

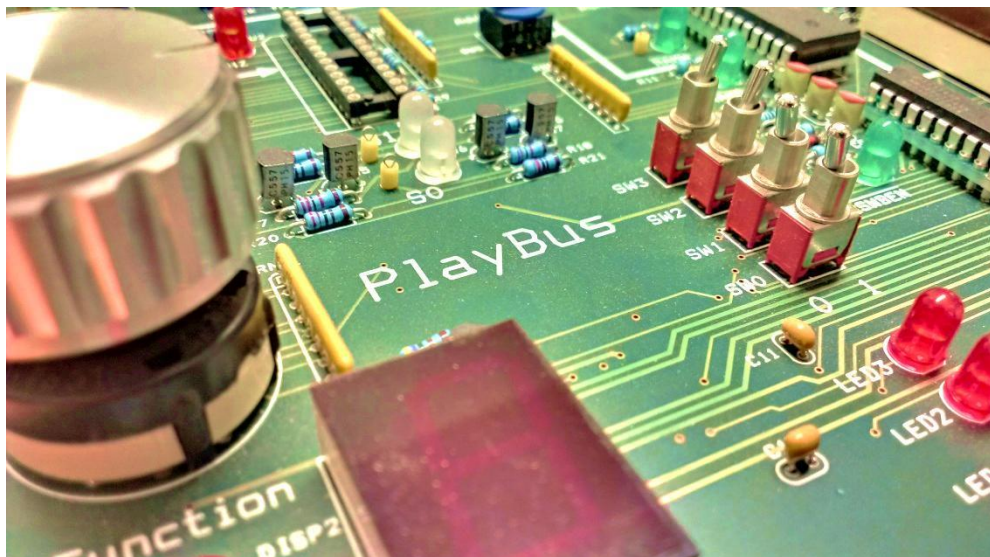
# T4

---

## Bus Operation and Control

---

In this lab, you will understand how bidirectional busses work, and how tristate logic is used to avoid contention on a bus shared by multiple data sources (RAM, EPROM and switches) and sinks (RAM, and LEDs). It will also give you an opportunity to develop your skills in SystemVerilog, and provide you with experience of digital systems which use a separate data, address and control bus.



---

**Schedule**

---

Preparation time : 3 hours

Lab time : 3 hours

---

**Items provided**

---

Tools :

Components :

Equipment : PlayBus (with ispGAL22V10), Oscilloscope, Digital Testbed

Software : Modelsim, Synplify, ispLever

---

**Items to bring**

---

Essentials. A full list is available on the Laboratory website at  
<https://secure.ecs.soton.ac.uk/notes/ellabs/databook/essentials/>

**Before** you come to the lab, it is essential that you read through this document and complete *all* of the preparation work in section 2. If possible, prepare for the lab with your usual lab partner. Only preparation which is recorded in your laboratory logbook will contribute towards your mark for this exercise. There is no objection to several students working together on preparation, as long as all understand the results of that work. Before starting your preparation, read through all sections of these notes so that you are fully aware of what you will have to do in the lab.

**Academic Integrity** – *If you undertake the preparation jointly with other students, it is important that you acknowledge this fact in your logbook. Similarly, you may want to use sources from the internet or books to help answer some of the questions. Again, record any sources in your logbook.*

---

**Revision History**

---

July 16, 2013	Mark Zwolinski (mz)	Minor corrections. Appendix 2 Fig corrected
November 20, 2012	Mark Zwolinski (mz)	Revised version of previous D4 lab

---

## 1 Aims, Learning Outcomes and Outline

This laboratory exercise aims to:

- give you an introduction to the practicalities of a digital bus system;
- give you experience of a tristate-able bus, including controlling its operation;
- provide you with opportunities to extend your PLD design skills.

Having successfully completed the lab, you will be able to:

- understand how a bidirectional data bus works;
- demonstrate with waveform diagrams the transfer of data across a data bus;
- understand the differences between a data bus, an address bus and a control bus;
- design a state machine for controlling data transfers between multiple sources and sinks;
- control the reading and writing of RAM data.

In this lab, you will understand how bidirectional busses work, and how tristate logic is used to avoid contention on a bus shared by multiple data sources (RAM, EPROM and switches) and sinks (RAM, and LEDs). It will also give you an opportunity to develop your skills in SystemVerilog, and provide you with experience of digital systems which use a data, address and control bus.

## 2 Preparation

Read through the course handbook statement on safety and safe working practices, and your copy of the risk assessment form. Make sure that you understand how to work safely. Read through this document so you are aware of what you will be expected to do in the lab.

