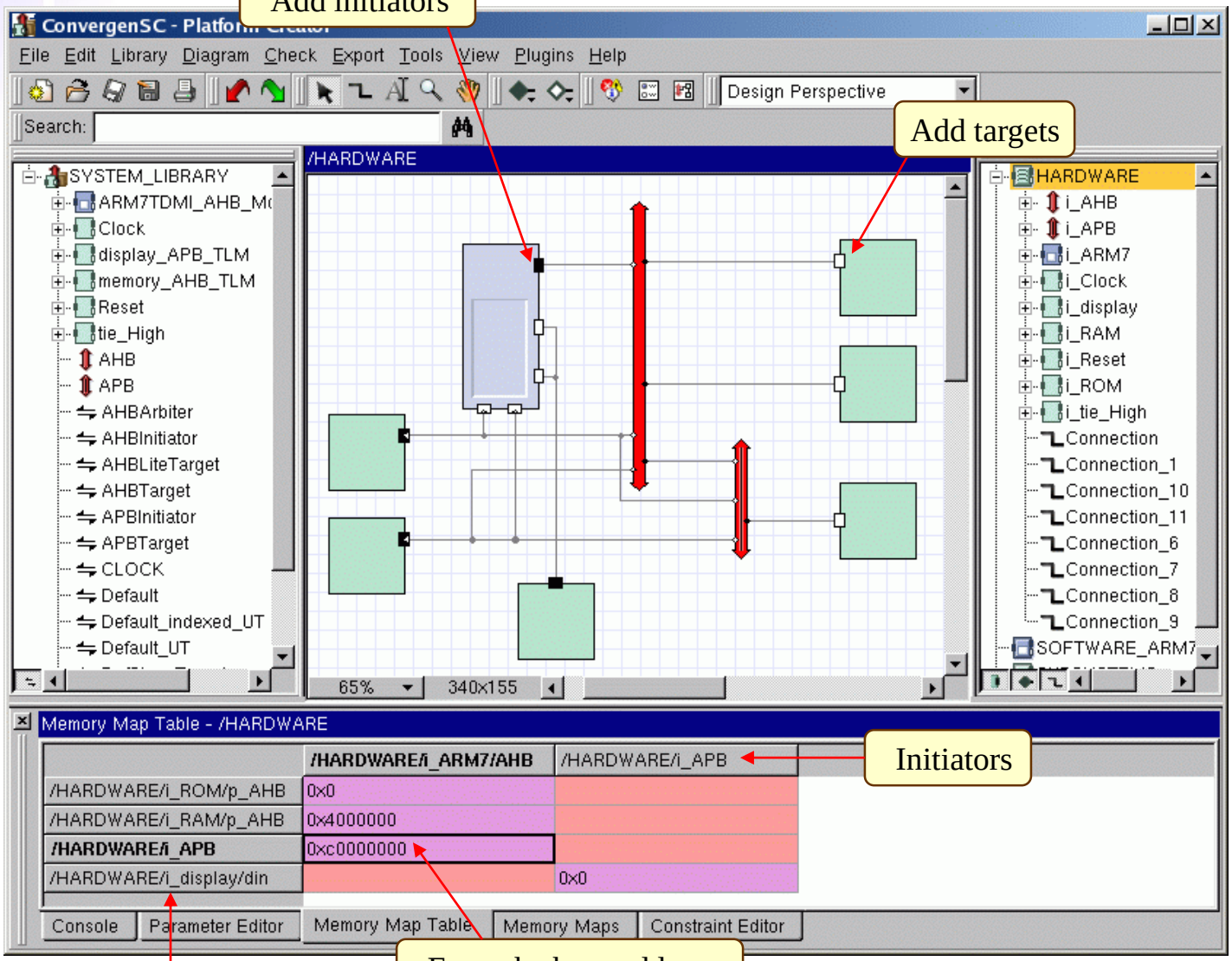
2. Complete Connections

Pin Stub

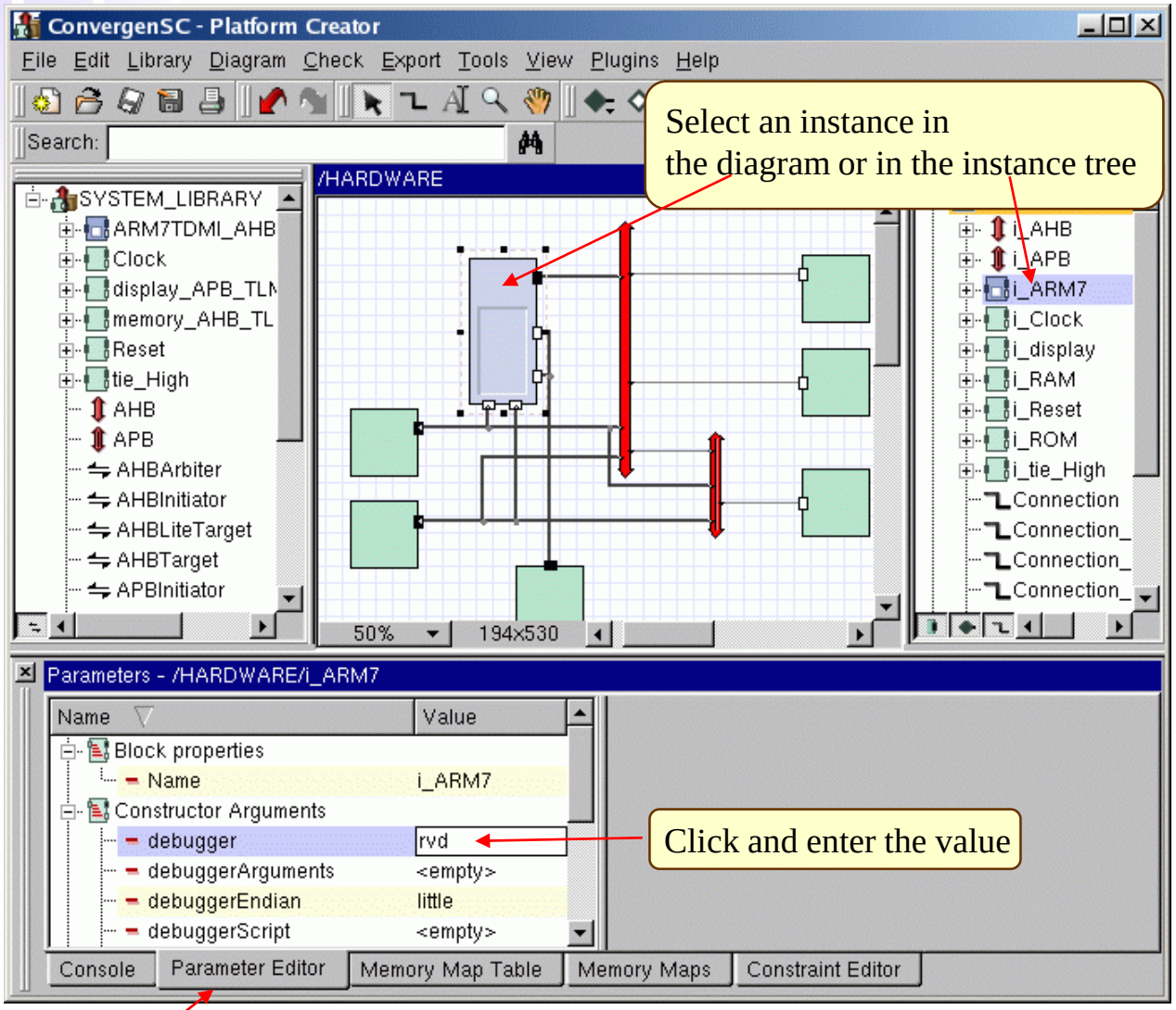




