Application Note

Interfacing the Z16C32 IUSC with Motorola Processors

AN000802-0801

ZiLOG Worldwide Headquarters • 910 E. Hamilton Avenue • Campbell, CA 95008
Telephone: 408.558.8500 • Fax: 408.558.8300 • www.zilog.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

Totally Logical

INTERFACING THE Z16C32 IUSC WITH MOTOROLA PROCESSORS

DIRECT-REGISTER ADDRESSING SIMPLIFIES SOFTWARE REQUIREMENTS AND INCREASES RELIABILITY

INTRODUCTION

This application note describes an Altera Embedded Programmable Logic Device (EPLD) design that comprises a single-chip interface between a ZiLOG Z16C32 Integrated Universal Serial Controller (IUSC) and a Motorola MC68340 processor.

The Z16C32's ancestor, the Z16C30 USC, was originally designed as a peripheral for the ZiLOG Z8000 processor. The USC and IUSC feature seamless/*glueless* interface to the Z8000's successors, the Z16C0x processors. These processors feature a multiplexed address/data bus. Other processors that offer multiplexed address and data, such as the Intel 80186, can also be interfaced to the Z16C32 with fairly minimal glue logic.

Processors with separate address and data buses, such as those from Motorola, can easily be interfaced to the USC and IUSC if the IUSC's *indirect register addressing* feature is used. Each time that software must access any of the IUSC's numerous registers, a register address must be written into the Z16C32. While indirect addressing allows a simple hardware interface, it makes the software more complex and degrades performance. Indirect addressing also requires software to disable interrupts prior to writing the register address, in addition to reenabling interrupts after reading or writing the register. Programmers must remember to disable and reenable before and after every register access, or use centralized *read and write register* subroutines, which further degrade performance.

Typically IUSC-based products and applications that use indirect register addressing take more time to develop and finalize than those that use direct register addressing (the IUSC User Manual notes this difference in Chapter 2). For complete *Bus Demultiplexing*, which indicates the conversion between a nonmultiplexed bus and the IUSC's multiplexed bus, ZiLOG's IUSC User Manual falls short of the detail necessary to build such an interface. To aid in your development and understanding of Bus Demultiplexing, this Application Note is intended to provide that detailed interface between an IUSC and a typical Motorola processor, the MC68340.

OVERVIEW

Figure 1 shows the interconnection of a Z16C32 IUSC and a Motorola MC68340 processor with an Altera EPM7128EQC100 EPLD. When the 68340 is in control of its address and data buses, the 7128 converts its bus cycles for the IUSC, including time-multiplexing of the address and data. When the IUSC takes control of the bus, the 7128 demultiplexes its address and data onto the 68340's separate address and data buses, so that memory can be configured for the 68340 bus. This configuration occurs without much regard for the characteristics of the IUSC.

Appendix A includes the logic equations for the Altera 7128, written in Altera's Hardware Description Language (AHDL). For those who want to convert this logic to other languages and devices, or for those not familiar with AHDL, a short summary of AHDL appears on the page 3.

Appendix B shows the results of simulation of the 7128 logic using the Altera simulator.

Note: The logic has been simulated according to the author's understanding of the Motorola 68340. It has been used successfully by at least one customer; however, readers should critically examine the schematic and both Appendices, in view of their knowledge of their particular processor and application.

All Signals with an overline, "$\overline{\ }$" are active Low, for example, B/$\overline{W}$ (WORD is active Low); $\overline{B}$/W (BYTE is active Low, only).