### 2.1 Background – PlayBus

The exercise uses an experimenter unit called “PlayBus” with the block schematic shown in Figure 1 below. This provides a set of data sources and sinks. Data transfers are controlled by an ispGAL22V10 PLD. The RAM memory IC is both a source of data (read) and a sink (write). The other data units are either purely sources (read-only) or purely sinks (write only). PlayBus also has a display which shows the hexadecimal value on the data bus. This display shows ‘Z’ if the bus is not driven. If two or more data sources are active simultaneously, the display shows ‘?’ and the blue “Contention” LED will be on – this also happens at power-up.

Number	Operation
0	Read data from EPROM at address onto bus
1	Read data from RAM at address onto bus
2	Read data from Switches onto bus
3	Copy data from Switches into RAM at address
4	Copy data from EPROM into RAM at address
5	Copy data from Switches into LEDs
6	Copy data from EPROM at address into LEDs
7	Copy data from RAM at address into LEDs

TABLE 1 – PlayBus Functions

PlayBus can perform a total of eight functions. The function is selected by the “Function” switch as shown in TABLE 1 below:

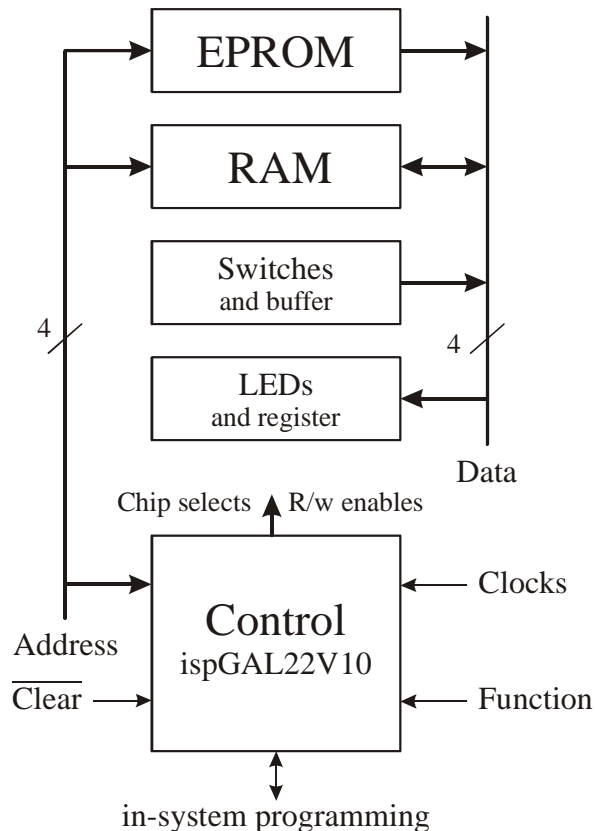


FIGURE 1: Block schematic of PlayBus.

At the start of the exercise, the PLD will be erased. You will use a prepared design to investigate the unit set to functions 0 and 5, then reprogram the PLD in stages to implement as many of the eight functions as possible. Note that functions 0, 1 and 2 are “static” or “stateless” (there is no *sequence* of control signals required for their implementation) while the other functions are “dynamic” and require the control signals to step through a set of states. PlayBus has two “Clocks”: a time-standard clock (for stepping through the states of the control PLD’s state machine) and a “GO” signal which initiates dynamic functions 3:7.

## 2.2 Filestore

As in previous System Verilog exercises, set up a folder on your H: drive server account for all your T4 work. See section 2.1 of the SystemVerilog walkthrough notes for further information. Download into this working directory the System Verilog source file “starter.sv”, constraint file “playbus.sdc” and testbench test\_starter.sv from the Part I labs website.

## 2.3 Understanding TristateL

Using the information from a) your lecture notes, b) elsewhere in these notes, c) your course text *Introductory Digital Design* and d) the ECS Website, answer the following questions:


- ❖ Using ‘Z’ as a possible output state and ‘X’ as a possible input state, what is the full truth-table for a 74HCT125 tristate buffer?
- ❖ What would be the full set of System Verilog statements to implement a function identical to that of a single ’125 buffer? Do not simply call a standard library function! You can start from the following fragment:

```


module clone125 (output logic C, input A, B)
  always_comb
    begin
      end
    endmodule

```

How can you build a 2-to-1 line multiplexer from a 74HCT125 and a 74HCT04?


 *Hint: see section 2.6.3 of Introductory Digital Design if you have forgotten how a multiplexer works.*

The PlayBus switches do not drive the Data Bus directly, but via a 74HCT245 configured as a unidirectional tristateable buffer.