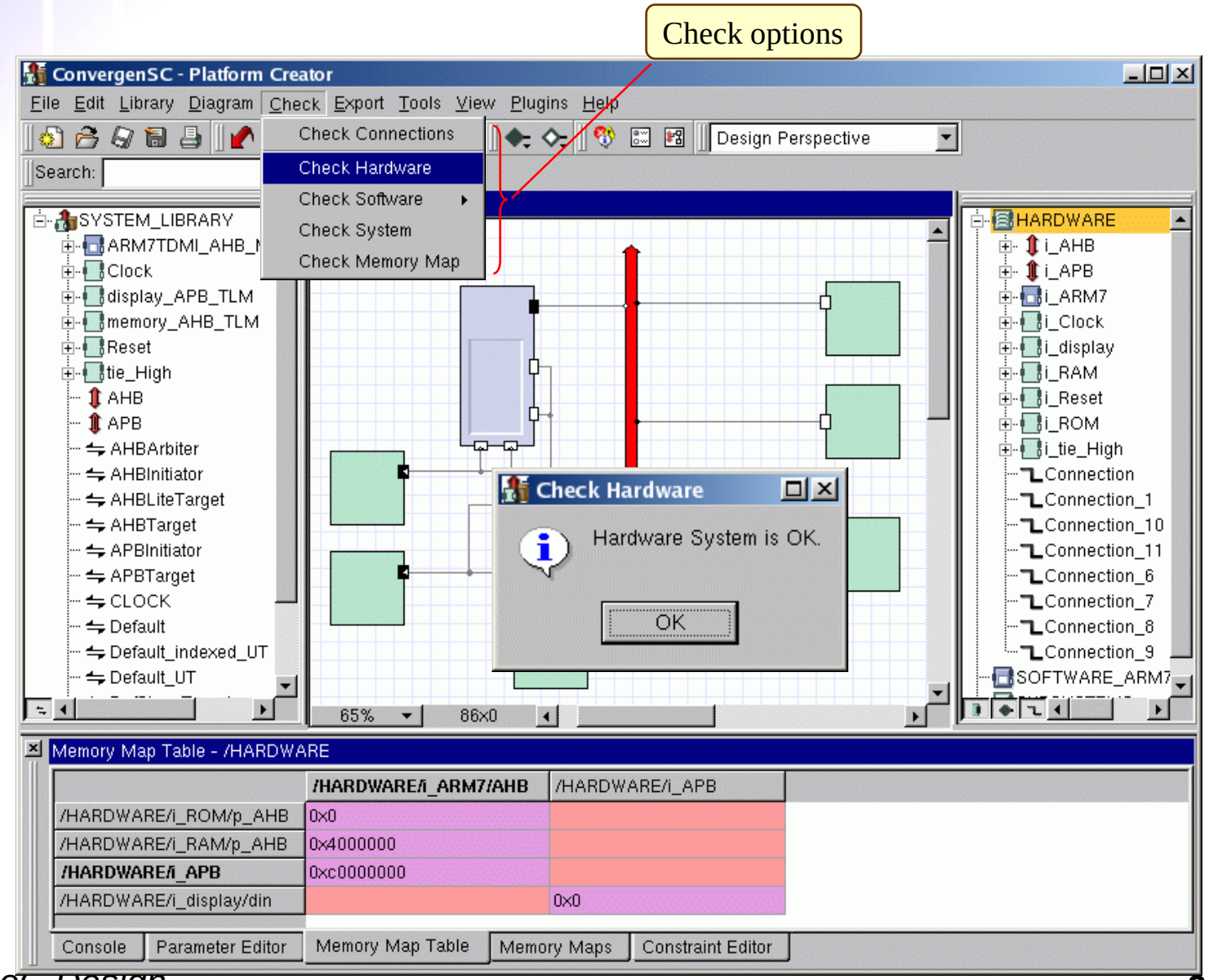
3. Create the Memory Map



4. Set Parameters



