AN ABSTRACT OF THE THESIS OF

Kirby R. Headrick for the degree of Honors Baccalaureate of Science in Electrical Engineering presented on November 4, 2011. Title: Serial Peripheral Interface Router.

Abstract approved: _____
Roger Traylor

Many modern embedded systems have multiple devices that require communication between themselves. Many serial communication standards have been created to provide communication between such devices. Modern serial protocols do have downsides. The ones that provide high communication speeds only allow for one device to act as a master and initiate communication and they also require more overhead when connected to multiple slave devices. Protocols that allow for multiple masters operate at speeds that are factors of ten or more slower. The goal of this project is to devise a system that will allow for fast serial communication between multiple masters with minimal overhead. The solution developed works with the existing SPI (serial peripheral interface) protocol, thus allowing high speed communication, multiple masters and minimized overhead. The system accomplishes this by using a router to act as an intermediary in the SPI bus, dynamically creating links between master and slave devices.

Keywords: Electrical Engineering, Digital Hardware

Corresponding e-mail address: headrick@lifetime.oregonstate.edu

Serial Peripheral Interface Based Router

by

Kirby R. Headrick

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements of the
degree of

Honors Baccalaureate of Science in Electrical and Electronics Engineering

Presented on November 4, 2011

Honors Baccalaureate of Science in Electrical and Electronics Engineering project of Kirby R. Headrick presented on November 4, 2011.

APPROVED:

Mentor, representing Electrical and Electronics Engineering

Committee Member, representing Electrical and Electronics Engineering

---

Committee Member, representing Electrical and Electronics Engineering

---

Director, School of Electrical Engineering and Computer Science

---

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any ready upon request.

Kirby R Headrick, Author

Table of contents

List of Figures

Intro

The ability for microcontrollers to efficiently communicate with other devices is important in embedded systems. Most embedded systems contain many specialized peripheral devices to allow the systems to function properly. Different peripheral devices have differing needs that impose various requirements on the communication protocol used.

Communication in embedded systems has a variety of specific implementations but they all have some areas in common. Parallel communication is not usually used as it takes a large number of costly I/O lines, thus some form of serial communication is often employed. Most systems use a master/slave communication model in which one device is the master and is responsible for initiating and controlling the communication. The master is usually a general purpose device that acts as a central processing unit for incoming data and generates the desired response. The slave is usually a specialized device that is designed for a specific task such as receiving specific inputs or outputting specialized control signals. When multiple slave devices are used in a master/slave system, some method of selecting a specific slave to communicate with is required. This can take the form of address information being sent over the communication bus before transmission of data, or separate control lines being used to select individual slave devices.

Because slave devices are generally designed to perform a specific task they have a myriad of requirements for communication based on their function. Some devices, like Analog to digital converters (ADC) or external memory devices can generate or receive data very quickly, on the order of 1 Mb/s or more. These devices need a high speed

method of communication to allow them to be fully utilized. Other devices such as identification chips or smart batteries are designed to be removable and thus need a protocol that allows for devices to be added and removed from the communication bus without causing errors.

Systems with more than master device have greater needs in communications protocol to accommodate the interactions between multiple master devices. If multiple master devices need to be able to communicate to the same peripheral device then there needs to be a system in place to allow both master devices to communicate without interfering with each other. This requires the master devices to some ability to check if the communication bus is in use before attempts to communicate are undertaken. If the system requires the two master devices to communicate directly with each other then it is often necessary to implement some method of flow control to slow down or temporarily stop communication. A master device is often performing multiple tasks, thus the amount processing it can dedicate to communications can vary greatly depending on what other events are occurring. Thus there needs to be a way for the master device acting as a slave to request a pause in the communication to give it time to finish processing the required data. This ability to throttle communication is usually not needed for slave devices as they are usually dedicated to a single task and thus usually have a fixed known communication speed.

Existing solutions

There currently exist many protocols for serial communication between microcontrollers and other devices. Each protocol has unique advantages and drawbacks as is discussed in the following.

Universal serial bus (USB) is a protocol commonly utilized for communication between personal computers and peripherals, although it is used in some embedded systems as well. A USB interface consists of four wires, two are used to share power and ground connections between the connected devices and the other two are used to transfer data. These communication lines are used to convey data and status information at a rate up to 480 Mb/s. The USB protocol allows for devices to be connected and disconnected from the communication lines at anytime. This feature along with the ability to power a device off the supplied power lines of the connection makes USB useful for peripheral devices, such as mice, keyboards and storage drives, used with personal computers.

While USB has a very high data rate it has a variety of downsides that make it non-ideal to use in embedded applications. The protocol only allows for a single master device, all other devices must act as slave devices which limits use in systems with more than one microcontroller. The high speeds and ability to connect on the fly add complexity to the protocol requiring more sophisticated hardware and software. The software to implement a USB interface can take hundreds of lines of code to implement taking up most of the program space and system resources to handle communication. Other communication protocols designed for embedded systems such as I2C can be implemented in only about a hundred lines of code and are often simple enough to have them implemented almost entirely in hardware. The benefits of speed from using USB

are usually overshadowed by the increased complexity of hardware and software required to implement the protocol and thus it is not often used for communication within an embedded system.

