Linux I2C Driver

Content

- Getting Started with I2C protocol
  - I2C Features
  - I2C Data Transfer Modes
  - I2C Terminology
  - I2c Protocol format
    - Data Validity
    - Start & Stop Conditions
- AM335X I2C Controller
- DS1307 I2C Slave Device

Getting Starting with I2C communication Protocol

LINUX I2C DRIVERS

I2C Features

- Philips Semiconductors (now NXP Semiconductors, Qualcomm) developed a simple
  - Serial
  - 8-bit oriented,
  - bidirectional
  - 2-wire
  - synchronous communication protocol.
- Only 2 lines:
  - SDA (Serial Data Line)
  - SCL (Serial Clock Line)
- I2C protocol derived from System Management BUS (SMBUS – developed by INTEL)

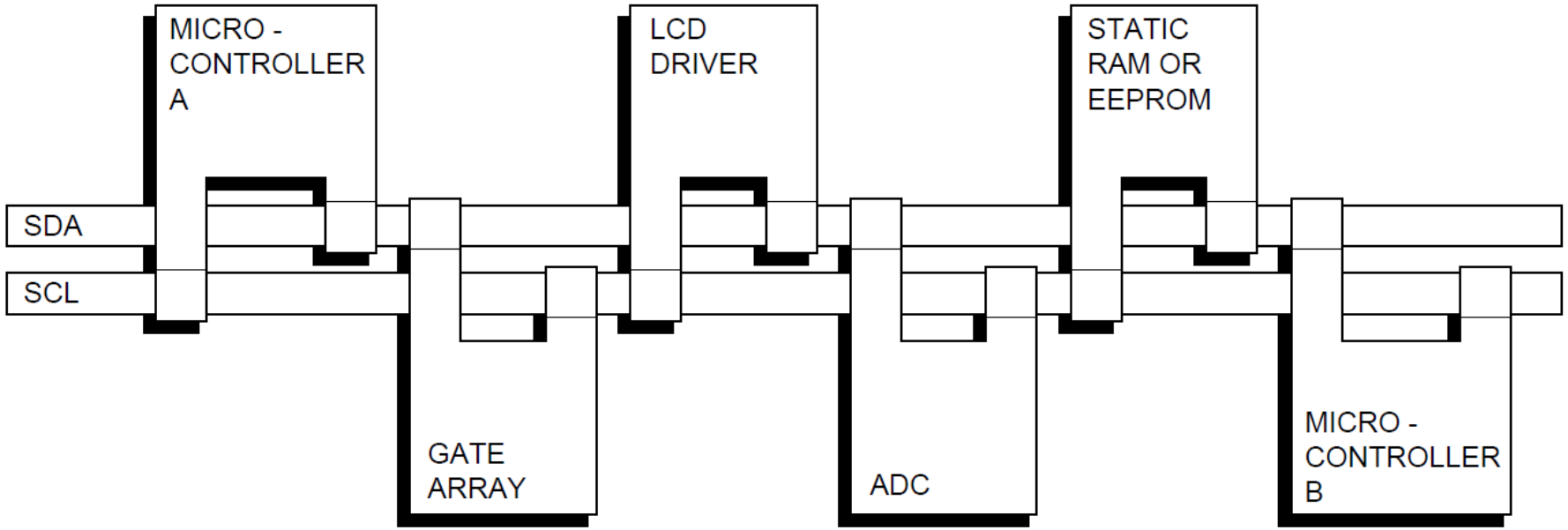
I2C Features

- I2C various Data transfer modes:
  - Standard-mode: 100kbit/s
  - Fast-mode: 400kbit/s
  - Fast-mode Plus (Fm+): 1Mbit/s
  - High-speed mode: 3.4 Mbit/s
  - Ultra Fast-mode: 5 Mbit/s (Unidirectional mode)
- Each device connected to the bus is software addressable by a unique address.
- simple master/slave relationships.
- masters can operate as master-transmitters or as master-receivers

I2C Features

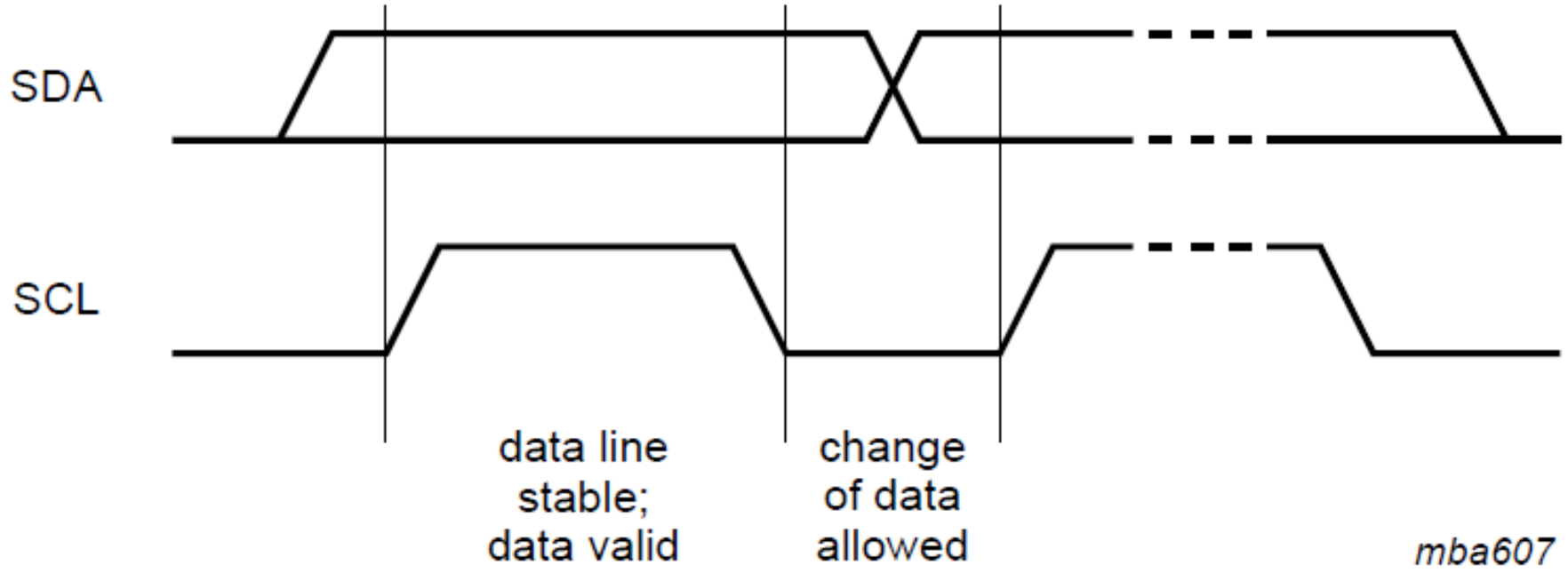
Term	Description
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	the device which initiates a transfer, generates clock signals and terminates a transfer
Slave	the device addressed by a master
Multi-Master	more than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted