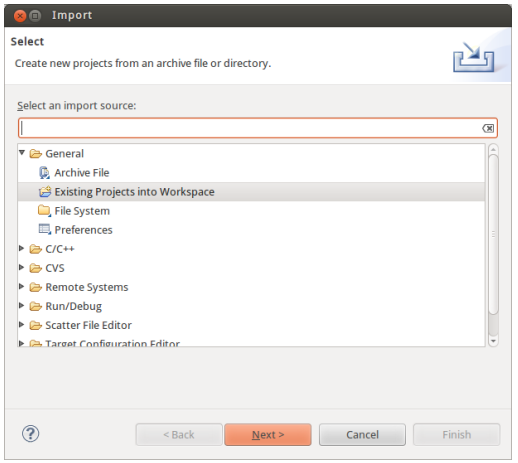
Check the System



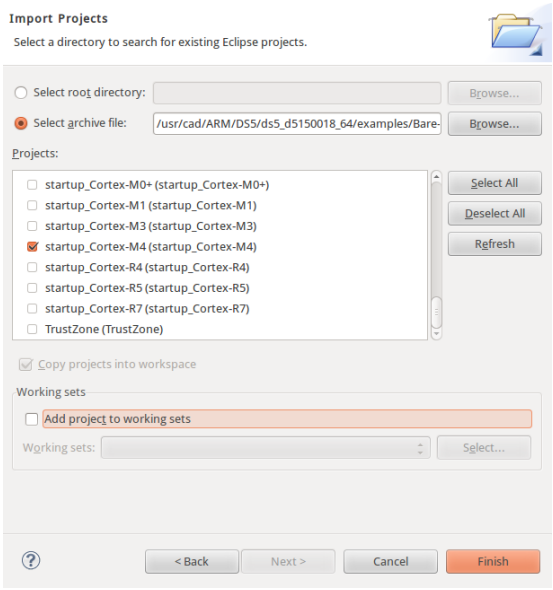
SOFTWARE PROGRAMS WRITING (DS-5)