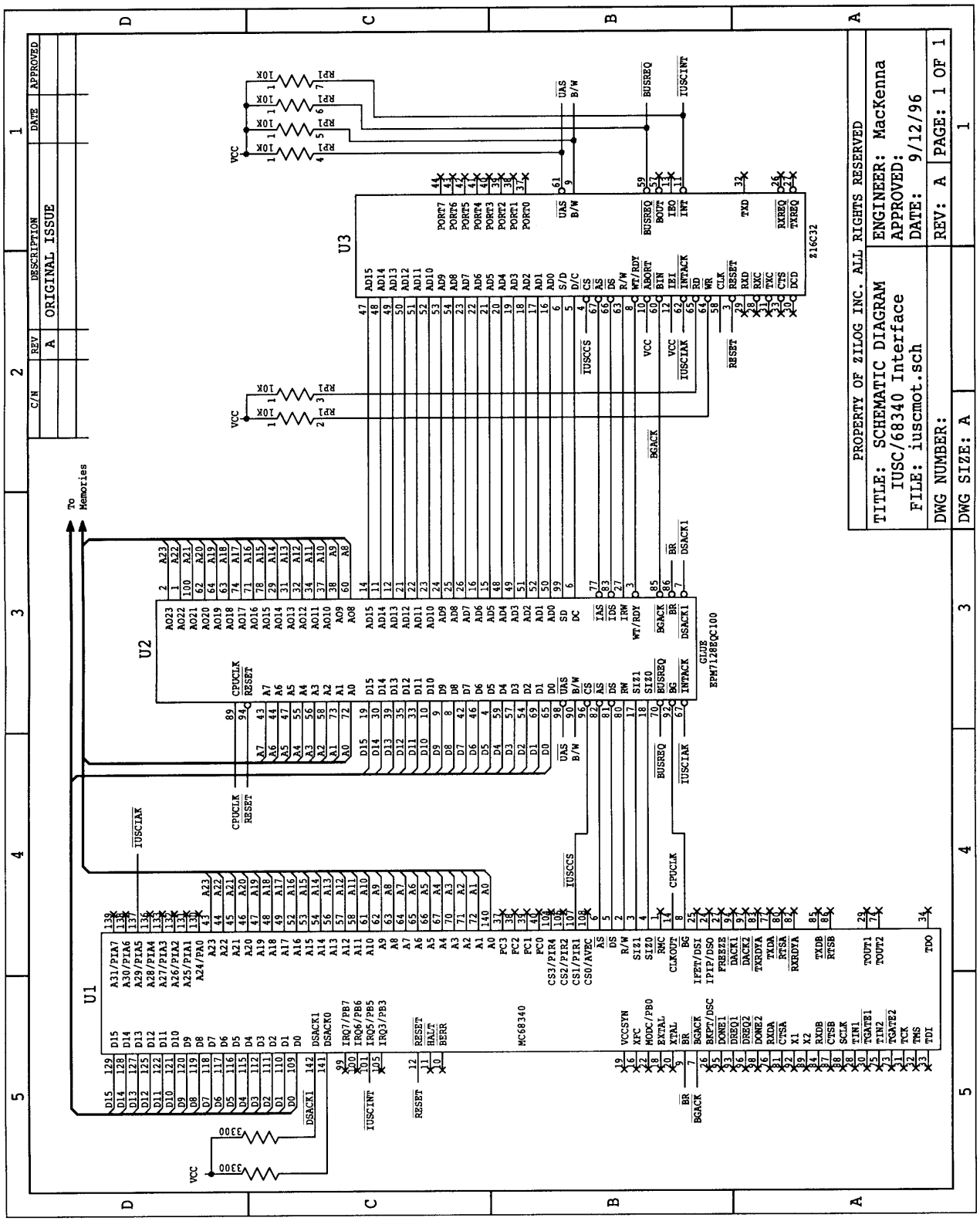

Figure 1. Z16C32 Schematic Diagram

ABOUT THE SCHEMATIC

Many pins on the 68340 are left unconnected in the schematic. This arrangement is intended to show just the hookup of the three parts (68340, 7128 and Z16C32). Similarly, the serial interface pins and port pins of the IUSC are left unconnected.

Four tri-statable outputs of the IUSC are pulled up with 10K resistors to avoid floating inputs to the 68340 or 7128. Two unused input/outputs of the IUSC ($\overline{RD}$ and $\overline{WR}$) are similarly pulled up. Pull-up resistors are not included on signals between the 7128 and IUSC, such as $\overline{AS}$, $\overline{DS}$, R/W, and $\overline{WT}/\overline{RDY}$, because the two parts should exchange the driving of these signals without long dormant periods. Neither the 7128 nor the IUSC drive the signal with the Pull-up resistors included.

ABOUT APPENDIX A

The SUBDESIGN block near the start of Appendix A lists the inputs and outputs of the design. The VARIABLEs block defines *buried* (internal) signals and registers. The body of Appendix A consists of logic equations, which are arranged in no particular order.

AHDL Summary

- ! indicates negation
- & indicates logical And
- # indicates logical Or

Unless parentheses indicate otherwise, the sequence is !, followed by &, then #.

- name[n..m] is an n–m + 1 wide bus, whose individual signals can be referred to without brackets, for example, a6 rather than a[6].
- Comments are enclosed in % signs
- NODEs are intermediate signals
- DFFs are D-type flip-flops that clock their .d inputs on rising edges of their .clk inputs, and have low-active asynchronous .prn presets and .clrn clear inputs.
- LATCHes pass their .d inputs to their outputs when their .ena inputs are High (transparent latches).
- TRIs drive their first operand to their outputs, when their second (enable) operand is High.

ABOUT APPENDIX B

The simulation waveforms of Appendix B are arranged with all of the inputs of the IUSC first: [I]RESET through [I]ad[15..0]. Next, [O]br through [O]ad[6..0] are the outputs of the IUSC.

Many of the pins of the Z16C32 are input/outputs, and therefore appear twice; the upper [I] instance shows whether outside influences are driving the pins, while the second [O] instance includes input states, including states driven by the IUSC. The reader should take care to observe the [O] trace to see how the Altera part is operating, and the [I] trace to see how it is being stimulated.

Finally, [B]drv_wait through [B]ar[23..0] shows the state of the buried (internal) registers and latches in the Z16C32.

High impedance states are shown as either ZZ… for buses, or a thick line between a High and a Low for non-bused signals. XX… for outputs and buried registers indicates an *undefined* state, which typically results from an internal register or latch that is not preset or reset. This condition may also result from enabling a driver, the input of which is in Z state.

Notes at the bottom of each page show the type of cycle or event occurring above. Initially, the part is Reset. The 68340 runs cycles not targeted for the IUSC, and follows by performing a read and write for the IUSC. The IUSC then requests bus control, the 68340 grants it, and the IUSC runs a read and write cycle with memory. Finally, the IUSC returns bus control to the 68340, which then acknowledges an interrupt from the IUSC.