A communication protocol better suited for embedded systems communications is Inner IC communication (I2C). I2C was developed by Philips Semiconductors with the goal of creating a communication protocol for embedded systems as an alternative to parallel communication. The goal was to reduce the number of lines used for communication thus reducing I/O pins used and board spaced taken up by communication lines. I2C communication has the advantages that it allows for multiple master devices and the protocol supports bus arbitration and flow control. The protocol consists of two wires, one that carries data and the other carries a clock signal. Each device on the bus has its own unique address which is used to specify the target device in communication. In I2C any device can act as a master by driving the clock line and sending out an address of the destination device. Slave devices send an acknowledge signal when they detect their address being sent allowing the master to know the target device is active and ready for communication. After the acknowledgement the master or slave device can send data across the communication line, but the master is always responsible for controlling the clock line. The I2C protocol also allows for flow control. The slave device can hold the clock line to delay communication in the event data is being sent to them or requested from them at too high of a rate.

Despite being designed for usage in embedded systems I2C does have some drawbacks to its usage. Communication operates at a much slower speed compared to some device needs. Normal I2C speeds are in the range of 100-400 kb/s. The standard

does allow for communications up to 3.4 Mb/s but it is not supported widely supported in devices and imposes more stringent requirement on bus capacitance characteristics. The I2C protocol achieves the design goal of creating a method of communicating that requires fee I/O pins and less board space to implement parallel communication, however its slow communication speeds makes it not suitable for situations that involve lots of data transfer such as communicating with an external memory device.

Another communication protocol designed for embedded systems is 1-wire communication. This protocol has that advantage that it only requires a single wire to supply power and communicate with devices. The when idle the communication line supplies power, during communication it turns off and back on in short pulses. The standard defines a bit as being a one or a zero base on the duration of the pulse. Slaves are designed with internal capacitors to hold charge during communication to remain powered. Because the power is supplied along the communication line this protocol is well suited for devices that are designed to be removed and inserted into the system as it requires minimal connections and the removable salve device does not require its own internal power source.

The downside of 1-wire communication is it does not support multiple master devices and communications are slow operating on order of 10 kb/s. The main advantage of 1-wire communication is the simplicity of connecting to removable devices, which makes it useful in specific situations but the slow communication speed makes it not practical for usage outside of cases of those cases.

The last existing communication protocol considered is Serial Peripheral Interface (SPI). SPI has that advantage of being very simple to implement and allows for very fast

communication speeds. The communication takes place one three wires, one that contains the clock, one contains data going out of the master and into the slave, the final wire is used for data coming out of the slave and into the master. The clock is driven by the master, at each pulse of the clock a single bit of the message is sent by the master and slave simultaneously along their respective output lines. The process continues until all bits of the message have been sent. This design can be implemented easily in hardware with a standard serial shift register, or can be implemented in code with a simple loop and a few lines of code. The simplicity of the protocol whether implemented in hardware or software contributes to the high speeds the communication can run at. SPI communication can run at speeds on the order of 10 Mb/s or more.

The simplicity of SPI does present some downsides. The protocol only supports a single master device. In order for multiple slaves to be on a single communication bus each one requires a dedicated select signal to enable communication with it. Thus for each added slave device on an SPI bus the master needs another output to be able to select it. SPI protocol's fast speed makes it ideal for embedded applications that involve large amounts of data such as reading data from a high speed ADC. The simple interface makes it well suited for constructing custom SPI devices. However the large overhead of select lines required for multiple slave devices makes it not useful as a general purpose serial protocol.

Proposed Solution

While many communication protocols do exist, there are none without significant downside. USB requires complex hardware and software to implement and is designed

for more sophisticated computers not embedded systems. I2C and 1 wire communication suffer from slow communication speeds. SPI requires more control lines as more slave devices are added to the bus. The proposed solution to this problem is to modify the existing SPI protocol to remove its drawbacks of not supporting multiple masters and requiring individual control lines for each slave device.

SPI was chosen to be modified as the communication already operates at high speeds and the protocol definition is simple enough to provide some flexibility for modification. In order to solve the problem of the large number of control lines needed to communicate there needs to be an alternative way to select a slave device for communication. The alternative method chosen was to implement addressing as it allows for devices to be chosen but only using the communication bus without the need for extra external signals. To allow for multiple masters to communicate on the same bus there needs to be a way to prevent all but one master from driving the clock and outgoing data lines. The method chosen to accomplish this was to have an intermediary device on the communication bus, such that each master device would be connected via its own SPI communication bus to the intermediary and it would select which master's communication to relay to the slave's SPI bus.

Another important part of the new system is that it needs to be compatible with standard SPI slave devices. Slave devices designed to communicate with SPI can not be modified to accommodate a new protocol, and thus the new system must be able to communicate with a standard SPI slave. Master devices on the other hand can be expected to have some degree of customizations available to them, although it is desirable to keep modifications to a minimum to ease implementation.

