# ECE 337: ASIC Design Lab 6

Simplified I$^2$C Transmit-only Slave

Individual Mini-Design Project

*Week of 9/23/2013*

## Down the road a bit: The Project

- Think of a design that is applicable for an ASIC (or a FPGA). **A microcontroller should not be adequate (no elevator controllers)**
- Think about a parallel design
- Blackboard contains past project ideas. Good ideas include protocol interfacing and signal processing… (no simple-math co-processors)
- Due to time limitations, a subset of a given protocol may be acceptable
- Work in groups of 3-4 **only in your lab section**
- See Project Idea Guidelines posted online (due next Friday), you are encouraged to submit multiple ideas.

## Lab 6 High-Level

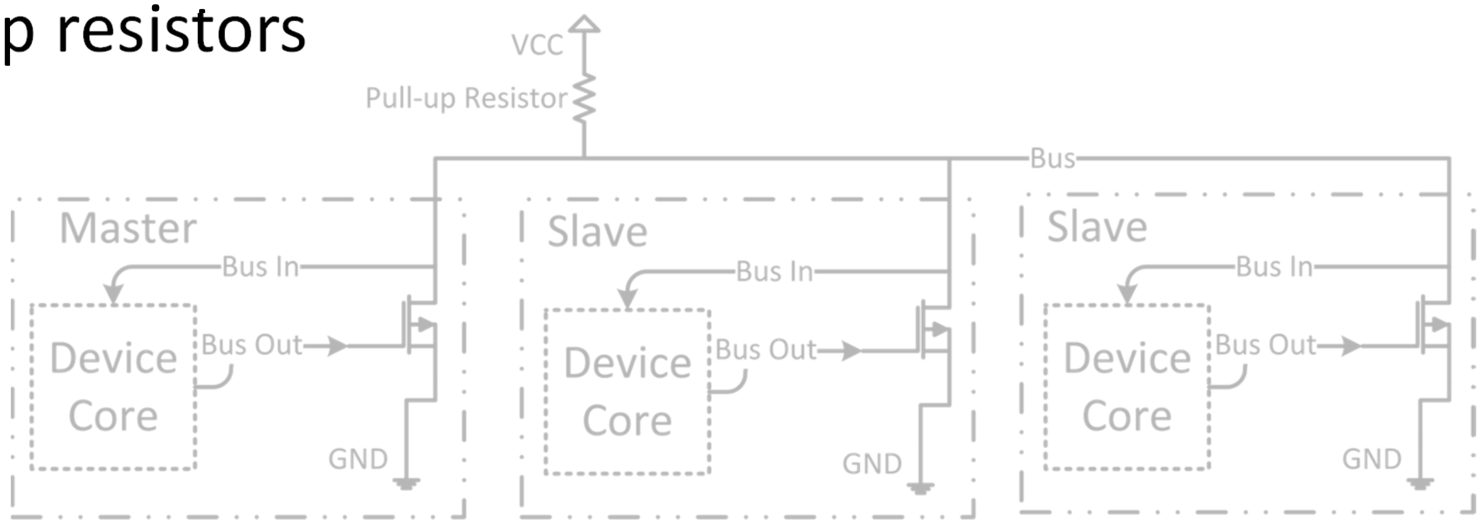
- Will be designing a simplified I$^2$C Transmit-only Slave
- Two-week, **individual lab**
- Two main phases with optional source grade check
  - Preparation Diagrams
    - Due by end of 4th TA office hours after your lab
  - Phase 1: Sub-block code & test benches
    - Due late night the day of your lab next week
  - Optional complete source only grade submission
    - 48 hours prior to Phase 2 deadline
  - Phase 2: Complete mapped version
    - Start of Lab week after next (Week of 10/7)

## Simplified I$^2$C Slave Interface

- Transmit-only I$^2$C Slave interface, subset of I$^2$C Slave spec.
  - Transmits data to a master upon request
  - Takes data to transmit out of a FIFO
  - Higher level functions not handled for lab feasibility (Bus arbitration, multiple masters on the same bus, etc.)
- Uses separate buses for data and the data clock
  - SDA line is the data bus
  - SCL line is the data clock bus
- “Starting”/”Target” byte for a transmission
  - Contains address/id of target device
  - Contains the data transfer mode (read/write)
- Clock synchronization (using data transitions)
- Real I$^2$C includes “clock-stretching” (we won’t)
  - used to supply flow control feedback from a slave to a master
- Uses START conditions to signal the start of a transmission
- Uses a STOP condition to signal the end of a transmission
- Transmit Data written to FIFO Queue from interface to external device (provided)

