EECE 218
Microcontrollers

The HCS12 Architecture
The 9S12DP256
The 9S12DP256

- **Block diagram**
- **CPU12:** the processing unit
- **Main operating modes:**
  - Single chip – Everything on chip, incl. memory
  - Extended – Memory, devices connected to bus
    - Time-Multiplexed Address/Data bus
- **Memory:** *Layout*
  - 256kbyte Flash EEPROM (‘paged’)
  - 12kbyte RAM
  - 4kbyte EEPROM
  - If external: address/data bus
The 9S12DP256

Highlights of peripheral interfaces:

- Parallel ports: A, B, E, H, J, K, M, P, S, T
  » 8 bit bytes are transferred (read/written) in one step
- Timers – in Enhanced Capture Timer (ECT)
  » HW units for generating pulses/measuring pulses
- Serial interfaces:
  » Data (bytes) are transferred bit-wise
  » Serial Communication Interface (SCI) (2)
  » Serial Peripheral Interface (SPI) (2)
  » CAN, IIC, BDLC: Specialized serial interface
- Analog to Digital converters:
  » Convert analog voltage into digital value
The 9S12DP256

Other elements:

- PLL: master oscillator and clock generator
  » For lab: 24MHz
- Background Debug Module (BDM): hardware support for debugging
- System integration module
  » Support for ‘extended’ configuration: external memory, peripheral interface devices, etc.
Generic I/O Principles

Major methods for digital interfacing and communications:

- Parallel: send/receive one word (byte) in one step (multiple wires)

- Serial: send/receive one bit in one step (~1 wire)
Parallel vs. serial

<table>
<thead>
<tr>
<th></th>
<th>Parallel</th>
<th>Serial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cable</td>
<td>Complex</td>
<td>Simple</td>
</tr>
<tr>
<td>Speed</td>
<td>Faster</td>
<td>Slower</td>
</tr>
<tr>
<td>Interface</td>
<td>Simple</td>
<td>Complex</td>
</tr>
</tbody>
</table>

Parallel interface used for:
- disks, graphics devices, switches, LEDs, etc.

Serial interface used for:
- serial communication devices, LAN, WLAN, USB, Firewire, disks, etc.
I/O on 9S12DP256

- Principle: Memory-mapped I/O
  - Each peripheral device has register/s that are accessed at designated physical memory addresses. (See: ‘hcs12.inc’)
    - Example: PORTA is at address 0
  - Read and write happens HCS12 instructions
    - \texttt{STAA PORTA}; Write A into PORTA
  - Reading and writing at the same address may deliver different results! (See details later)
  - Sometime device registers are 2bytes that need to be written/read in one instruction (e.g. STD/LDD)
Parallel ports on the 9S12D

- Parallel ports: A, B, E, H, J, K, M, P, S, T
  - Bits are transferred (read/written) in one step
  - Not all ports have all 8 bits!
  - Port bits are bi-directional:
    - Can be used for input OR output (but not the same time!)
    - Direction is determined by specific bit/s in the corresponding port Data Direction Register (0=in, 1=out)
  - Example:
    ; Use PORTB bits 0-3 for input, 4-7 for output
    LDAA #$F0
    STAA DDRB
    ; Read from PORTB:
    LDAA PORTB
    ; Write to PORTB
    STAA PORTB
Parallel ports on the 9S12D

- Parallel port behavior:

<table>
<thead>
<tr>
<th>DDR Setting</th>
<th>Read on bit</th>
<th>Write on bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Read from pin (momentary)</td>
<td>No effect</td>
</tr>
<tr>
<td>1</td>
<td>Undefined</td>
<td>Write to pin (latch)</td>
</tr>
</tbody>
</table>

- In summary:
  » Port DDR bits should be initialized (0=in, 1=out)
  » Reading/writing = simple load/store
Handshaking: A Parallel Port Technique

- NOT on 9S12 (but on many other systems)
- Simple (non-handshake) output port:

![Diagram of output port with 8 bit data and output device](image)

**Problems:**

1. Output device does NOT know when a new byte is ‘stable’.
2. Port does NOT know when the output device is ready to receive
Handshaking: A Parallel Port Technique

● Simple (non-handshake) input port:

Problems:
(1) Input device does NOT know whether port is able to receive new byte.
(2) Port does NOT know when the input device sends new data.
Handshaking: A Parallel Port Technique

- Solution: two extra lines to indicate status

<table>
<thead>
<tr>
<th></th>
<th>H1</th>
<th>H2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output handshake</strong></td>
<td>Data ready for device</td>
<td>Data has been taken by device</td>
</tr>
<tr>
<td><strong>Input handshake</strong></td>
<td>Port is ready to receive data</td>
<td>Data is ready for port</td>
</tr>
</tbody>
</table>

### Diagram

- **Input Device** (e.g. switches)
  - Port 1
  - Port 2

- **Output Device** (e.g. Printer)
  - Port 1
  - Port 2

- Handshake signals:
  - H1
  - H2

- Data flow:
  - 8 bit data

- Status:
  - Data is ready for port
  - Port is ready to receive data
  - Data has been taken by device
  - Data ready for device
Input handshake protocol

- Step 1. The port asserts (or pulses) H1 to indicate its intention to input data.
- Step 2. The input device puts data on the data port pins and also asserts (or pulses) the handshake signal H2.
- Step 3. The port latches the data and de-asserts H1. After some delay, the input device also de-asserts H2.

Figure 7.3 Input Handshakes
Output handshake protocol

- Step 1. The port places data on the port pins and asserts (or pulses) H1 to indicate that it has valid data to be output.
- Step 2. The output device latches the data and asserts (or pulses) H2 to acknowledge the receipt of data.
- Step 3. The port de-asserts H1 following the assertion of H2. The output device then de-asserts H2.

![Diagram of output handshake protocol]

Figure 7.4 Output Handshaking
Lab setup

- Demo board:
  » LED-s, DIP switches, etc.
  » H1 connector!
  » PORTH drives the 8+2 LED-s
  » 8-DIP switches are connected to PORTT.

- Satellite board --- See class pack!
  » 7-segment display, D/A converter.
  » H2 connector
  » PORTB drives display segments (act L), D/A
  » PORTK0/1 select left/right digits (act H)