The method final devised to overcome the limitations of SPI is to use an SPI based router that acts as an intermediary on the SPI bus. All SPI devices are directly connected to the router. To initiate communication the master device sends the address of the desired slave device using normal SPI protocol. The router will then dynamically create a link between the master and the desired slave device allowing for standard SPI communications to take place.

This approach fixes one of the problems of normal SPI by reduces the number of control lines needed when communicating with many slave devices. The master device selects a slave to communicate with by sending an address into the router treating the router as a standard SPI slave device. This reduces the number of select lines the master needs to control from one per slave device to just one to select the router. The addresses consist of 3 bits unique to each device connected to the router and 5 bits which are the same for all devices on that router. The fixed 5 bit address can be used to allow for a master to select between multiple routers without needing to use more than one select line.

The other issue with normal SPI that the router addresses is the lack of multi master support. The normal SPI bus only supports a single master because multiple devices controlling the clock and slave select lines is not feasible. The router allows for multiple masters as each device is connected to the router with its own SPI interface, thus when a master drives its SPI lines it does not interfere with the SPI lines another master is attempting to drive. The router dynamically creates the connections between master and slave on requests of a master device, allowing multiple masters to access the same slave in a sequential manor. In the event the masters are communicating with different slaves

the links can be constructed simultaneously allowing the communication to take place in parallel.

Design description

Top Level – Router

The main purpose of the SPI router is to address the lack of multi master support and control line overhead of normal SPI. This is accomplished by adding addressing functionality to SPI communications. It is designed to reduce the I/O overhead of communicating with a multitude of devices while still being compatible with normal SPI to allow for ‘dumb’ slave devices to still function. The router also allows for multiple master devices to communicate to the slave devices and for multiple communications to occur simultaneously. This is accomplished by having each device connected to the router thought its own dedicated SPI interface as opposed to being connected to a common SPI bus. Masters make communication requests to the router, which will then link the master’s SPI interface with the desired slave’s interface to allow for master/slave communication. To allow for flow control status messages, the router has two additional lines added to each SPI interface. They provide notifications for communication requests (REQ) and acknowledgments (ACK) but are not required for basic functionality.

To begin communication the master device must first send in the address of the port it wishes to communicate with using the standard SPI protocol, treating the router as a normal SPI slave device. Once the address byte is received by the router, it constructs a path from the source device to the requested destination slave device using an algorithm outlined in the routing algorithm section below. Upon reaching the destination device the two interfaces are not yet connected, first a request for communication is sent out on the

REQ line of the destination device's SPI interface. When the destination device is ready for communication, it responds to the request by sending an acknowledgement on the ACK line of its SPI interface. In the event the destination device is a simple slave device that cannot be configured to send acknowledgement signals the device's ACK and REQ lines can be externally connected. This will cause all incoming request signals to trigger an acknowledgement allowing communication to continue. Upon receiving the acknowledgement the router connects the master's SPI interface to the slave device's interface allowing for normal SPI communication to take place.

Once a link has been established, the acknowledgement signal is relayed back to the master's port and output on its ACK line. This signals to the master device that the slave device is connected and ready to communicate. During communication a slave device can stop sending the acknowledge signal which will signal to the master device to pause communication. This allows for a more sophisticated slave device to pause communications in the event it needs more time to process the incoming or outgoing data. The connection persists until the master device deselects the router by driving the slave select line high, in which case the router will disconnect the two ports and return to an idle state.

Internal Structure

Internally, the router is comprised of eight routing nodes. Each node is directly connected to a single SPI interface and is responsible for initiating routing for that port. Each node has an input and output communication port connected to each neighboring node. The communications port consists of an SPI interface with added ACK and REQ

signals and destination address data. Nodes also have a bi-directional port connecting them to an external SPI interface with added ACK and REQ signals as shown in figure 1.

Node neighbors are determined by the nodes address. Each node is assigned a unique three bit address which is the master uses to select a destination device. Nodes with addresses that differ by only one bit are considered neighboring and are directly connected. By treating the address of the device as a coordinate point the internal network topology can be visualized as a cube with nodes located at the vertices and connections made along the edges as seen in figure 2. To differentiate between the various connections at a node, they are labeled according to the 'dimension' the link travels in based on the coordinate interpretation of addresses. For example the connection between node 000 and node 001 is referred to as the X connection.