## I$^2$C Bus Characteristics

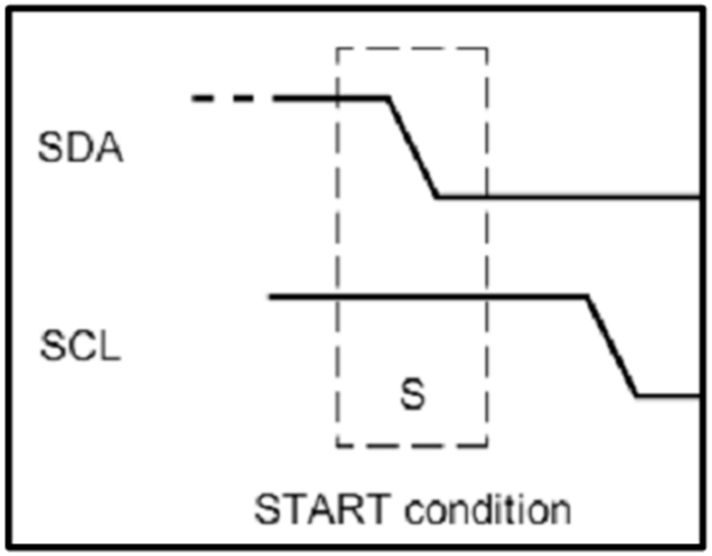
- Each wire (SDA, SCL) is a bi-directional bus
- Each wire (SDA, SCL) operates with a 0's priority
  - 0's override 1's if written to the bus simultaneously
- Implemented using open-drain connections with pull-up resistors



## Start Condition

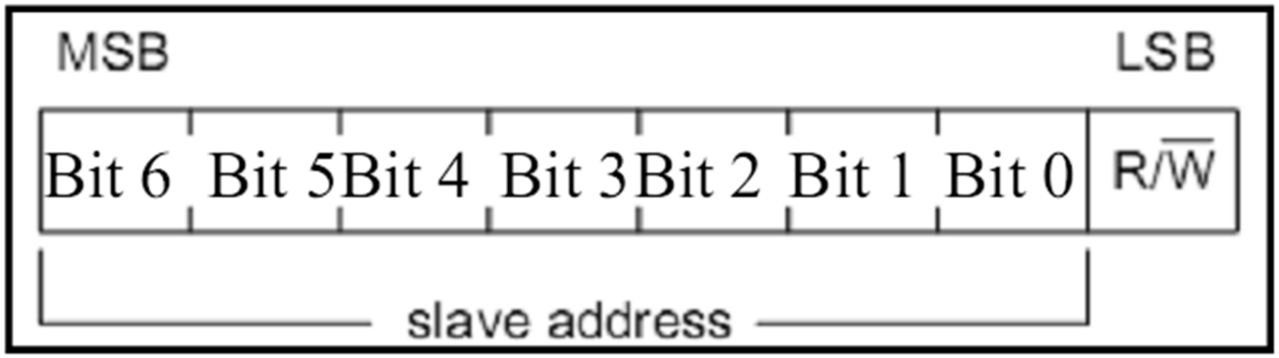
- High to low transition on the SDA Line while the SCL line is high
- Signals the start of a new transmission

  (wakes up all slaves)



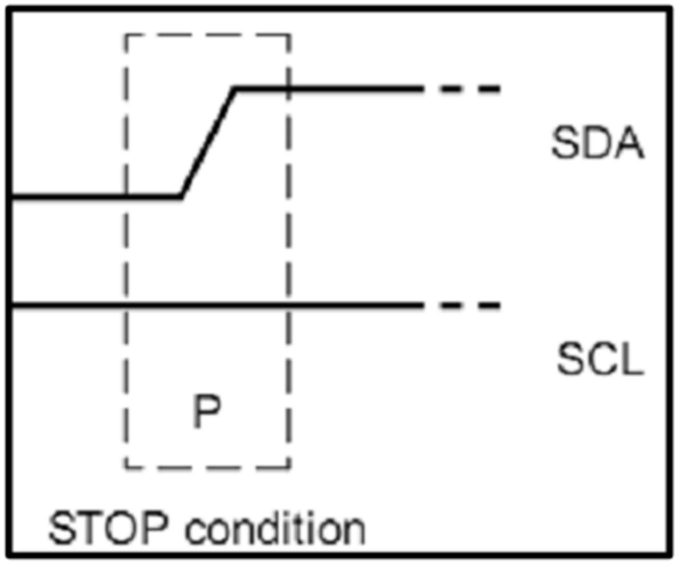
## Starting/Target Byte

- The first byte sent following a START condition
- Must be acknowledged by the identified slave to start transfer, otherwise transmission is terminated
- Read/Write mode is from the perspective of the master
  - Read: Master reads from bus/Slave writes to bus
  - Write: Master writes to bus/Slave reads from bus



