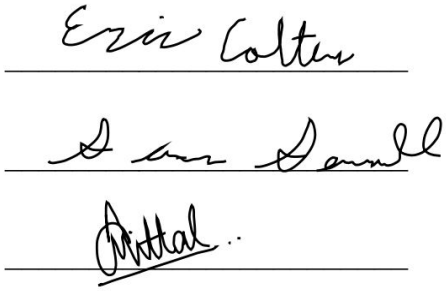
Phase 3

I^2C Interface for APB

Team Members:
Eric Colter
Sam Sowell
Arnav Mittal

ECE 337
Asic Design Laboratory
Lab # 5: Friday 11:30 AM - 2:20 PM
Friday, 11th March'16

Lecturer: Dr. Mark Johnson
GTA: Nick Pfister



1.0 Executive Summary

In current day and age it is often difficult and tedious to connect the current generation of chips to an external device. Most devices require an I^2C bus to connect, however the current chip iteration does not contain an I^2C bus. To rectify this issue, our team proposes that an I^2C bus is implemented on the next iteration of the chip. By implementing this common device standard, your chip will soon be able to communicate with even more devices (additional 1008 devices).

While the I^2C is a universally used standard, our iteration will set the new industry standard. Our current design includes flexibility for both master and slave control, an interface for connecting to an APB bus, and FIFO registers for storing data. Master mode allows the I^2C to control any devices connected to the I^2C bus, while slave allows these devices to send data to your chip. Including both a master and slave mode increases the flexibility and possible uses for your chip. In addition to a master and slave mode the I^2C bus will have flexible data transmission rates, opening the doorway for communication with even more devices.

Another important aspect of our proposal is the inclusion of an APB slave. This APB slave allows our interface to be quickly implemented into any pre-existing chip with an already created APB bus. In addition to an APB slave our proposal includes FIFO registers. FIFO registers provide a safe and efficient way of storing data for use on or off the chip. This registers work in conjunction with the APB slave to create an efficient and universal method of communication to the rest of your chip.

In addition, the device features two circular FIFO buffers, one for data to be transmitted, and one for data that has been received. This allows the module to send or receive several bytes of data consecutively without requiring intervention from the core. This simplifies software design and also frees up more clock cycles for the core.

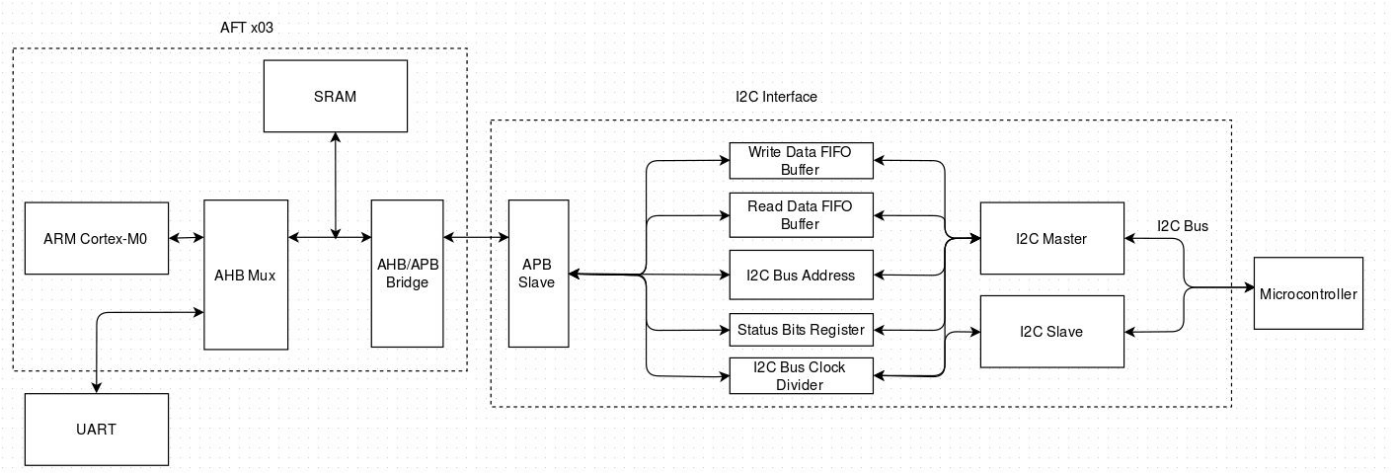
While all of these features sound great, it is also important to note the cost of implementing our interface. Our interface comes with the benefit of easy integration onto a pre-existing chip, so you will not have to worry about developing another device. In addition our interface will take up a relatively small amount of area on the chip, so the increase in chip complexity will not be an issue.

The rest of the proposal will provide the necessary technical details of the chip. Future proposals will delve even further into these details.

2.0 Design Specifications

2.1 System Usage

2.1.1 System Usage Diagram



2.1.2 Implemented Standards and Algorithms Overview

Inter Integrated Circuit Communication (I^2C):

- Mode (Master/Slave): Both
- Master Baud Rate: Adjustable (10kHz, 100kHz, 400kHz, 1MHz)
- Addressing mode (10bit/7bit): Adjustable
- Repeated Start: Implemented

ARM Advanced Peripheral Bus (APB)

- 32 bit wide data
- 32 bit wide address

First In First Out Receive/Transmit Buffers (FIFO):

- 16 byte long circular buffer
- Method of determining Empty/Full: An external buffer counters

2.2 Design Pinout

Table 2.2.1 : APB Interface Design Pinout

Signal Name	Type (Input/Output)	Number of Bits	Description
PCLK	Input	1	The System Clock. (Variable frequency)
PRESETn	Input	1	The Active Low reset signal.
PADDR	Input	32	The address bus line coming from the APB Slave Interface.
PSELx	Input	1	The select signal generated by the APB slave interface to select a slave ‘x’ for data transmission.
PENABLE	Input	1	The Active High enable signal that indicates subsequent cycles of APB data transfer.
PWRITE	Input	1	The write signal that indicates the direction of movement of data. This signal indicates APB write access when Active High and APB read access when Active Low.
PWDATA	Input	32	This is the signal that contains the data that needs to be written to the Slave.
PREADY	Output	1	The ready signal used by Slave to extend an APB transfer.
PRDATA	Output	32	This is the signal that contains the data that needs to be read from the Slave. This signal is driven by the selected slave during read cycles when PWRITE is LOW.
PSLVERR	Output	1	This is the Slave error signal that indicates the a failure in APB transfer.

Table 2.2.2 : I^2C Interface Design Pinout

SDA	Bi-directional	32	This is the signal that contains the data. Data is always placed on SDA when SCK goes LOW and sis sampled when the SCK goes HIGH.
SCK	Bi-directional	1	This is the clock signal that is generated by the current master. It might be forced to LOW by some slave devices when they need more time in generating data before the data being clocked out by the current master.

