Timers

Material taken from Dreslinski, Dutta, Le, Ramadas, Smith, Tikhonov & Mahal
Agenda

- A bit on timers

- Project overview

- Serial Buses (we won’t finish this today, bring slides on Wednesday)
  - Introduction
  - UART
  - SPI
  - I2C
Timer

Program
//Setup Timer

Hardware Timer
Count: -1
Interrupt

ISR
//Do Something
Virtual Timer

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Virtual Timer Code
//Update Event Queue
//Adjust Timer if needed

Virtual Event Queue

Hardware Timer
Count: -1
Interrupt

Virtual ISR
//Figure out source
bl bl bl
//Insert new event?
//Set new time

Program 1-Handler
//Do Something

Program 2-Handler
//Do Something

Program 3-Handler
//Do Something

Program 4-Handler
//Do Something
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

Null

Hardware Timer
Count: -1
Event Queue

Program 1 //Setup Timer @3
//Setup Timer @5

Program 2 //Setup Timer @D2

Program 3 //Setup Timer @D7

Program 4 //Setup Timer @4

Head Ptr

Time: 3
Delta: -1
Mode: One

Hardware Timer
Count: 3
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

- Time: 3
- Delta: -1
- Mode: One

Hardware Timer

- Count: 3
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Hardware Timer
Count: 2
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Hardware Timer
Count: 2
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Event Queue

Head Ptr

Time: 2
Delta: 2
Mode: Delta

Time: 3
Delta: -1
Mode: One

Time: 5
Delta: -1
Mode: One

Hardware Timer
Count: 2

Time: 7
Delta: 7
Mode: Delta
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr
Time: 2
Delta: 2
Mode: Delta

Hardware Timer
Count: 2

Time: 3
Delta: -1
Mode: One

Time: 4
Delta: -1
Mode: One

Time: 5
Delta: -1
Mode: One

Time: 7
Delta: 7
Mode: Delta
**Event Queue**

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

![Diagram of Event Queue with pointers and timers](image-url)
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

Hardware Timer
Count: 0

Virtual ISR
//Figure out source
//Remove head
//Insert new event?
//Set new time

Time: 2
Delta: 2
Mode: Delta

Time: 1
Delta: -1
Mode: One

Time: 2
Delta: -1
Mode: One

Time: 3
Delta: -1
Mode: One

Time: 5
Delta: 7
Mode: Delta
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

Time: 1
Delta: -1
Mode: One

Hardware Timer
Count: 0

Virtual ISR
//Figure out source
//Remove head
//Insert new event?
//Set new time

Time: 2
Delta: -1
Mode: One

Time: 2
Delta: 2
Mode: Delta

Time: 3
Delta: -1
Mode: One

Time: 5
Delta: 7
Mode: Delta

Time: 5
Delta: 7
Mode: Delta

Time: 3
Delta: -1
Mode: One

Time: 2
Delta: 2
Mode: Delta

Time: 2
Delta: -1
Mode: One

Time: 1
Delta: -1
Mode: One

Time: 2
Delta: 2
Mode: Delta
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Event Queue

Head Ptr
- Time: 1
- Delta: -1
- Mode: One

Hardware Timer
- Count: 1

Virtual ISR
- Figure out source
- Remove head
- Insert new event?
- Set new time

Hardware Timer
- Time: 5
- Delta: -1
- Mode: One

Hardware Timer
- Time: 7
- Mode: Delta

Virtual ISR
- Time: 3
- Delta: -1
- Mode: One

Virtual ISR
- Time: 2
- Delta: 2
- Mode: Delta
Event Queue - Caveats

No new event added for one-shot

Handler may need to loop to handle multiple events at same time
Caveats

• Previous slides assumed scheduling of events all when timer was first set.

• What if we need to schedule an event and we have already expired some of the timer?
  - Need to update the entire virtual time queue before inserting new event.
Agenda

• Serial Buses
  - Introduction
  - UART
  - SPI
  - I2C

• Glitches
  - Asynchronous resets and glitches
  - Design rules

• Set-up and hold time.
  - Review
  - Dealing with external inputs
    • Design rules
Serial interfaces

EECS 370

Top: traps & exceptions

SVC#

Internal Interrupts

GPIO/INT

Timers

Serial Bus

I2C

SPI

UART

Interrupts

DAC/ADC

System Buses

AHB/APB

Assembly Machine Code

C

Internal & External Memory

Fault

INT#

SFI

Str (write)

Internal & External

I/O

Input Output

Compare Capture

ADC DAC

EMC
External memory attaches to the processor via the external memory controller and bus.
• **Universal Asynchronous Receiver/Transmitter**
  - a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms.
  - UARTs are commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485.
  - The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) typically are handled by a special driver circuit external to the UART.

