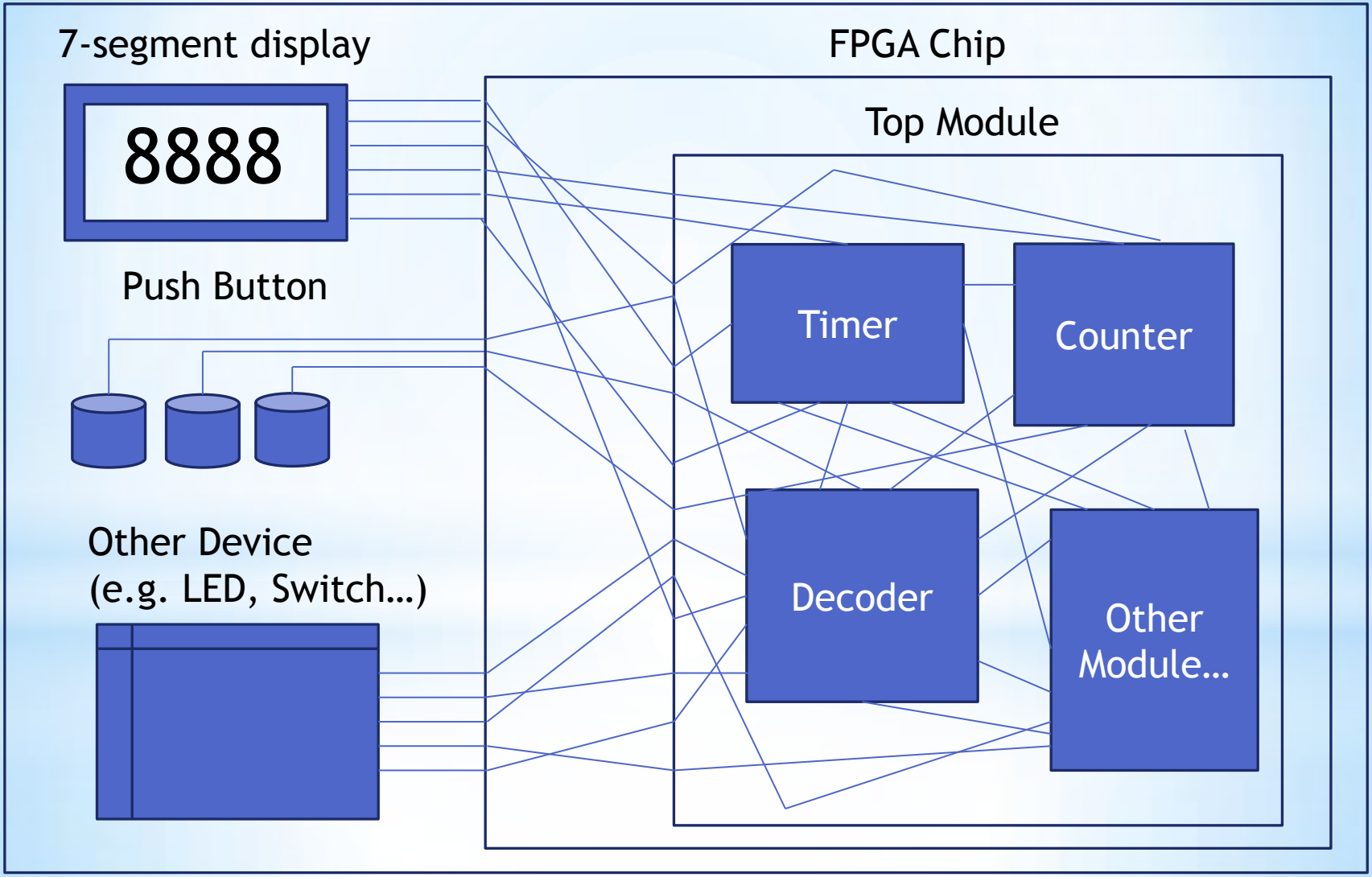
Introduction to SOPC Lab

Bring you from Logic Design Lab to SOPC Lab!