Table 2.2.3: Control Register Values (Values that can be written to by the APB bus)

Number of bits	**Flag**
10[9:0]	Bus address
1[10]	Address mode(7/10 bit) [10 bit = 1, 7 bit = 0]
1[11]	Master/Slave Select [Master = 1, Slave = 0]
2[13:12]	Clock Rate (10kHz/100kHz/400kHz/1MHz)
4[17:14]	Packet Size(up to 16 bytes)
1[18]	Read/Write Select [Read = 1, Write = 0]

Table 2.2.4: Status Register Values (Values that can be read by the APB bus)

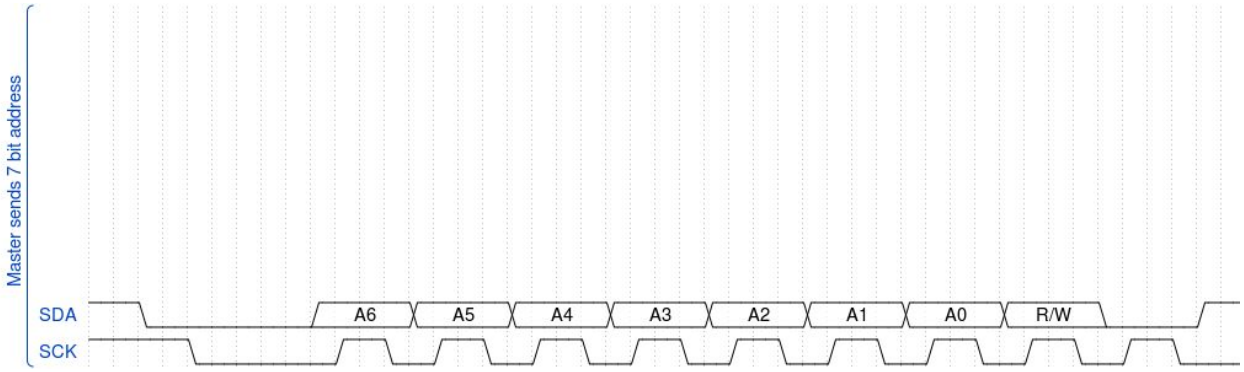
Number of bits	Flag
1	Error
1	Transmit Fifo Full
1	Received Fifo Full
1	Transmit Fifo Empty
1	Receive Fifo Empty
1	Transmit Underflow Error
1	Receive Overflow Error

2.3 Operational Characteristics

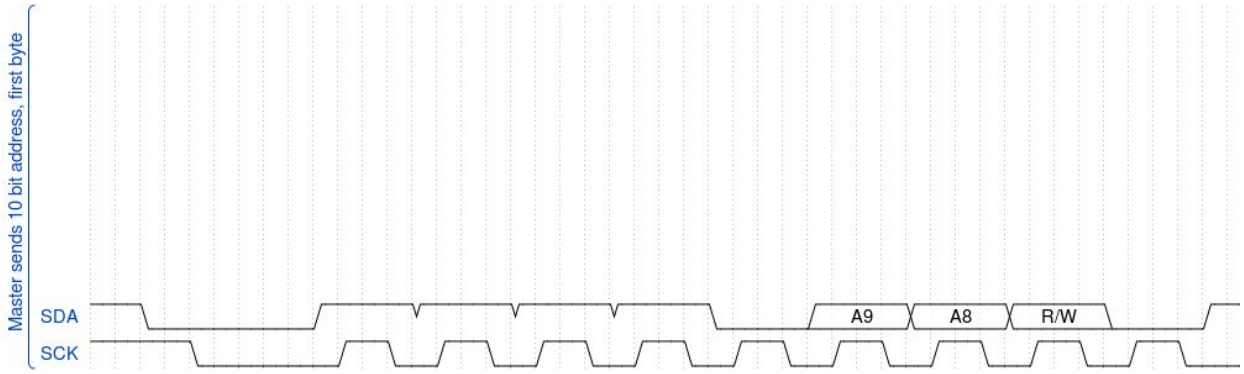
2.3.1 Overall Characteristics

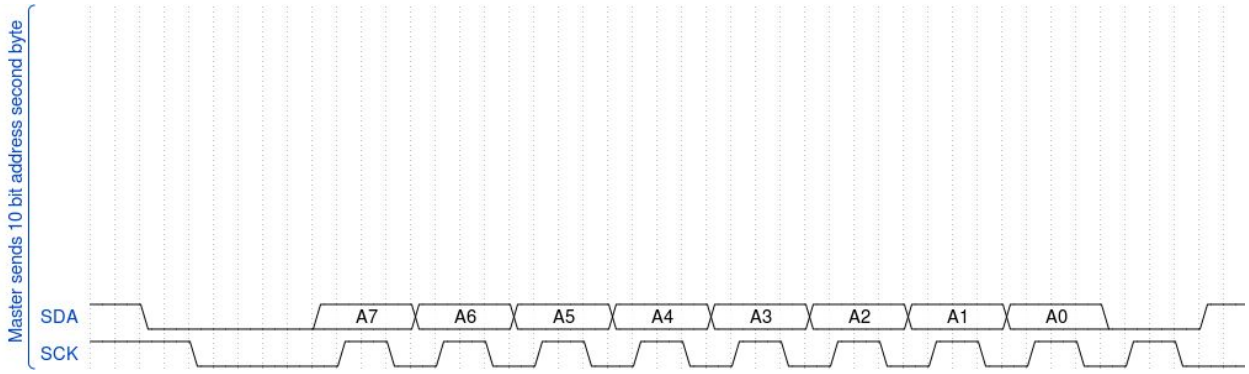
The I^2C communicates to the master and slave using 2 lines which are the Serial Data (SDA) and Serial Clock (SCK).

The communication starts with the Master controlling the SDA and SCK lines and asserting the Start condition to let the slaves know that it is about to start communication with one of the slaves.



After the assertion of the Start condition by the Master it proceeds with sending the 7 bit address to the I2C Bus Address register. This address is then passed onto the slaves which checks the address with respect to its' address and provides an Acknowledgement bit.



