

# Cyber-Physical Systems and Discrete Controllers

CS684: Embedded Systems

Topic 1

Paritosh Pandya

Indian Institute of Technology, Bombay

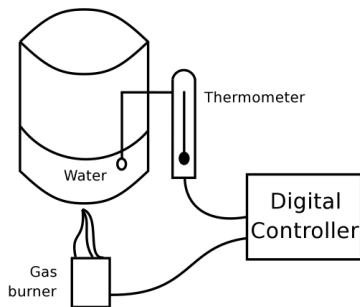
January 12, 2023

- Physical system consisting of mechanical and electronics components controlled by computerised controller.
- Languages for Programming **Discrete Controllers**: C, SystemC, Verilog/VHDL, State charts, Simulink/Stateflow, Lustre/SCADE/Heptagon.

## Synchronous Dataflow Programming

- Using language **Lustre/Heptagon**
- Compiler: Lustre/Heptagon  $\rightarrow$  C/Verilog
- Why Lustre/Heptagon?
  - Very high level modelling language for Discrete Control
  - Functional Dataflow Programs
  - Logical concurrency with clean semantics
  - Deterministic execution
  - Automatic generation of Efficient C code

# Example: Cyber-Physical System



# Example: Cyber-Physical System

- Behavior of the temperature in the tank
  - When the gas burner is OFF the temperature *evolves* according to  $\mathbf{x}(t) = \mathbf{l} e^{-Kt}$   
i.e.  $\dot{x} = -Kx$
  - When the gas burner is ON the temperature *evolves* according to  $\mathbf{x}(t) = \mathbf{l} e^{-Kt} + \mathbf{h} (1 - e^{-Kt})$   
i.e.  $\dot{x} = K(h-x)$

Where  $\mathbf{l}$  is the initial temperature of the water,  $K$  is a constant that depends on the nature of the tank (how much it conducts heat for example),  $\mathbf{h}$  is a constant that depends on the power of the gas burner, and  $t$  models time.

- We will refer to ON and OFF as *modes* of the tank evolution.

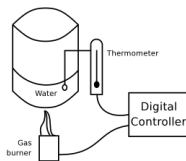
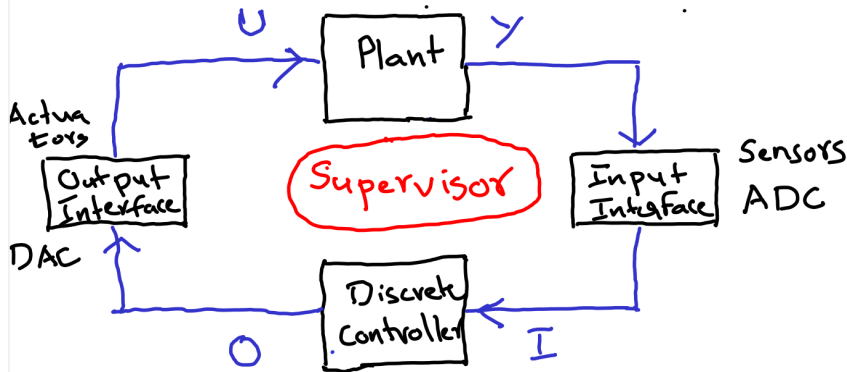


Fig. 1. Our running example

# Model of Cyber-Physical System



# Physical Plant

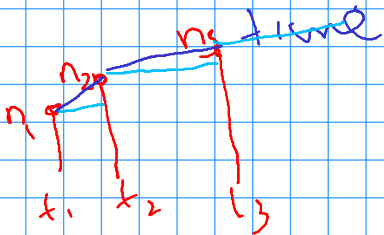
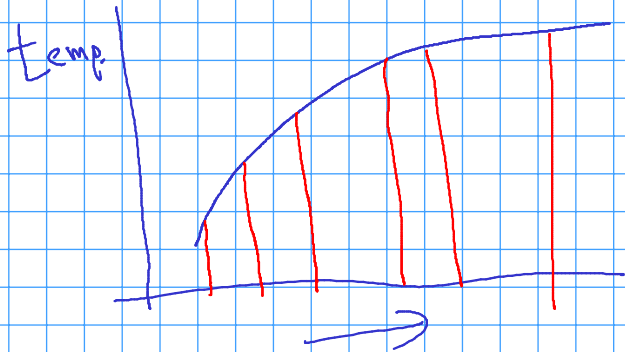
- Includes physical device and environment
- Continuous Dynamics  
Input  $U$ , Output  $Y$  and State of the system  $X$ .  
Continuous trajectory  $X : \mathbb{R}_0 \rightarrow Val_X$
- Mathematical Model using Differential Equations