DESCRIPTION OF THE LOGIC EQUATIONS

The following descriptions follow the order of the logic equations in Appendix A.

$\overline{BGACK}$ is asserted Low to the 68340 and IUSC when the IUSC drives $\overline{BUSREQ}$ low, and the 68340 asserts $\overline{BG}$ low. Thereafter, it is held Low until the IUSC drives $\overline{BUSREQ}$ high.

$\overline{BR}$ is asserted Low to the 68340 when the IUSC drives $\overline{BUSREQ}$ until the 68340 asserts $\overline{BG}$.

The ar[23..16] latches open to capture the high-order address on ad[7..0] from the IUSC, when it drives $\overline{UAS}$ low. The ar[15..0] captures the low-order address from ad[15..0], when the IUSC drives its $\overline{AS}$ pin (called IAS to distinguish it from the 68340's $\overline{AS}$) low.

When $\overline{BGACK}$ High indicates the 68340 is controlling the bus, the part drives the states of A7 and A6 onto the IUSC S/$\overline{D}$ and D/$\overline{C}$ pins, thereby selecting between serial registers, (Tx DMA registers and Rx DMA registers). Thus, the IUSC occupies a 256-byte block in 68340 address space.

The asd1 and asd2 provide time-delayed versions of 68340 $\overline{AS}$, and are used to time-multiplex the 68K address and data into the IUSC.

IAS, IDS, and IRW (IUSC $\overline{AS}$, $\overline{DS}$, and R/$\overline{W}$) are driven to the IUSC when $\overline{BGACK}$ High indicates the 68340 has bus control. Within this time, IAS is driven Low between the time $\overline{AS}$ goes Low from the 68K, and when the asd1 flip-flop goes Low about one CPU clock later. IDS is driven Low from when asd2 goes Low one-half CPU clock later, for the duration of the $\overline{DS}$ from the 68K. IRW simply tracks 68K R/W.

The 68K control signals (SIZ1 and SIZ0) are driven from the B/$\overline{W}$ output of the IUSC to indicate byte vs. word transfers, when $\overline{BGACK}$ Low indicates the IUSC has bus control.

$\overline{DSACK1}$ is driven to the 68K to indicate cycle completion, including the 16-bit width of the IUSC, when $\overline{BGACK}$ is High indicating the 68K has bus control. In this case, either $\overline{CS}$ and $\overline{DS}$ are both Low indicating an IUSC register access, or $\overline{INTACK}$ is Low indicating an interrupt vector fetch from the IUSC. $\overline{DSACK1}$ is driven from the $\overline{WT}$/RDY signal from the IUSC, which has the proper timing to indicate cycle completion.

Conversely, the $\overline{WT}$/RDY signal to the IUSC is driven from $\overline{DSACK1}$, when the IUSC has bus control. This state assumes that memory is always 16 bits wide, causing the memory to "wait" for the IUSC. This feature should be deleted if memory is fast enough, and never needs to wait for the IUSC when it is bus master. A DFF named drv_wait is used as an RS latch. This DFF drives $\overline{WT}$/RDY from UAS Low, until a High $\overline{BGACK}$ signals that the IUSC has given up control of the bus.

The signal d_out enables drivers from the IUSC's AD bus to the 68K D bus in three cases:

1. $\overline{BGACK}$ is Low indicating that the IUSC has bus control, and IRW is Low indicating that it is writing to memory.
2. $\overline{BGACK}$ is High indicating that the 68K has bus control; $\overline{CS}$ and $\overline{DS}$ are Low and R/W is high, indicating that it is reading from an IUSC register.
3. $\overline{BGACK}$ is high, indicating that the 68K has bus control, and $\overline{INTACK}$ is low, indicating that the 68K is fetching an interrupt vector from the IUSC.

The signal d_in enables data from the 68K D bus to the IUSC's AD bus, in two cases:

1. $\overline{BGACK}$ is low, indicating that the IUSC has bus control; In addition, IRW is High and IDS is Low, indicating that the IUSC is reading from memory.
2. $\overline{BGACK}$ is high, indicating that the 68K has bus control, and $\overline{CS}$ and R/W are both Low, indicating that the 68K is writing an IUSC register.

The signal drv_ad enables drivers to the IUSC's AD bus for address from the start of 68K $\overline{AS}$ until ASD1 goes Low about one CPU clock later. The same is true for data when d_in is High as described above.

When AD6 though AD0 is driven in from the 68K, it is multiplexed between address and write data by the signal asd2, which satisfies the timing requirements of the IUSC.

Note: AD6 is driven from SIZ0 in the address time to allow the 68K of software to access either bytes or words. This capability is very important when writing certain IUSC registers.

The final block of equations drive AD[15..7] from D[15..7], and should be enabled by d_in rather than drv_ad. Because the Altera 7000E family allows only 6 tri-state enable terms in one part, drv_ad is used instead of d_in.

CONCLUSION

By performing the Altera logic equations contained within this document, and incorporating the power of the Z16C32, a strong single-chip interface can be obtained using a variety of well-known processors, including those from Motorola or Intel.