Written by Geng You Chen
(gengyouchen@gmail.com)

Logic Design Lab's Final Project: A Simple Electronic Clock

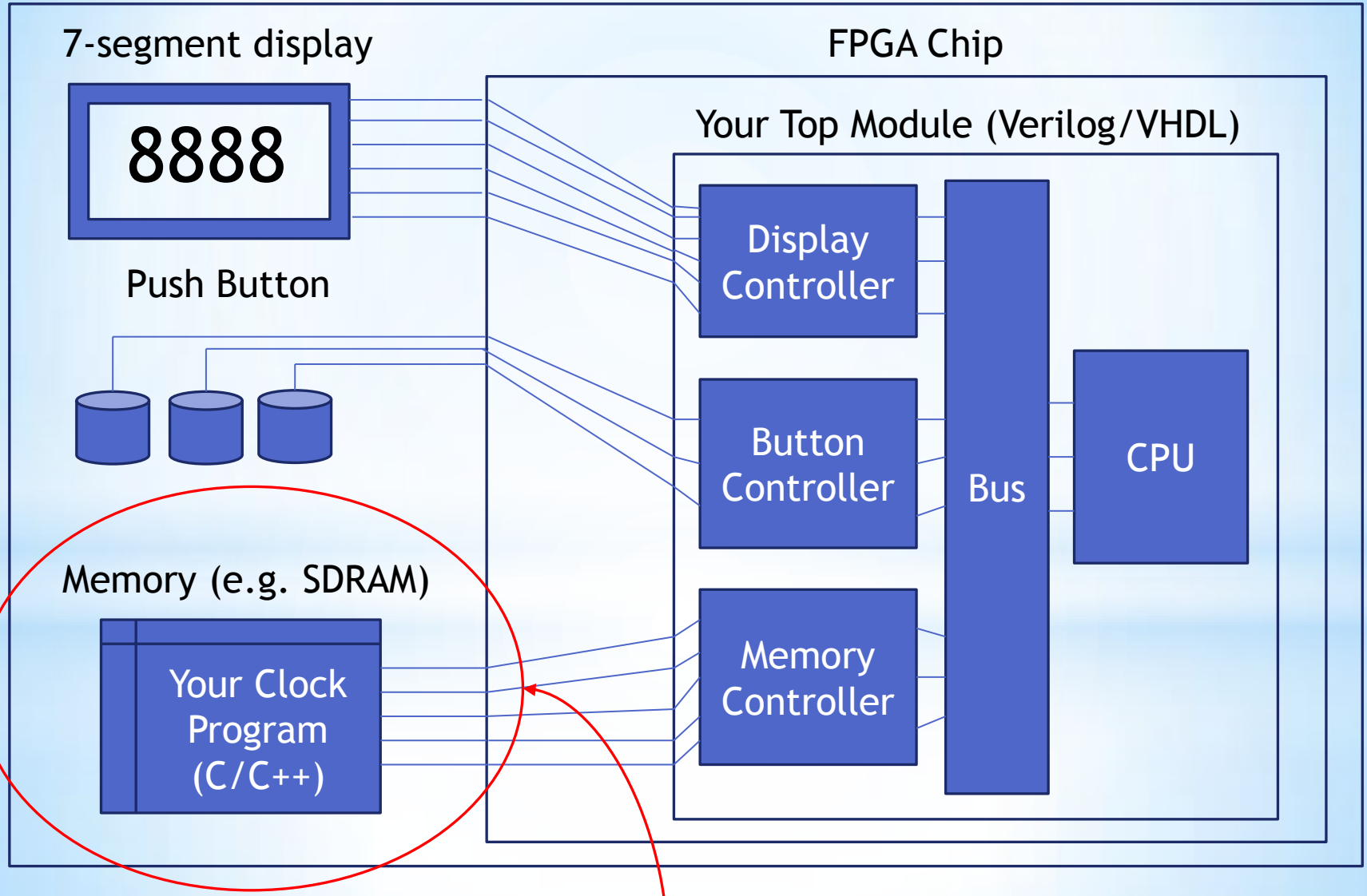


The hardware implemented by FPGA chip will be complicated if you want to design "not just a simple electronic clock", for example, a cell phone that can play games and surf the Internet. So what should we need to do?

* In your logic design lab, all the functions are directly implemented by your hardware (FPGA). If your functions become more complicated, your hardware become more complicated, too!
* A cell phone may have thousands of functions, so you need to design thousands of hardware and put all of them in FPGA. This is impossible!
* However, if the behavior of your hardware is not always the same, but according to some codes you give to it, it is possible to implement complicated functions in a simple hardware.

