Hardware/Software Co-design using Primitive Interface

Navin Chourasia 1 , Puran Gaur 2

1. M Tech Scholar Department Of Electronics & Communication, NIIST Bhopal, India
Email: navin_chourasia@yahoo.com

2. Asst. Prof. Department Of Electronics & Communication, NIIST Bhopal, India
Email: purangour@rediffmail.com

Abstract

Most engineering designs can be viewed as systems, i.e., as collections of several components whose combined operation provides useful services. Components can be heterogeneous in nature and their interaction may be regulated by some simple or complex means. Interface between Hardware & Software plays a very important role in co-design of the embedded system. Hardware/software co-design means meeting system-level objectives by exploiting the synergism of hardware and software through their concurrent design. This paper shows how hardware & software interfaces can be implemented using primitive interface design.

Key words: *Co-design; Interface, Latch ,Primitive Interface; Processor*

Introduction

Hardware/Software interface design supports the simultaneous of both hardware and software to implement a desired function. It is widely used in design of Embedded System. With hardware and software interface design, a system specification is divided into hardware and software parts according to the system architecture.

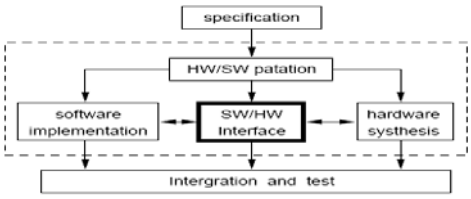

Figure 1: Hardware/Software Co-design Partition

Embedded System usually requires high performance at low cost. To meet such requirements in application, powerful chips, such as Micro-Processors (MP), Micro-controllers (MC) and Digital Signal Processors (DSP), and high level programming languages are used for the software design.

Primitive Interface Design

The primitive interface design implements input and output, where the input interface supports the reading function and the output interface support the writing function

Input Interface for Processor

The input interface for processor is consisting of the hardware interface and the software interface.
The behavior of the hardware interface can be described by a Verilog program as shown.

```
1 ‘define BUSwidth 8 // Width of BUS

2 module Data_input( x, y, g );

3 input [‘BUSwidth-1:0] x;

4 output [‘BUSwidth-1:0] y;

5 input g;

6 assign y = (g) ? x : {‘BUSwidth{1’bz}} ;

7 endmodule
```

Here line 1 defines the Bus width. Line 2 introduces the module ‘Data_input’ as an input hardware interface, where the parameter x stands for input, y for output and g for control. A digital device “Latch “has the same behavior as the hardware interface

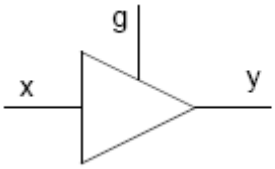

Figure 2: Latch

The gate has its input wire x directly connected to the input source, whose selection is determined by the address value sent by the software input interface shown below. Its output wire y ! and control wire " g are linked to the data Bus and the control Bus respectively. If g is valid, x ! is assigned the value of y, otherwise y ! is equal to z 3 (which denotes high impedance).

The software interface reads the data from the hardware interface. It is implemented as a procedure Data_Read (v,c,RD) where its value parameters c and RD identify the selected address and the control signal and the result parameter 8 is the holder of the incoming message.

$$\begin{aligned} Data\_Read(v, c, RD) \triangleq \quad & addr\_Bus := c; \\ & ctrl\_Bus := RD; \\ & \Delta\delta; \\ & v := data\_Bus; \\ & ctrl\_Bus := idle; \end{aligned}$$

In the multi-input hardware interface, the selection of the input can be implemented in two ways. One is based on the control signal, see figure 3. The other one is based on the input source, see figure 4.

