## Asynchronism close to hardware

### Interrupts

Interrupt control...
- at the individual device level
- at the system interrupt controller level
- at the operating system level

#### LM12L458 – Data transfer options

<table>
<thead>
<tr>
<th>Mode</th>
<th>Interrupt control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read interrupts status</td>
<td>Interrupt status register, 15 bits</td>
<td>Includes current active interrupts, shows the total number of completed and processed instructions, and is readable through the register main interface (control, byte 13, 7).</td>
</tr>
</tbody>
</table>

#### LM12L458 – Register bank

<table>
<thead>
<tr>
<th>Register</th>
<th>Purpose</th>
<th>Type</th>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Status</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>Priority</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### A/D, D/A & Interfaces: Examples

**Interrupt service routines**
- Does not involve processes
  - All actions except for some diagnostics will be deferred for later.
- Communicating interrupts to processes
  - Synchronization of events or data structure.
- Communicating interrupts to signals
  - Notifies the scheduler that a process is to be activated.

#### Asynchronism close to hardware

Interrupt control...
- at the individual device level
- at the system interrupt controller level
- at the operating system level

**Interrupt service routines**
- Available in real-time operating systems only
- Purpose
  - Allows processes to be interrupted and their execution to be deferred for later.
- Parameters
  - Provides a mechanism for handling different types of interrupts.
- Actions
  - Processes might be interrupted at any time.
- Limitations
  - Cannot operate on the level of threads or tasks!
Interrupts and interrupts close to hardware

Interrupt service routines

Interrupt service routines are still used in some operating systems and languages only (for response to interrupts indirectly (OS) interrupting software or mismatch) with/without system programming language.

Interrupts, especially in real-time operating systems, may be handled by means of special hardware interrupts.

Software interrupts are handled through the system interrupt controller.

Interrupts may be classified based on the level of the system:

1. System level
2. Process level
3. Subroutine level

Interrupts may be classified based on their purpose:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the method of handling:

1. Directly
2. Indirectly

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts

Interrupts may be classified based on the type of interrupting software:

1. Operating system interrupts
2. Library function interrupts
3. User program interrupts

Interrupts may be classified based on the type of interrupting hardware:

1. Peripheral interrupts
2. Device interrupts
3. System interrupts

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting software:

1. Library function interrupts
2. User program interrupts
3. Operating system interrupts

Interrupts may be classified based on the type of interrupting event:

1. User-defined interrupts
2. System-defined interrupts
3. Hardware-defined interrupts

Interrupts may be classified based on the type of interrupting condition:

1. External interrupts
2. Internal interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting resource:

1. Physical interrupts
2. Logical interrupts
3. Software interrupts

Interrupts may be classified based on the type of interrupting operation:

1. Input interrupts
2. Output interrupts
3. Communication interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting context:

1. User context
2. System context
3. Hardware context

Interrupts may be classified based on the type of interrupting task:

1. User task
2. System task
3. Hardware task

Interrupts may be classified based on the type of interrupting environment:

1. User environment
2. System environment
3. Hardware environment

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts
3. User program interrupts

Interrupts may be classified based on the type of interrupting event:

1. User-defined interrupts
2. System-defined interrupts
3. Hardware-defined interrupts

Interrupts may be classified based on the type of interrupting condition:

1. External interrupts
2. Internal interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting resource:

1. Physical interrupts
2. Logical interrupts
3. Software interrupts

Interrupts may be classified based on the type of interrupting operation:

1. Input interrupts
2. Output interrupts
3. Communication interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting context:

1. User context
2. System context
3. Hardware context

Interrupts may be classified based on the type of interrupting task:

1. User task
2. System task
3. Hardware task

Interrupts may be classified based on the type of interrupting environment:

1. User environment
2. System environment
3. Hardware environment

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts
3. User program interrupts

Interrupts may be classified based on the type of interrupting event:

1. User-defined interrupts
2. System-defined interrupts
3. Hardware-defined interrupts

Interrupts may be classified based on the type of interrupting condition:

1. External interrupts
2. Internal interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting resource:

1. Physical interrupts
2. Logical interrupts
3. Software interrupts

Interrupts may be classified based on the type of interrupting operation:

1. Input interrupts
2. Output interrupts
3. Communication interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting context:

1. User context
2. System context
3. Hardware context

Interrupts may be classified based on the type of interrupting task:

1. User task
2. System task
3. Hardware task

Interrupts may be classified based on the type of interrupting environment:

1. User environment
2. System environment
3. Hardware environment

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts
3. User program interrupts

Interrupts may be classified based on the type of interrupting event:

1. User-defined interrupts
2. System-defined interrupts
3. Hardware-defined interrupts

Interrupts may be classified based on the type of interrupting condition:

1. External interrupts
2. Internal interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting resource:

1. Physical interrupts
2. Logical interrupts
3. Software interrupts

Interrupts may be classified based on the type of interrupting operation:

1. Input interrupts
2. Output interrupts
3. Communication interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting context:

1. User context
2. System context
3. Hardware context

Interrupts may be classified based on the type of interrupting task:

1. User task
2. System task
3. Hardware task

Interrupts may be classified based on the type of interrupting environment:

1. User environment
2. System environment
3. Hardware environment

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts
3. User program interrupts

Interrupts may be classified based on the type of interrupting event:

1. User-defined interrupts
2. System-defined interrupts
3. Hardware-defined interrupts

Interrupts may be classified based on the type of interrupting condition:

1. External interrupts
2. Internal interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting resource:

1. Physical interrupts
2. Logical interrupts
3. Software interrupts

Interrupts may be classified based on the type of interrupting operation:

1. Input interrupts
2. Output interrupts
3. Communication interrupts

Interrupts may be classified based on the type of interrupting process:

1. User interrupts
2. System interrupts
3. Hardware interrupts

Interrupts may be classified based on the type of interrupting context:

1. User context
2. System context
3. Hardware context

Interrupts may be classified based on the type of interrupting task:

1. User task
2. System task
3. Hardware task

Interrupts may be classified based on the type of interrupting environment:

1. User environment
2. System environment
3. Hardware environment

Interrupts may be classified based on the type of interrupting device:

1. Input device interrupts
2. Output device interrupts
3. Communication device interrupts

Interrupts may be classified based on the type of interrupting source:

1. Hardware interrupts
2. Software interrupts
3. User program interrupts
### Exception Forms

#### POSIX

- **Synchronous**
  - Exception propagation is blocked.
  - Exception information is lost.

- **Asynchronous**
  - Exception propagation is not blocked.
  - Exception information is preserved.

#### Ada

- **Synchronous**
  - Exception propagation is blocked.
  - Exception information is lost.

- **Asynchronous**
  - Exception propagation is not blocked.
  - Exception information is preserved.

### Exception Granularity

#### Block Level

1. **Find an exception handler.**
2. **Execute the exception handler (and potentially raise another exception).**
3. **After completion of the exception handler (and possibly other handlers):**
   - Return to the beginning of the code block in which the exception occurred.

### Exception Recovery

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. After completion of the exception handler (and possibly other handlers):
   - Return to the beginning of the code block in which the exception occurred.

#### Terminating Mode

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. After completion of the exception handler (and possibly other handlers):
   - Return to the beginning of the code block in which the exception occurred.

### Exception Attributes

- **Exception forms**
  - Synchronous
  - Asynchronous

- **Real-time Java**
  - Exception granularity
    - Block level

### Exception Scope and Propagation

1. All procedures and functions declare every potentially raised exception.
2. All exceptions are raised in the context of the current task.
3. Exception granularity is block size (only model in: Ada, Real-time Java; offered in: Pearl, Mesa).
4. Exception-handlers are determined at compile-time (either by propagation or in the static scope).

### Exception Granularity

- **Block-Resumption Model**
  1. Find an exception handler.
  2. Execute the exception handler (and potentially raise another exception).
  3. After completion of the exception handler (and possibly other handlers):
     - Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. After completion of the exception handler (and possibly other handlers):
   - Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. Return to the beginning of the code block in which the exception occurred.

### Exception Handling

**Block-Resumption Model** (block level)

1. Find an exception handler.
2. Execute the exception handler (and potentially raise another exception).
3. Return to the beginning of the code block in which the exception occurred.
Asynchronism in general program flows

Exception handling overview

<table>
<thead>
<tr>
<th>Real-time Java</th>
<th>Ada</th>
<th>CHILL</th>
<th>Modula-3</th>
<th>SELF</th>
<th>Ada/Modula-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction</td>
<td>State</td>
<td>Thread</td>
<td>Propagation</td>
<td>Exception</td>
<td>Handler</td>
</tr>
<tr>
<td>Recovery</td>
<td>State</td>
<td>Stack</td>
<td>State name</td>
<td>State name</td>
<td>Task</td>
</tr>
<tr>
<td>atomic</td>
<td>failed</td>
<td>blocked</td>
<td>null</td>
<td>failure</td>
<td>failure</td>
</tr>
<tr>
<td>Fault</td>
<td>atomic</td>
<td>fault</td>
<td>task</td>
<td>atomic</td>
<td>atomic</td>
</tr>
<tr>
<td>Atomic action</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atomic actions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Atomic actions (Definitions)

An action is atomic if the processes performing it ... 
- ... are not aware of the existence of any other action processes, and no other action process is aware of the activity of the processes during the time the processes are performing the action.
- ... do not communicate with other processes while the action is being performed.
- ... cannot detect any outside state change and do not reflect own state changes until the action is complete.

A failure in atomic action ...
- ... is propagated to both involved tasks.
- ... is declared as failed, if any part of the action fails.
- ... is either performed fully or not at all.
- ... is a part of an atomic action ... 
- ... is either performed fully or not at all.