APPENDIX A

User Friendly Glue: Z16C32 IUSC/Motorola 68340

```
% An Altera FPGA to act as the interface between ZiLOG Z16C32 IUSC and Motorola MC68340
     processor bus.


Begun by CAM, ZiLOG  2/23/98 %


SUBDESIGN IUSCMOT ( cs, busreq, bg, uas, cpuclk, reset, bw, intack : INPUT;
     ao[23..8], br, bgack, sd, dc, siz1 : OUTPUT;
     a[7..0], d[15..0], ad[15..0], ias, as, ids, ds, irw, rw,
     siz0, dsack1, wait_rdy : BIDIR; )


VARIABLE
     d_out, d_in, a_to_ad, drv_ad : NODE;
     asd1, asd2, asff, drv_wait : DFF;
     ar[23..0] : LATCH;


BEGIN
     % Make BGACK, assuming the IUSC is the only requester %
     bgack = !reset
                    #busreq
               # (bg & bgack);


          % Make BR for 68K %
     br = !reset # busreq # !bgack;


% Address latching when IUSC is bus master %
     ar[23..16].d = ad[7..0];
     ar[23..16].ena = !uas;
     ar[15..0].d = ad[];
     ar[15..0].ena = !bgack & !ias;


% Drive S/D and D/C from A7-A6 in slave mode %
     sd = TRI(a7, bgack);
     dc = TRI(a6, bgack);


% Delayed versions of address strobe from processor %
     asd1.d = as;
     asd1.clk = !cpuclk;
     asd1.prn = bgack;
```

```
     asd2.d = asd1;
     asd2.clk = cpuclk;
     asd2.prn = bgack;
     a_to_ad = bgack & !as & asd1;



% Drive IAS, IDS, IRW from processor in slave mode %
     ias = TRI(as # !asd1, bgack);
     ids = TRI(asd1 # asd2 # ds, bgack);
     irw = TRI(rw, bgack);



% Synthesize a 68K type AS in master mode %
     asff.d = VCC;
     asff.clrn = ias;
     asff.prn = reset;
     asff.clk = ids;         % negate 68K /AS at end of IUSC /DS %
     as = TRI(asff, !bgack);



% Drive 68K DS and R/W from IUSC in master mode %
     ds = TRI(ids, !bgack);
     rw = TRI(irw, !bgack);



% Drive SIZ[1..0] in master mode %
     siz1 = TRI(!bw, !bgack);
     siz0 = TRI( bw, !bgack);



% Drive 68K DSACK1 (16 bit port) in slave mode %
     dsack1 = TRI(wait_rdy, bgack & !cs & !ds
                    # bgack & !intack);



% Drive IUSC /WAIT line in master mode %
     drv_wait.d = VCC;
     drv_wait.clk = VCC;
     drv_wait.prn = uas;
     drv_wait.clrn = !bgack;
     wait_rdy = TRI(dsack1, drv_wait); % 16 bit memory only %



% Ad bus to d bus (slave and master) %
     d_out = !bgack & !irw
     # bgack & !cs & rw & !ds
           #  bgack & !intack;
```

```
% D bus to ad bus (slave and master) %
     d_in = !bgack & irw & !ids
         #  bgack & !cs & !rw;



% AD6-0 <-- address or data %
     drv_ad = a_to_ad # d_in;



ad6 = TRI(siz0 & asd2 # d6 & !asd2, drv_ad);
     ad5 = TRI(a5   & asd2 # d5 & !asd2, drv_ad);
     ad4 = TRI(a4   & asd2 # d4 & !asd2, drv_ad);
     ad3 = TRI(a3   & asd2 # d3 & !asd2, drv_ad);
     ad2 = TRI(a2   & asd2 # d2 & !asd2, drv_ad);
     ad1 = TRI(a1   & asd2 # d1 & !asd2, drv_ad);
     ad0 = TRI(a0   & asd2 # d0 & !asd2, drv_ad);



  % Other tri-state drivers %



     ao23 = TRI(ar23, !bgack);
     ao22 = TRI(ar22, !bgack);
     ao21 = TRI(ar21, !bgack);
     ao20 = TRI(ar20, !bgack);
     ao19 = TRI(ar19, !bgack);
     ao18 = TRI(ar18, !bgack);
     ao17 = TRI(ar17, !bgack);
     ao16 = TRI(ar16, !bgack);
     ao15 = TRI(ar15, !bgack);
     ao14 = TRI(ar14, !bgack);
     ao13 = TRI(ar13, !bgack);
     ao12 = TRI(ar12, !bgack);
     ao11 = TRI(ar11, !bgack);
     ao10 = TRI(ar10, !bgack);
     ao9 = TRI(ar9, !bgack);
     ao8 = TRI(ar8, !bgack);
     a7 = TRI(ar7, !bgack);
     a6 = TRI(ar6, !bgack);
     a5 = TRI(ar5, !bgack);
     a4 = TRI(ar4, !bgack);
     a3 = TRI(ar3, !bgack);
     a2 = TRI(ar2, !bgack);
     a1 = TRI(ar1, !bgack);
     a0 = TRI(ar0, !bgack);



       d15 = TRI(ad15, d_out);
     d14 = TRI(ad14, d_out);
     d13 = TRI(ad13, d_out);
     d12 = TRI(ad12, d_out);
     d11 = TRI(ad11, d_out);
     d10 = TRI(ad10, d_out);
```

```
     d9 = TRI(ad9, d_out);
     d8 = TRI(ad8, d_out);
     d7 = TRI(ad7, d_out);
     d6 = TRI(ad6, d_out);
     d5 = TRI(ad5, d_out);
     d4 = TRI(ad4, d_out);
     d3 = TRI(ad3, d_out);
     d2 = TRI(ad2, d_out);
     d1 = TRI(ad1, d_out);
     d0 = TRI(ad0, d_out);


% The following should be enabled by d_in, but Altera MAX7000E family only allows 6 output
     enable terms %
     ad15 = TRI(d15, drv_ad);
     ad14 = TRI(d14, drv_ad);
     ad13 = TRI(d13, drv_ad);
     ad12 = TRI(d12, drv_ad);
     ad11 = TRI(d11, drv_ad);
     ad10 = TRI(d10, drv_ad);
     ad9 = TRI(d9, drv_ad);
     ad8 = TRI(d8, drv_ad);
     ad7 = TRI(d7, drv_ad);

END;
```