Import your code from DS-5

- File -> Import
  - Existing Project into Workspace
- Check “Select archive file” and Browse to find your file

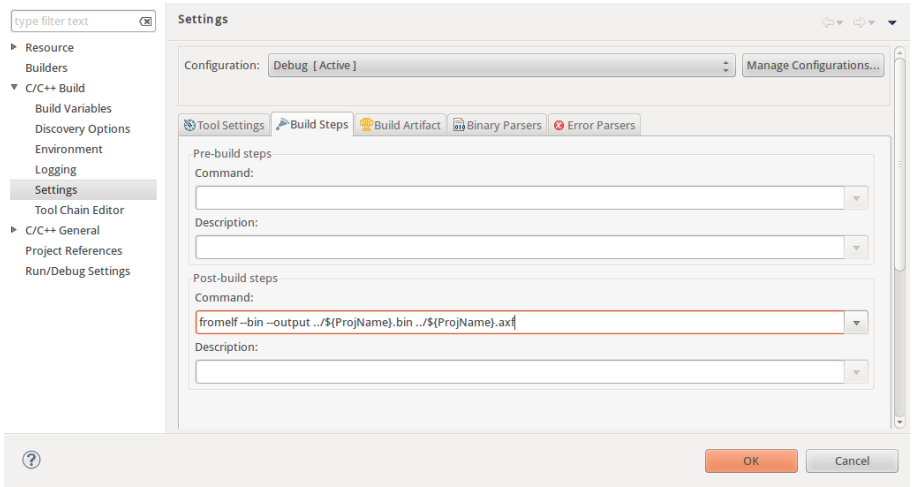


- Or you can import the sample code
  - /usr/cad/ARM/DS5/cur/examples/Bare-metal_examples.zip
  - Press “Deselect All” button
  - Select startup_Cortex-M4 (startup_Cortex-M4) project



Setting and generating application images

- Set linker to generate binary
  - Project -> Properties -> C/C++ Build -> Settings
  - In Tab “Build Steps” -> Post-build steps -> Command
    fromelf --bin --output ../${ProjName}.bin ../${ProjName}.axf
- Generate execution file and binary
  - Project -> Build project
  - Files will be generated
    Startup_Cortex-M4.axf
    Startup_Cortex-M4.bin
- Use this as template to modify main() function and build your own application image.



The application software is a simple routine for our testing purposes

mySoftware.c

```
#include "defines.h"
extern void InterruptEnable(void);

extern __irq void p_nIRQ_ISR (void) { ; }
extern __irq void p_nFIQ_ISR (void) { ; }

void myFunction(void) {
    int i;
    for (i=0; i < 20; i++){
          myDevice = i;
    }
    return;
}

int main(void) {
  InterruptEnable();
  myFunction();
  return 0;
}
```

Include header file
Prototype for InterruptEnable()