Atomic actions failure

Failure in outer atomic action.
- Preserve full atomicity.
- Break atomicity and propagate failure immediately.

Atomic actions failure

Failure in inner atomic action.
- Preserve full atomicity.
- Break atomicity and propagate failure immediately.

Exception handling overview

Exception handling issues

Exception handling in Real-time Java

Exceptions are objects in Real-time Java (exceptions have a runtime relation).

Exception handlers can catch:
- One individual exception.
- Exceptions of a class (throwing in Ada).
- All exceptions.
- The kinds of exceptions which are to be handled can be denoted precisely, completely and safely.

Exception handling in C / POSIX

There is none.

Unavailable: using POSIX long jumps or signals, or a lock in a language exception handling method.

Atomic actions (Concurrent)

An atomic action ...
- ... is either performed fully or not at all.
- ... is declared as failed, if any part of the action fails.

Atomic actions

Atomic actions failure

Failure in outer atomic action.
- Preserve full atomicity.
- Break atomicity and propagate failure immediately.

Atomic actions failure

Failure in inner atomic action.
- Preserve full atomicity.
- Break atomicity and propagate failure immediately.
Asynchronism

Hiding asynchronism

Atomic actions implementations

Atomic actions in Ada (specification)

Atomic actions in Ada (implementation)

Atomic actions failure recovery

Atomic actions in real-time environments

Asynchronous events and transfer of control

Asynchronous events

Asynchronous events and transfer of control

Asynchronous events
Asynchronous events

Real-time Java: Asynchronous events

public class RealtimeThread { ... }

Java: Interrupting exception

While asynchronous events lead to a non-blocking behavior, other event classes are modeled to stop the control flow directly:

Real-time Java

There is a standard exception interrupting the flow of execution within a Java statement, namely the RuntimeException. Asynchronously interrupting exceptions can be handled only when a set of different methods handle different kinds of events:

Ada: Interrupt handlers

package Ada.Interrupts is
...
Asynchronism

Asynchronous Transfer of Control

Case 1: Rendezvous starts and completes before timeout
1. The trigger is matched, and the optional statement following the trigger is executed. When the select statement is completed, the abortable part is never started.
2. The trigger is matched and the select statement is completed. If the body is abortable, the abortable part is executed.
3. If the abortable part completed before the trigger is completed, the exceptions from the abortable part are lost!
4. If the abortable part completed after the trigger is completed, the abortable part is aborted, and the optional statement following the trigger is executed.

Case 2a: Rendezvous completes before ST
1. The trigger is matched, and the optional statement following the trigger is executed. When the select statement is completed, the abortable part is never started.
2. The trigger is matched and the select statement is completed. If the body is abortable, the abortable part is executed.
3. If the abortable part completes before the trigger is completed, the exceptions from the abortable part are lost!
4. If the abortable part completes after the trigger is completed, the abortable part is aborted, and the optional statement following the trigger is executed.

Note about exception handling:

All parts of a select-then-abort statement raise exceptions, set
in order of interruption of the abortable part, the select-then-abort statement.

Ado Asynchronous Transfer of Control
Asynchronism in Ada and Real-time Java:

**Ada: Asynchronous Transfer of Control**

- **Asynchronous transfer of control**: A mechanism for allowing tasks to run concurrently without blocking each other.

- **Interrupts / Signals**: Mechanisms for handling unexpected events or signals from the environment.

- **Atomic Actions**: Code segments that are executed without interference from other tasks.

- **Exceptions**: Support for handling exceptional situations, including user-defined exceptions.

- **Definitive / Requirements / Failure Cases / Implementation / Error Recovery**: Detailed specifications and guidelines for implementing asynchronous control.

Case 2b: ST completes before rendezvous

Case 3a: ST completes before rendezvous starts

Case 3b: Rendezvous completes before ST starts

Case 3c: Rendezvous completes before ST completes

Language Support for Asynchronism in Ada and Real-time Java:

- **ATC-enabled regions**: Regions of code that are enabled for asynchronous transfer of control.

- **Asynchronous Transfer of Control (ATC)**: A mechanism for enabling tasks to run concurrently in Ada.

- **Mechanisms**: Support for handling asynchronous events, including interrupts and signals.

- **Atomic Actions**: Code segments that are executed independently of other tasks.

- **Exceptions**: Support for handling exceptions, including user-defined exceptions.

- **Recovery**: Mechanisms for handling errors and failures in asynchronous environments.

**Summary**

- **Asynchronism**: The ability of tasks to run concurrently without blocking each other.

- **Interrupts and Signals**: Mechanisms for handling unexpected events or signals from the environment.

- **Atomic Actions**: Code segments that are executed independently of other tasks.

- **Exceptions**: Support for handling exceptional situations, including user-defined exceptions.

- **Recovery**: Mechanisms for handling errors and failures in asynchronous environments.