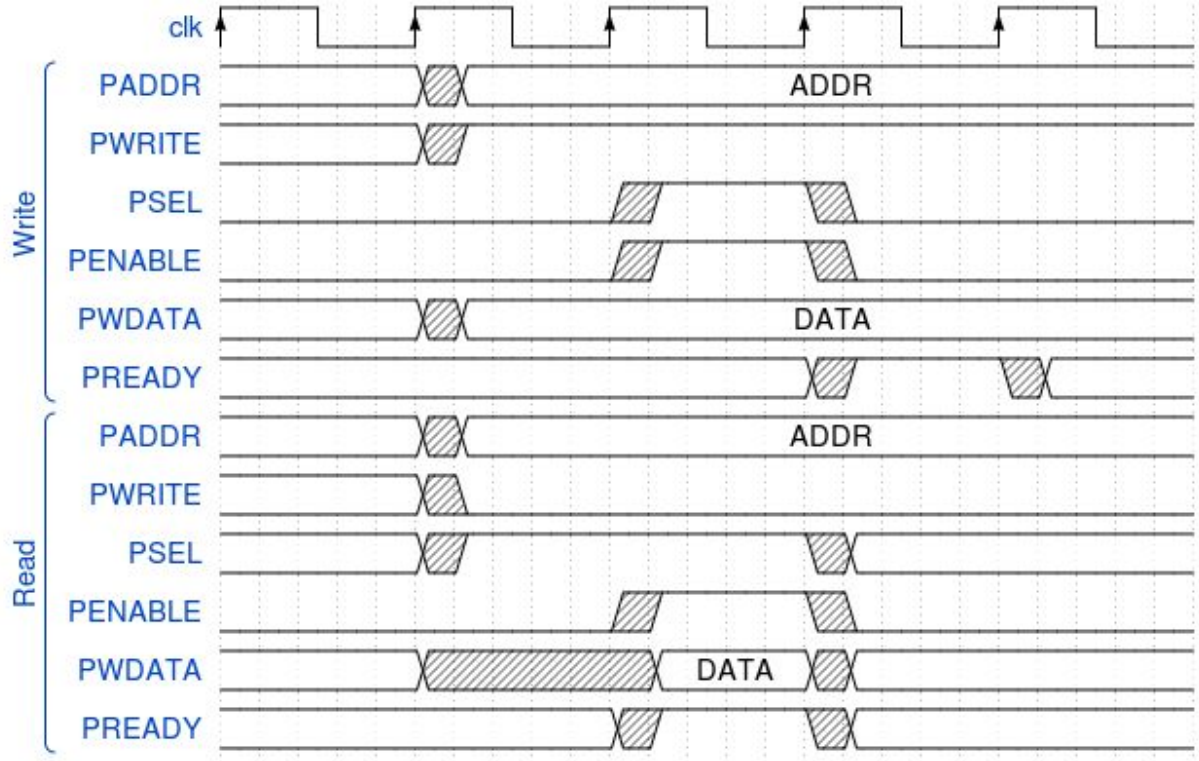


If in case of 10 bit mode, after the assertion of the Start condition by the Master it proceeds with sending the 10 bit address to the I2C Bus Address register. This address is then passed onto the slaves which checks the address with respect to its' address and provides an Acknowledgement bit.

After the address phase, the master proceeds to the data phase where it proceeds with sending the data to the Read/Write FIFO buffer. This data is then passed onto the selected slave until another Start or Stop condition is received.

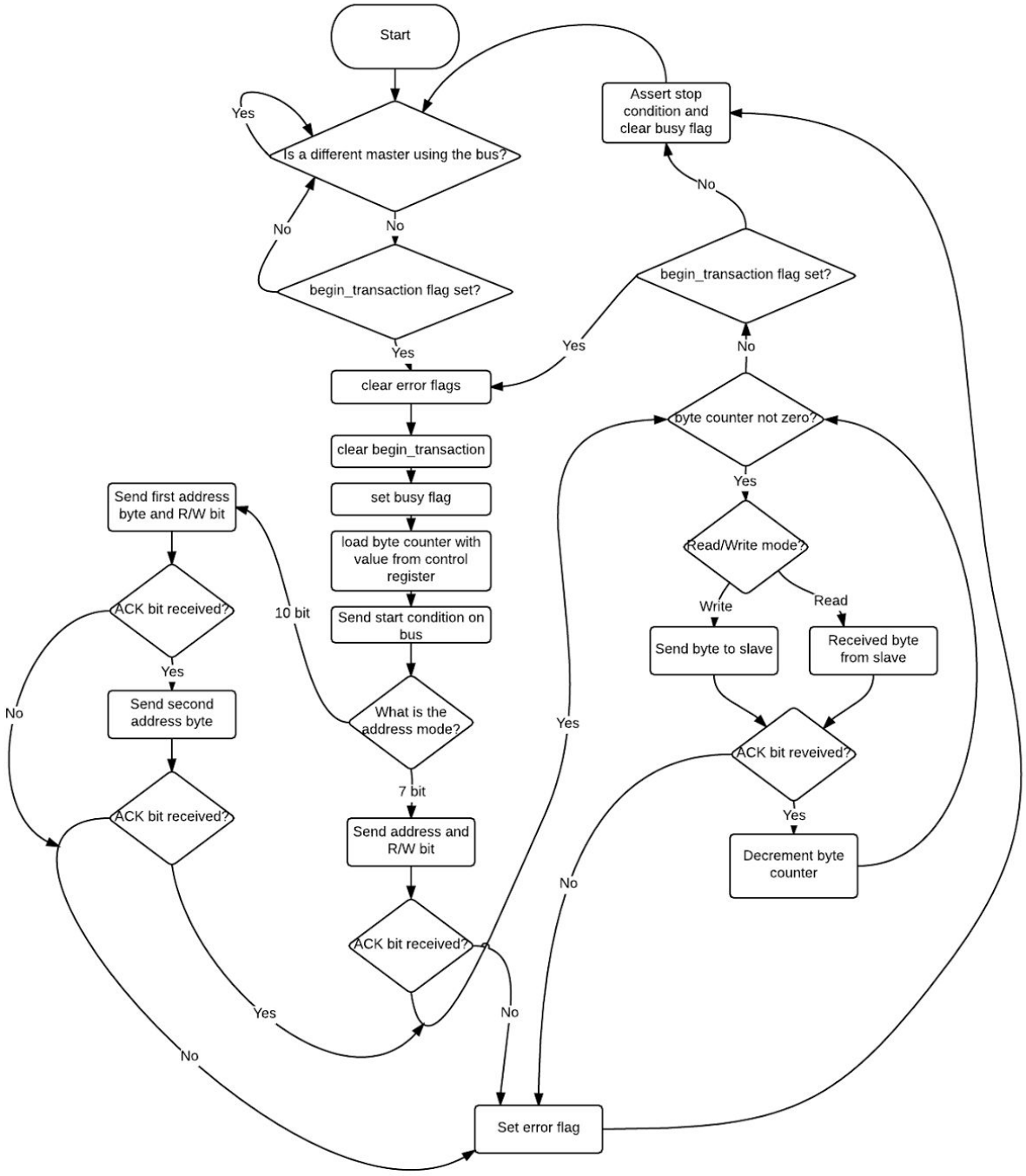
2.3.2 APB Slave

The I^2C will communicate with the rest of the chip by using the standard APB protocol. When the chip wants to write data using the I^2C the APB master will assert the write signal and declare the data as well as the address to write to. One clock cycle later the master will assert the enable and select line to tell which APB slave to write to. Lastly the ready signal will be asserted once the selected slave is ready to write and will stay asserted for one clock cycle. When the chip needs to read from the I^2C the write line will be asserted to low and the select line for the I^2C will be asserted high. One clock cycle later the slave will send the data to the master.