I2C Bus Configuration



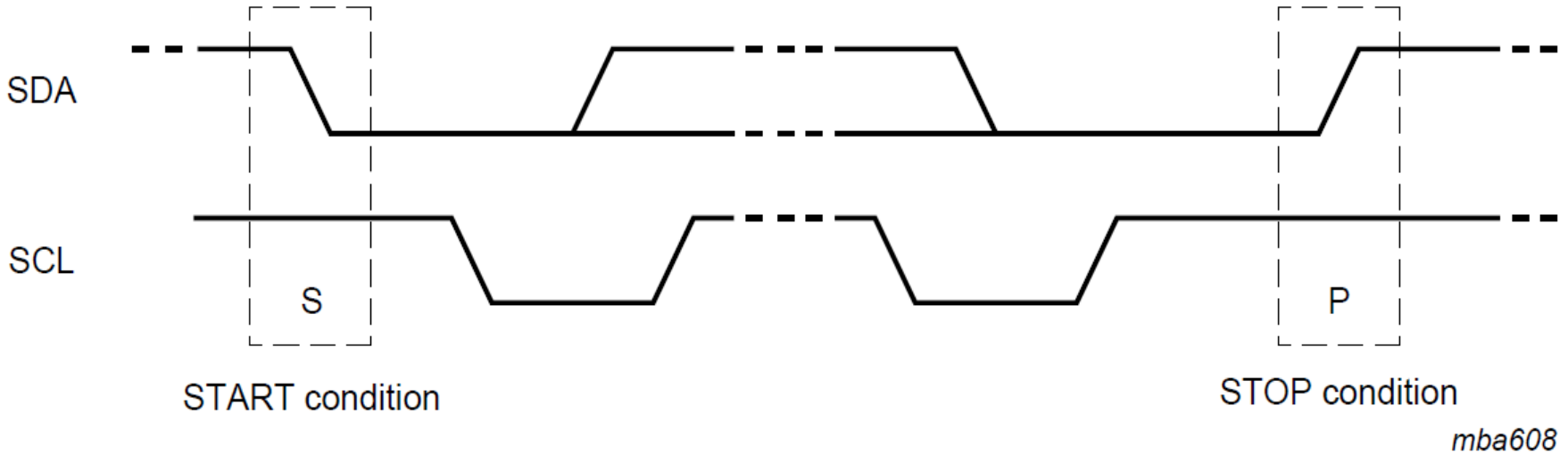
Data Validity

- The data on the SDA line must be stable during the HIGH period of the clock.
- The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW (see Figure).
- One clock pulse is generated for each data bit transferred



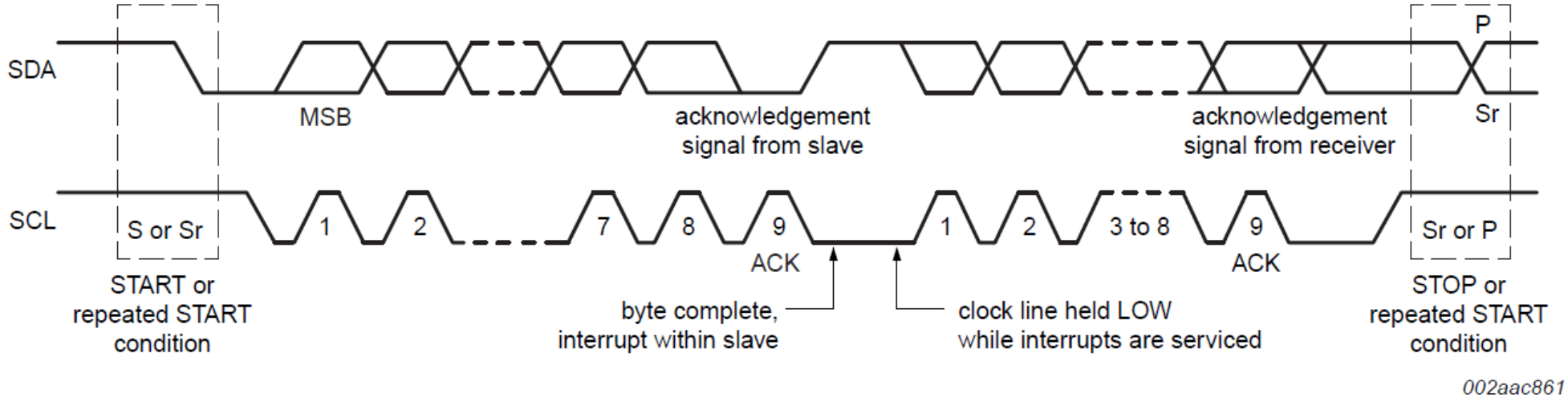
Start and Stop Conditions

- A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition.
- A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.
- START and STOP conditions are always generated by the master.



Byte Format

Every byte put on the SDA line must be eight bits long. The number of bytes that can be transmitted per transfer is unrestricted.
Each byte must be followed by an Acknowledge bit.
Data is transferred with the Most Significant Bit (MSB) first.



Acknowledge (ACK) and Not Acknowledge (NACK)

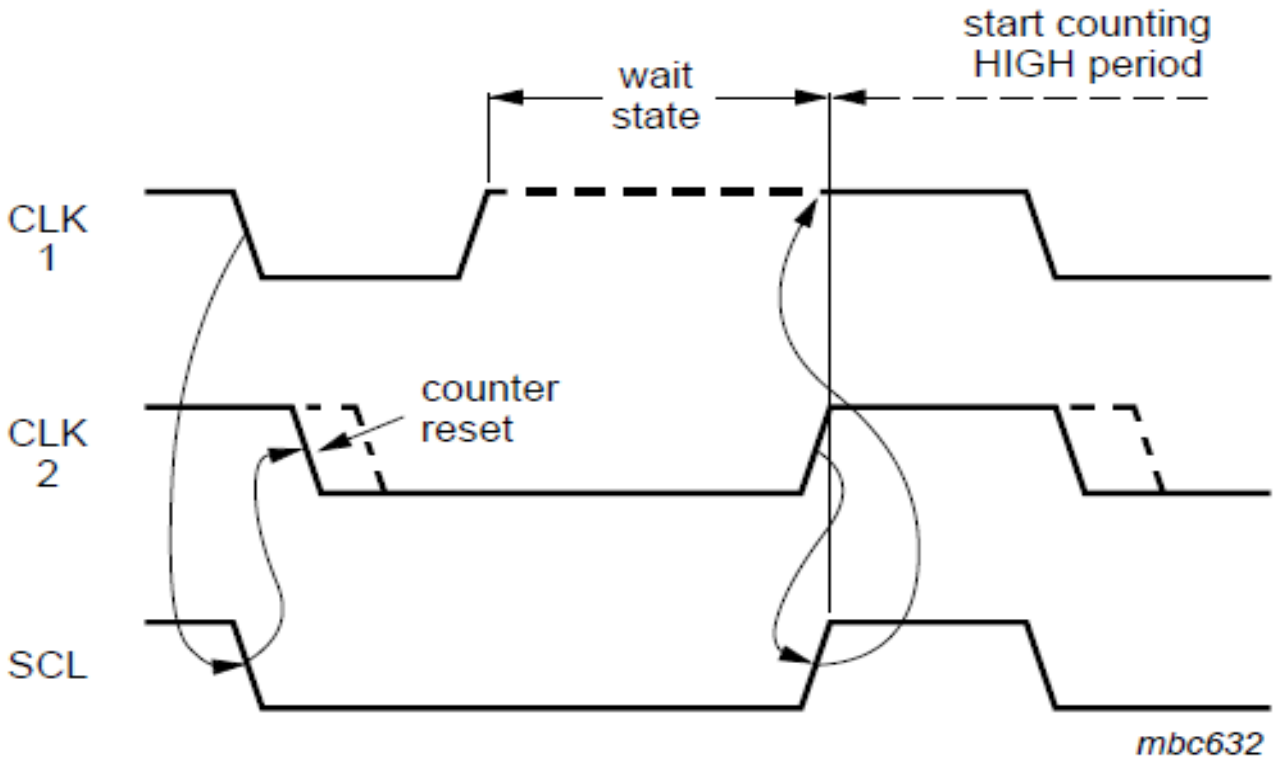
- **ACK Defines:** the transmitter releases the SDA line during the acknowledge clock pulse so the receiver can pull the SDA line LOW and it remains stable LOW during the HIGH period of this clock pulse.
- **NACK Defines:** When SDA remains HIGH during this ninth clock pulse, this is defined as the Not Acknowledge signal.