APPENDIX B

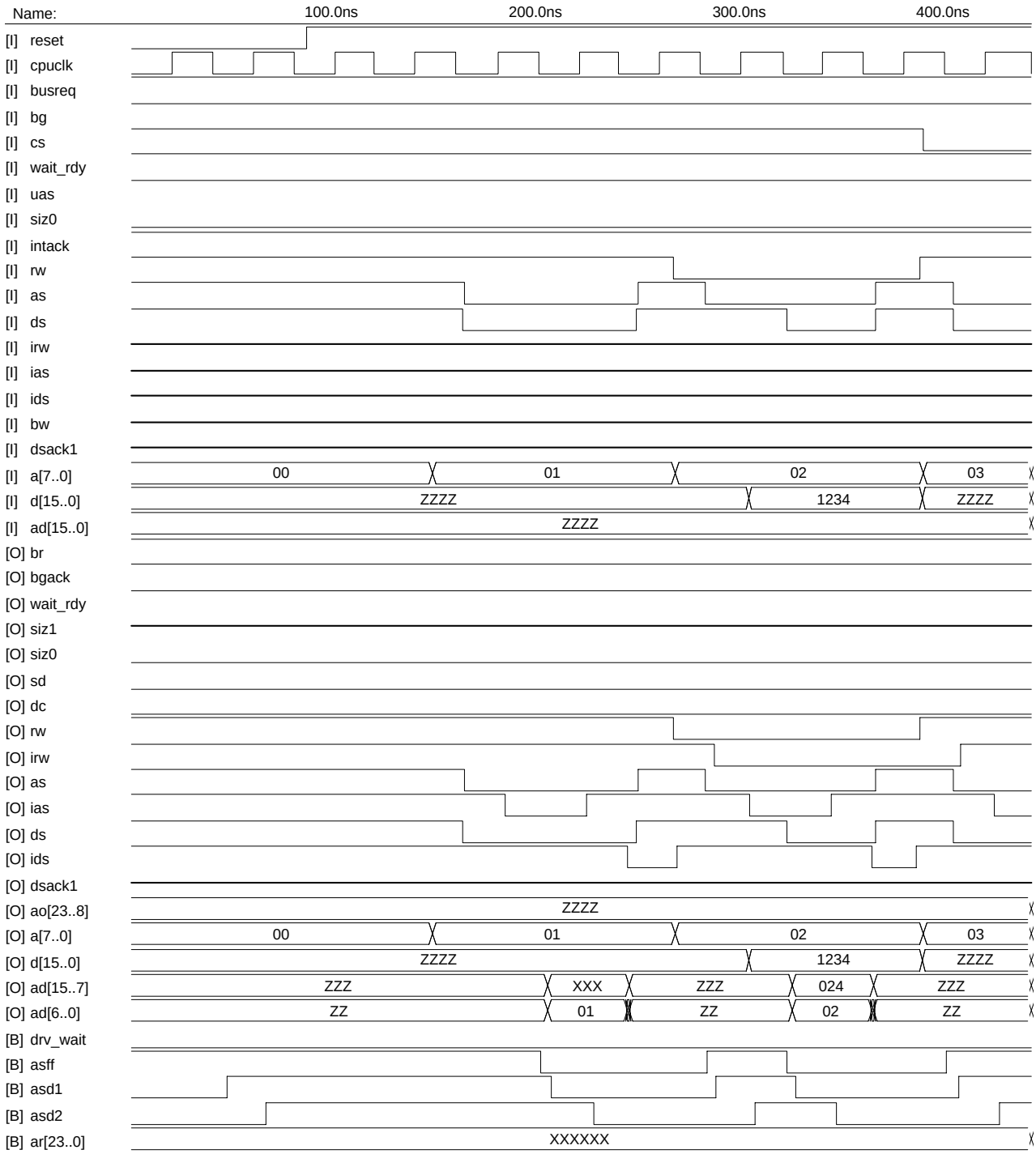

Figure 2. Simulation Waveforms (1 of 5)