Most of the UART stuff (including images) Taken from Wikipedia!
Fun with buses

• A multidrop bus (MDB) is a computer bus in which all components are connected to the same set of electrical wires. (from Wikipedia)
  - In the general case, a bus may have more than one device capable of driving it.
  • That is, it may be a “multi-master” bus as discussed earlier.
How can we handle multiple (potential) bus drivers? (1/3)

• Tri-state devices, just have one device drive at a time. Everyone can read though
  - Pros:
    • Very common, fairly fast, pin-efficient.
  - Cons:
    • Tri-state devices can be slow.
      - Especially drive-to-tristate?
    • Need to be sure two folks not driving at the same time
      - Let out the magic smoke.
  - Most common solution (at least historically)
    • Ethernet, PCI, etc.
How can we handle multiple (potential) bus drivers? (2/3)

- **MUX**
  - Just have each device generate its data, and have a MUX select.
  - That’s a LOT of pins.
    - Consider a 32-bit bus with 6 potential drivers.
      » Draw the figure.
      » How many pins needed for the MUX?
  - Not generally realistic for an “on-PCB” design as we’ll need an extra device (or a lot of pins on one device)
    - But reasonable on-chip
      - In fact AHB, APB do this.
How can we handle multiple (potential) bus drivers? (3/3)

- “pull-up” aka “open collector” aka “wired OR”
  - Wire is pulled high by a resistor
  - If any device pulls the wire low, it goes low.

- Pros:
  - If two devices both drive the bus, it still works!

- Cons:
  - Rise-time is very slow.
  - Constant power drain.

- Used in I2C, CAN
Agenda

- Serial Buses
  - Introduction
  - UART
  - SPI
  - I2C
• Universal Asynchronous Receiver/Transmitter
• Hardware that translates between parallel and serial forms
• Commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485
• The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) typically are handled by a special driver circuit external to the UART.

Most of the UART stuff (including images) Taken from Wikipedia!
Protocol

• Each character is sent as
  - a logic low start bit
  - a configurable number of data bits (usually 7 or 8, sometimes 5)
  - an optional parity bit
  - one or more logic high stop bits.
  - with a particular bit timing ("baud" or "baudrate")

• Examples
  - “9600-N-8-1” → <baudrate><parity><databits><stopbits>
  - “9600-8-N-1” → <baudrate><databits><parity><stopbits>
Variations and fun times

• UART is actually a generic term that includes a large number of different devices/standards.
  - RS-232 is a standard that specifies
    • “electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin out of connectors.”
Signals (only most common)

- The **RXD** signal of a UART is the signal receiving the data. This will be an input and is usually connected to the TXD line of the downstream device.

- The **TXD** signal of a UART is the signal transmitting the data. This will be an output and is usually connected to the RXD line of the downstream device.

- The **RTS#** (Ready to Send) signal of a UART is used to indicate to the downstream device that the device is ready to receive data. This will be an output and is usually connected to the CTS# line of the downstream device.

- The **CTS#** (Clear to Send) signal of a UART is used by the downstream device to identify that it is OK to transmit data to the upstream device. This will be an input and is usually connected to the RTS# line of the upstream device.
Wiring a DTE device to a DCE device for communication is easy. The pins are a one-to-one connection, meaning all wires go from pin x to pin x. A straight through cable is commonly used for this application. In contrast, wiring two DTE devices together requires crossing the transmit and receive wires. This cable is known as a null modem or crossover cable.
RS-232 transmission example

RS232 Transmission of the letter 'J'

Signal levels at the UART output pin

Signal levels at the Transceiver output pin

Meaning: IDLE, START, B0, B1, B2, B3, B4, B5, B6, B7, STOP, IDLE
Agenda

- Serial Buses
  - Introduction
  - UART
  - SPI
  - I2C
Introduction

- What is it?
- Basic Serial Peripheral Interface (SPI)
- Capabilities
- Protocol
- Pro / Cons and Competitor
- Uses
- Conclusion
What is SPI?