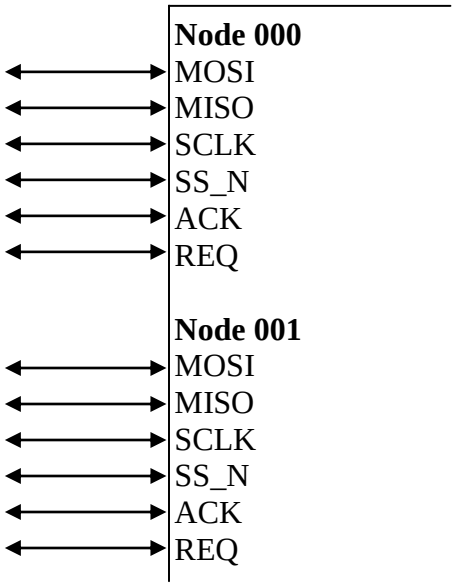

Figure1
External router interface

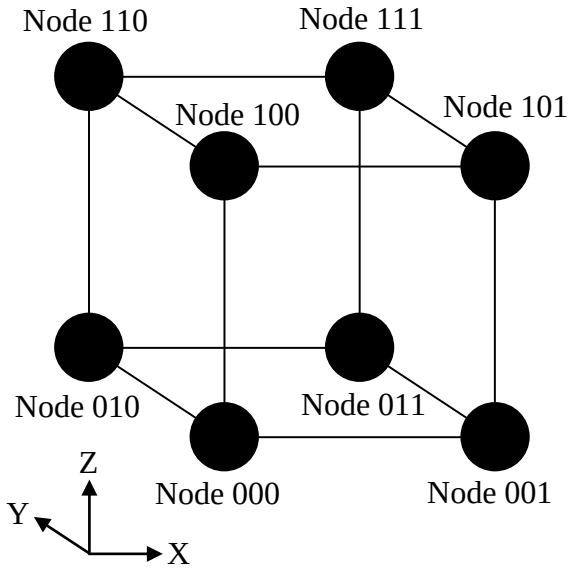

Figure 2
Internal router topology

Operation overview

Routing Algorithm

The routing algorithm is responsible for determining the path between the source and destination and provides arbitration when between multiple requests attempt to use the same internal links. The routing process begins when a device sends a destination address into the router's SPI interface using standard SPI protocol. Upon receiving the address, the upper five bits are checked against the fixed address of the router. If the upper bits do not match, the router remains idle and will not accept any new information from the SPI interface until the slave select line is driven high again. This allows for a master device to be connected to multiple routers without any change in the protocol or control lines needed, assuming each router is assigned a unique fixed address.

If the upper five bits are correct, a routing request is generated and the path from source to destination begins construction. The lower three bits of the input address are used as the destination node address. The node compares the three bit destination address with its own address, starting with the least significant bit. When a difference is found, the node attempts to forward the routing request to the neighboring node with the correct address in that bit, by linking the SPI, ACK, REQ and address lines of the incoming communication port with the output port. The routing request will not be forwarded along if the communication port between the two nodes is currently being used. In that case, it will wait until the port is no longer in use or the routing request has been canceled. In the event multiple incoming requests simultaneously request the same output link, priority is given to a specific incoming communication port on a rotating basis. Upon successfully

being forwarded to the next node, the routing request will be processed the same way by the new node until it reaches the node with the correct destination address.

Upon reaching the destination node the routing behavior differs slightly. If a node determines that all the address bits match its own address it forwards the request to the external SPI interface. This routing request will wait like previous ones if the output port is currently in use. When the node outputs a routing request on its external interface it only dives the REQ high, it does not yet link the other SPI or ACK lines. When the slave device is ready to communicate it responds to the routing request by driving its ACK line high. When the node receives acknowledgement signal it links the incoming SPI, ACK and REQ lines to the external interface making a complete channel from source to destination for communication to begin. In the event the slave device cannot be configured to respond to a routing request the ACK line of the slave can be directly wired to the REQ line, providing instantaneous acknowledgement of any routing requests.

Once a link between master and slave has been established normal SPI communication can commence between the two devices. The master is informed when the link has been established by the ACK line on its interface being driven high. During communication the slave device can use the ACK line to request a pause in communication. If the slave device needs more time for processing it can drive the ACK line low, signaling to the master that slave needs more time. The master is then responsible for pausing the communication until the slave drives the ACK line high again, the router keeps the two interfaces fully linked even while the ACK line is low.

Communication is ended when the master device returns the router’s slave select line to a high state. When this occurs the router will remove the internal routing request

causing the slave's REQ line to go low and all internal links between master and slave will disconnect from each other. When the internal nodes disconnect they become free to establish connections for any pending routing requests. In the event there are multiple pending routing requests the request with the current highest arbitration priority is processed first.

Functional unit - Routing node

The routing nodes all have the same internal structure to manage the routing of incoming signals and arbitration to output ports. Each of the three incoming communication ports from neighboring nodes and the external interface have their own state machine that takes incoming requests on that port and determines the correct output port to continue routing on. The three output communication ports and the external interface have state machines that arbitrate which incoming port gets linked to the output port via a multiplexer. The internal structure of a node is shown in figure 3.

The input port router state machine is responsible for taking a routing request and determining the correct output port to request a link with. The state machine takes inputs of a destination node address and a routing request and outputs requests to establish a link with an output port. The state machine initializes to an idle state where it waits for an incoming routing request which is signaled when its REQ line is driven high. When it receives a routing request, the state machine looks at the incoming destination address and determines which output port is next according to the routing algorithm. The incoming address is compared bit by bit, starting with the least significant bit (X address) against the node's address. When a bit is found to differ, the state machine moves to a