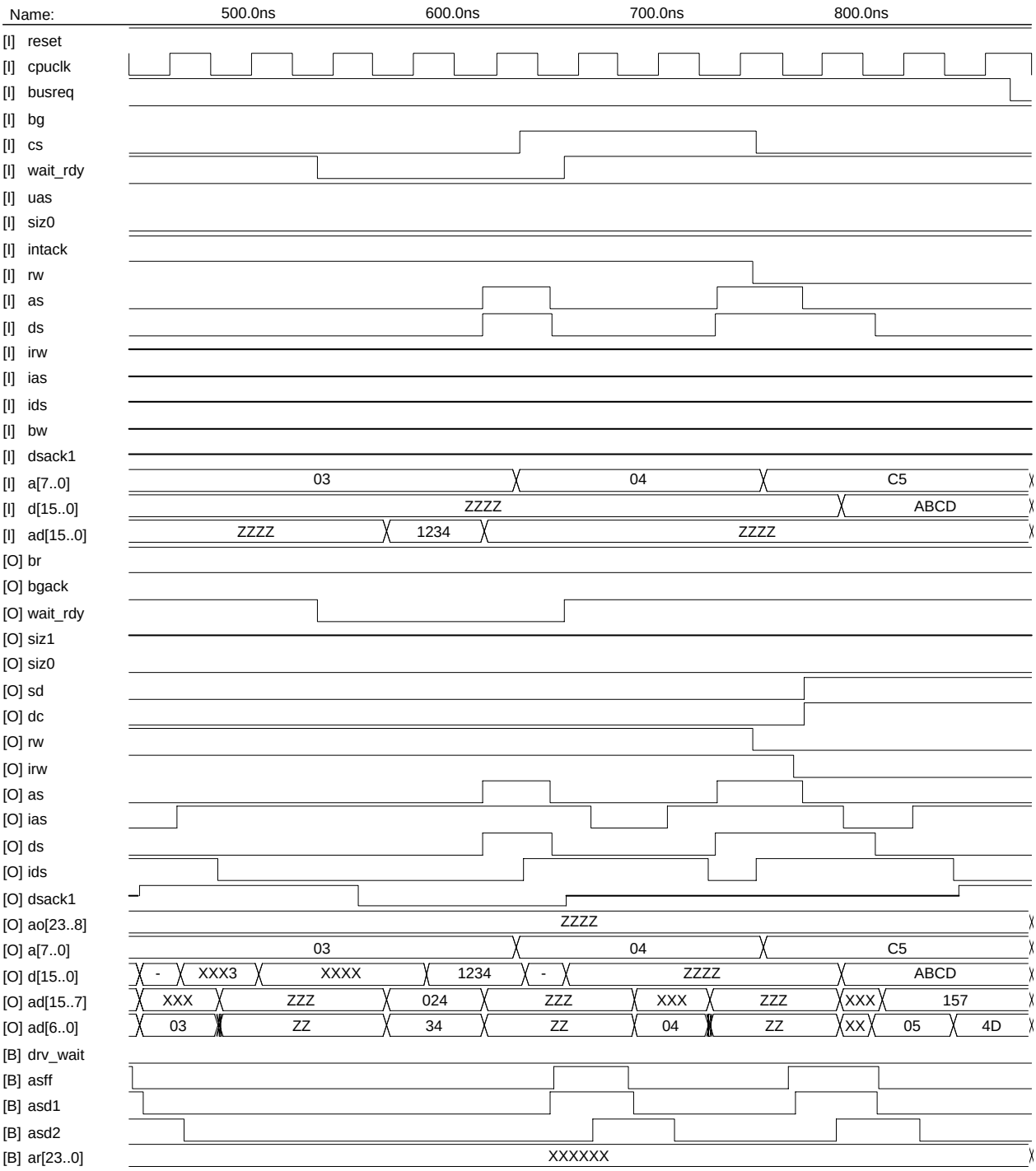

Figure 3. Simulation Waveforms (2 of 5)