* a complete system = hardware + software

A Complete Electronic Clock System

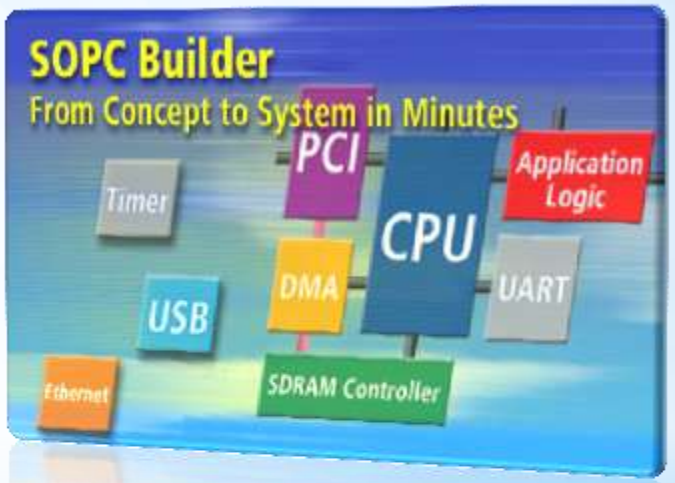


Now the CPU sense the push buttons and control the 7-segment display. Because the CPU behavior is according to the codes you write in memory, you can implement very complicated functions just by writing more codes.

SOPC = System on Programmable Chip

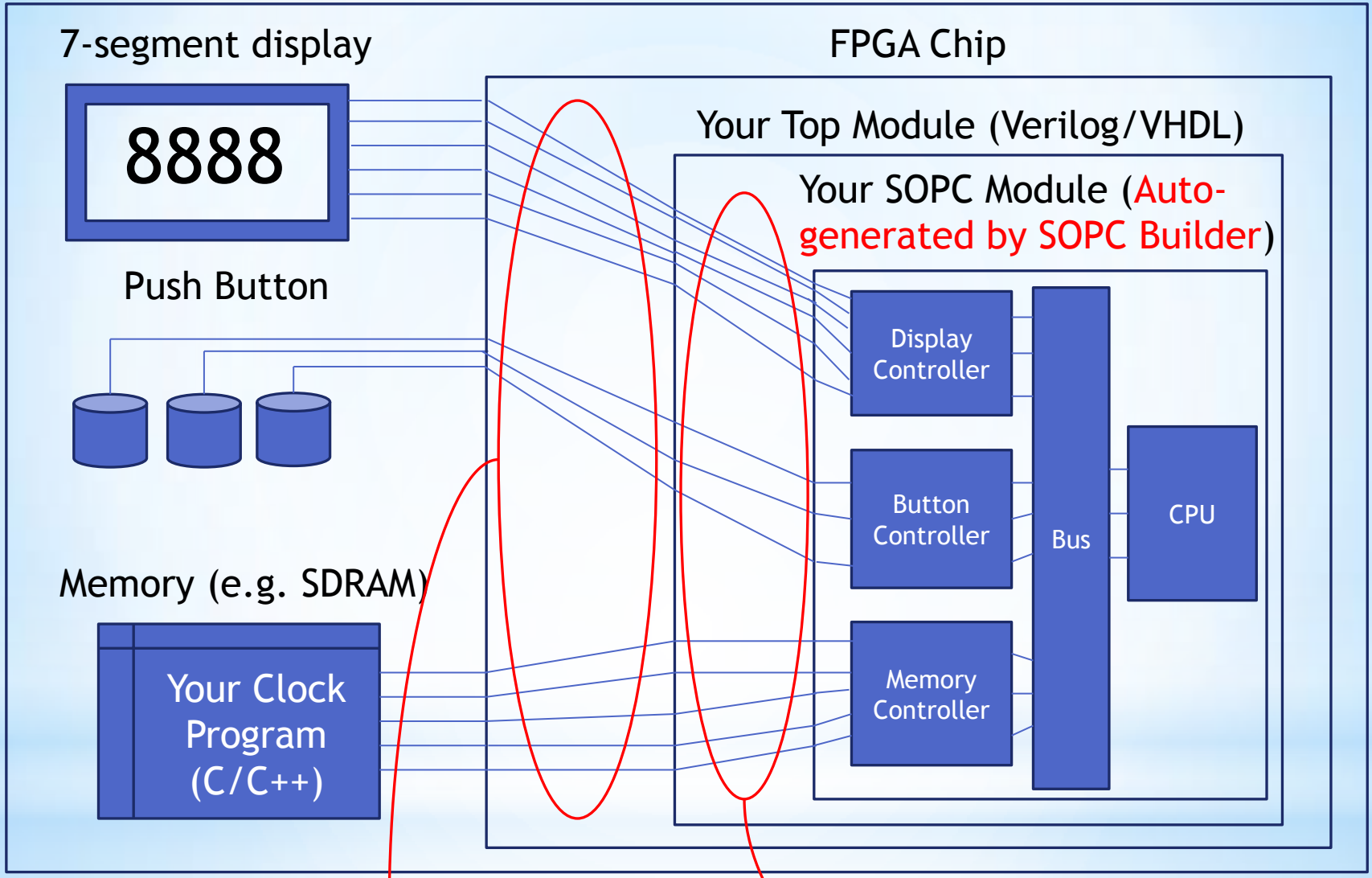
- System = Hardware and Software
  - Design hardware (Verilog or VHDL) using Altera Quartus II
  - Design software (C or C++) using Altera Nios II EDS
- Programmable Chip = FPGA
  - FPGA Chip: Altera Cyclone II EP2C70F896C6N
  - Development Board: Terasic DE2-70 Board (with D5M and LTM)