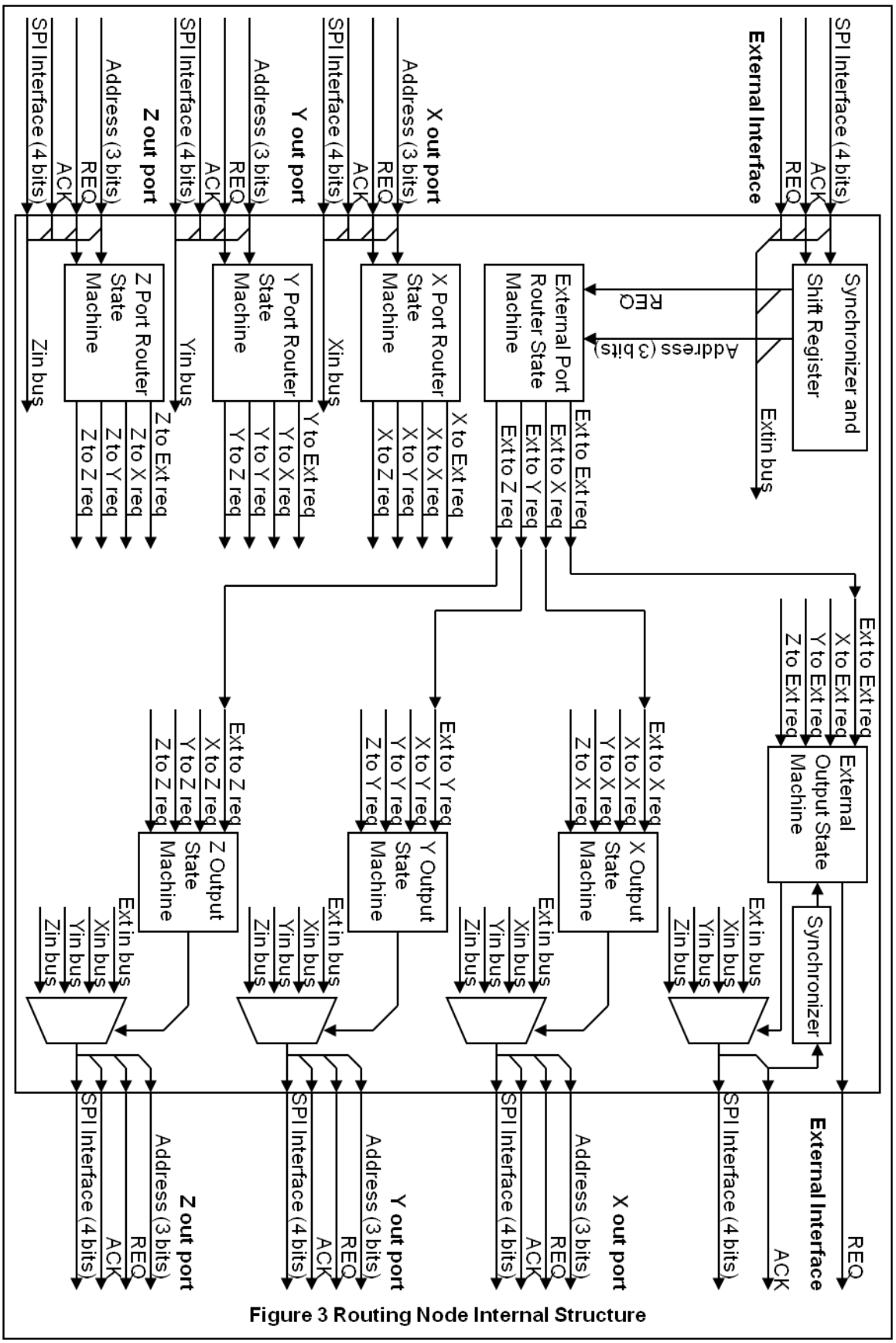

Figure 3 Routing Node Internal Structure

routing state and outputs a request to link the input port with the necessary output port. In order to speed up the routing process, the state machine performs address bit checks in parallel as shown in figure 4. Upon transitioning to a routing state the routing state machine outputs a link request internal to the node to the state machine that arbitrates the desired output port.

The state machine that routes incoming signals for the external interface is connected differently than the other input state machines. The master provides the destination address in a serial form and thus it needs to be converted into a parallel form for generate the routing request and destination address. This is accomplished using a shift register that only accepts the first 8 bits transferred after the slave select line has been driven low. The shift register outputs the data in a parallel fashion, the lower 3 bits are used as the address of the destination port and sent into the state machine. The upper 5 bits are checked against the router's fixed address, if it matches the routers address a routing request signal is sent to the routing state machine. The shift register used to convert the input address from serial to parallel allow for the state machine controlling the external interface to be identical in function to the state machines that control the internal input ports.