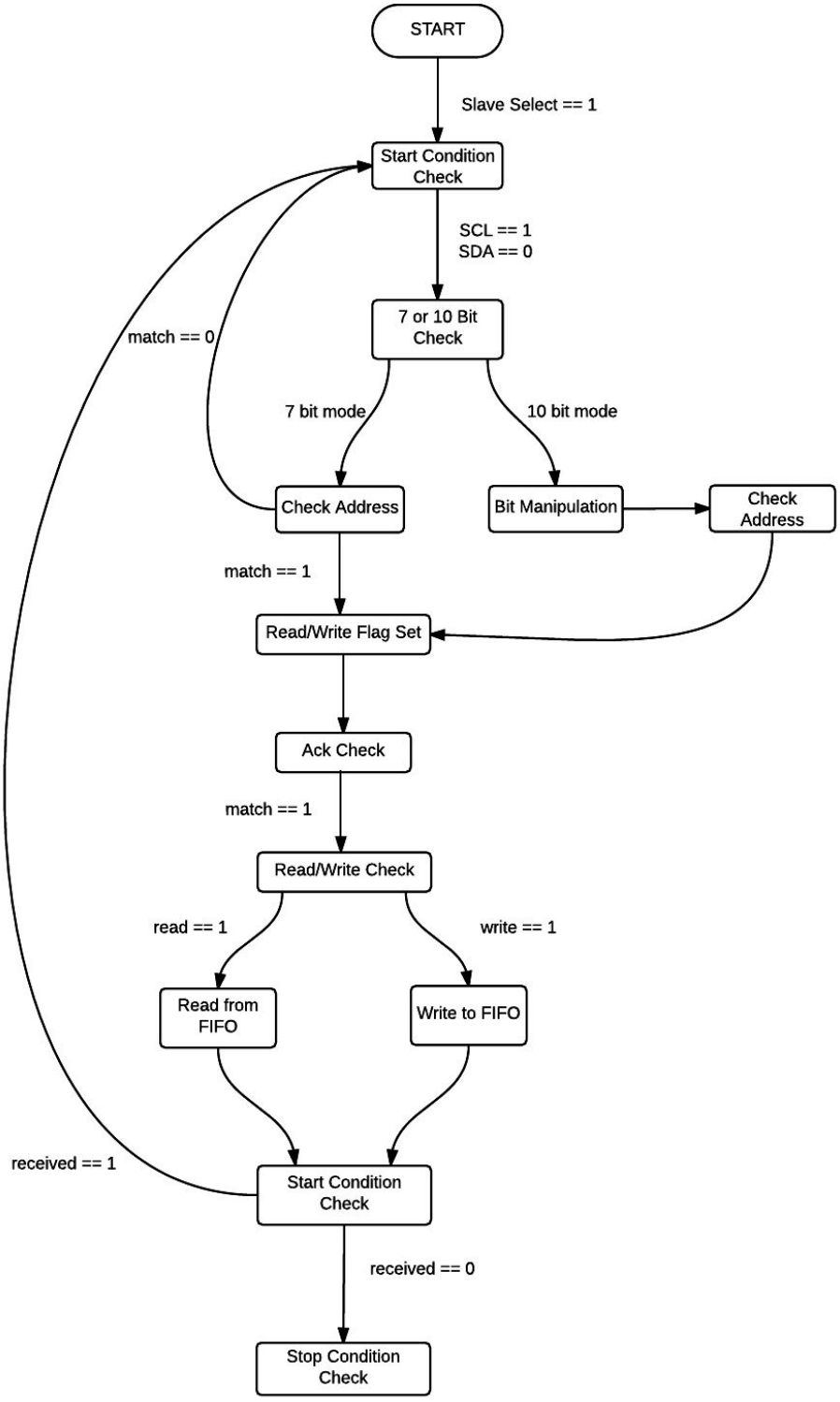
2.3.3 I^2C Master Algorithm

1. If a different master on the bus is currently sending/receiving data, wait until a stop condition has been received.
2. If the begin_transaction flag is set in the control register, clear the begin_transaction flag, set the busy status flag and move to the next step, otherwise end.
3. Load the number_of_bytes counter with the value specified in the control register. Load the timer with the baud rate specified in the control register.
4. Assert start condition on the data bus.
5. If in 7 bit mode, send address in control register and R/W bit and then check ACK bit. If in 10 bit mode, send both bytes of address in control register and R/W bit and then check ACK bit after each byte. If ACK bit is incorrect, assert stop condition, set the error status flag, and end the algorithm.
6. If in read mode, receive byte from slave and store in rx buffer. If in write mode, write byte from tx buffer to slave. Wait for ACK bit. If ACK bit incorrect, assert stop condition and set error status flag.
7. Decrement the number_of_bytes counter. It if is nonzero, go to step number 6. If it is zero, continue.
8. If the begin_transaction flag is set, go back to step 4. Otherwise, continue to next step.
9. Assert stop condition and clear the busy status flag.


Start
Is a different master using the bus?
Yes
No
begin_transaction flag set?
No
Yes
clear error flags
clear begin_transaction
set busy flag
load byte counter with value from control register
Send start condition on bus
What is the address mode?
10 bit
7 bit
Send first address byte and R/W bit
ACK bit received?
Yes
No
Send second address byte
ACK bit received?
Yes
No
Send address and R/W bit
ACK bit received?
Yes
No
Set error flag
byte counter not zero?
Yes
No
Read/Write mode?
Write
Read
Send byte to slave
Received byte from slave
ACK bit reveived?
Yes
No
Decrement byte counter
begin_transaction flag set?
Yes
No
Assert stop condition and clear busy flag

2.3.4 I2C Slave Algorithm

1. Check if the slave mode is selected from the Control register.
2. Check if the Start condition has been received i.e. check if SCK is pulled High and SDA is pulled Low.
3. Check the Control register for 7 bit mode or 10 bit mode.
4. Check if the address is ready from the I^2C Bus Address register.
5. If the Slave is in 7 bit mode, then grab the address sent by the master from the I2C Bus Address Register and check if the address matches the Slave address and then set the flag for Read/Write based on the bit following the 7 bits of address. Also if a particular Slave address matches the acquired address, then it should pull SDA line Low before 9th clock cycle to indicate Acknowledgment / Agreement with read and write of data.
6. If the Slave is in 10 bit mode, the I2C Bus Address Register has the address in the form {1111xyz,last{7:0}}. Perform bit manipulation i.e grab x and y bits and check if the 2 partial bits of the address match the first two MSB of the Slave address. Also, set the flag for Read/Write based on the bit following the z bit of the address. Following this perform another slave address check with the last[7:0] bits. If a particular Slave address matches the acquired address, then it should pull SDA line Low before 9th clock cycle to indicate Acknowledgment / Agreement with read and write of data.
7. Check the Read/Write flag, get or set the data to and from the Read/Write FIFO Buffer.
8. Check if Start condition has been received, If yes then Second Start condition has been reached. Start over from Step 3.
9. Check if the Stop condition had been received by checking a High to Low transition in the SDA followed by a High to Low transition in SCK, with SCK remaining High.


START
Slave Select == 1
Start Condition Check
SCL == 1
SDA == 0
7 or 10 Bit Check
match == 0
7 bit mode
10 bit mode
Check Address
Bit Manipulation
Check Address
match == 1
Read/Write Flag Set
Ack Check
match == 1
Read/Write Check
read == 1
write == 1
Read from FIFO
Write to FIFO
received == 1
Start Condition Check
received == 0
Stop Condition Check

2.4 Requirements

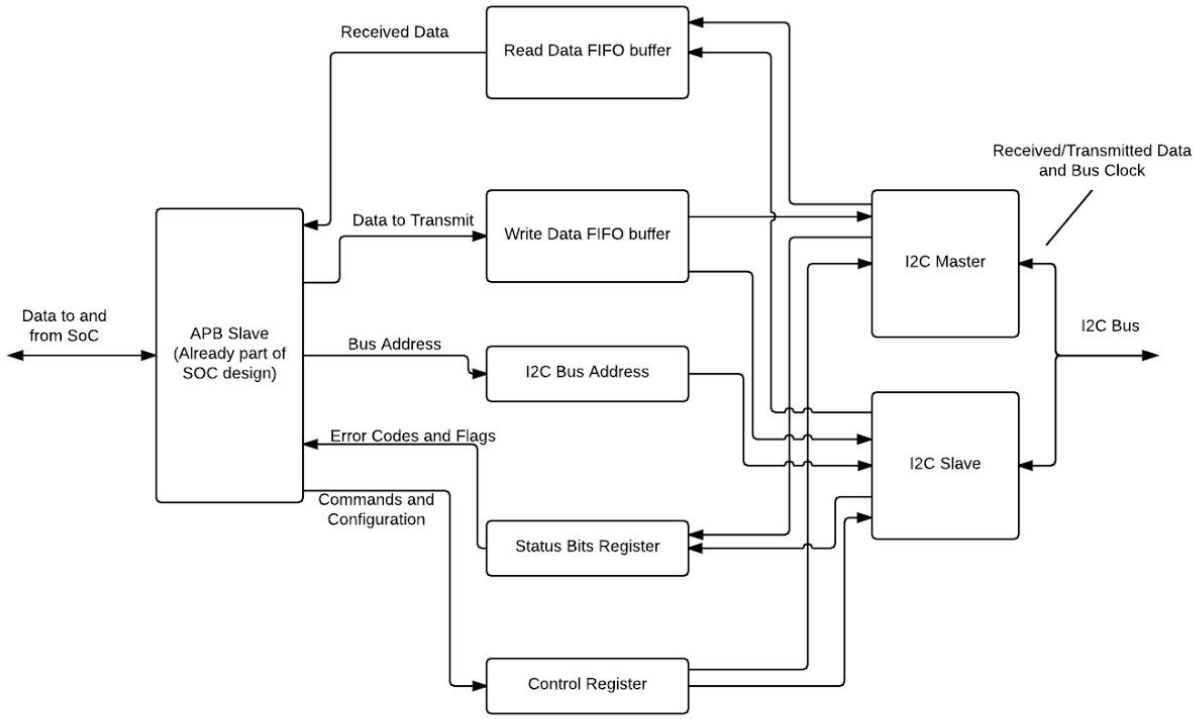
The I^2C module will be implemented as part of a preexisting System on a Chip (SoC). The SoC currently runs with a clock speed of about 60MHz, a clock speed which is liable to increase as the design is still being adapted and improved upon. The I^2C protocol define a maximum frequency (operating in 'high speed mode') of 3.2 MHz due to its open drain nature. This is considerably slower than the clock frequency of the system, which gives the I^2C module upwards of 20 cycles per bus clock. This means that speed is not a very important factor in the design.