## STOP Condition

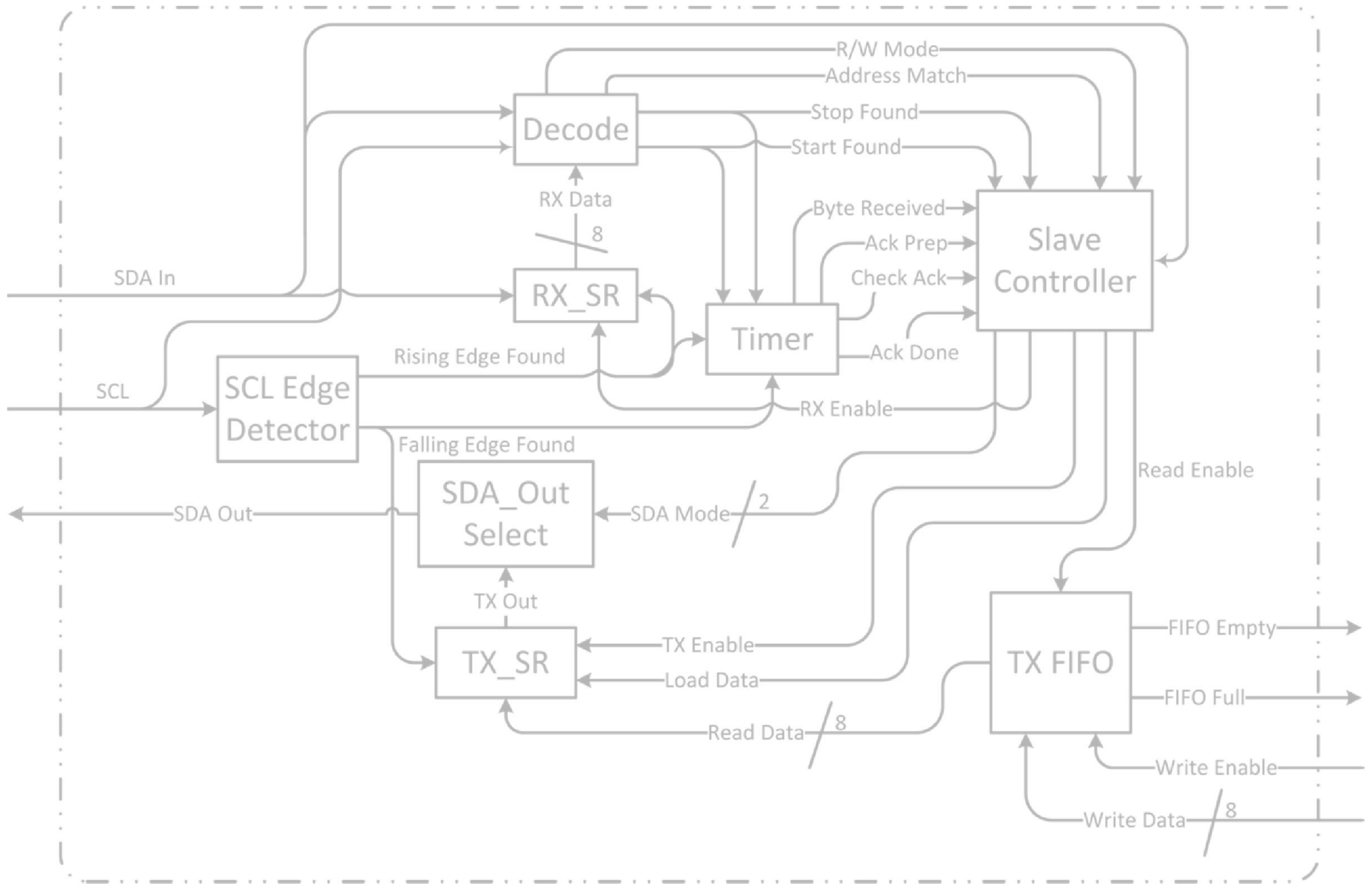
- Low to High transition on the SDA line while SCL line is high
- Signals the end of a transmission



# Complete Transmission

1. Begins with START Condition, followed by a “Starting” byte
2. Identified Slave acknowledges (ACK) the master
3. The slave sends data byte from FIFO
4. The slave checks for ACK from Master
5. Repeat steps 3 & 4 until master not-acknowledges (NACK)
6. Transmission is done
   - Need to hold on to last data byte for a retransmission/future transmission requested by the master.