The external output state machines are responsible for linking input ports with output ports based on the requests from the routing state machines. It take a node internal routing request inputs from each of the four input port routers and outputs a control signal for the multiplexer that controls which input port is linked to the output port. The state machine initializes to an idle state, in which the output selection multiplexer is set to output none of the input ports, until it receives a node internal routing request from an

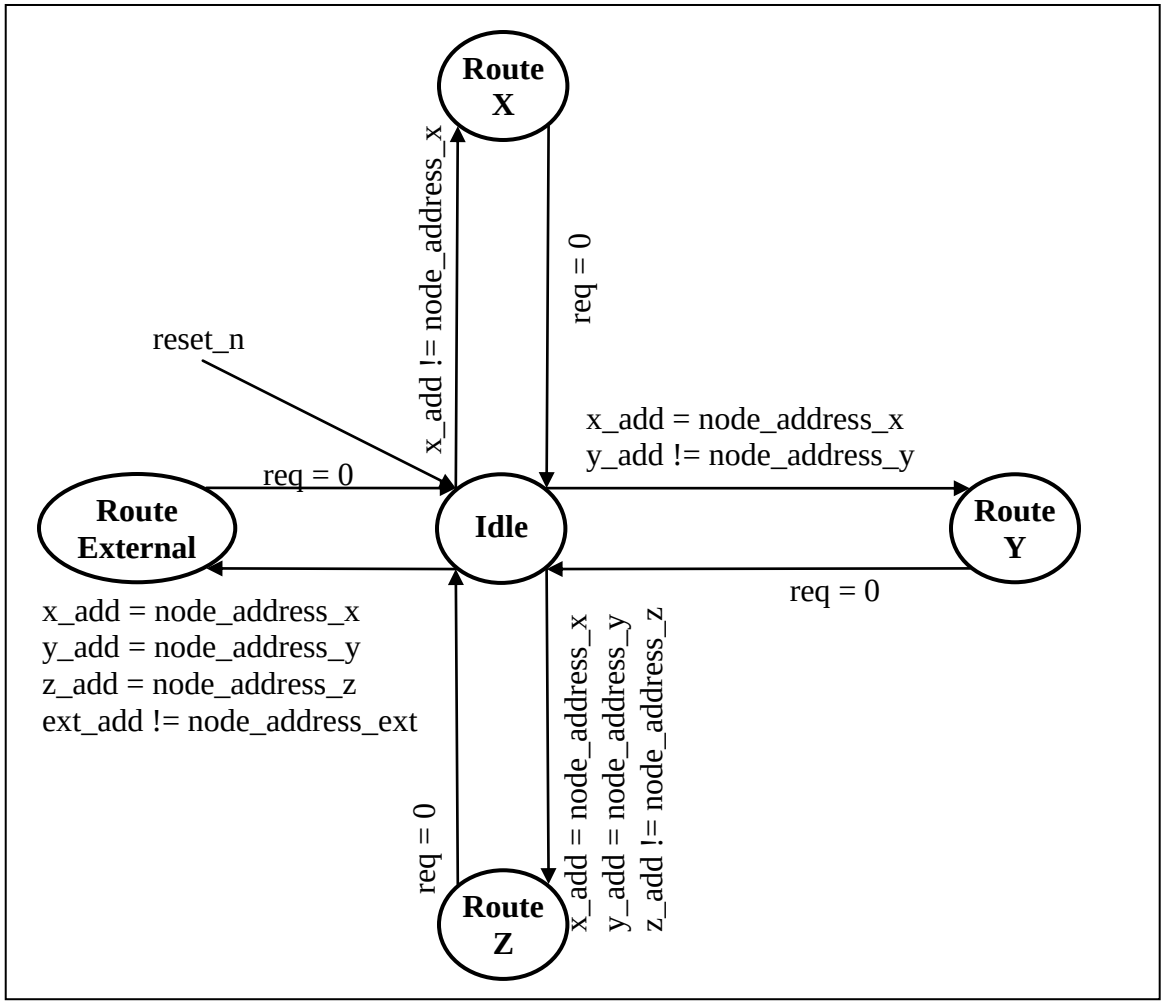

Figure 4: Input Port Router State Machine

input port state machine. The state machine takes all node internal routing requests and accepts the request coming from the port that has the highest arbitration priority as seen in figure 5. The arbitration priority is determined by the arbitration priority state machine described below. Once it has established a link between an input and output port, it will remain in that state until the request from the input port goes low again.

The state machine that controls the routing link for the external output port is slightly different than the other state machines. When a node internal routing request is received instead of moving to a routing state that links the two ports together, it instead moves to