Acknowledge (ACK) and Not Acknowledge (NACK)

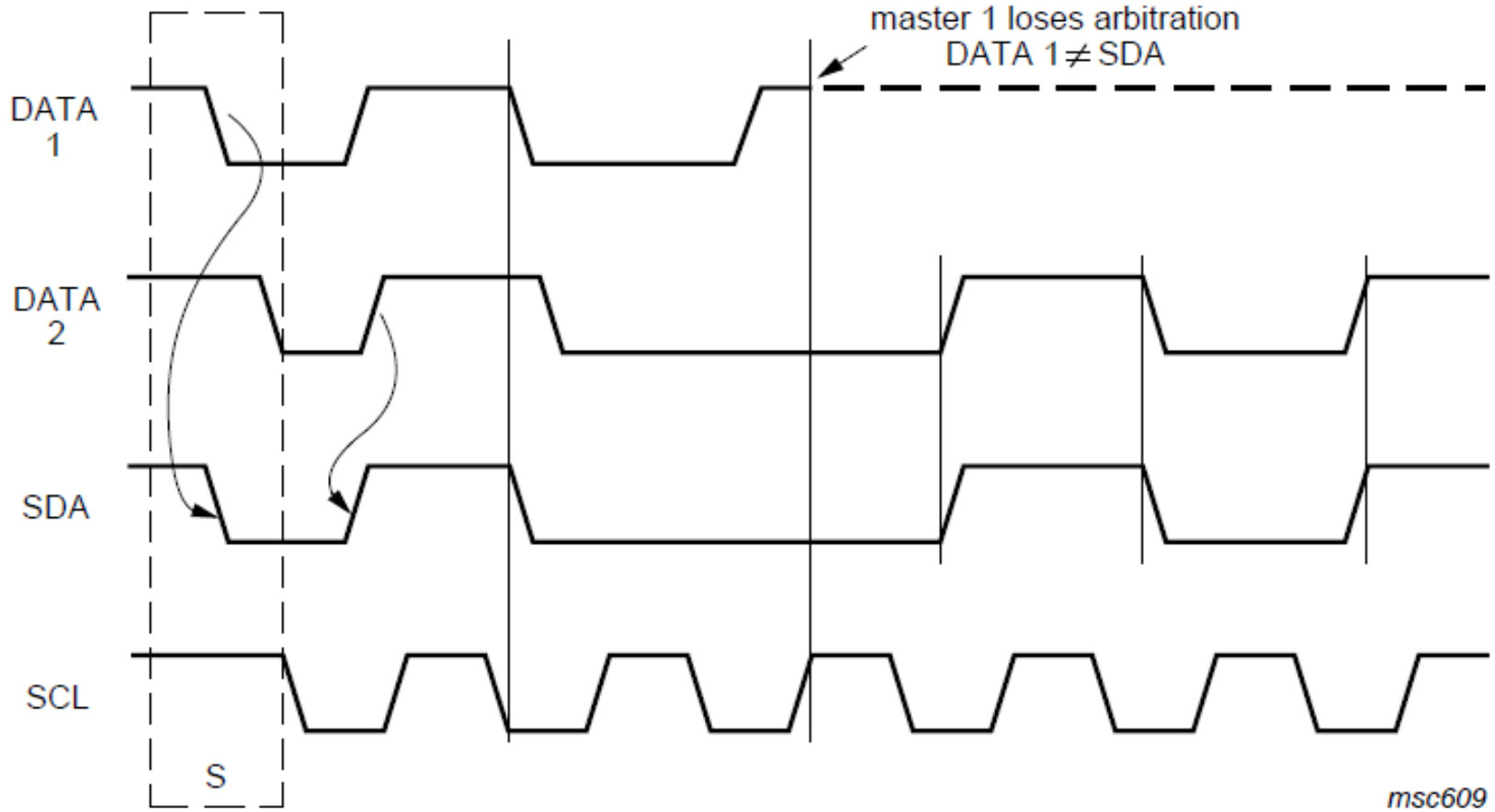
There are five conditions to the generation of a NACK:

1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start communication with the master.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A master-receiver must signal the end of the transfer to the slave transmitter.

Clock Synchronization



Arbitration



I2C Bit Banging

It is hard to tell from your schematic but in some instances I2C interfaces are implemented using general purpose I/O port pins and then bit banged in software.

Sometimes the implementer may not operate the I/O pins in this configuration using an open drain methodology and this may play a factor on why an interface without pullup resistors may seem to work.

I2C FAQ

Q. What happens if I omit the pullup resistors on I2C lines?

A. There will be no communication on the I2C bus. At all. The MCU will not be able to generate the I2C start condition. The MCU will not be able to transmit the I2C address.

Q. The lack of pullups is likely to damage any of those two ICs in my board?

A. Even without the internal pull-ups, a lack of any pull-ups will not damage either IC. The internal build of i2c device SCl and SDA lines are like NPN transistors. They are Open Collectors, essentially current controlled/switched diodes.

.

I2C FAQ

Why need pull up resistors in I2C?