## System Overview

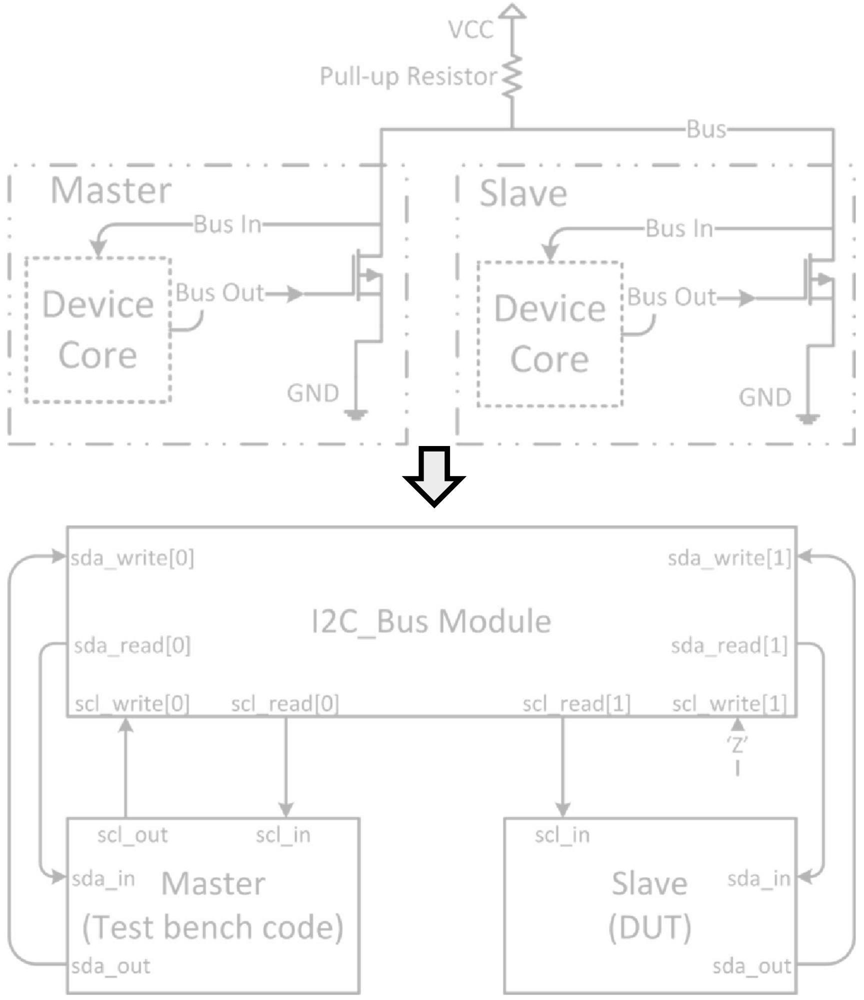


## TX FIFO

- The FIFO is **provided**, you need to make a “wrapper” file.
- FIFO writes data from write_data on a clock edge, while write_enable is high.
- FIFO outputs data on read_data, increments to next byte when read_enable is high and there is a clock edge.
- fifo_full and fifo_empty indicate FIFO status
- Provided FIFO has two clocks
  - Why? This FIFO was designed to provided interface between circuits running on different clocks
  - you will tie them to same system clock

# Simulating the Open-drain buses

- Use the provided i2c_bus module
  - Connect your slave design to it
  - Connect the test bench's emulated master signals to it
- It will implement and enforce all of the open-drain & pull-up resistor characteristics
- It has assertions to check if multiple devices are trying to drive the bus low
- You can view what happens to the bus by adding the internal signals (bi_dir_sda & bi_dir_scl)


VCC
Pull-up Resistor
Bus
Master
Bus In
Device Core
Bus Out
GND
Slave
Bus In
Device Core
Bus Out
GND
I2C_Bus Module
sda_write[0]
sda_read[0]
scl_write[0]
scl_read[0]
scl_read[1]
scl_write[1]
sda_write[1]
sda_read[1]
'Z'
scl_out
scl_in
sda_in
Master
(Test bench code)
sda_out
scl_in
Slave
(DUT)
sda_in
sda_out