Furthermore, as the SoC is still in development, it is desirable to reduce the number of logic gates and flip flops in order to make the design easier to fit on a Field Programmable Gate Array (FPGA) for testing purposes.

This gives the design a very clear optimization goal of reducing area and number of gates and flip flops. Algorithms that take more clock cycles but fewer gates should be implemented instead of ones that are fast and take up more area.

3.0 Design Implementation

3.1 Design Architecture

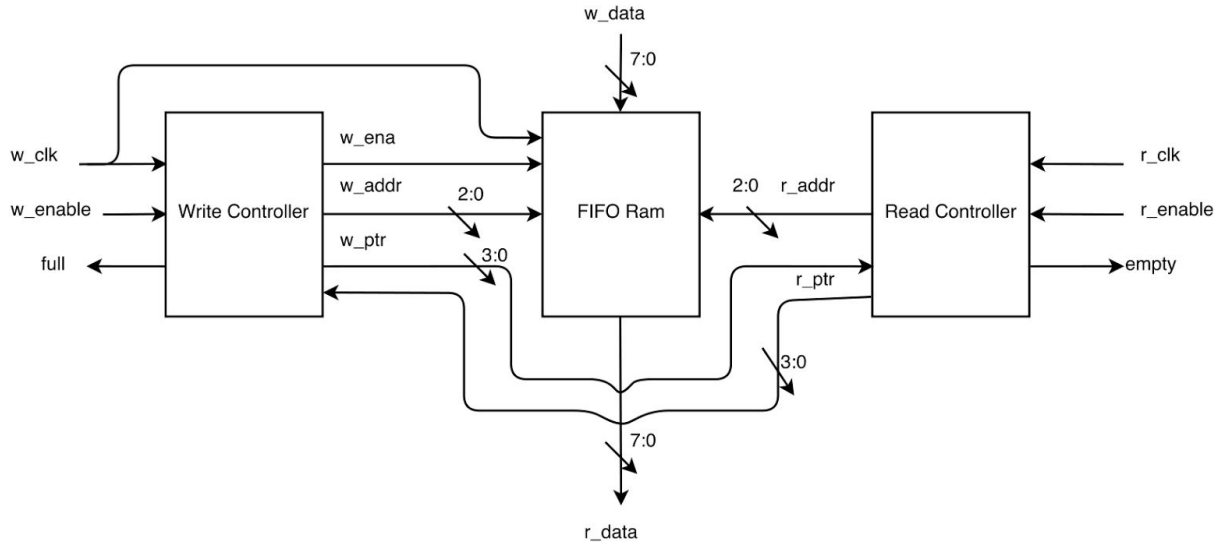


Component	Description
APB Slave	This component allows the SoC to communicate with the I^2C by reading from and writing to its registers
Read Data FIFO Buffer	A circular buffer to store data that has been received
Write Data FIFO Buffer	A circular buffer to store data to be transmitted
I^2C Bus Address	A register to store the address that the I^2C slave will respond to
Status Bits Register	A register to hold flags such as buffer overflow, or transmission complete
Control Register	A register to store configuration information, for example 7 bit address mode vs 10 bit address mode.
I^2C Master	The I^2C Master that will read and write to slaves on the I^2C bus

I²C Slave	The I²C Slave that will respond to masters on the I²C bus

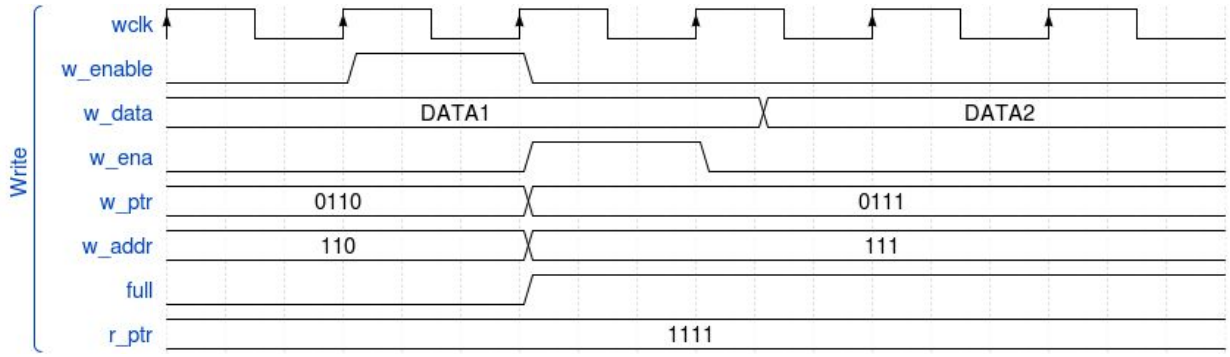
3.2 Functional Block Diagram

3.2.1 FIFO and Control Registers



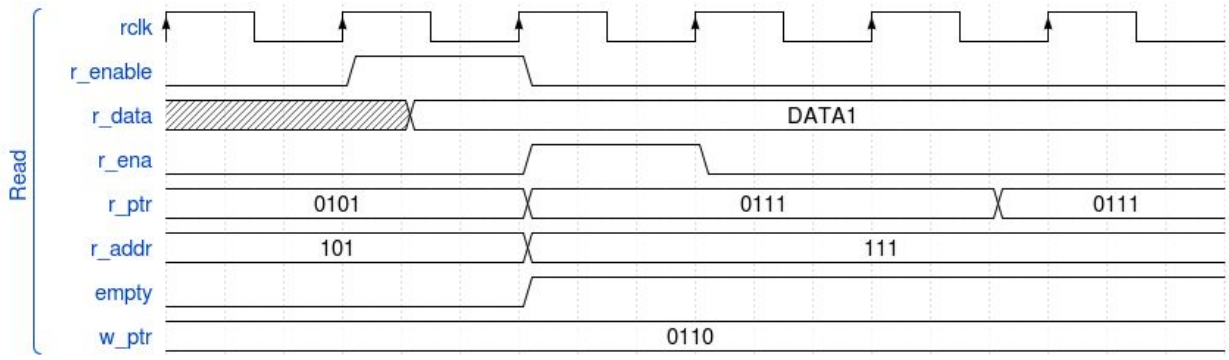
Writing to the FIFO

Writing to the buffer is simple, all the user is required to do is ensure the w_data is steady and then assert the w_enable signal for one clock cycle. The w_ena signal is asserted high for one clock cycle alerting the FIFO to write the new data to the address pointed to by the w_addr signal. After writing to the specified address the address pointer will then increment. The w_addr signal is equal to the 3 LSBs of the w_ptr, and the MSB of the w_ptr is used in both full and empty checks. The full flag will be set high if the MSB of w_ptr and r_ptr are different, and if the other 3 bits of each signal are equal to the last 3 bits of the other signal.

