Lattice™
Semiconductor Corporation

GAL®6002: 4-to-1 RS232 Port Multiplexer

Introduction

The GAL6002 is the most versatile 24-pin PLD available today. Its FPLA architecture offers buried macrocells, D/E registers, programmable clocks and dedicated input pins which can be individually configured as latches or registers. These features combine to provide the designer with an ideal platform on which to build complex state machines and other complex logic functions.

This application note provides an example of how a GAL6002 can be used in a system and shows how software tools are used to maximize some of the device's unique features. The circuit to be described is a 4-to-1 RS-232 serial port multiplexer (Port MUX). The concept for the circuit arose from the need to replace a mechanical switch used to connect four computers to a high speed laser printer.

The Port MUX application uses every input and output pin, as well as all eight State Logic Macrocells. Other than a single GAL6002, the only ICs needed are RS-232 line driver/receiver chips and a clock source.

Basic RS-232 Protocol

To understand the operation of the port MUX, it is necessary to have a basic knowledge of RS-232 communications protocol.

Though the RS-232 protocol is standardized, its definition is loose enough to allow liberties to be taken in its implementation. When the port MUX was designed, the assumption was made that communication can take place with only four signals: transmit data, receive data, printer ready/busy, and computer ready/busy. In RS-232 parlance these signals are called TxD, RxD, DTR, and DSR, respectively.

An RS-232 link is digital (bistable) in nature, but the voltages used to represent logic ones and zeros are not TTL level. Instead, -12VDC represents a logic one and +12VDC represents a logic zero. Another consideration is that the idle or deasserted state of an RS-232 signal is a logic one, although data is transmitted in its "true" form. A typical single byte transfer can be seen in Figure 1.

Figure 1. TxD During Single Byte Transfer
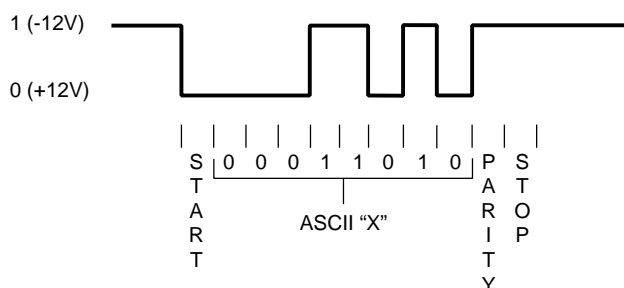
A typical RS-232 data transfer between a computer and a printer would proceed as follows:

1) The printer, being powered-up and ready to accept data, has its DTR line asserted (logic '0'), while its TxD line is idle (logic '1'). The computer, also powered-up but not yet sending data, is in a similar state: TxD is idle and DSR is asserted.

2) When the computer is ready to send a byte of data, it asserts its TxD line for one "bit period." This is called the start bit. A "bit period" is dependent on the data transmission speed (300 baud (bits/sec), 9600 baud, etc.) . After the start bit, seven or eight bits of data will follow, optionally followed by a parity bit, and ending with one or two stop bits (logic '1'). Data is transmitted least significant bit first.

     Note: Asserted = 0 (+12V)

              "Idle"= 1 (-12V)

The condition of TxD and DSR after sending a byte of data is the same as before sending it. From an electrical perspective, there is no indication whether or not the computer is going to send another byte.

3) Somewhere in the middle of the transfer, the printer runs out of paper, or its print buffer fills up, or for some other reason it must suspend communications. When this happens, the printer deasserts its DTR line, telling the computer to stop sending data. When the printer is again ready to accept data, it will reassert DTR.

As alluded to in #2 above, there is no way to tell when the computer is finished sending data. In fact, the computer can be said to have "finished" its transmission after sending only the first byte of a multi-byte transmission.

GAL6002: 4-to-1 RS232 Port Multiplexer

Each subsequent byte transfer can be viewed as an entirely new transaction. Extended periods of time may even elapse between byte transfers if the computer has to access a disk or is interrupted for some reason. Remember, RS232 is an asynchronous communications protocol.