Unused Interrupts

Stub off the ISR's

User Function

Write some integers out to the peripheral

Main

defines.h

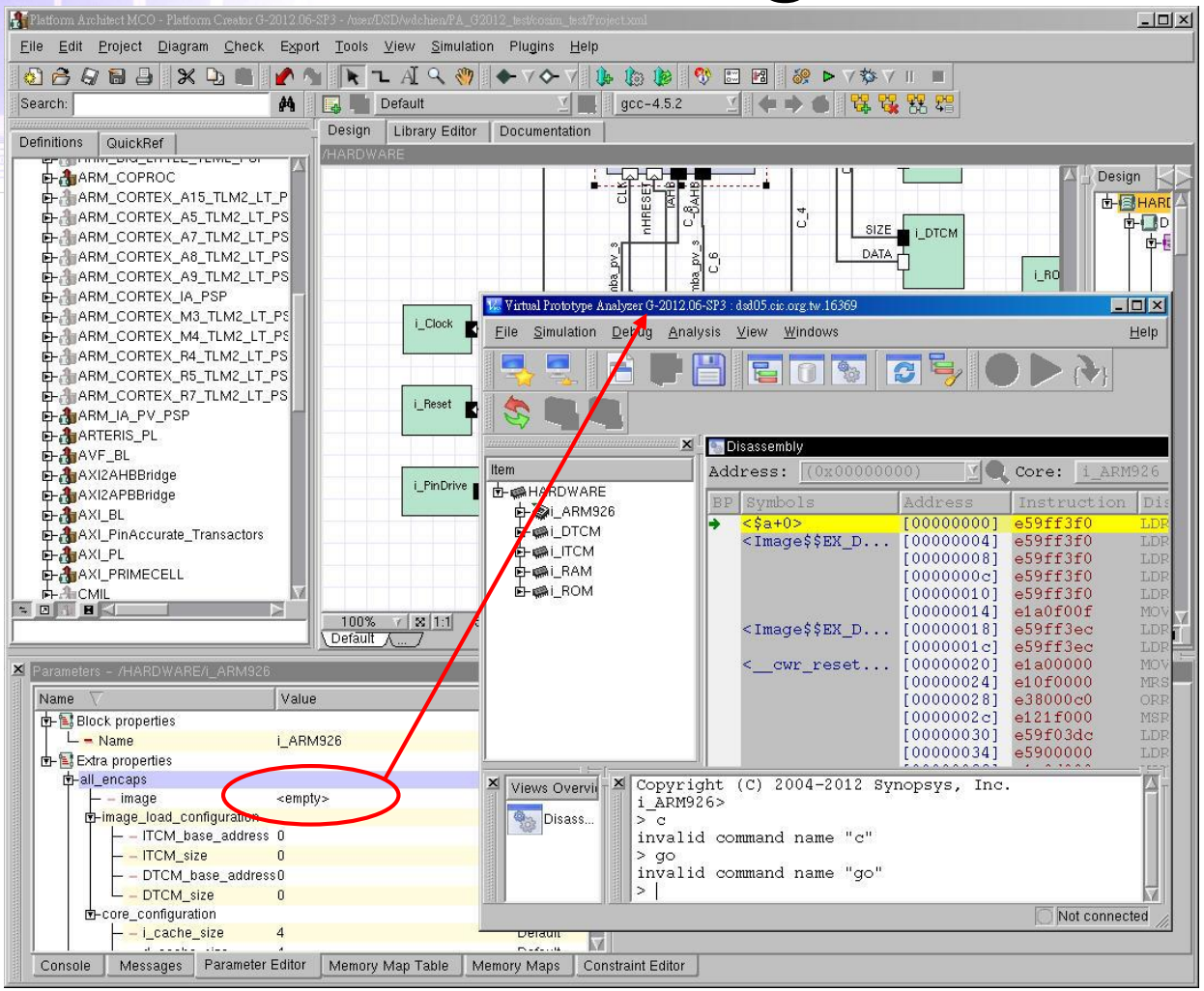
```
#define  myDevice  (*((volatile unsigned int *)0xc0000000))
```

Addressing

```
1 #define ALU_BASE         0x40001000
2 #define DISPLAY_BASE     0x41000000
3 #define DUAL_TIMER_BASE  0x41001000
4
5 #define OPCODE           (*((volatile unsigned int *)(ALU_BASE)))
6 #define OPERAND1         (*((volatile unsigned int *)(ALU_BASE+0x04)))
7 #define OPERAND2         (*((volatile unsigned int *)(ALU_BASE+0x08)))
8 #define RESULT           (*((volatile unsigned int *)(ALU_BASE+0x0C)))
9
10 #define DISPLAY         (*((volatile unsigned int *)(DISPLAY_BASE)))
11
12 #define TIMER1LOAD      (*((volatile unsigned int *)(DUAL_TIMER_BASE)))
13 #define TIMER1VALUE     (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x04)))
14 #define TIMER1CONTROL   (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x08)))
15 #define TIMER1INTCLR    (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x0C)))
16 #define TIMER1RIS       (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x10)))
17 #define TIMER1MIS       (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x14)))
18 #define TIMER1BGLOAD    (*((volatile unsigned int *)(DUAL_TIMER_BASE+0x18)))
```