* If you want to design all hardware in SOPC system such as CPU, Bus, Memory Controller, you need a lot of background knowledge in other course (e.g. computer architecture), also, you need a lot of time to write Verilog/VHDL and “try to” debug it.
* Luckily, Altera Corporation provides a powerful tool named SOPC Builder, which let us design a “workable” SOPC system without writing any Verilog/VHDL codes.



* Altera SOPC Builder speeds up your development!

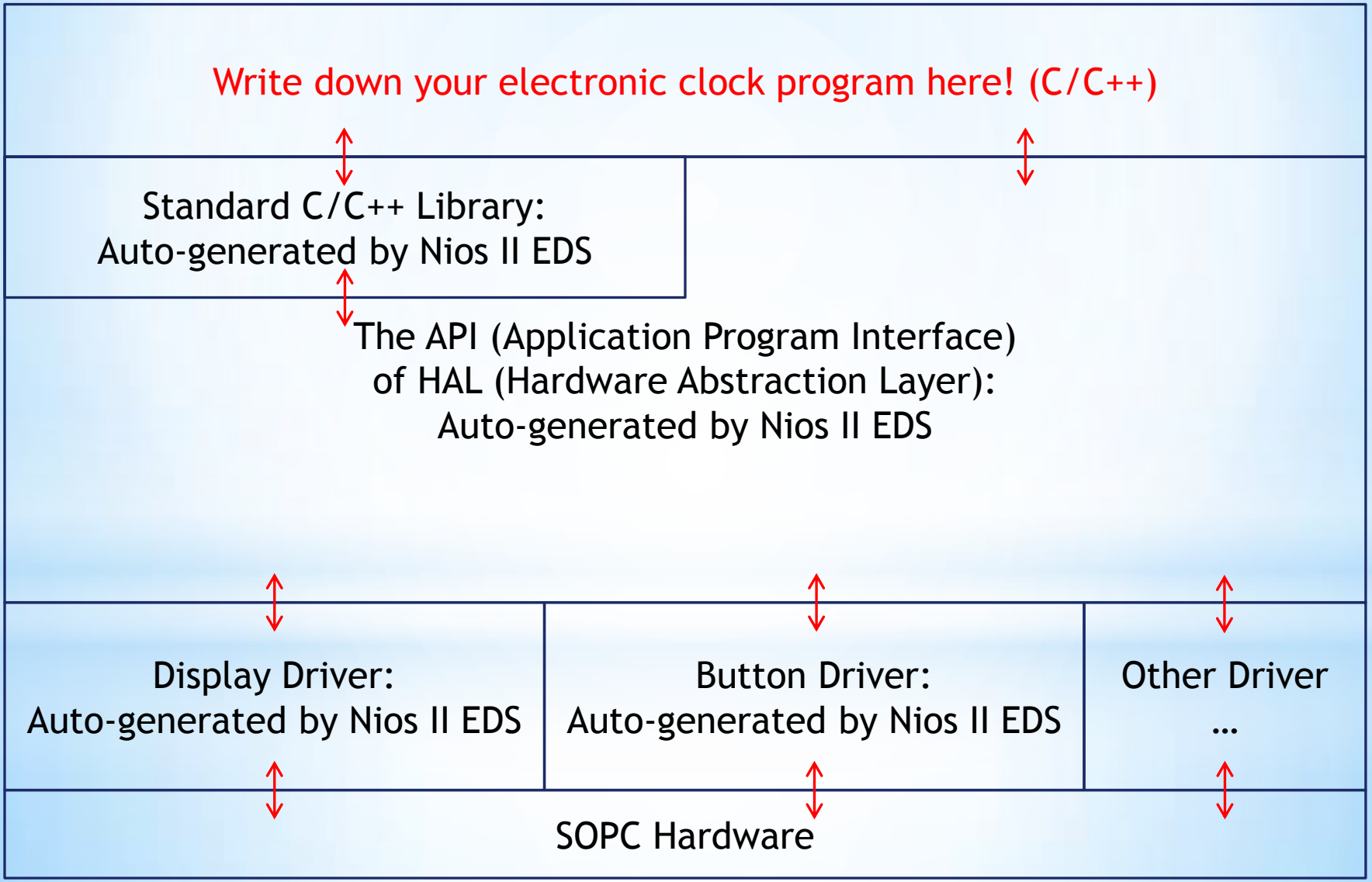
For your hardware design, all you need to do is connect wires and assign pins.