$$\begin{aligned}X' &= G(X, U) \\ Y &= E(X, U)\end{aligned}$$

$U$



- Converts a continuous time physical signal and gives its value to the discrete controller at discrete time points (sampling points).
- **Example** Analog to Digital Converter (ADC). Organized as Input Driver Library of API calls.
- Conversion and data transfer is done at discrete time points (**sampling points**) decided by the **supervisor**.
- General mathematical model of Interface: Combined Differential+Difference equations.





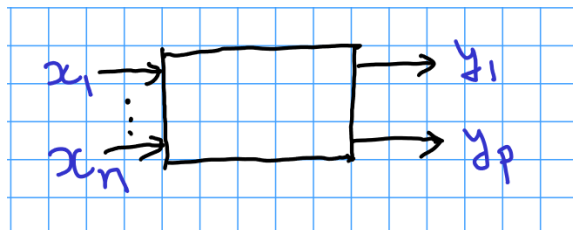
- Converts a discrete time signal from controller into continuous physical signal to the plant.
- **Example** Digital to Analog Converter (DAC). Some outputs may require further computation provided by output drive library.
- Conversion and data transfer is done at discrete time points (**actuation points**) decided by the **supervisor**.
- General model of Interface: Combined Differential+Difference equations.

# Discrete Controller

- Invoked by supervisor repeatedly.
- At each invocation supervisor provides input value.
- Discrete controller produces corresponding output value.
- current output can depend on the sequence of past inputs.
- Example `node MINSOFAR( I : int) returns (O : int)` returns minimum of inputs seen so far.

<i>I</i>	4	0	2	-3
<i>O</i>	4	0	0	-3
<i>clock</i>	0	1	2	3 ...

# Discrete Controller Input Output



# Discrete Controller

- Works in discrete steps of computation.
- Realised using computational devices like Microcontroller, or FPGA, or Sequential Circuit with Flipflops and Gates.
- **Inputs  $I$** , **Outputs  $O$**  and **State  $Z$** .
- **Time is Discrete**, i.e.  $\mathbb{N}$ . A time points is called a clock cycle, tick, or reaction. **Discrete flow**:  $I : \mathbb{N} \rightarrow Val_I$

# Discrete Controller

- Works in discrete steps of computation.
- Realised using computational devices like Microcontroller, or FPGA, or Sequential Circuit with Flipflops and Gates.
- **Inputs**  $I$ , **Outputs**  $O$  and **State**  $Z$ .
- **Time is Discrete**, i.e.  $\mathbb{N}$ . A time points is called a clock cycle, tick, or reaction. **Discrete flow**:  $I : \mathbb{N} \rightarrow Val_I$

## Model (Forward): Difference Equation / Recurrence

$$Z_0 = R(I)$$

$$Z^+ = F(Z, I)$$

$$O = H(Z, I) \leftarrow$$

$$O_0 = H(Z_0, I_0)$$

*(Handwritten note:  $R(I_0)$  with an arrow pointing to  $Z_0$ )*

Model (backward): Difference Equation / Recurrence.

$$Z_0 = R(I)$$

$$Z = F(Z^-, I)$$

$$O = H(Z, I)$$

# Discrete Controller

$$Z_2 = F(Z_1, I_2),$$

$$Z_1 = F(Z_0, I_1) \quad O_1 = H(Z_1)$$

Model (backward): Difference Equation / Recurrence.

$$Z_0 = R(I_0)$$

$$Z = F(Z^-, I)$$

$$O = H(Z, I)$$

## Behaviour of controller

$I$	$I_0$	$I_1$	$I_2$	...
$Z^-$	—	$Z_0$	$Z_1$	...
$Z$	$Z_0$	$Z_1$	$Z_2$	...
$O$	$O_0$	$O_1$	$O_2$	...
<i>clocktick</i>	0	1	2	...