 *Why is the buffer needed?*

*Hint: it is nothing to do with current drive requirements.*

The PlayBus LEDs are not driven directly from the Data Bus, but from a 74HCT373 latch.

 *Why is the latch needed?*

*Hint: it is nothing to do with current drive capabilities.*


 *Extend the 2-input NAND truth table of Table 2 (below) so as to cover situations where an unknown 'X' input can be 'Z' tristate (undriven) as well as 'H' or 'L'.*

IN A	IN B	OUT
L	X	H
X	L	H
H	H	L

TABLE 2 – Truth table for a 2-input NAND gate.

## 2.4 Control signals

Using the ECS Website document “Memory ICs”, and datasheets for the 74HCT245 octal buffer and 74HCT373 octal latch<sup>1</sup>:

 *Complete Table 3 (below) listing the control signals SigA1:SigC3 for the 6116 SRAM, 74HCT245 buffer and 74HCT373 latch, with their corresponding functions FuncA1:FuncC3.*

Note that the same signal and function names may be used for more than one device, and that not all devices have three control signals. The signal names and the functions these signals control on the 2716 EPROM have been entered for you, so that you can see what is required: SigA0 is “/OE” with FuncA0 being “Output enable”, SigB0 is /CS with FuncB0 being “Chip select”.

<sup>1</sup> <https://secure.ecs.soton.ac.uk/notes/ellabs/databook/comps/ics/index.htm>

2716 EPROM		6116 SRAM		'245 Buffer		'373 Latch	
/OE	Output enable	SigA1	FuncA1	SigA2	FuncA2	SigA3	FuncA3
/CS	Chip select	SigB1	FuncB1	SigB2	FuncB2	SigB3	FuncB3
-	-	SigC1	FuncC1				

TABLE 3 – Control signals for PlayBus devices.

From the circuit schematic given in Appendix 2, you should be able to identify the control signals grouped in Figure 1 as “Chip selects” and “R/w enables” as follows:  $n\_RAMO$  (also written as  $\overline{RAMO}$ , to show that it is active LO);  $n\_RAMW$ ;  $CONTEND$ ;  $n\_ROMO$ ;  $n\_SWBEN$ ;  $LEDLTCH$ .



Complete Table 4 (below) showing, for each of the data sources and sinks, the required state or sequence of all the control signals to achieve a read from or write to that source/sink, taking into account the connections and signal names shown in Appendix 2. Assume signal “ $CONTEND$ ” of Appendix 2 is always LO/‘0’. The state of the signals for reading from the RAM is filled in for you as an example. The active signal is shown in **BOLD**.

Data Source	To read from source		Data Sink	To write to sink	
RAM	<b><math>n\_RAMO</math></b> $n\_RAMW$ $n\_ROMO$ $n\_SWBEN$ $LEDLTCH$	<b>LO</b> HI HI HI LO	RAM	SignalA SignalB SignalC SignalD SignalE	state/sequence state/sequence state/sequence state/sequence state/sequence
ROM	SignalA SignalB SignalC SignalD SignalE	state/sequence state/sequence state/sequence state/sequence state/sequence	LEDs	SignalA SignalB SignalC SignalD SignalE	state/sequence state/sequence state/sequence state/sequence state/sequence
Switches	SignalA SignalB SignalC SignalD SignalE	state/sequence state/sequence state/sequence state/sequence state/sequence			

TABLE 4 – Control signal sequences for reading/writing to data sources/sinks.



Sketch the waveforms for the full set of control signals for each of functions 3 through 7 in Table 1. At this stage, you do not need to include the state machine clock or the “GO” signal, but you do need to show clearly the distinct sequence of signal states for the dynamic functions.

Hint: inspection of *starter.sv* should show you how to read from EPROM and switches, and how to write to the LEDs.

## 2.5 Start-up Functionality


Use “starter.sv” and constraint file “playbus.sdc” to create a Synplify project for the PlayBus PLD. The constraint file ensures PLD signals appear on the correct pins – the PCB’s connectivity is fixed! As you did in the SystemVerilog walkthrough exercise, synthesise these source files to an EDIF file starter.edf, then use that as the basis of an ispLever project to generate a downloadable “starter.jed” file. For both projects, the PLD should be “ispGAL22V10-15LK”.

The source files are guaranteed to provide correct behaviour for functions 0 and 5 – verify that this is the case by exercising the design using testbench “test\_starter.sv” in ModelSim. Note how the CK2HZ, GO and state signals relate to the control signal waveforms.