- Serial Bus protocol
- Fast, Easy to use, Simple
- Everyone supports it
SPI Basics

- A communication protocol using 4 wires
  - Also known as a 4 wire bus
- Used to communicate across small distances
- Multiple Slaves, Single Master
- Synchronized
Capabilities of SPI

- Always Full Duplex
  - Communicating in two directions at the same time
  - Transmission need not be meaningful
- Multiple Mbps transmission speed
- Transfers data in 4 to 16 bit characters
- Multiple slaves
  - Daisy-chaining possible
Protocol

- Wires:
  - Master Out Slave In (MOSI)
  - Master In Slave Out (MISO)
  - System Clock (SCLK)
  - Slave Select 1...N
- Master Set Slave Select low
- Master Generates Clock
- Shift registers shift in and out data
Wires in Detail

- **MOSI** – Carries data out of Master to Slave
- **MISO** – Carries data from Slave to Master
  - Both signals happen for every transmission
- **SS_BAR** – Unique line to select a slave
- **SCLK** – Master produced clock to synchronize data transfer
Shifting Protocol

Master shifts out data to Slave, and shift in data from Slave

http://upload.wikimedia.org/wikipedia/commons/thumb/b/bb/SPI_8-bit_circular_transfer.svg/400px-SPI_8-bit_circular_transfer.svg.png
Master and multiple independent slaves


Some wires have been renamed

Master and multiple daisy-chained slaves

http://www.maxim-ic.com/appnotes.cfm/an_pk/3947
Pros and Cons

Pros:
- Fast and easy
  - Fast for point-to-point connections
  - Easily allows streaming/Constant data inflow
  - No addressing/Simple to implement
- Everyone supports it

Cons:
- SS makes multiple slaves very complicated
- No acknowledgement ability
- No inherent arbitration
- No flow control
Some Serial Encoders/Decoders, Converters, Serial LCDs, Sensors, etc.

Pre-SPI serial devices
Summary

- SPI – 4 wire serial bus protocol
  - MOSI MISO SS SCLK wires
- Full duplex
- Multiple slaves, One master
- Best for point-to-point streaming data
- Easily Supported
Agenda

- Serial Buses
  - Introduction
  - UART
  - SPI
  - I2C
- Glitches
  - Asynchronous resets and glitches
  - Design rules
- Set-up and hold time.
  - Review
  - Dealing with external inputs
    - Design rules
What is I²C?

- Inter-Integrated Circuit
- Pronounced “eye-squared-see”
- Two-wire serial bus protocol
- Invented by Philips in the early 1980’s
  - That division now spun-off into NXP
Where is it used?

- Originally used by Philips inside television sets
- Now very common in peripheral devices intended for embedded systems use
  - Philips, National Semiconductor, Xicor, and Siemens, ...
- Also used in the PC world
  - Real time clock
  - Temperature sensors
Technical Description

- Two-wire serial protocol with addressing capability
- Speeds up to 3.4 Mbit/s
- Multi-master/Multi-slave
Wiring

- Two lines
  - SDA (data)
  - SCL (clock)
- Open-collector
  - Very simple interfacing between different voltage levels
Clock

- Not a traditional clock
- Normally high (kept high by the pull-up)
- Pulsed by the master during data transmission (whether the master is transmitter or receiver)
- Slave device can hold clock low if it needs more time
A Basic I²C Transaction

- Master always initiates transactions
- Start Condition
- Address
- Data
- Acknowledgements
- Stop Condition

Source: ATmega8 Handbook
A Basic I²C Transaction

- Transmitter/Receiver differs from Master/Slave
- Master initiates transactions, slave responds
- Transmitter sets data on the SDA line, Receiver acknowledges
  - For a read, slave is transmitter
  - For a write, master is transmitter
Start Condition

- Master pulls SDA low while SCL is high
- Normal SDA changes only happen while SCL is low
Address Transmission

- Data is always sampled on rising edge of clock
- Address is 7 bits
- An 8th bit indicates read or write
  - High for read, low for write
- Addresses assigned by Philips/NXP (for a fee)
Data Transmission

- Transmitted just like address (8 bits)
- For a write, master transmits, slave acknowledges
- For a read, slave transmits, master acknowledges
- Transmission continues with subsequent bytes until master creates stop condition

Source: ATMega8 Handbook
Stop Condition

- Master pulls SDA high while SCL is high
- Also used to abort transactions
Another Look

Source: ATMega8 Handbook