RUN SIMULATION & DEBUG

Software debug



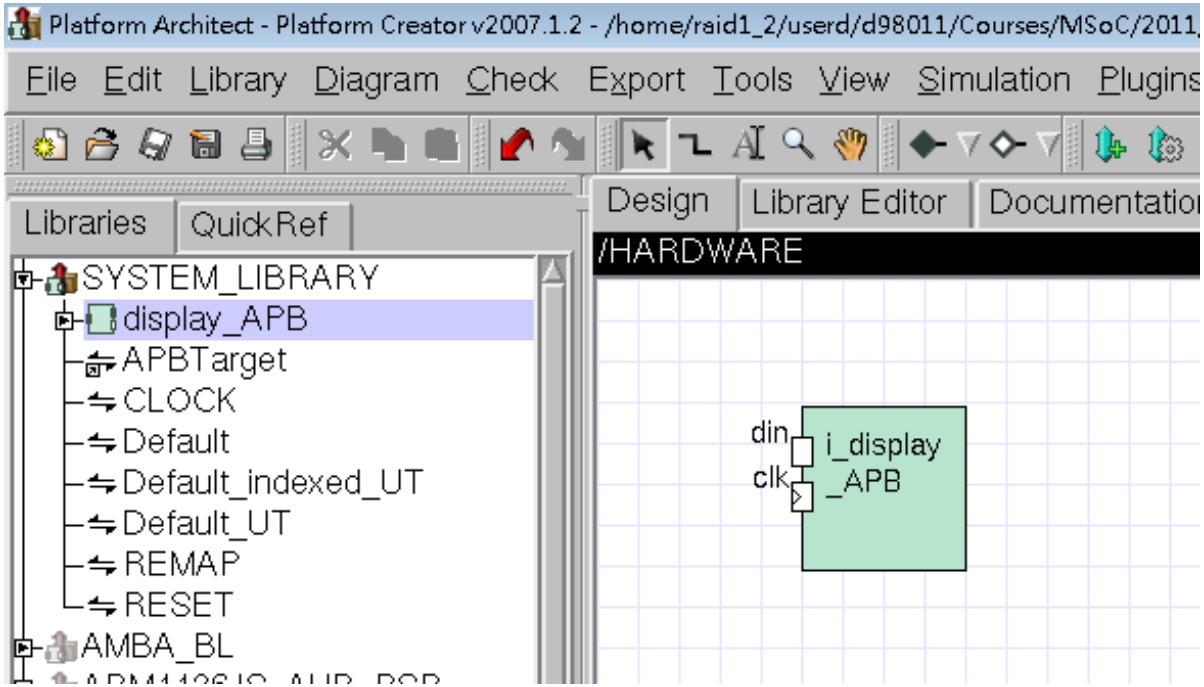
Virtual Platform Analyzer (VPA): PA build-in software debugger / analyzer

Lab Exercise

Platform Architect (I)

