Advanced Micro-controllers

Agenda

- Q & A
- ARM Assembly Language
  - Functions -- APCS
  - Advanced Instructions
- I2C Protocol

Q & A

- Stack types

QEmu

- https://www.qemu.org/docs/master/system/arm/stm32.html

ARM Assembly Programming

Functions

- Function arguments are passed via registers: r0, r1, r2, r3
- Function return value: r0
- Called function should save r4-r11 before use and restore them after use.
- Calling function should save r0-r3,r12 before call and return them after call.
- Example1:

```
@ calling function
main:
    push {lr}
    mov r0, #11     @ copy arguments in r0, r1
    mov r1, #22     @ copy arguments in r0, r1
    bl add2         @ keep addr of next instruction in LR and jump to add2.
    mov r7, r0      @ copy result into r7
    pop {lr}
    mov pc, lr

@ called function
add2:
    add r0, r1      @ r0 = r0 + r1
    mov pc, lr      @ copy ret addr in lr to pc (return from function)
```

- Example2:

```
@ calling function
main:
    push {lr}
    mov r0, #11     @ copy arguments in r0
    mov r1, #22     @ copy arguments in r1
    mov r2, #33     @ copy arguments in r2
    bl add3         @ keep addr of next instruction in LR and jump to add2.
    mov r7, r0      @ copy result into r7
    pop {lr}
    mov pc, lr

@ called/calling function
add3:
    push {r7, lr}
    push {r2}
    bl add2         @ r0 = add2(r0,r1)
    pop {r2}
    mov r1, r2      @ arg1 in r0 and copy arg2 in r1
    bl add2         @ r7 = add2(r0,r1)
    mov r7, r0
    pop {r7, lr}
    mov pc, lr      @ return from function

@ called function
add2:
    add r0, r1      @ r0 = r0 + r1
    mov pc, lr      @ copy ret addr in lr to pc (return from function)
```

Serial Communication vs Parallel Communication

- Serial: One bit is transferred at a time.
- Parallel: Multiple bits transferred at a time.
- Initially Parallel Communication was faster than Serial Communication.
- Parallel Communication needs more wires and hence bigger circuit.
- Several Serial Communication protocols:
    - RS232
    - I2C
    - SPI
    - CAN
    - USB
    - PS2
    - JTAG
- Few serial protocols are
    - For short distance communication: I2C, SPI.
    - For long distance communication: RS232.
    - For noisy environment: CAN.
- Few serial protocols are
    - Simplex:
    - Half duplex: I2C, CAN
    - Full duplex: RS232, SPI
- Few serial protocols are
    - Peer to Peer: RS232, PS2.

- Bus: SPI, I2C, CAN, USB.

I2C Protcol

Physical characteristics

- Type
  - Bus protocol (Master Slave)
  - Half duplex
- Wires/Connectivity
  - 2 wire protocol
    - SDA -- Serial Data
    - SCL -- Serial Clock
- Voltage levels
  - TTL voltage
- Frequency
  - 100 KHz -- Standard Mode
  - 400 KHz -- Fast Mode
  - 1 MHz -- Fast plus Mode

Logical characteristics

- I2C Device Modes
  - Master Transmitter
  - Master Receiver
  - Slave Transmitter
  - Slave Receiver
- Data Bit transfer
- Data frame
  - 8 Data Bits + ACK
  - Transmitter ---> 8 Data Bits + (ACK=1) ---> Receiver
  - Transmitter <--- Acknowledge (ACK=0) <--- Receiver

- Address frame
  - 7 I2C-Addr Bits + Read/Write' + ACK
  - Master ---> 7 I2C-Addr Bits + Read/Write' + (ACK=1) ---> Slave
  - Master <--- Acknowledge + (ACK=0) <--- Slave

Single byte data write

- START (M) + I2CSlaveAddr_WR (MT) + ACK (SR) + InternalAddress (MT) + ACK (SR) + DataByte (MT) + ACK (SR) + STOP (M)

Multi byte data write

- START (M) + I2CSlaveAddr_WR (MT) + ACK (SR) + InternalAddress (MT) + ACK (SR) + DataByte1 (MT) + ACK (SR) + ... + STOP (M)

Single byte read write

- START (M) + I2CSlaveAddr_WR (MT) + ACK (SR) + InternalAddress (MT) + ACK (SR) + REPEAT START (M) + I2CSlaveAddr_RD (MT) + ACK (SR) + DataByte1 (ST) + ACK (MR) + STOP (M)

Multi byte data read

- START (M) + I2CSlaveAddr_WR (MT) + ACK (SR) + InternalAddress (MT) + ACK (SR) + REPEAT START (M) + I2CSlaveAddr_RD (MT) + ACK (SR) + DataByte1 (ST) + ACK (MR) + ... + STOP (M)

Clock stretching

- If slave is busy in processing the received data, it pulls clock line (SCL) low. So effective line voltage is low and clock is stretched (disabled).
- Due to clock disabled, no I2C data is transferred.

Arbitration

- If two devices send start signal simultaneously (to become master), whichever device send low (0) bit first, wins the arbitration.
- The winning device contiues master operation, while loosing device fallback to slave mode.

I2C Device - State machine

- I2C device as a state machine.
  - Each I2C operation has standard success/failure codes.
  - Status code is of 5 bits.

DS1307 (RTC)

- Slave address: 11010000 (D0) - wr, 11010001 (D1) - rd
- Internal address: 1 byte -- RTC registers (SEC, MIN, HOUR, ...), Memory registers (56 bytes)

Device Modes

- Master Transmitter
- Master Receiver
- Slave Transmitter
- Slave Receiver

Error conditions

- Bus error -- When START or STOP signal is received while communication is going on (Bus is busy). In master mode, it will keep owning the bus (will not release bus lines).
- Ack error -- After transmitting data/address, transmitter keeps data line high. The receiver after getting the data, pulls line low. This is ack to the transmitter. If no receiver devices, pulls line low will cause Ack error -- indicate that data is not received.
- Arbitration lost error
- Read overrun -- The next received data overwrites earlier received data before reading it into some regr.
- STM32 -- CRC Error -- CRC of transmitter and receiver mismatch.