Functional Description

The Port MUX has five ports, numbered 0 to 4. Port 0 is connected to the printer, while ports 1 through 4 are connected to the computers. The port MUX merely acts as an intelligent switch; data flows through it unhindered and unaltered. At any given time, there will always be one (and only one) computer connected to the printer.

Four signals per port are switched: TxD, RxD, DTR, and DSR. This arrangement is known to work for connecting IBM-PC compatible computers to an HP LaserJet. The RS-232 specification has no lack of ready/busy signals, so others could be substituted for DTR and DSR if necessary (CTS and RTS, for example). See Figure 2 for a block diagram of the port multiplexer.

Since RS-232 signal levels are not compatible with TTL levels, line driver/receiver circuits are needed for translation. For this project, Maxim MAX235 Driver/Receiver chips were used, though others devices will work as well. Each MAX235 IC has five drivers and five receivers; two MAX235 ICs are needed to build the Port MUX.

Figure 2. Port MUX Block Diagram
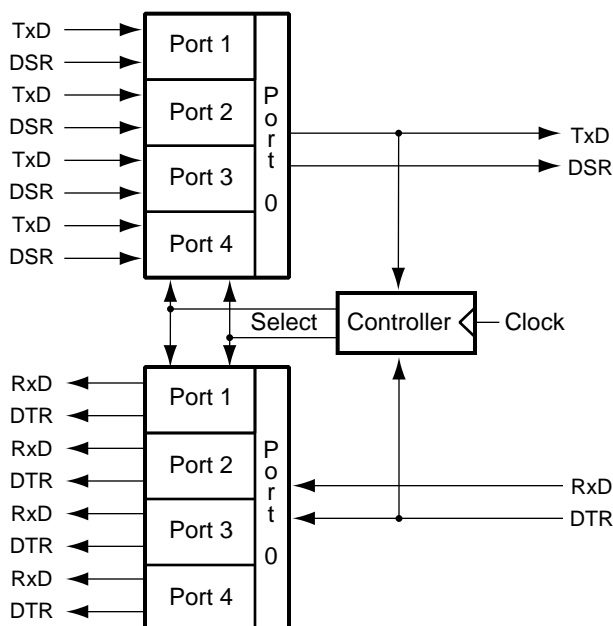
The multiplexer functions by sequentially scanning the four input ports until data appears at one of them.

Scanning a port involves connecting that port to the printer and waiting for data to flow. If no data appears within a predetermined time period, the period of the system clock (~ .25s), the process is repeated at the next port. When data does appear at a port, the port MUX "locks onto" that port and goes into the transmit mode. At the end of the transmission, the port MUX returns to the scan mode.

As mentioned in the discussion of RS-232 protocol, detecting the end of a transmission is non-trivial. To peripheral devices such as printers, the "end of transmission" concept is fiction — to them, life is one big data transmission. The port MUX, on the other hand, must be able to determine when it is permissible to resume scanning. It should not return to the scan mode before the end of a transmission, and at the same time it must not lock onto a port for an inordinately long time. Both requirements are met by timing how long the computer's TxD line is idle, and returning to the scan mode if TxD is idle for longer than a predetermined time period (5 -10 seconds is reasonable).

The data routing logic of the port multiplexer is controlled by two loosely coupled state machines and a status register. The state machines and the status register use the State Logic Macrocells (labeled as state bits S0 - S7). The status register determines the operating mode (scan or transmit); the first state machine determines the active port; and the second state machine is used as a timer.

The basis for most state machines is the simple binary counter, with added logic to allow branching, state skipping, etc. The most efficient way to build a binary counter in the GAL6002 is to configure the registers to emulate T-flip flops. This way, only the conditions that should cause the state bits to change state need to be specified. In the case of simple up counters, there is only one condition when all lower order bits are ones. The equations for a 4-bit up counter are as follows:

```
BO.D = /BO.Q;
BO.E = l;

Bl.D = /Bl.Q;
Bl.E = BO;

B2.D = /B2.Q;
B2.E = Bl*BO;

B3.D = /B3.Q;
B3.E = B2*Bl*BO;
```

As you can see, counters of any size can be built using only two product terms per bit.

In the following discussions POTx and PODT are TxD and DTR respectively.