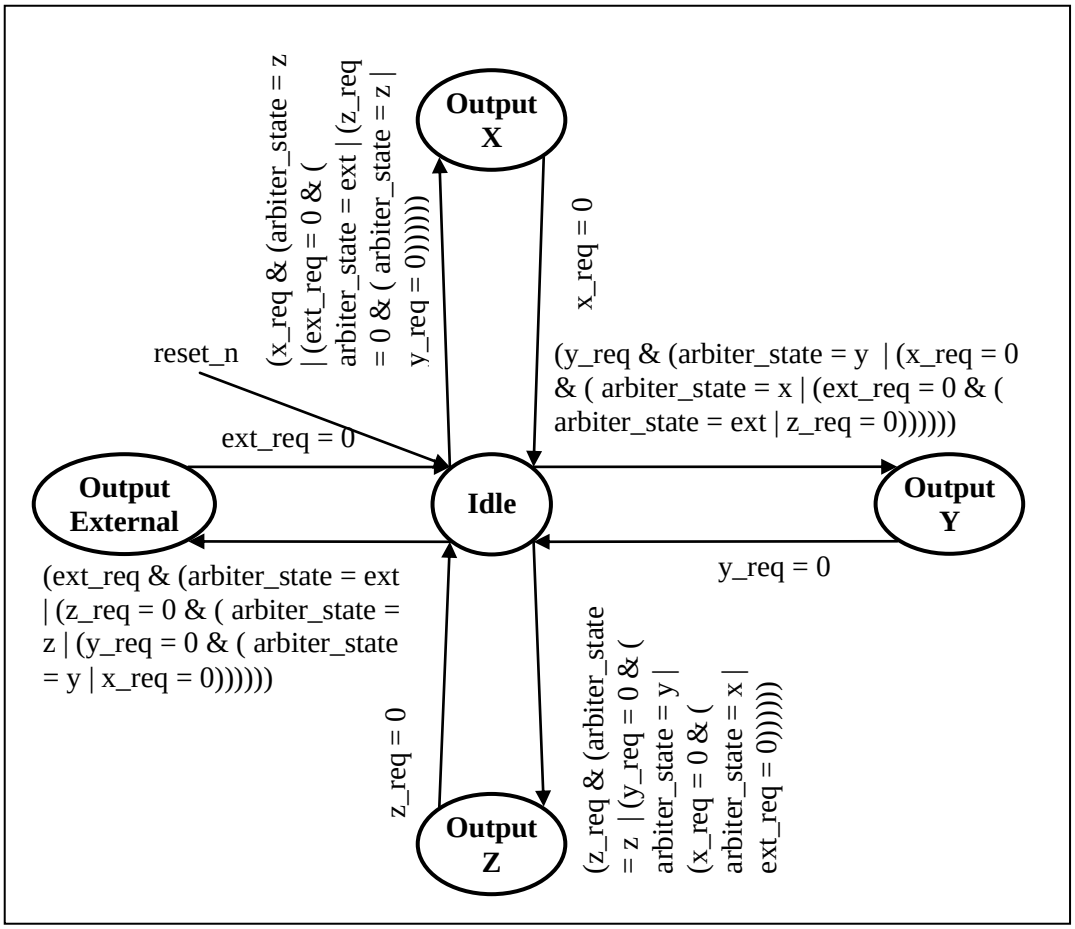

Figure 5: Output Port Arbiter State Machine

an intermediary pre-routing state. The two ports are not linked in this state, but the state machine does drive the external ports REQ line high. Upon receiving an acknowledgement from the slave, the state machine will move into a link state during which the input and output ports are linked. In the event the input port making the link request is the external input port, there is no pre-routing state. It immediately links the two ports as seen in figure 6. This is done as the only situation where the external input port of a node would request a link with the external output port is when a master device sent in its own port's address. This is done to test connectivity between the master device and the router. The request and acknowledge protocol is not utilized when a device is