Before burning your Verilog/VHDL codes into FPGA, you need to do the pin assignment according to your Terasic DE2-70 board.

In top module, you need to connect these wires by yourself. These wires are not auto-generated.

For your software design, because Nios II EDS will generate the software core from SOPC Builder, it's easy to write C/C++ program to access your hardware: Simply call the API functions provided in HAL, just like you're calling a normal C/C++ function, and your program can sense the button or control the display.



* To sum up, a very simple SOPC design flow is:

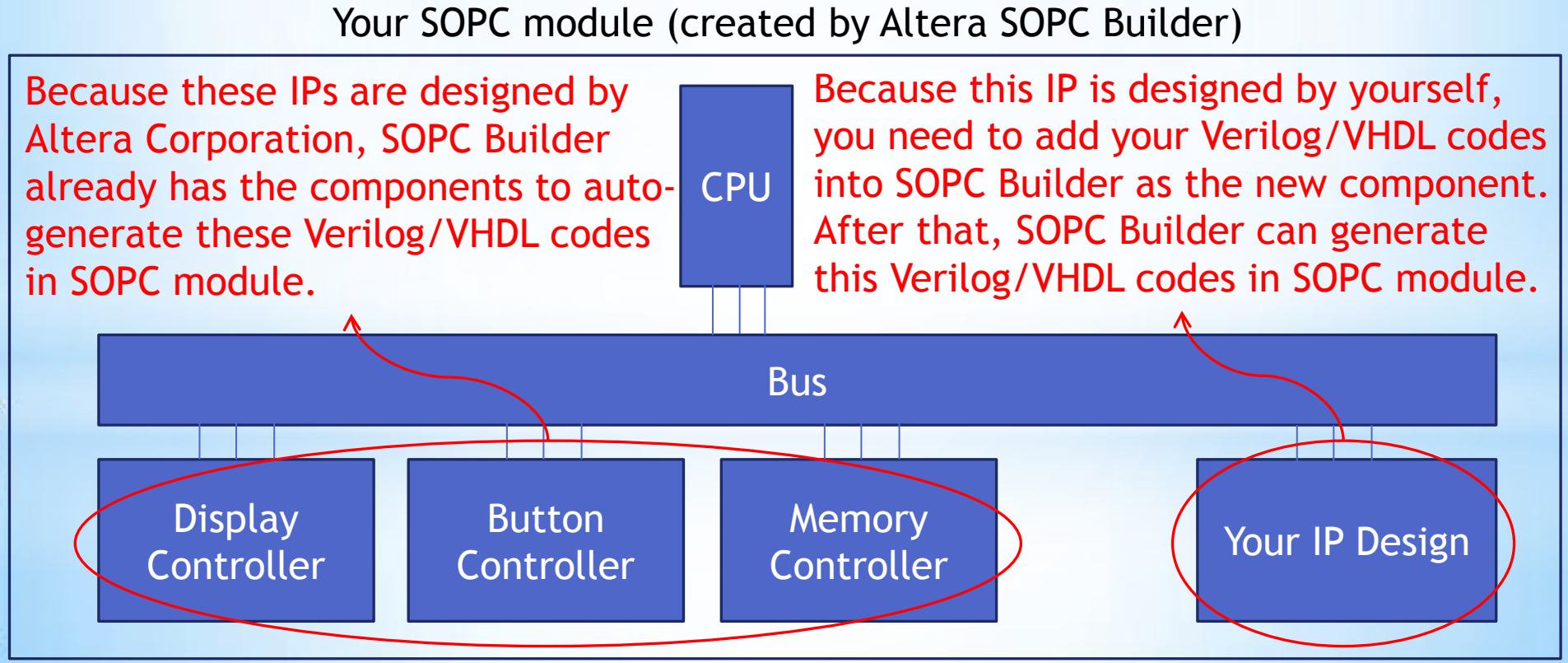
1. Use SOPC Builder to design your SOPC module. Then, SOPC Builder can auto-generate the Verilog/VHDL files.
2. Write your top module file (Verilog/VHDL) by yourself. Make sure I/O pins are connected to the SOPC module.
3. Do pin assignment according to Terasic DE2-70 board. Then, burn all Verilog/VHDL files into Cyclone II chip.
4. Use Nios II EDS to auto-generate your software core based on your design in SOPC Builder.
5. Write your C/C++ program to access the SOPC hardware using the software core. Then, burn all C/C++ files into the memory (e.g. SDRAM) on Terasic DE2-70 board.