- Generally you will need to have the pullup resistors for an I2C interface circuit. If the interface is truly a full spec I2C on both ends of the wires then the signal lines without the resistors will never be able to go to the high level.
- They may remain low or go to some intermediate level determined by the leakage current in the parts at each end.
- The reason for this is because true I2C is an open drain bus.
- I2C is a TTL-logic protocol; so your data and clock lines are open-drain. In other words, the I2C hardware can only drive these lines low; they are left floating when not a zero. That's where the pull-up resistors come in.
- Some devices may actually have on-chip pullup resistors in the 20K to 100K ohm range just to hold the interface pins at a high inactive level when the I2C interface on the part is not in use. For simple and short interfaces these pullup resistors may be just enough to provide the current needed to pull the lines high while clocks and/or data is being signaled.
- i2c pull up resistors allows for features like concurrent operation of more than one I2C master (if they are multi-master capable) or stretching (slaves can slow down communication by holding down SCL).

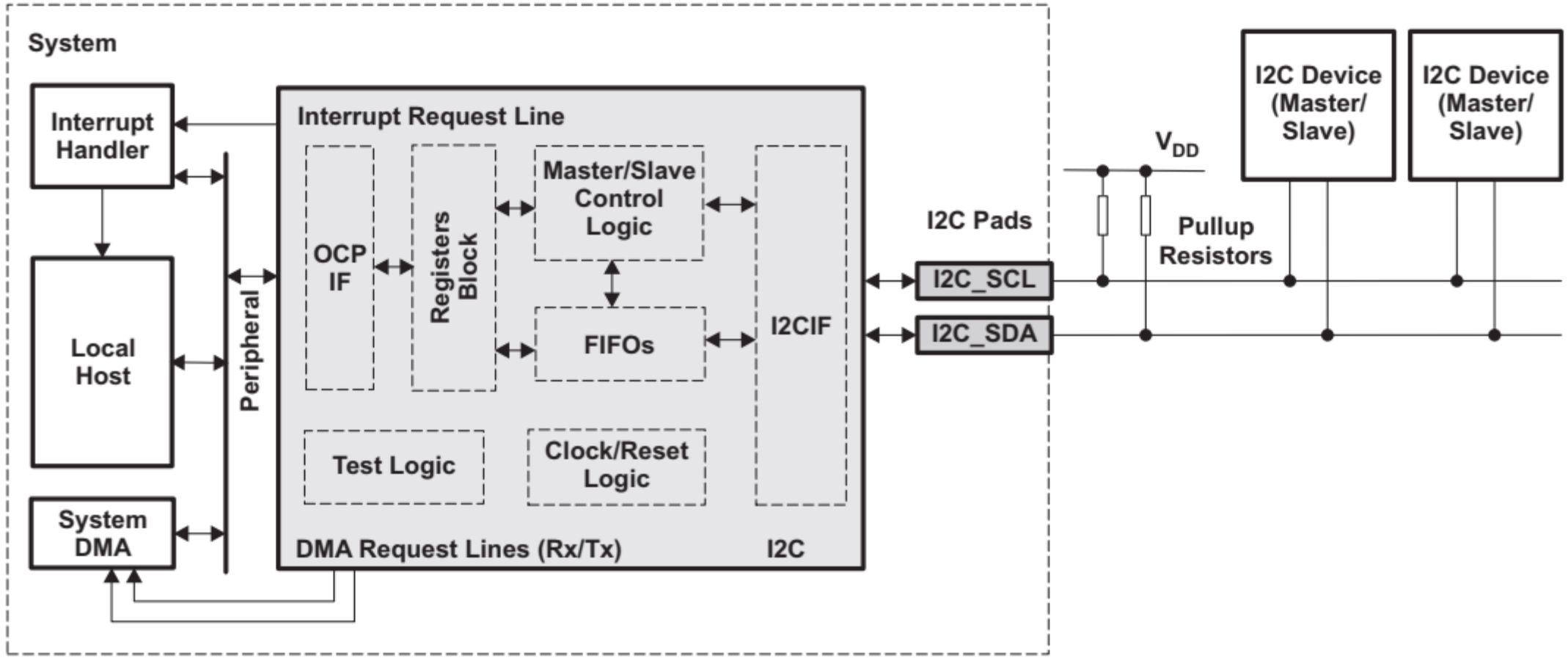
AM335X
I2C Controller

AM335X I2C Controller

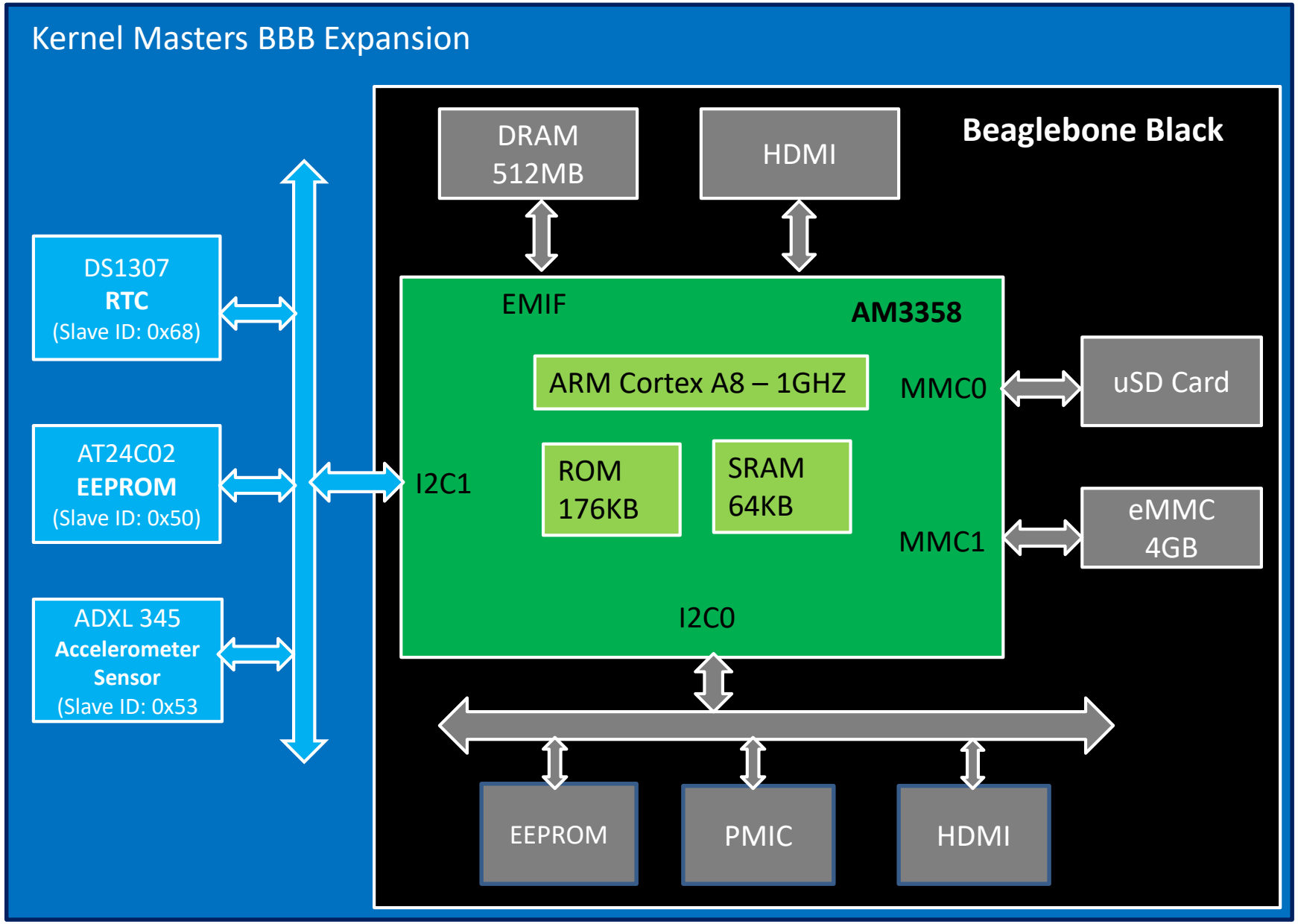
Specifications:

- The general features of the I2C controller are:
- Compliant with Philips I2C specification version 2.1.
- Supports standard mode (up to 100K bits/s) and fast mode (up to 400K bits/s).
- Multimaster transmitter/slave receiver mode
- Multimaster receiver/slave transmitter mode
- Combined master transmit/receive and receive/transmit modes
- 7-bit and 10-bit device addressing modes
- Built-in 32-byte FIFO for buffered read or writes in each module
- Programmable clock generation
- Two DMA channels, one interrupt line

AM335X I2C Controller Functional Block Diagram



I2C Slave Devices interface with AM335x I2C Masters



KERNEL MASTERS

AM335x I2C Interrupts

Bus Free interrupt (BF) is generated to inform the Local Host that the I2C bus became free (when a Stop Condition is detected on the bus) and the module can initiate his own I2C transaction.

Start Condition interrupt (STC) is generated after the module being in idle mode have detected (synchronously or asynchronously) a possible Start Condition on the bus (signalized with WakeUp).

Arbitration lost interrupt (AL) is generated when the I2C arbitration procedure is lost.
No-acknowledge interrupt (NACK) is generated when the master I2C does not receive acknowledge from the receiver.

Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed address, data, and command have been performed and the status bits have been updated.
This interrupt is used to let the CPU know that the I2C registers are ready for access.

AM335x I2C Interrupts

Receive interrupt/status (RRDY) is generated when there is received data ready to be read by the CPU from the I2C_DATA register (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to read the received data from the I2C_DATA register.

Transmit interrupt/status (XRDY) is generated when the CPU needs to put more data in the I2C_DATA register after the transmitted data has been shifted out on the SDA pin (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to write the next transmitted data into the I2C_DATA register.

Receive draining interrupt (RDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be transferred and to enable the draining mechanism.

Transmit draining interrupt (XDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be written and to enable the draining mechanism.

How to Program I2C

- Module Configuration Before Enabling the Module
- Initialization Procedure
- Configure Slave Address and DATA Counter Registers
- Initiate a Transfer
- Receive Data
- Transmit Data

I2C Slave Device

DS1307 RTC

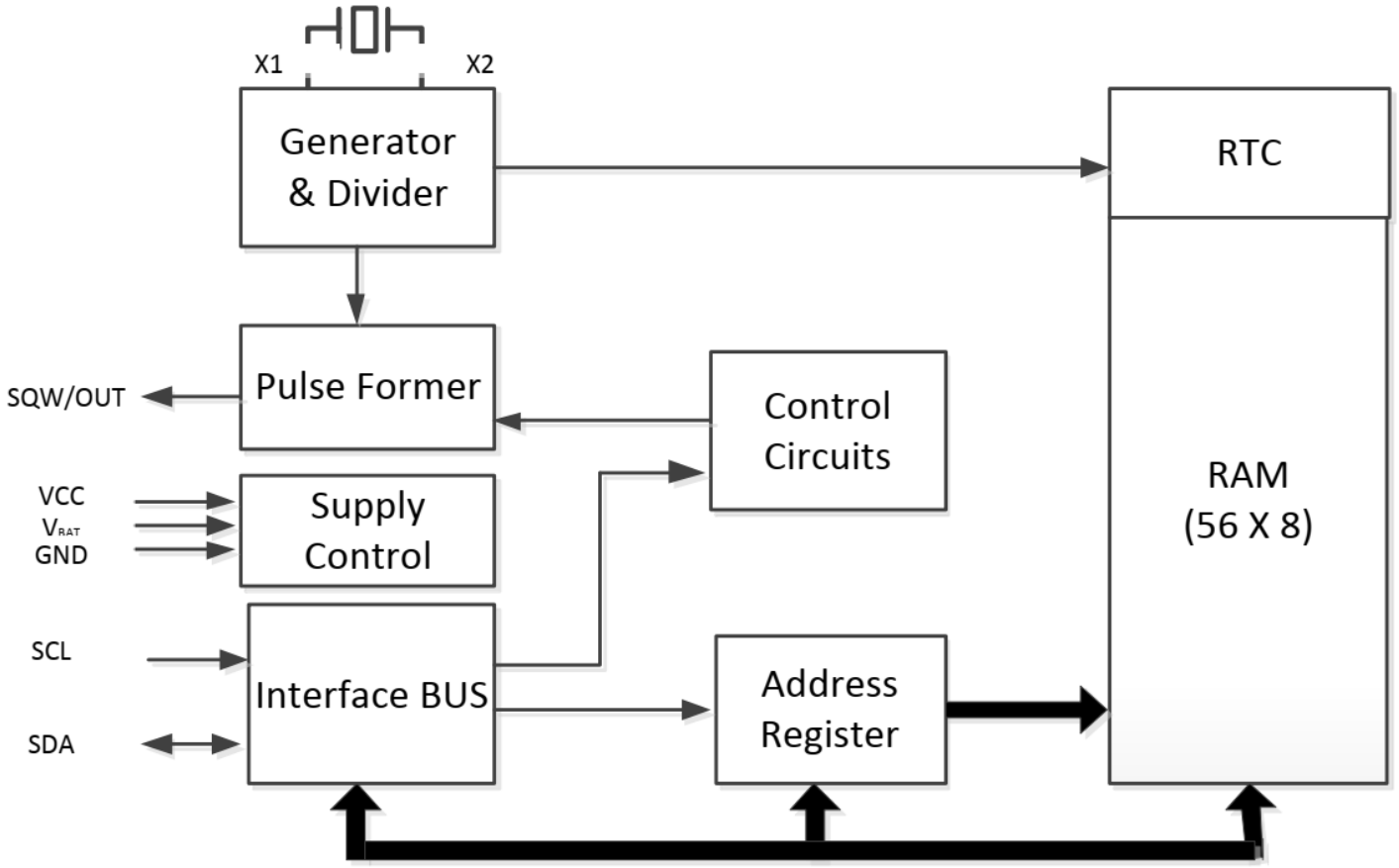
DS1307

DS1307 Specifications
DS1307 Functional Block Diagram
DS1307 Register Programming Model
DS1307 Communication Protocol