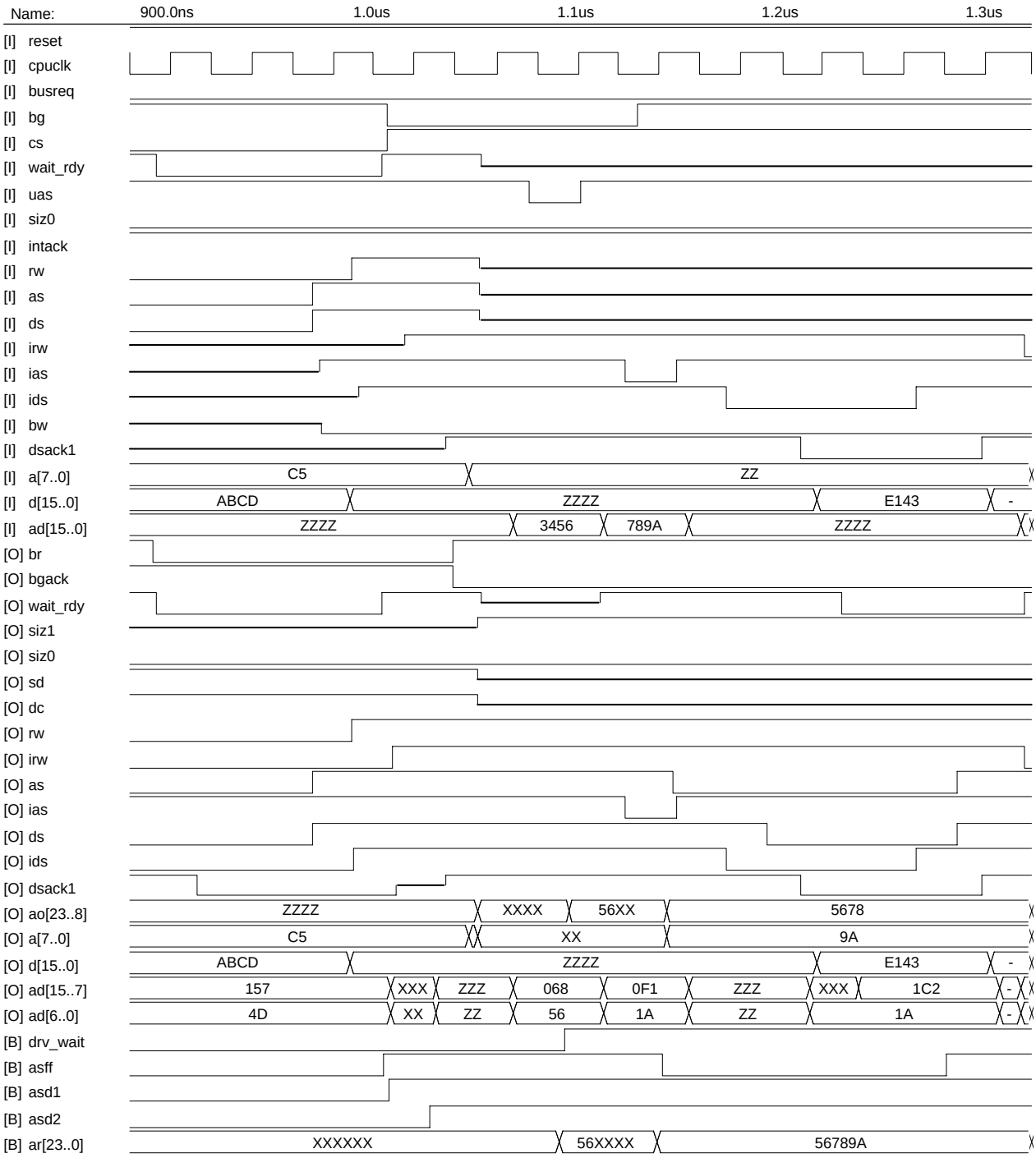

Figure 4. Simulation Waveforms (3 of 5)