* So… No hardware design? Let's think a little bit deeper…

Software do anything! Why we need to design hardware?

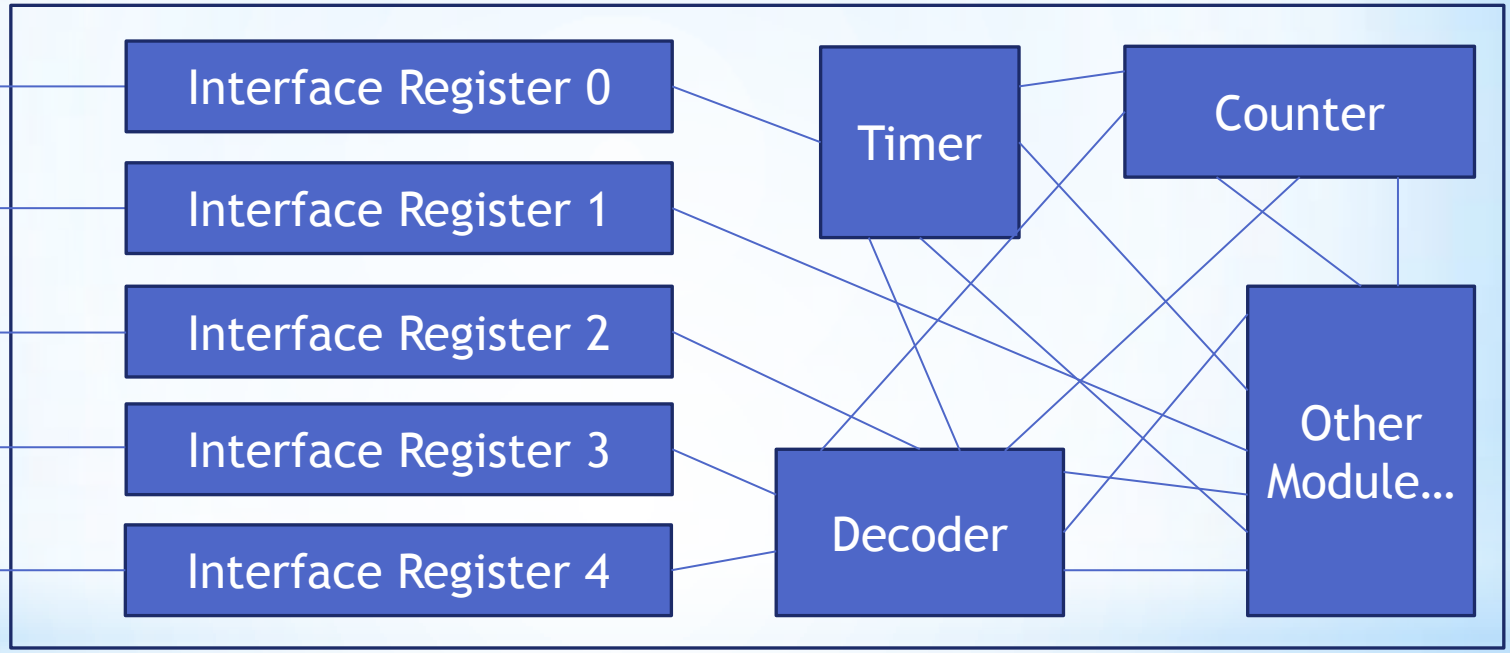
- Hardware Acceleration
  - A cell phone may have the ability to play high-compressed videos (e.g. H.264/MPEG-4) or play 3D games, so your C/C++ program needs a lot of time to calculate the color of each pixel.
  - Assume the screen has 960 x 640 pixels (for iPhone 4). Because software can only do pixel-by-pixel calculation, if you want to show 30 frames per second, your program must calculate total 18432000 pixels per second, which means your program has only 54 nanoseconds for each calculation. This is impossible!
  - However, if you design a hardware to calculate each pixels, hardware can parallel calculate all pixels at the same time.
- Non-Altera Device Controller
  - Because your Terasic DE2-70 Board is not 100% made by Altera Corporation, some device (e.g. 7-segment display) are buy from other company. Therefore, Altera SOPC Builder doesn't have the components to auto-generate device controllers for these devices.