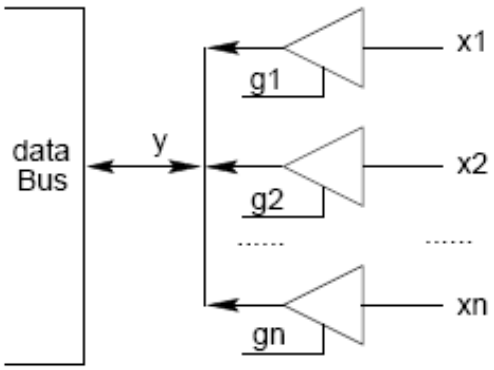

Figure 3: Control based input

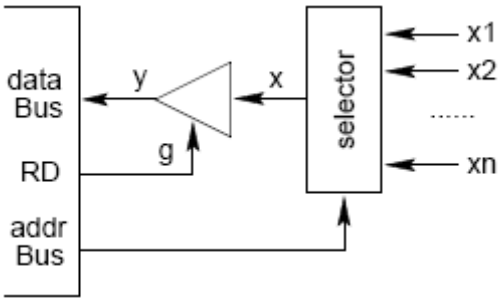

Figure 4: Source based input

In figure 3, the multi-input hardware interface is a collection of the latches connected to the bus in parallel. As in the case of single three-state gate, the inputs x1,…..xn are connected to their input sources. The control wire gi becomes valid only when both RD _ signal and the corresponding address are available. The behavior of this multi-input hardware interface can be modeled by

$$M\_I \triangleq \begin{array}{l} if \\ \left(\begin{array}{l} g_1 \rightarrow y := x_1; \\ g_2 \rightarrow y := x_2; \\ ..... \\ g_n \rightarrow y := x_n; \\ \neg g_1 \wedge \neg g_2 \wedge ... \wedge \neg g_n \rightarrow y := z; \end{array} \right) \\ fi \end{array}$$

Where the input message passed to the data Bus is selected by the control signals gi. To avoid interference between multiple inputs, we must ensure that only one of control signals can be active at any time.

In figure 4, the multi-input hardware interface is built by a latch together with a selector. The input wires P x1,…..xn are connected to their corresponding input resource. One of them is selected by the address and linked to the input wire _ of the three-state gate. The control wire " is solely controlled by the RD signal. This multi-input hardware interface can be modeled by the program

$$M\_I \triangleq y := \begin{array}{l} if \\ \left(\begin{array}{l} (addr\_Bus = addr_1) \rightarrow x_1; \\ (addr\_Bus = addr_2) \rightarrow x_2; \\ ..... \\ (addr\_Bus = addr_n) \rightarrow x_n; \\ (addr\_Bus \notin addr_{1..n}) \rightarrow z; \end{array} \right) \\ fi \end{array} \lhd RD \rhd z$$

Where the input message assigned to! Is selected by the address. To avoid interference between multiple
Inputs, users must ensure that only one of addresses can be active and must be active at any time.

The procedure Data_Read(v,c,RD) can also be used to implement the multi-input software interface.
These input interfaces for processor are the typical asynchronous interfaces. They require that the input
data should be ready before the processor executes the reading function.

Output Interface for Processor

The output interface for processor is composed of the hardware interface and the software interface. The hardware interface can be described by the Verilog program

```
1 ‘define BUSwidth 8; // Width of BUS

2 module Data_output( x, y, k );

3 input [‘BUSwidth-1:0] x;

4 output [‘BUSwidth-1:0] y;

5 input k ;

6 reg [‘BUSwidth-1:0] y;

7 always @( k or x or y )

8 begin

9 if (k) y=x; else y=y;

10 end

11 end module
```

A digital device ‘Latch’ has the same behavior as the hardware interface

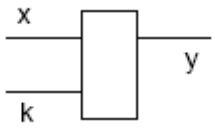

Figure 5: Latch

Where the output wire y ! is directly connected to output destination. The input wire x and the control wire k are linked to the data Bus and the control Bus respectively. When k is valid, the value of x propagates to the output wire y, otherwise y remains unchanged.

The software interface writes data to the hardware interface, and is implemented by the procedure Data_Write(e,d,WR)

$$Data\_Write(e,d,WR) \triangleq \begin{array}{l} addr\_Bus := d; \\ data\_Bus := e; \\ ctrl\_Bus := WR; \\ delay(\delta); \\ ctrl\_Bus := idle; \end{array}$$

where the parameter e gives the output message, d delivers the selected address and WR outputs a control signal for writing. The procedure is invoked when an application program wants to perform the writing function

For the multi-output interface, the selection of the output in the hardware interface can be implemented in two ways. One is based on the control signal, see figure 6. The other one is based on the output destination, see figure 7. In figure 8, the multi-output hardware interface is composed of a collection of