DS1307 Features/Specifications

- Count of seconds, minutes, hours, week days, date, months and years with consideration of the leap years (before 2100)
- 56 bytes of the power self-sufficient RAM for the data storage;
- Two-wire consecutive interface;
- Programmable rectangular output signal;
- Automatic determination of the supply voltage drop and the switching diagram;
- Consumption of less than 500 nA in the back-up supply mode with the operating generator;
- Temperature range of the industrial application: -40degreeCent to –+85degreecent
- Accuracy is better than ±1 minute per month

RTC Functional Block Diagram



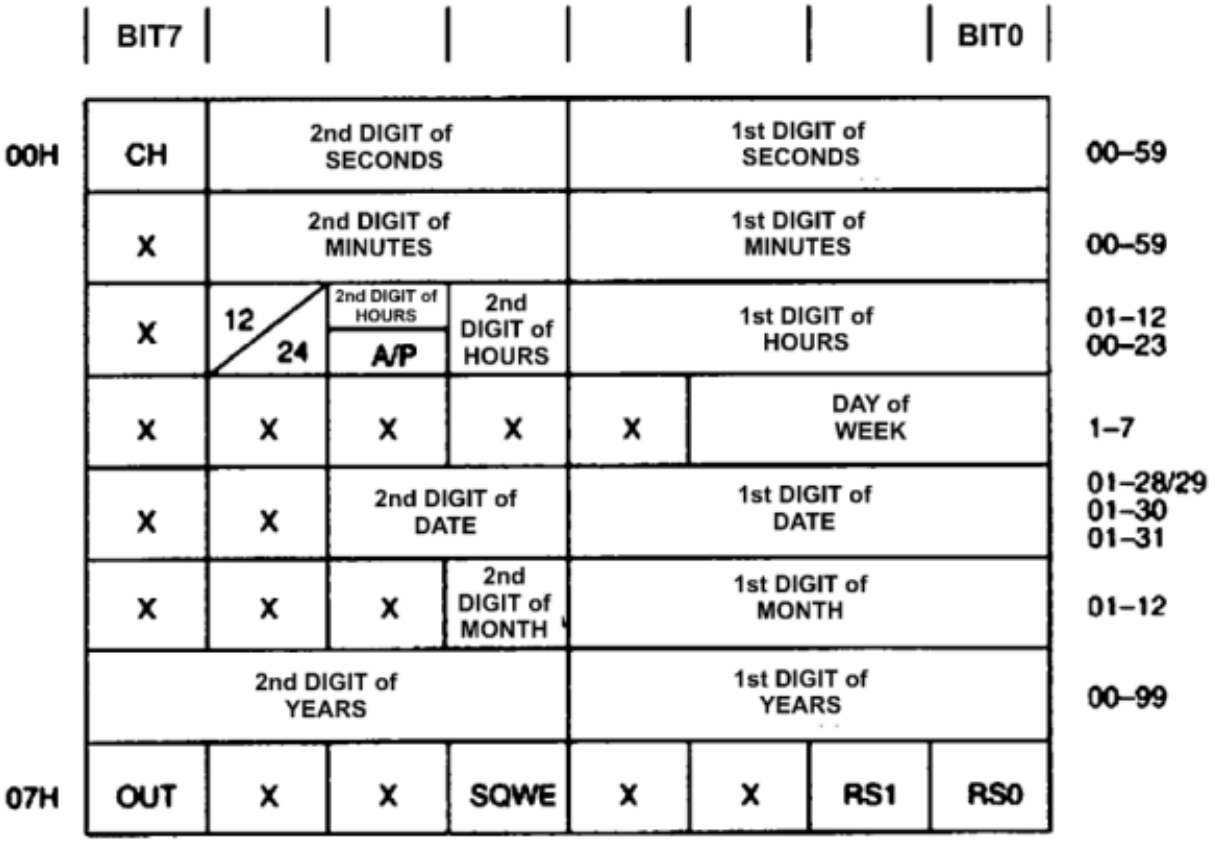
RTC Pin Description

PIN DISCRIPTION

Pin	Symbol	I/O	Pin Description
1	X1	In	Pin for connection of the quartz resonator
2	X2	In	Pin for connection of the quartz resonator
3	VBAT	In	Pin for battery
4	GND	In	Ground pin
5	SDA	Bi	Input / output of serial data
6	SCL	In	Input of the consecutive cycle signal
7	SQW/OUT	Out	Output of rectangular signal
8	VCC	In	Power supply pin