*You can design your own IP (Intellectual Property) in your SOPC module!



*How to talk with your IP?

Your IP Design (Assume you attach your IP at the bus address 69, also, you want 5 registers which can be accessed from software)



* In your C/C++ program, if you want to read Interface Register 3, you can simply call the function IORD(69, 3) in your software core, which will return the value of Interface Register 3.
* Again, if you want to write the value 1 into Interface Register 4, you can call the function IOWR(69, 4, 1) in your software core.

Register Map for IORD/IOWR

* For other programmers who want to control your IP, because they don't know what's inside your hardware, it's impossible for them to control your IP by calling the IORD(base, offset) and IOWR(base, offset, data). You can provide a C/C++ file which defines each Interface Register IDs as "understandable" names for other programmers.
* If you design a 2-to-4 decoder, which will convert (x, y) to (a, b, c, d), you may need 6 registers which can be accessed from CPU. Assume your Interface Registers 0~3 are a~d and 4~5 are x~y, you can write down:
  * #define THIS_IS_MY_DECODER_OUTPUT_A 0
  * #define THIS_IS_MY_DECODER_OUTPUT_B 1
  * #define THIS_IS_MY_DECODER_OUTPUT_C 2
  * #define THIS_IS_MY_DECODER_OUTPUT_D 3
  * #define THIS_IS_MY_DECODER_INPUT_X 4
  * #define THIS_IS_MY_DECODER_INPUT_Y 5
* Now with the Register Map you provide, assume your 2-to-4 decoder is attached on bus address 117, other programmers can set the input x = 1 by calling IOWR(117, THIS_IS_MY_DECODER_INPUT_X, 1).