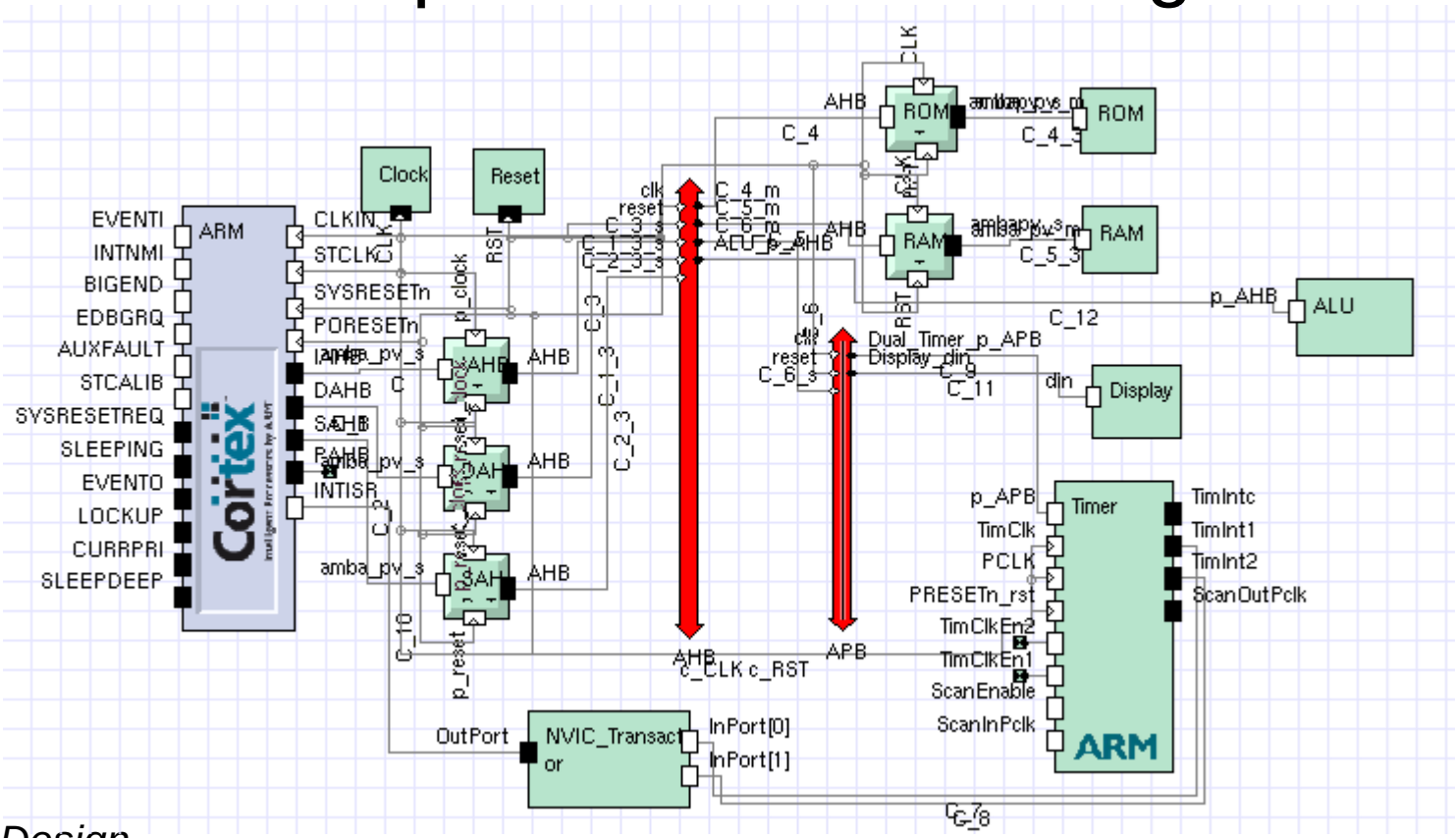
Target

- Building a self-developed SystemC and Verilog module into a library

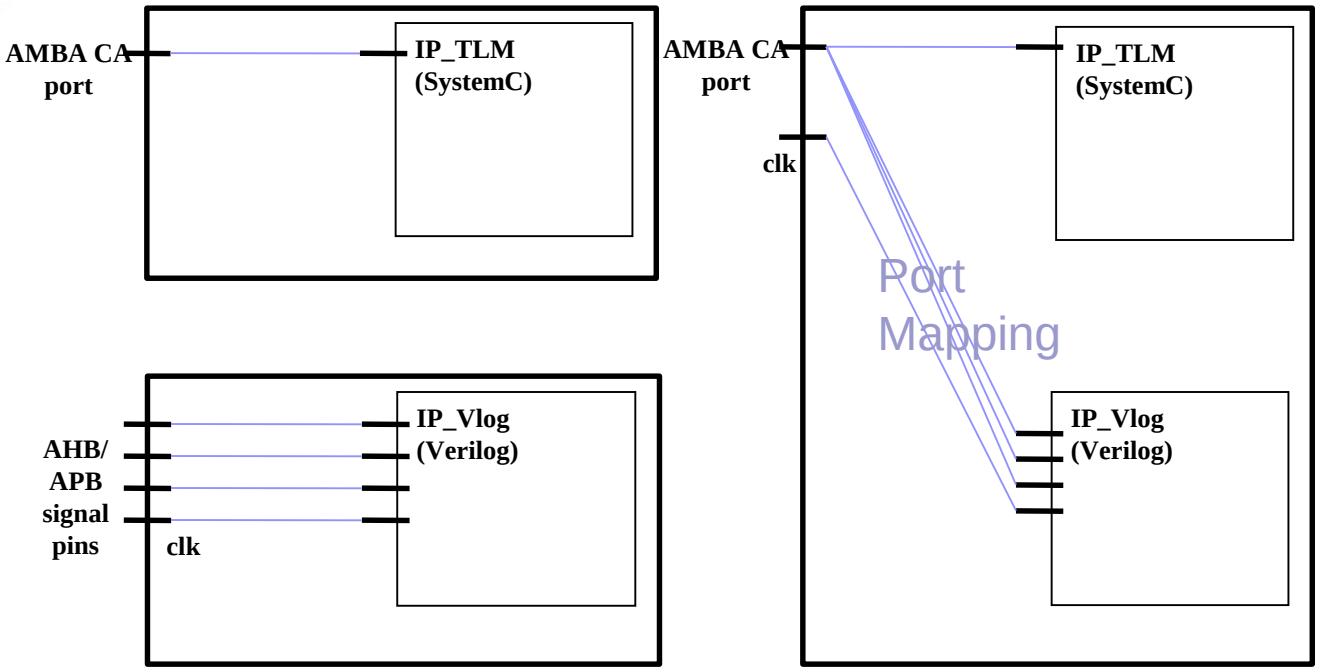


Target

- Familiar with the platform-based design flow



Packing Verilog module



Commands

- 解壓縮
  - >tar -xvf PA_DS5_ALU_v1.05.tar.gz
- Source File
  - >source pa_setup.csh
  - >source run.csh

Modify it :
#31 ARM -> arm
#32 ARM -> arm

run.csh #6 ARM->arm

- Cad lab server
  - http://cad.ee.ntu.edu.tw/ws_list.htm
  - Linux OS

Use cad42 !!

- Remember to execute X-ming or X-win

IP	Name	Type	CPU	CPU Clock	Memory	OS
140.112.20.59	cad16	IBM X3400	Intel Xeon 64	2.4 GHz * 16	100 G	RHEL 5
140.112.20.60	cad17	IBM X3550	Intel Xeon 64	2.4 GHz * 16	20 G	RHEL 5
140.112.20.61	cad18	SUN Blade 2500	UltraSPARC	1.28 GHz*2	8 G	Solaris 10
140.112.20.62	cad19	SUN Blade 2000	UltraSPARC	1.2 GHz * 2	8 G	Solaris 10
140.112.20.63	cad20	SUN Blade 2000	UltraSPARC	1.2 GHz * 2	8 G	Solaris 10
140.112.20.64	cad21	DELL R620	Intel Xeon 64	2 GHz * 32	48 G	CentOS 5
140.112.20.65	cad22	SUN Blade 2500	UltraSPARC	1.28 GHz * 2	8 G	Solaris 10
140.112.20.66	cad23	DELL R620	Intel Xeon 64	2 GHz * 16	96 G	CentOS 5