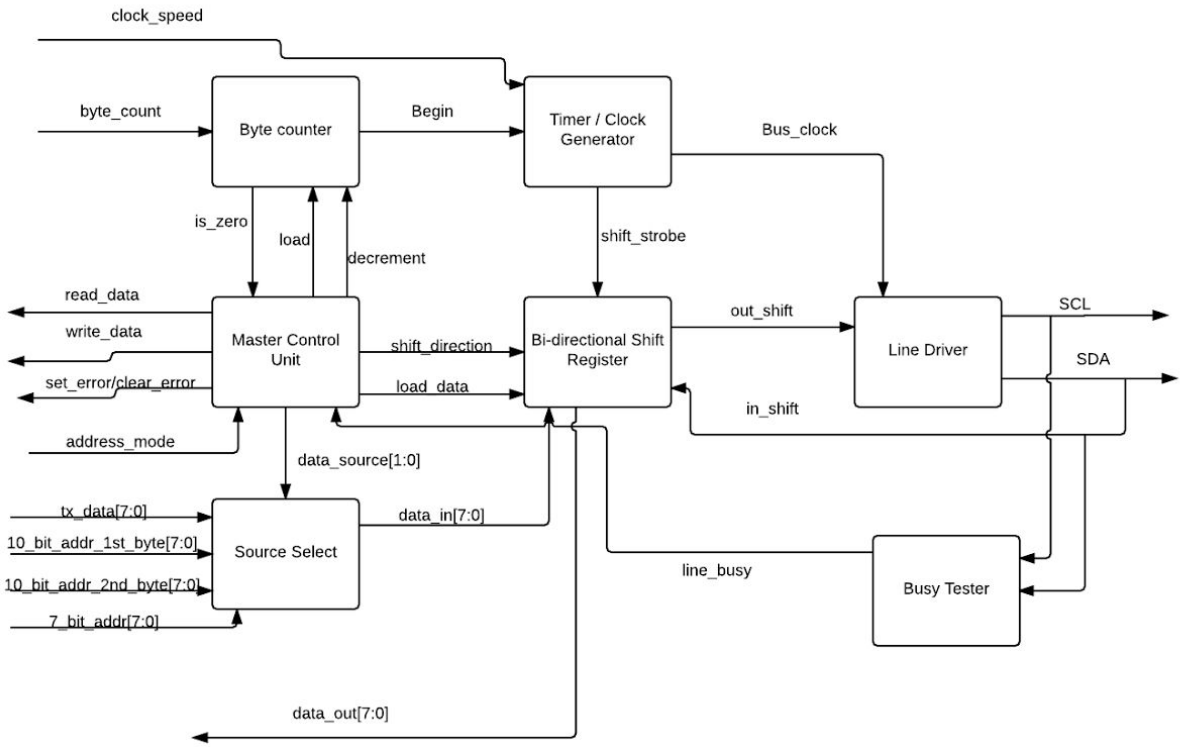


Reading from the FIFO

The protocol from reading from the FIFO is very similar to writing to the FIFO. To begin reading from the FIFO assert the r_enable line, this will alert the controller to increment the r_ptr. The FIFO Ram will detect this change in the r_addr and will change the value of the r_data to the data at the specified address. The empty flag will be set when the value of the r_ptr is equal to the value of the w_ptr.



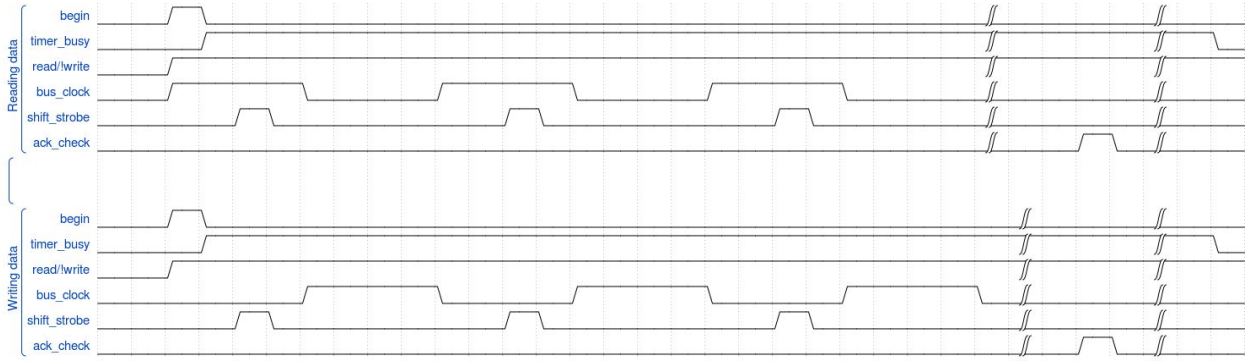
3.2.2 I2C Master



Master Control Unit

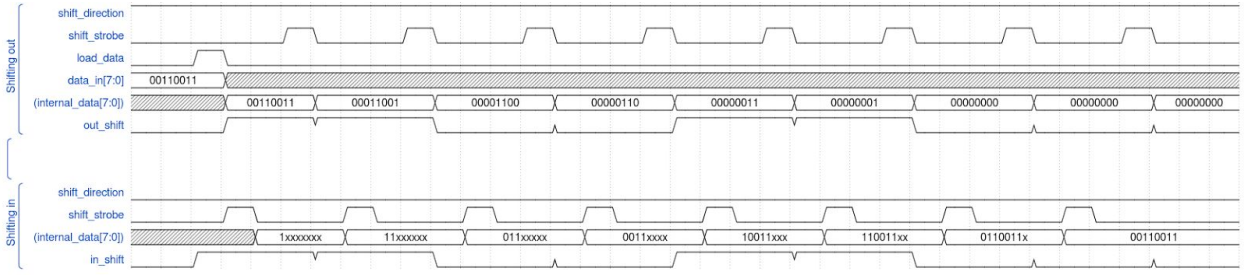
The master control unit is a Moore machine that controls the flow of the I2C master. It has several lines coming into and out of it and has a complicated, branching flow of operation. Because of this, a timing diagram would be ill suited to describe its operation. Please refer the the flow chart in section 2.3.3 to understand how this unit works.