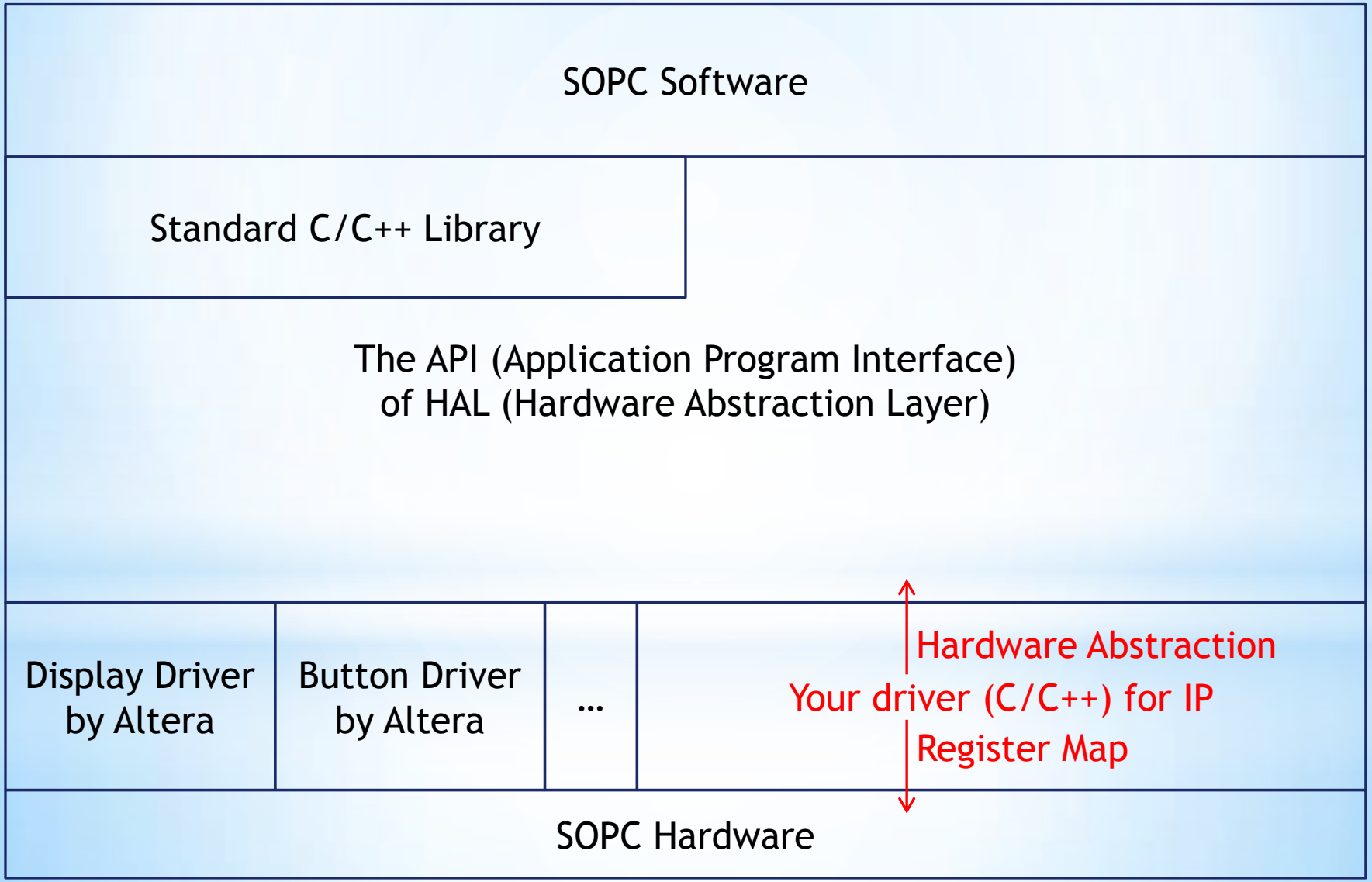
Hardware Abstraction for IP

- However, even with the Register Map of your 2-to-4 decoder, software designer still need to think like a hardware designer because they still handle the register-level access. To avoid it, you can write some C/C++ functions which “looks like” normal C/C++ functions, but in real, performs the register-level access, for example:

```
int get_condition_a(void *base, int from_x, int from_y)
{
    IOWR(base, THIS_IS_MY_DECODER_INPUT_X, from_x);
    IOWR(base, THIS_IS_MY_DECODER_INPUT_Y, from_y);
    return IORD(base, THIS_IS_MY_DECODER_OUTPUT_A);
}
```

- Now with the Hardware Abstraction you’ve done, programmers feel no different between accessing a normal C/C++ function or accessing a real hardware.

Because your IP is not designed by Altera Corporation, Altera Nios II EDS doesn't have the component to auto-generate your IP driver. You need to provide your Register Map and Hardware Abstraction in Nios II EDS as your IP driver.



*To sum up, a more complete SOPC design flow is:

1. Use Altera Quartus II to write your IP module file (Verilog/VHDL). Make sure you have included Interface Registers in your module.
2. Add your IP module into Altera SOPC Builder as a new component.
3. Use Altera SOPC Builder to design your SOPC module. Remember to attach your IP to the bus. Then, generate all Verilog/VHDL files of your SOPC module.
4. Use Altera Quartus II to write your top module file (Verilog/VHDL). Make sure the I/O pins are connected to your SOPC module.
5. Use Altera Quartus II to do Pin Assignment according to your Terasic DE2-70 board. Then, burn all Verilog/VHDL files into Cyclone II chip.
6. Use Altera Nios II EDS to auto-generate your software core based on your SOPC design in Altera SOPC Builder.
7. Use Altera Nios II EDS to write your IP driver file (C/C++) including Register Map and Hardware Abstraction.
8. Use Altera Nios II EDS to write your SOPC software (C/C++) based on the software core and your own IP driver. Then, burn all C/C++ files into the memory (e.g. SDRAM) on Terasic DE2-70 board.

*Done. Congratulations!

*Review Questions

1. Why we need to let our hardware run software? (Hint: Chip Area) How does this hardware work? (Hint: CPU, Memory)
2. Why we choose to use Altera SOPC Builder instead of designing all things by ourselves? (Hint: Working Time) What is the difference of hardware structure between these two ways? (Hint: Top Module)
3. Why we need to design our IP hardware? (Hint: Speed, Non-Altera) What is the major difference between designing a IP on a bus and designing a normal module in Logic Design Lab? (Hint: Interface)
4. Why we need to write a Register Map? (Hint: Black Box)
5. Why we need to do Hardware Abstraction? (Hint: Thinking Way)