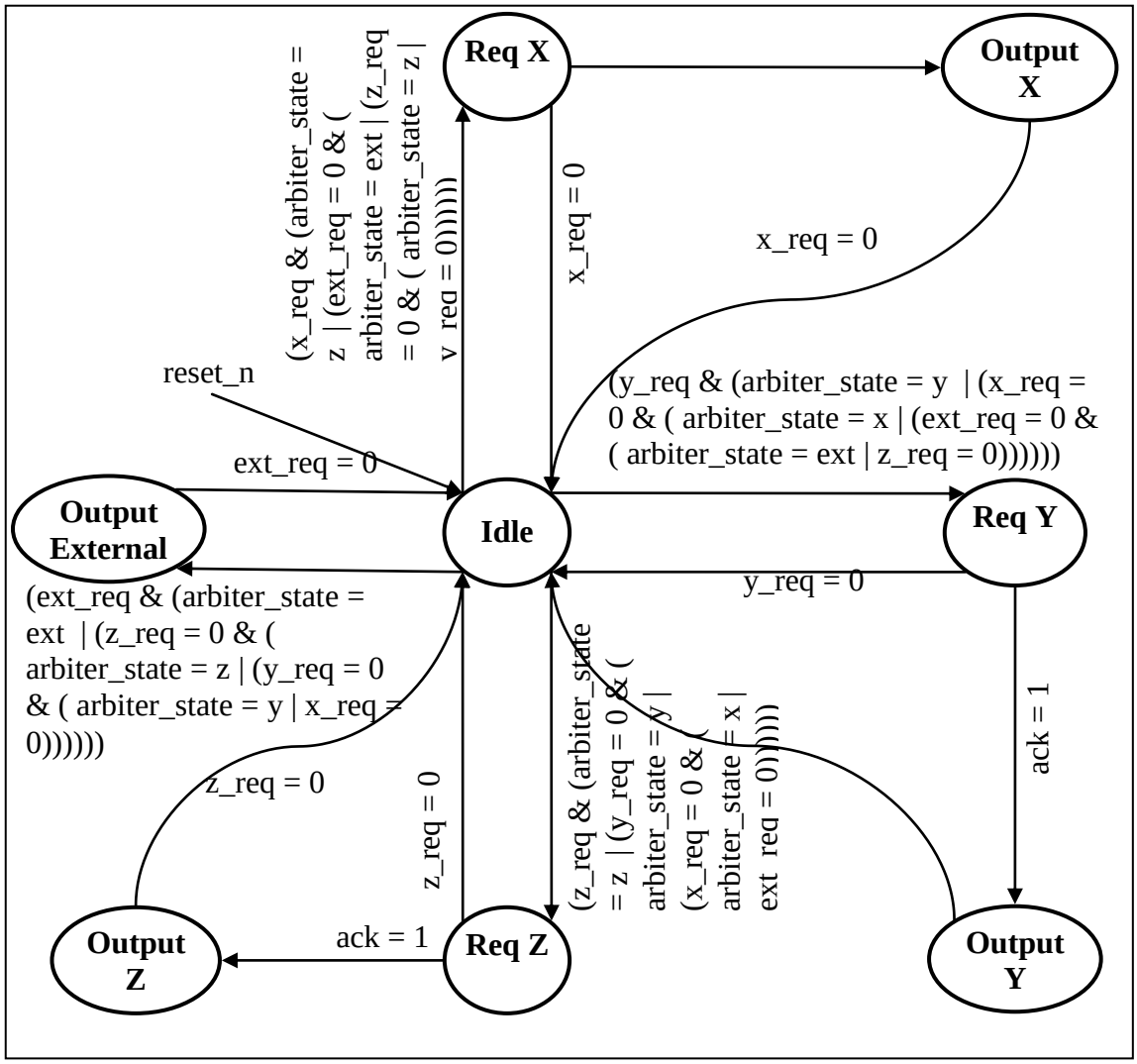

Figure 6: External Interface Arbiter State Machine

communicating with itself to differentiate between communication originating from the device itself or from a different device, which would require a different response than communication originating from the device itself. Testing of the connection can be conducted by sending data over the SPI connection and verifying the same data is read back in.

To implement the rotating priority arbitration scheme a state machine is used to determine arbitration priority. The state machine consists of four states, each one

corresponding to a different input port. The state machine has no inputs, and its output is just the current state representing the port that has arbitration priority. Every clock cycle the port with top priority is given lowest priority and all other ports are moved up in the priority list by one.

Design environment

The tools chosen for design and testing of the SPI router were primarily picked due to familiarity and availabilities of the tools. The design language chosen was system verilog due to prior experience using it in digital designs. Testing and simulation of the design was done with Mentor Graphics ModelSim as the tool is readily available for students. Synthesis of the design was conducted using Synopsis dc_shell. These tools were primarily accessed via command lines and scripts were heavily employed to provide repeatability and ease of development.

Testing Protocol

Testing is an important part of the design process and was conducted at various stages in the development of the SPI router. Individual state machines were tested using a simple test bench to check for proper transactions between states. The next stage of testing was done on a single routing node. The node was tested for functionality to ensure it made proper routing decisions and followed the arbitration scheme. The last level of testing was to test the entire router design to verify the nodes were connected properly and that the design as a whole functioned properly. After testing was completed on levels of the router code the design was synthesized and testing was conducted on the resulting

synthesized code. The synthesized code was tested in the same way as the pre-synthesized router to ensure that the synthesis process did not break any functionality. Through the testing the design was found to be fully functional.

Future work to be done

While this project did result in successful design and implementation of an SPI router, there are still areas that could be improved upon. The next step in the development process would be to synthesize the router for an FPGA. This would allow for testing to be conducted with real deices as opposed to all current test that have been run in a simulated environment.

The router protocol could be expanded to allow for multiple routers to communicate with each other. Currently, for two routers to operate in a chained fashion, the connecting port between the routers would need to have its ACK and REQ lines connected together for automatic request acknowledging. This would prevent slave devices connected to the second router from relaying acknowledge signals for flow control. Multiple routers also currently require multiple address bytes to be sent in sequential order, first to select the destination of the first router and then to select the destination of the second router, thus creating more communication overhead. With modifications to the router protocol, these issues could be alleviated.

The third area of improvement for the router design is that the router currently only supports data being clocked in on the rising edge of the clock for the destination address. Once the link to the destination port is established the master is able to communicate with data being clocked on rising or falling edges. However that would

require configuration changes on the part of the master when communicating. A better system would allow for the router to be configured to allow a port to receive input addresses clocked on either the rising or falling edge.

Conclusion

The SPI router design does result in improved serial communication over standard SPI communication. Use of the router adds support for multiple master devices as well as reducing the I/O overhead of SPI. The router protocol is compliant with standard SPI and thus will work with ‘dumb’ slave devices that cannot be modified. The design was tested via simulations on synthesized system verilog code and found to function according to the original design.

BIBLIOGRAPHY

[1] "The I2C-Bus Specification." Jan. 2000. Web. <www.nxp.com/documents/other/39340011.pdf>.

[2] "1-Wire Devices - Maxim." *Analog, Linear, and Mixed-signal Devices from Maxim*. Web. <http://www.maxim-ic.com/products/1-wire/>.

[3] "Universal Serial Bus Specification." *USB.org - Welcome*. 27 Apr. 2000. Web. <http://www.usb.org/developers/docs/usb_20_071411.zip>.