Timer/Clock Generator



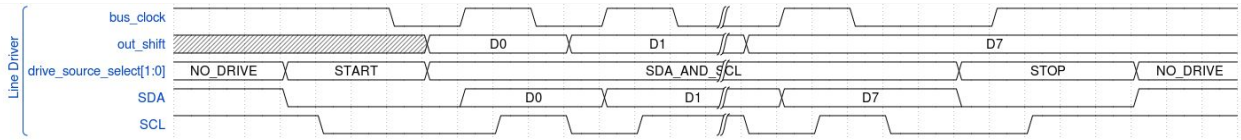
The timer keeps track of the length of the byte, shifts the shift register, generates the SCL signal, and also signals to the Master Control Unit when it is time to check the ACK bit. The timer behaves slightly differently depending on whether it is transmitting or receiving. When transmitting, the shift register is shifted while SCL is low, because the master is driving data on SDA. When receiving, the shift register is shifted while SCL is high because the master is reading data from SDA.

Bi-directional Shift Register



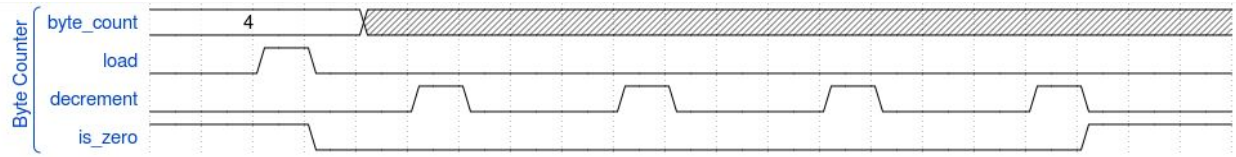
The shift register can both read a packet from the transmit fifo and write it to the line, and read a packet from the line and store it in the receive fifo. Doing this with the same shift register saves flip flops and helps reduce the area of the design.

Line Driver



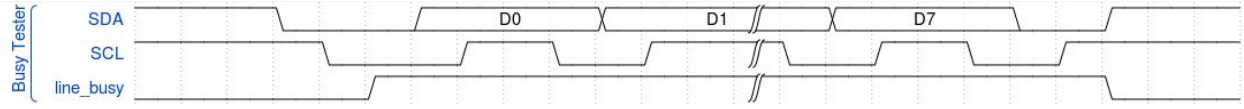
The line driver is a state machine that has 4 modes of operation. In NO_DRIVE mode, no values are driven on SDA and SCL, and so the lines default to '1' because they are open drain. In START mode, a start condition is driven on the line and then both lines are held low. If in SDA_AND_SCL mode, SDA follows the out_shift line and SCL follows the bus_clock line. Finally, in STOP mode, a stop condition is asserted on the line and then both output are held high.

Byte Counter



The byte counter keeps track of how many bytes of data have been sent/received. When the Master has sent/received all of the bytes (packet length is specified in a control register), then the is_zero line is asserted to inform the Master Control Unit.

Busy Tester

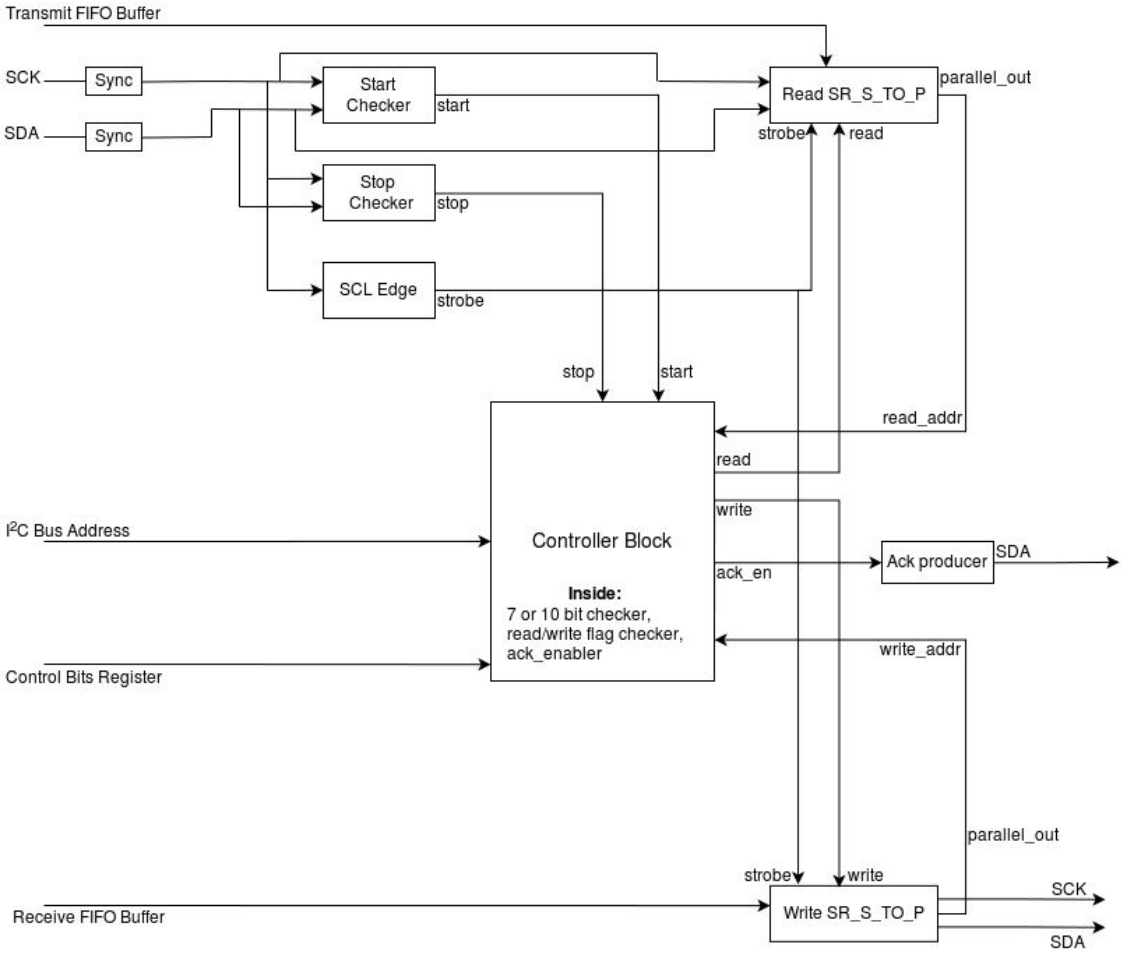


The busy tester is a simple state machine to let the master know if the line is currently busy, whether by this master, or another master on the bus. If the line is busy while this master wants to initiate a communication, it will wait. This prevents multiple masters from talking on the bus at the same time.

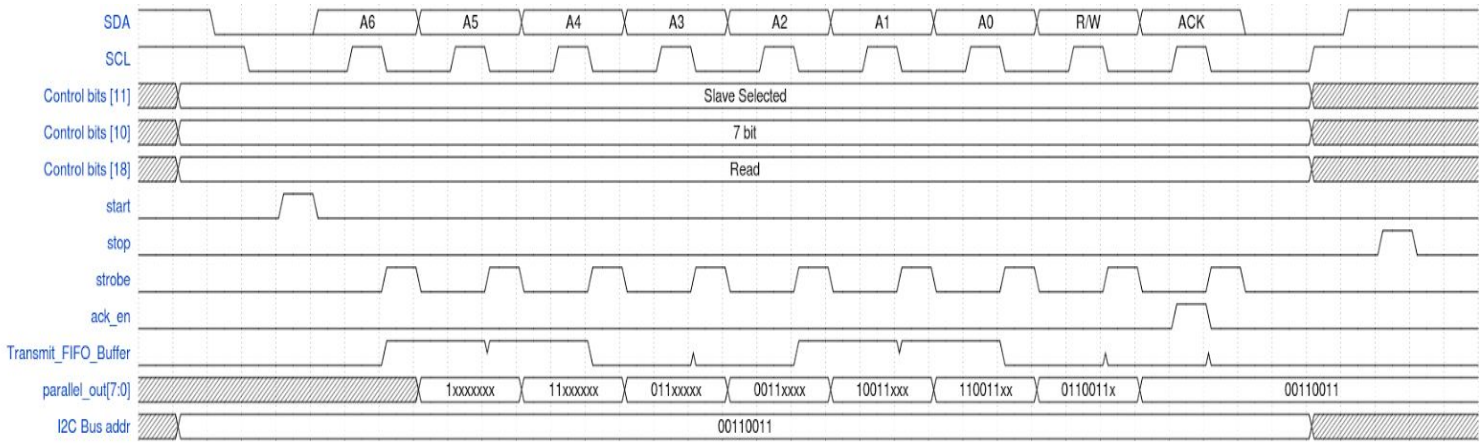
Source Select

Source select needs no timing diagram as it is simply a multiplexer. It determines which values gets loaded into the shift register to be driven on the line.