Starter.sv makes all “writing” to data sinks happen under the control of signals which do not “glitch”. This is achieved by having these signals set in the always\_ff process, so that they only change in response to a positive-going clock edge. It is acceptable for data source enabling to be glitchy, as long as these signals are stable throughout any write process, so these signals are defined in the always\_comb process.

## 2.6 Preliminary PLD design

“Save As” starter.sv with a different name, or copy starter.sv and rename the copy, then:

 *Edit this new file for use **in a new Synplify project** (which still uses playbus.sdc) so that it provides functions 1 and 2 as well as 0 and 5*

*Hint: think where you should add code for stateless functions.*

Check this in ModelSim with the “test\_starter.sv” testbench, which contains stimuli sufficient to exercise all PlayBus functions. When you are happy that your design simulates operation of functions 0, 1, 2 and 5 correctly, process the EDIF output in a new ispLever project to produce new downloadable .jed file (which needs to have a different name to starter.jed which you produced from the original starter.sv code!) Make sure that **all** your design, testbench, project and constraint files are kept on removable storage such as USB Flash as well as on your server account, and that both you and your lab partner have copies – this means no wasted lab time if there is any problem with a server or if your partner doesn’t turn up.

---

## 3 Laboratory Work

At the start of the laboratory session, log on to the bench PC and check that you can access and use your design files. If you have any problems with the system tell a member of the lab team straight away.

### 3.1 Preliminary Investigation of Bus Operation

Download your starter.jed into the PlayBus PLD. Check that PlayBus’s “Clock Frequency” switch is set to “2Hz”. Select function 0 and work through all sixteen addresses to confirm the EPROM contents appear on the Data Bus as per Appendix 1. This verifies function 0.

 *Is function 0 working correctly? How have you verified this?*

Monitor the switch buffer and LED latch control signals and confirm that the sequence for function 5 is produced correctly. Confirm that changing the switch value in function 5 does not change the LED display until the GO button has been pressed to initiate the transfer. GO will be active for a period of two to six clock cycles (this varies randomly each time the button is

pressed). The only GO-dependent transitions should be from states `idle_static` and `end_dynamic` of the controller state machine – as is the case for `starter.sv`.

◇ *Is function 5 working correctly? How have you verified this?*

### 3.2 More complex controller operation

Download your pre-prepared controller design into the PLD. Check that functions 0 and 5 still work correctly. Test new functions 1 and 2. Compare the PlayBus control signals with the simulated waveforms produced during your preparation. Investigate any errors, malfunctions or discrepancies. It is very important that you are confident with this basic controller functionality and with the way PlayBus behaves before moving on to more complex aspects of the exercise.

◇ *Are the signals that you observe consistent with the waveforms from your simulations? If not, why not?*

### 3.3 Development of a full-function controller

Save your four-function controller design under a new name, and set up new Synplify and ispLever projects (don't forget that every time you create a new project, you must set the target PLD in ispLever to ispGAL22V10-15LK and ensure that `playbus.sdc` is included in Synplify).

Edit the System Verilog source to add the remaining four functions 3, 4, 6 and 7 to the controller. *We strongly recommend that you only add one new function at a time.* It is also good practice to save each known good design and create new files, filenames and projects for each subsequent stage of the development.

With each development of the controller, use ModelSim and the testbench to validate the new functionality – always check all functions to ensure that known good functions have not been corrupted by your development of new functionality. When (and only when) you are satisfied that a new design simulates without error, process it through to a `.jed` file, download this into PlayBus and confirm that it works correctly. Compare the observed PlayBus control signals with those determined to be appropriate in your preparation (section 0).

◇ *For each of the functions, is the observed behaviour consistent with the simulations? If not, why not?*

Your logbook must contain evidence that you have confirmed correct operation of **all** functions at **every** stage of development. Absence of such notes will result in reduced marks for “Progress” and for “Understanding”. Similarly, you should be able to download and demonstrate every stage of development – not simply the most recent.

Note that it is very easy to create the problem known as “bus contention”, by generating a set of control signals whereby two or more data sources drive the data bus simultaneously. Bus controllers must ensure that only one source (or no source at all) is allowed to drive the bus at any instant. If your design produces bus contention, PlayBus will indicate that there is a problem – the bus display shows ‘?’ and the blue “Contention” LED lights (see the schematic of Appendix 2 for more information). In such situations you must change the function switches to remove the contention, and amend the design to stop it happening. Most real-life systems do not have a way of detecting bus contention, and if it occurs damage is quite common.