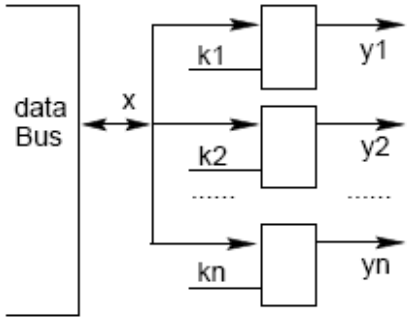

Figure 6: control-based output

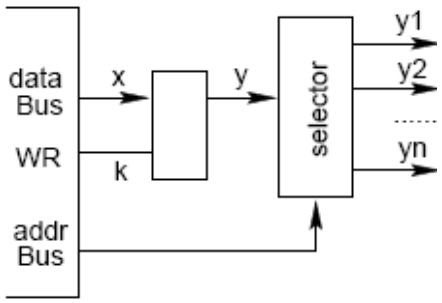

Figure 7: destination based output

the latches connected to the Bus. As in the case of single latch, each output wire yi Q is linked to its own output destination. The control wire ki is valid after both WR signal and the address become available.
The behavior of this multi-output hardware interface can be modeled by the multiple assignments

$$M\_O \triangleq y_1, y_2, \ldots, y_n := x \lhd k_1 \rhd y_1, x \lhd k_2 \rhd y_2, \ldots, x \lhd k_n \rhd y_n$$

In figure 7, the multi-output hardware interface is composed of a single latch connected to the Bus. This kind of the multi-output hardware interface can be modeled by,

$$M\_O \triangleq \begin{array}{l} if \\ \left(\begin{array}{l} (addr\_Bus = addr_1) \rightarrow y_1 \\ (addr\_Bus = addr_2) \rightarrow y_2 \\ \ldots\ldots \\ (addr\_Bus = addr_n) \rightarrow y_n \end{array} \right) := x \lhd WR \rhd y; \\ fi \end{array}$$

Where the output control signal is WR. The output message from the data Bus is determined by the address.

In this interface, at least one of the address must be active at any time.

The multi-output software interface can invoke the procedure Data_Write (e,d,WR) to write the data e to the multi-output hardware interface. All these output interfaces for processor are the typical asynchronous output interfaces. As the simplest case, the constraint of using these interfaces is that the receiver should be ready when the data is output by the processor.

Conclusion

This paper shows how to build the primitive input and output interfaces for processor. The technique discussed in this paper can also be used in design of various links between the components of the hardware/software mixed systems.

References

[1] Qin Shengchao and He jifeng, 'An algebraic Approach to Hardware/Software Portioning', In the proceeding of IEEE conference ICECS'2K, the IEEE press, Lebanon, Dec., 2000.

[2] Rockson Huang. 'Embedded Tools Target MPU/MCU Interface'. Aisys Inc., Electronic News, Oct 9,2000.

[3] Rolf Ernst. 'Codesign of Embedded System: Status and Trends'. IEEE Design & Test of Computers. 1998

[4] Ulrich Golze. 'VLSI Chip Design with the Hardware Description Language VERILOG'. Springer-Verlag Berlin Heidelberg, 1996.

[5] Bob Moore. 'SOC Should Mean 'Software-on-a-Chip' '. CoWare Inc., Electronic News, Oct 23,2000.

[6] M. Chiodo et al. 'Hardware-Software Codesign of Embedded Systems'. IEEE micro, Vol.14, No.4, Jul.-Aug. 1994.

[7] J.Van den Hurk and J.Jess. 'System-Level Hardware-Software Codesign'. Kluwer Academic,1998.

[8] Intel Inc. 'MCS 51 Microcontroller Family User's Manual'. Feb.,1994. http://www.intel.com/design/mcs51/manuals

[9] Xilinx,Inc., 'Development system reference guide'. 2000.

[10] Xilinx,Inc., 'Xilinx Synthesis Technology(XST) User Guide-3.1i'. 2000.

[11] Synopsys,Inc., 'FPGA Compiler II/FPGA Express Verilog HDL Reference Manual'. May 1999.

[12] Xilinx,Inc., 'Xilinx Netlist Format(XNF) Specification'. June 1,1995.

[13] Craig Peacock, 'Interfacing the Standard Parallel Port'. Feb.,1998. http://www.senet.com.au/ cpeacock