RTC Register Programming Model

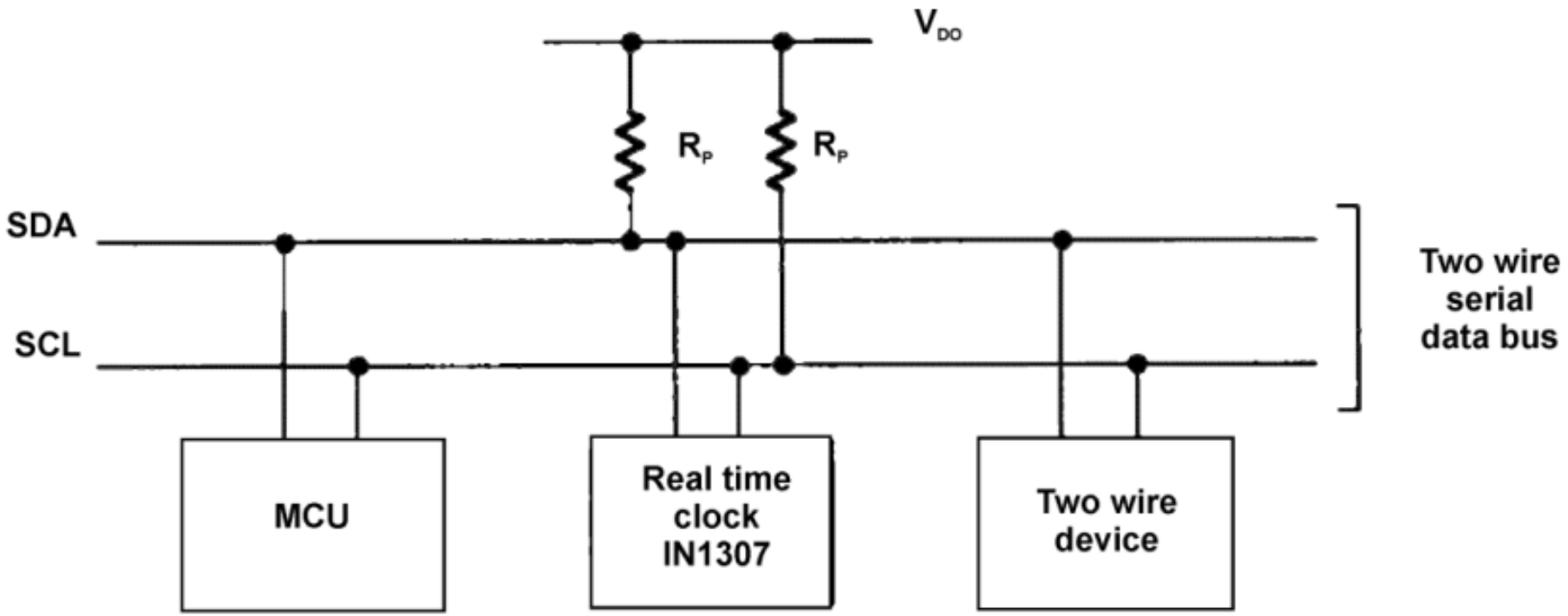
REGISTERS RTC IN1307



	BIT7							BIT0	
00H	CH	2nd DIGIT of SECONDS			1st DIGIT of SECONDS				00–59
	X	2nd DIGIT of MINUTES			1st DIGIT of MINUTES				00–59
	X	12 / 24	2nd DIGIT of HOURS / A/P	2nd DIGIT of HOURS	1st DIGIT of HOURS				01–12 / 00–23
	X	X	X	X	X	DAY of WEEK			1–7
	X	X	2nd DIGIT of DATE		1st DIGIT of DATE				01–28/29 / 01–30 / 01–31
	X	X	X	2nd DIGIT of MONTH	1st DIGIT of MONTH				01–12
	2nd DIGIT of YEARS				1st DIGIT of YEARS				00–99
07H	OUT	X	X	SQWE	X	X	RS1	RS0	

RTC Interface

- I2C Comm. Protocol



RTC Write communication protocol

S - START

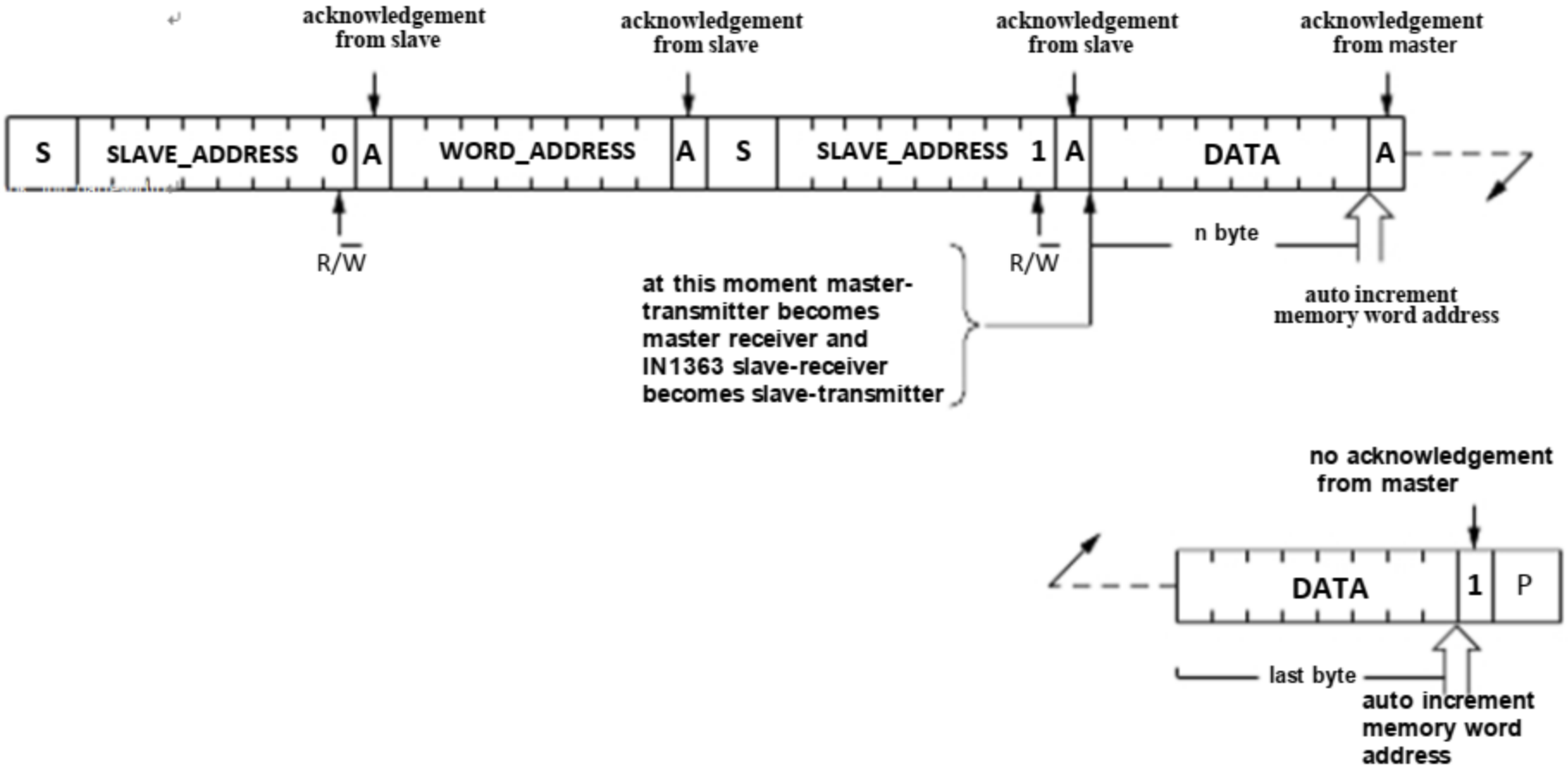
A - ACKNOWLEDGE

P - STOP

DATA TRANSFERRED (X+1 BYTES + ACKNOWLEDGE)