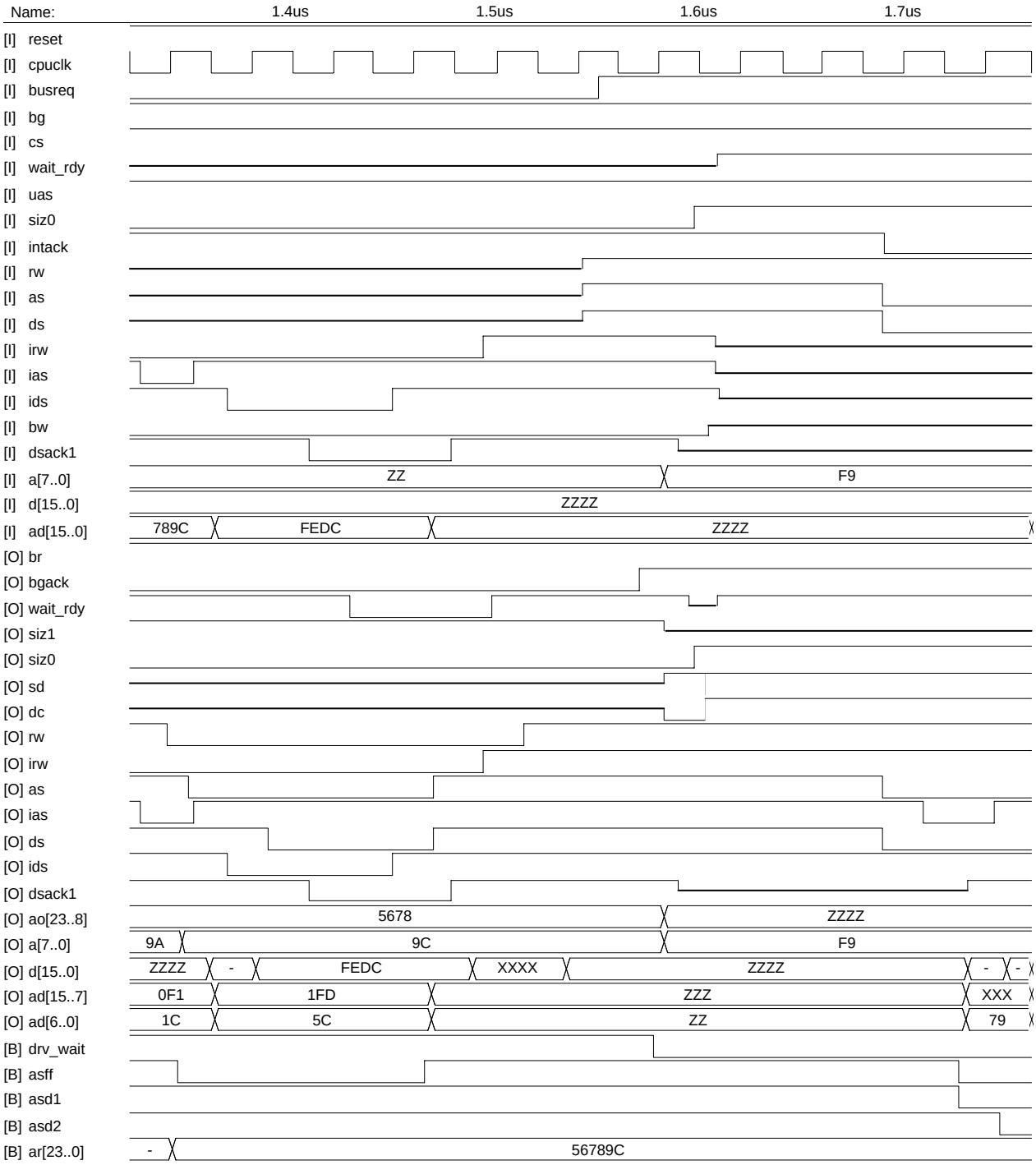

Figure 5. Simulation Waveforms (4 of 5)

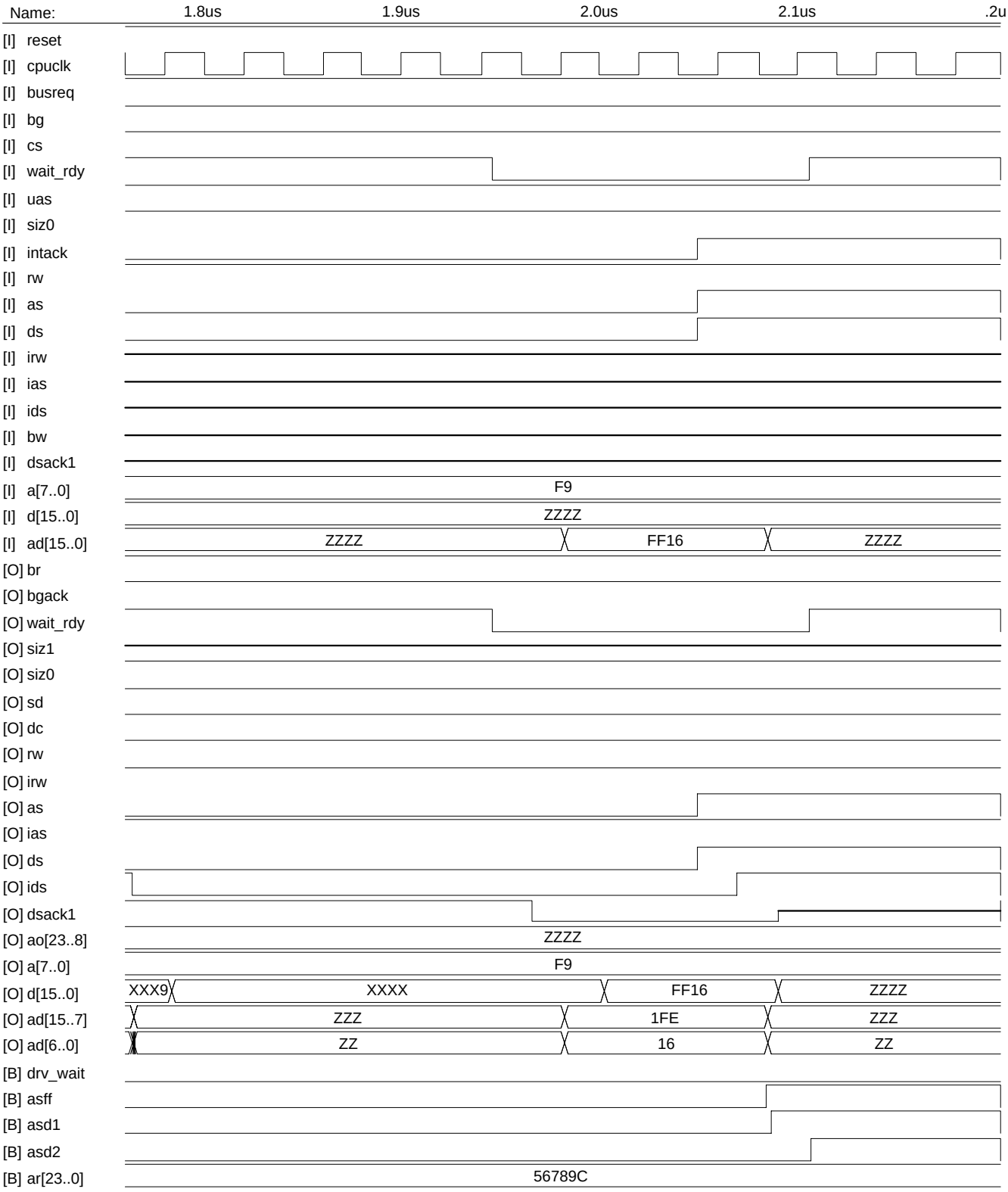

Figure 6. Simulation Waveforms (5 of 5)