# Example of Discrete Controller

A discrete controller which at each step returns the minimum of input values seen so far.

```
node MINSOFAR( I:int) returns (O:int)
var m:int;
```

$$m_0 = I_0$$

$$m_i = \min(m_{i-1}, I_i)$$

$$O_i = m_i$$

Behaviour

<i>I</i>	4	0	2	-3
<i>M</i>	4	0	0	-3
<i>O</i>	4	0	0	-3
<i>tick</i>	0	1	2	3 ...

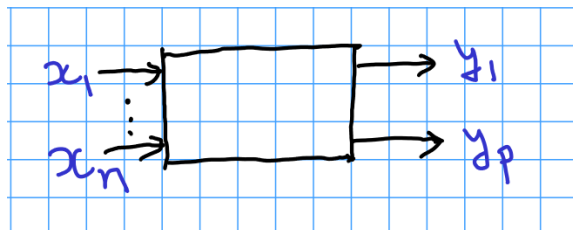


## Event and Time Triggered Systems

```
InitializeControllerMemory;  
for each input event do  
  Read Inputs;  
  Controller step{  
    <Compute Outputs>  
    <Update ControllerMemory>}  
  Perform Actuation  
end
```

```
InitializeControllerMemory;  
for each clock tick do  
  Read Inputs;  
  Controller step{  
    <Compute Outputs>  
    <Update ControllerMemory>}  
  Perform Actuation  
end
```

# Reactive Kernel Input Output



- Definition of **Controller Memory** data type `Controller_f_mem`  
Definition of Structure defining **Output data type** `Controller_f_out`
- Declaration of **controller Memory** variable `mem :Controller__f_mem`
- **Reset function** Procedure resetting controller to **initial state**  $Z_0$ .
- **Step function** Procedure computing **next memory state**  $Z_{i+1}$  and **current output**  $O$ . *using  $Z^i$  and  $I^i$  current*

```
void Controller__f_reset(Controller__f_mem* self)
```

```
void Controller__f_step( t1 x1 , ..., tn xn ,  
                        Controller__f_out* _out,  
                        Controller__f_mem* self );
```

# A Supervisor for Simulating Discrete Controller

```
#include "controller.h"
int main(int argc, char * argv[]) {
  /* declare state variables */
  Controller__f_m mem;
  /* declare input variable */
  t1 x1 ;
  ...
  tn xn ;
  /* declare output variables */
  Controller__f_out ans;
```

# Supervisor for Simulating Discrete Controller

```
/* initialize memory instance */
Controller__f_reset(&mem);
/* repeatedly perform reaction cycle */
while(1) {
    /* read inputs */
    scanf("...", &x1 , ..., &xn );
    /* perform step */
    Controller__f_step(x1 , ..., xn , &ans, &mem);
    /* write outputs */
    printf("...", ans.y1 , ..., ans.yp );
}
```

- Cyber physical system consists of physical elements controlled by the conglomerate of Supervisor, Discrete Controller and the Input-output Interfaces.
- For interacting with physical devices, the supervisor uses API calls from Input Interface Drivers and Output Interface Drivers.
- Control is achieved by **repeated** execution of **Sense**, **ComputeStep**, **Actuate** cycle as directed by the supervisor.
- A Discrete Controller transforms an input flow into an output flow, synchronously.
- A discrete controller can be given as **reactive kernel** consisting of a **reset** and a **step** function with associated declarations.

# High Integrity Embedded Systems

- **Examples** Nuclear reactors, Air Craft Controller, Space Ships (Launch Vehicles, Landers), Defense equipment, Transportation systems, Medical equipment, Smart ...
- **Safety Critical and Mission critical** Cost of failure high

## Challenges

- Low productivity. High development costs. High time-to-market.
- **Certification** Software and its development lifecycle  
DO-178B (Aircrafts)
- **Validation and Verification**

# Model Based Design of Discrete Controller

- Build a **high level model** of System (Discrete Controller) Behaviour
- Executable: Deterministic, concurrent, modular, hierarchical.
- Validation and Verification:
  - Simulation
  - Testing: Requirement based, Coverage.
  - Formal Verification
- Automatic code generation: certified.

## Languages for Model Based Design

State Charts, Simulink+State-flow, Verilog, **Lustre/Scade/Heptagon**