*R/$\overline{W}$ - READ/WRITE OR DIRECTION BIT ADDRESS = D0h

RTC Read communication protocol

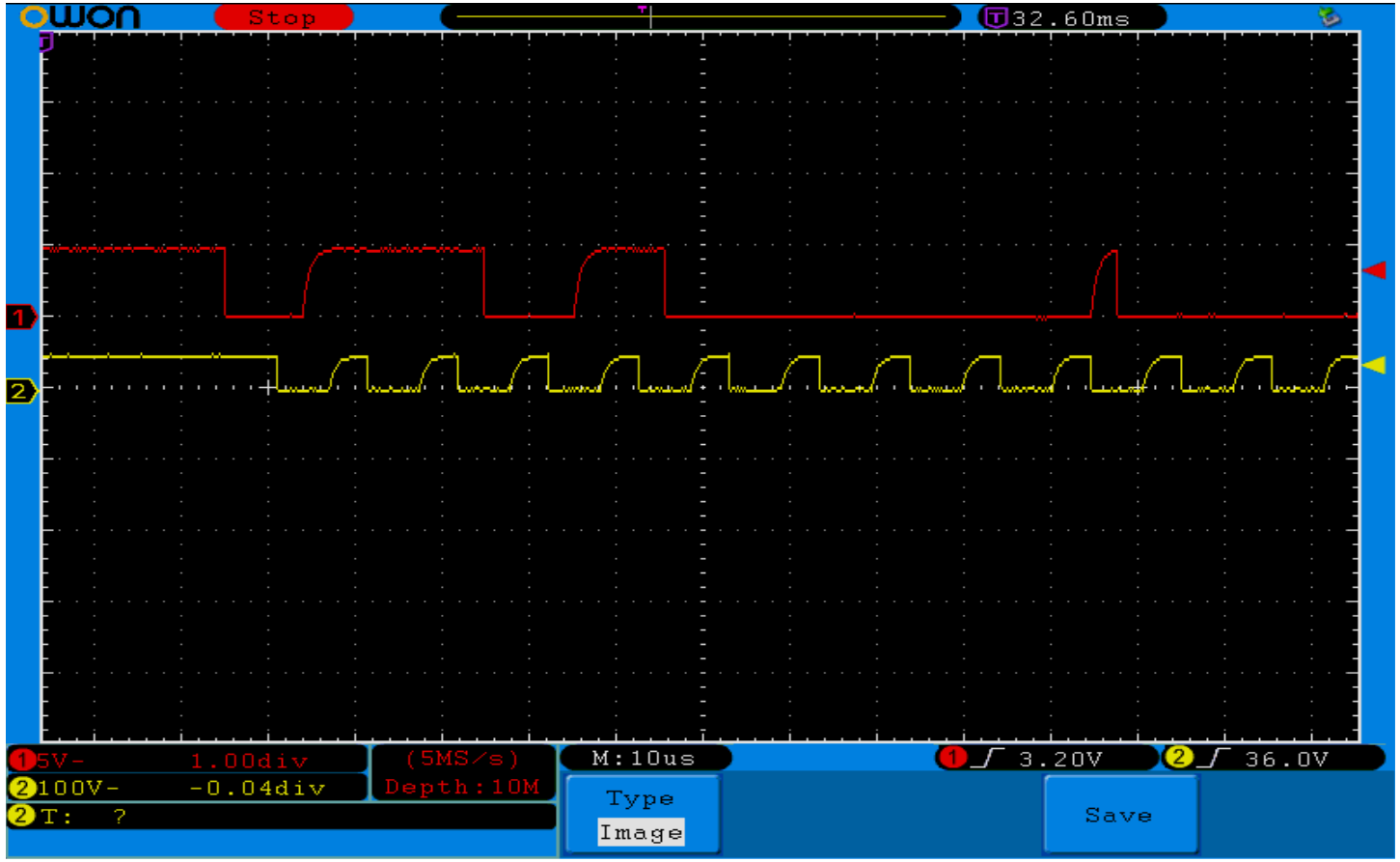


Master reads after setting word address (write word address ; read data)

I2C Waveform

I2C_Send2(0x68,0x00,15);

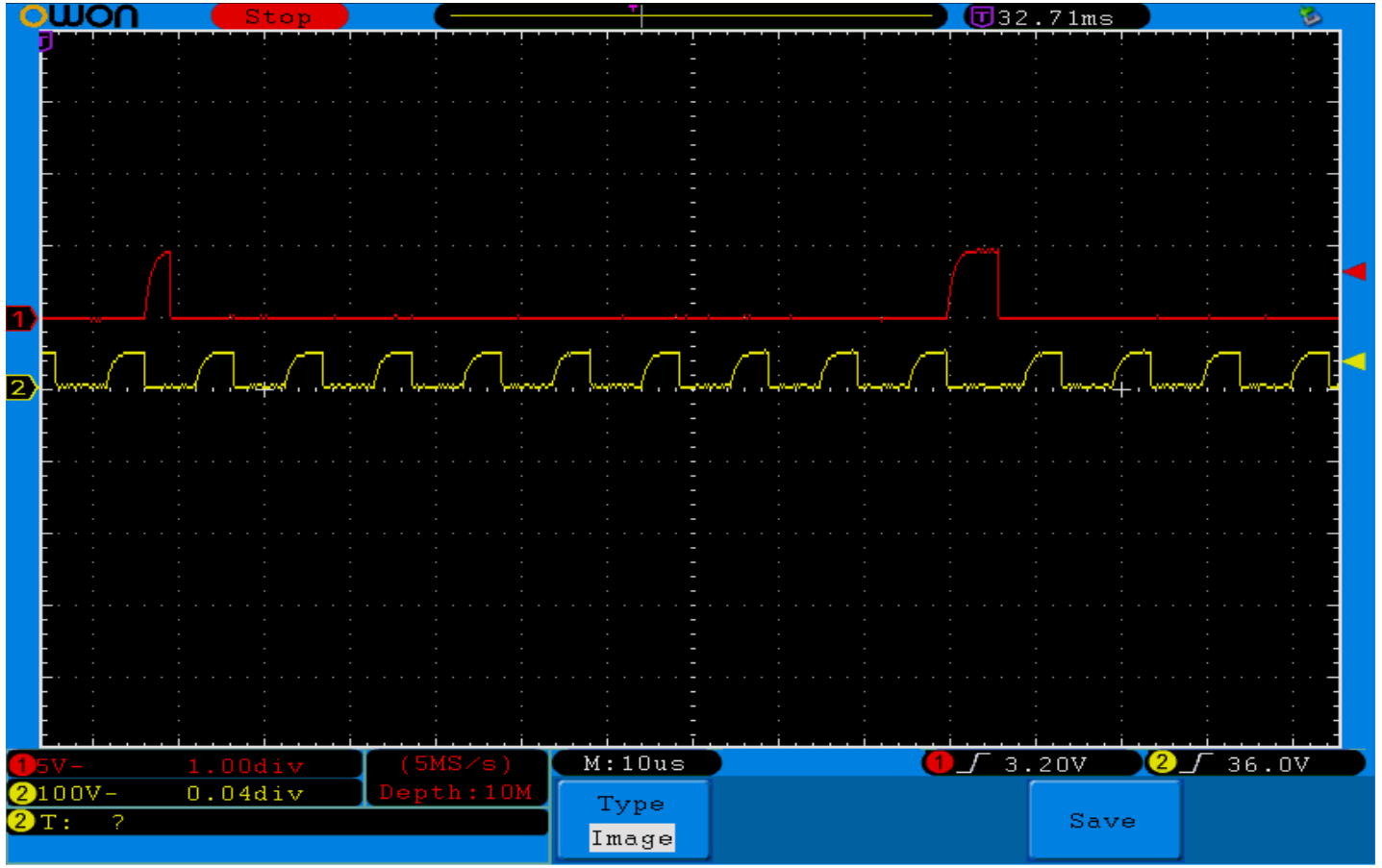
<S><Slave Address><W><ACK> <Sec Reg Address><ACK> <Data><ACK>



I2C Waveform

I2C_Send2(0x68,0x00,15);

<S><Slave Address><W><ACK> **<Sec Reg Address><ACK>** <Data><ACK>



I2C Waveform

I2C_Send2(0x68,0x00,15);
<S><Slave Address><W><ACK> <Sec Reg Address><ACK> **<Data><ACK>**