140.112.20.67	cad24	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.68	cad25	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.69	cad26	SUN Fire 280R	UltraSPARC	1.2 GHz * 2	4 G	Solaris 9
140.112.20.70	cad27	IBM x260	Intel Xeon 64	3.2 GHz * 4	8 G	RHEL 4
140.112.20.71	cad28	IBM x260	Intel Xeon 64	3.2 GHz * 4	8 G	RHEL 4
140.112.20.72	cad29	IBM e336	Intel Xeon 64	3.2 GHz	5 G	RHEL 4
140.112.20.73	cad30	IBM X3650	Intel Xeon 64	2 GHz * 16	12 G	SUSE 11
140.112.20.74	cad31	DELL R620	Intel Xeon 64	2.2 GHz * 32	48 G	CentOS 6
140.112.20.75	cad32	SUN Blade 2000	UltraSPARC	1.015 GHz * 2	8 G	Solaris 8
140.112.20.76	cad33	IBM X3500	Intel Xeon 64	2 GHz * 16	20 G	RHEL 4
140.112.20.77	cad34	IBM X3650	Intel Xeon 64	2.4 GHz * 8	12 G	RHEL 5
140.112.20.78	cad35	SUN V20z	AMD Opteron	2.4 GHz * 2	4 G	RHEL 4
140.112.20.79	cad36	SUN V20z	AMD Opteron	2.4 GHz * 2	4 G	RHEL 4
140.112.20.80	cad37	SUN V20z	AMD Opteron	2.4 GHz * 2	4 G	RHEL 4
140.112.20.81	cad38	IBM X3650	Intel Xeon 64	2.4 GHz * 16	96 G	RHEL 5
140.112.20.82	cad39	IBM X3650	Intel Xeon 64	2 GHz * 24	40 G	CentOS 5
140.112.20.83	cad40	IBM X3550	Intel Xeon 64	2 GHz * 24	8 G	CentOS 5
140.112.20.84	cad41	IBM X3550	Intel Xeon 64	2.1 GHz * 24	64 G	CentOS 6
140.112.20.85	cad42	IBM X3500	Intel Xeon 64	2 GHz * 24	32 G	CentOS 5