3.2.3 I2C Slave

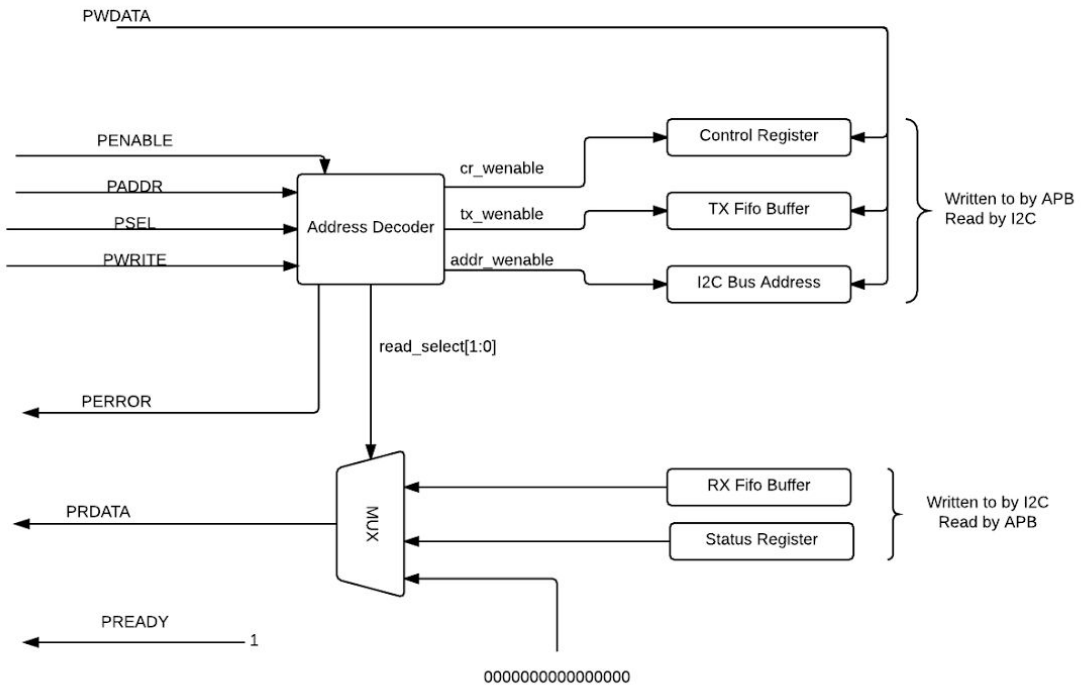


This is the I^2C Slave module. The Controller Block performs most of the tasks like checking for 7 or 10 bit mode and storing the value for the same in it. Similarly, it does the task of checking the read/write flag. It also outputs a signal ‘ack_en’ which holds the SDA line Low to acknowledge connection to the Master device. Also, please note that the SCK edge is actually the SCK Rising Edge Detector which serves as the ‘shift_en’ signal for the serial to parallel shift registers. Also when the parallel_out data from the shift register is the same as the addr on the I^2C bus, then the Controller Block generates ‘ack_en’ signal. Following to this description is a timing waveform diagram of how the different signals progress through the slave and finally produce the desired output.



The ‘start’ signal is asserted when SDA is pulled Low before SCK is pulled Low. The ‘stop’ signal is asserted when SCK is pulled High before SCK is pulled High. The control bits 11, 10 and 18 correspond to Master/Slave Select, 7 or 10 bit Mode Select and Read/Write Flag respectively. The ‘strobe’ signal is generated every-time the SCK has a 0->1 transition. The ‘parallel_out’ signal is the parallel_out of the shift register at the instantaneous time. The comparison of parallel_out and I^2C Bus Address is performed when the strobe signal has been asserted 8 times. Based on this comparison the ‘ack_en’ is generated.

3.2.4 APB Slave



The APB slave can be implemented entirely with combinational logic, with the exception of the registers and FIFO Buffers. This is because the design does not utilize the PREADY signal to extend the amount of time for writes and reads. The "state" can be determined from the external signals PSEL and PENABLE and do not need to be tracked within the slave.

Address Decoder

This is the main unit behind the APB slave. When both PENABLE and PSEL are high, the unit will use the PADDR line in conjunction with the PWRITE line to select the appropriate register. For example, if the PADDR is the control register address and PWRITE is 1, then the cr_wenable line will be asserted. If PADDER is the RX Fifo buffer address and PWRITE is 0, then the read_select line will be asserted so that the MUX outputs the value of the RX Fifo buffer on the line.

If an invalid PADDR and PWRITE combination is asserted (writing to a read register, reading from a write register, or an address that isn't mapped) while PSEL and PENABLE are asserted, then PERROR is asserted to inform the APB master that it attempted an incorrect operation.

4.0 Projected Timeline and Division of Tasks

4.1 Projected Timeline

Week	Goals
3/21/2016 - 3/25/2016	Create modules
3/28/2016 - 4/1/2016	Finish modules
4/4/2016 - 4/8/2016	Create Test benches and begin debugging
4/11/2016 - 4/15/2016	Debug
4/18/2016 - 4/22/2016	Test the I2C with the rest of the SOC using FPGA
4/25/2016 - 4/29/2016	Finish testing with the rest of the SOC
5/1/2016 - 5/6/2016	Final Presentation

4.2 Division of Tasks

Group Member	Tasks
Arnav Mittal	I2C Slave
Eric Colter	I2C Master
Sam Sowell	FIFO and APB Integration

5.0 Success Criteria

5.1 Fixed Criteria

1. **(2 Points)** Test benches exist for all top level components and the entire design. The test benches can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria.
2. **(4 Points)** Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings.
3. **(2 Points)** Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors expect at time zero.
4. **(2 Points)** A complete IC layout is produced that passes all geometry and connectivity checks.
5. **(2 Points)** The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determine by courses staff on your design review. Failure to reach any of the targets will result in a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0 out of 2.

5.2 Design Specific Criteria

6. **(2 Points)** Demonstrate by simulation of a Verilog test bench that the final design is APB compliant.
7. **(2 Points)** Demonstrate by simulation of a Verilog test bench that the final design correctly runs on a FPGA.
8. **(2 Points)** Demonstrate by simulation of a Verilog test bench that the final design correctly writes the data provided on the APB bus to the FIFOs.
9. **(2 Points)** Demonstrate by simulation of a Verilog test bench that the final design correctly writes values read from the I2C to the FIFOs.