#### Important:

- Include `playbus.sdc` in all your Synplify projects for this exercise
- Specify the target device as ispGAL22V10-15LK in all ispLever projects
- Create a new `.sv` source file, new Synplify project and new ispLever project as each function is added



- Don't add states, signals, inputs or outputs to the design
- Never allow any PLD output to be undefined (never have an "if" without an "else" unless an appropriate default assignment has been declared)
- Record in your logbook what is changed at each stage of development
- Check every design with ModelSim and test\_starter.sv before generating a Jedec file and downloading to PlayBus
- Confirm that PlayBus runs all implemented functions correctly for each stage of development and record how this has been done in your logbook
- Record in your logbook any problems, faults, errors or difficulties, together with explanations of how these were overcome

## 4 Optional Additional Work

*Marks will only be awarded for this section if you have already completed all of Section 3 to an excellent standard and with excellent understanding.*

### 4.1 Address-mapped functionality

In a microcomputer system with address, data and control buses, it is usual for the role of PlayBus's "function" signals to be incorporated into the address bus. That is, the various data sources and sinks are "address mapped", with the address bus defining which data source/sink is to provide/take CPU data. Design a PLD where the function signals FUNC2:0 are ignored, and instead address lines ADD3 and ADD2 determine the data source driving the data bus as per Table 5 below – 'Z' indicating the bus should be undriven. Before downloading this design, use function 3 (not function 4 – why not?) to write known data to all 16 RAM addresses, and confirm that they have been written correctly by checking with function 1. Work out what data bus value you expect to see for each of the 16 addresses when the new address-mapped PLD design is downloaded, then check these predictions by observation.

ADD3:2 =	0	1	2	3
Data placed on bus from:	EPROM	RAM	Switches	Z

TABLE 5 – Address-mapped static operation

You may chose to provide address-mapped static operation as an alternative to (static) functions 0, 1 and 2 in your full-function PLD design, leaving (dynamic) functions 3:7 unaffected. That is, signals FUNC2:0 are ignored unless 'GO' is pressed, and functions 0, 1 and 2 are unavailable – being replaced by address-mapped static behaviour.

### 4.2 Using an oscilloscope to observe the behaviour

Look at the behaviour of PlayBus using an oscilloscope. For this purpose, set the Clock Frequency switch to "Fast". This produces a 2.5MHz clock, allowing the delays between stimulus and response to be observed. By using the Agilent mixed-signal oscilloscope, you can see concurrent displays of up to 16 digital signals (eg all the control lines) *and* one or two analogue signals (eg two data bus lines – where the analogue display can show clearly both the Z undriven state and rise/fall times of the data signals). Ask a supervisor for assistance in using the digital channels.



*What can you observe with an oscilloscope, that you cannot see from the PlayBus LEDs?*

**Appendix 1 – PlayBus EPROM contents**

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Data	0	1	3	2	6	7	5	4	C	D	F	E	A	B	9	8

(The relationship between address and data is non-random – do you recognise it?)

The diagram illustrates a digital test-bed for an 8051 microcontroller, featuring various components and their interconnections:

- Address:** Includes a 74HC148 decoder (U9) and a 74HC153 decoder (U6). It also features a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Function:** Includes a 74HC148 decoder (U9) and a 74HC153 decoder (U6).
- Clear State:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Individual indicators and test points:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- LEDs:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- RAM:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- EPROM:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Clock Frequency:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- PlayBus Controller:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Switches:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Bus Monitoring:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Power from Digital Test-Bed or compatible source of +5V:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).
- Data Bus value or condition:** Includes a 74HC147 decoder (U5) and a 74HC147 decoder (U4).

Column	Row 6	Row 5	Row 4	Row 3	Row 2	Row 1	Row 0	Commoned Row 3
Col 0	13	9	14	12	10	7	2	5
Col 1	11	8	13	11	8	5	2	5
Col 2	10	7	12	10	7	4	1	4
Col 3	9	6	11	9	6	3	0	3
Col 4	8	5	10	8	5	2	0	2
Col 5	7	4	9	7	4	1	0	1
Col 6	6	3	8	6	3	0	0	0
Col 7	5	2	7	5	2	0	0	0

For “fast” clock, CK2HZ is LO until the Go button is pressed. That sets “GO” HI, then there will be four cycles of CK2HZ before GO returns LO, after which there is one further cycle of CK2HZ. FOLLO will be inactive throughout the time GO and/or CK2HZ are active. Pressing the Go button within half a second of GO returning inactive has no effect.