Status Register

Recall that the beginning of a data transfer is signaled by POTx becoming active for one bit period. By using the start bit event to asynchronously set a status register, set = transmit, the operating mode of the port MUX is determined. Once set, the status bit will remain set until a time-out occurs.

The status register is implemented using state bit S0, configured to emulate a T-flip flop with a programmable clock. With such an arrangement, meeting the specified state transition conditions doesn't just allow a transition at the next clock, but actually causes the transition.

There are two situations that must cause the status register to toggle: if it is clear, clear = scan, and data is flowing, or if it is set and a timeout has occurred. The equations for the status register are:

```
SO.D   =  /SO.Q;
SO.CK  =  /SO.Q*/POTx +
          sO.Q*POTx*PODT*57.Q
          *56.Q*55.Q*54.Q*53.Q;
```

The same function could have been implemented by "building" a latch from combinational equations, but the approach taken here is more efficient in terms of product term usage and is less prone to functional hazards.

Primary State Machine

The primary state machine directly determines the active port. It is simply a 2-bit counter with a hold function. The conditions necessary for the counter to increment are that the status register be clear and that PODT be active.

The primary state machine uses state bits S1 and S2 in the D/E configuration to emulate T-flip flops. The equations for S1 and S2 are:

```
Sl.D = /Sl.Q;
Sl.E = PODT*/SO.Q;

S2.D = /S2.Q;
S2.E = PODT*/SO.Q*Sl.Q;
```

Timer

The second state machine, a 5-bit counter/timer, will only count while the status register is set, POTx is idle, and PODT is active. The counter synchronously resets to zero if these conditions are not met. Thus, if PODT is active and POTx is idle on 31 consecutive OCLK edges, the timer will reach its maximum value, causing the status register to be cleared and the primary state machine to continue counting.

The 5-bit timer uses state bits S4 - S7 in the D/E configuration, again emulating T-flip flops. The equations for the timer are:

```
S3.D = /S3.Q*SO.Q*POTx*PODT;
S3.E = 1;

S4.D = /S4.Q*SO.Q*POTx*PODT;
S4.E = S3.Q
     + /SO.Q;

S5.D = /S5.Q*SO.Q*POTx*PODT;
S5.E = S3.Q*S4.Q
     + /SO.Q;

S6.D = /S6.Q*SO.Q*POTx*PODT;
S6.E = S3.Q*S4.Q*S5.Q
     + /SO.Q;

S7.D = /S7.Q*SO.Q*POTx*PODT;
S7.E = S3.Q*S4.Q*S5.Q*S6.Q
     + /SO.Q;
```

Time-out during a byte transfer, though statistically possible, is unlikely. If it should occur, however, it is not harmful. Because a time-out simply clears the status register and scanning does not resume until the next OCLK edge, the driving computer still has one OCLK period to finish the byte transfer, during which time a logic '0' on POTx returns the status register to the transmit mode. Thus, it is virtually impossible for a byte of data to be lost.

If the port MUX should time-out and resume scanning between byte transfers, but before the end of a transmission, and another computer is waiting to use the printer, then that computer will be serviced before the first computer is again granted use of the printer. This would cause the second computer's data to be inserted into the middle of the first computers transmission. This situation, though undesirable, is unavoidable. The good news is that the probability of this happening is very low.

Conclusion

The Port MUX provides an example of how the flexibility of the GAL6002 can simplify a complex design. Equally important, this example shows how various software tools are used to take advantage of the device's features. Since the Port MUX is not a speed-critical application, the GAL6002's 15ns t_{PD} is more than adequate.

Though this example is complex, it still does not push the GAL6002 to its limits. The state machine and data routing

equations use only 33 product terms, leaving over 48% of the AND array free for expansion. The GAL6002's FPLA architecture allowed five product terms to be merged. If this design were implemented using a standard 24-pin PLD, it would take at least two devices to accomplish this task.

![Lattice Semiconductor Corporation logo](Lattice™ Semiconductor Corporation)

LATTICE SEMICONDUCTOR CORPORATION
5555 Northeast Moore Court
Hillsboro, Oregon 97124 U.S.A.
Tel.: (503) 681-0118
FAX: (503) 681-3037
http://www.latticesemi.com November 1996