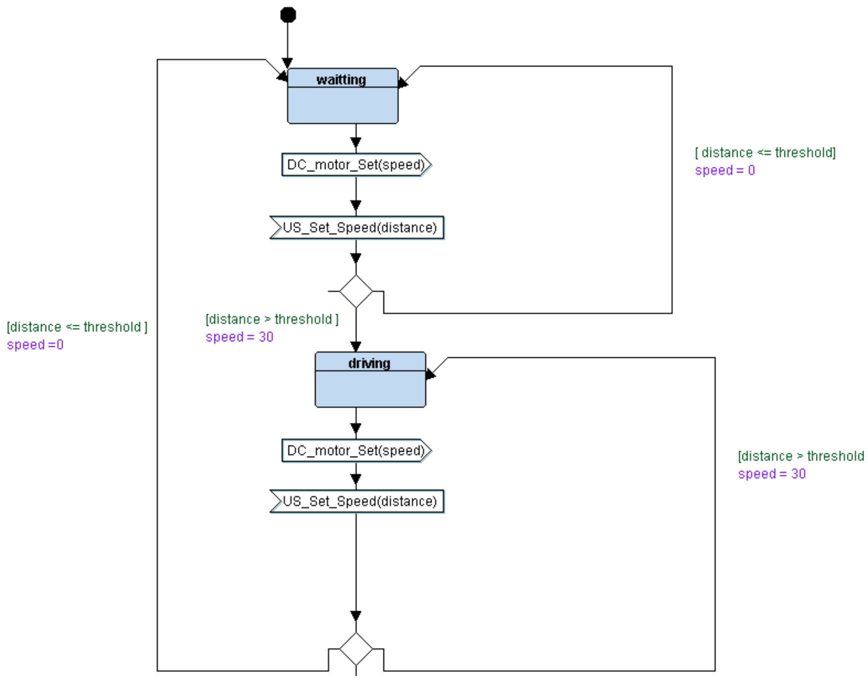
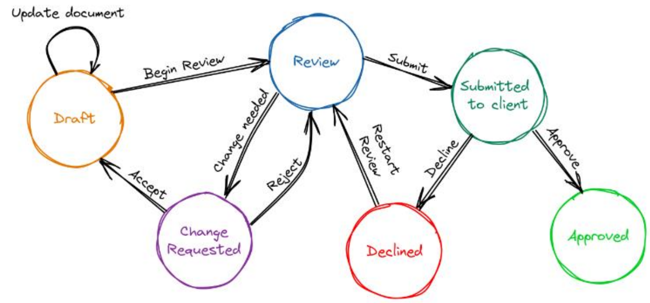
State Machines in C

a state machine is a design pattern used to manage and transition between different states of a system based on inputs or events. This pattern is particularly useful for implementing complex logic where the system needs to respond differently depending on its current state.





Key Concepts

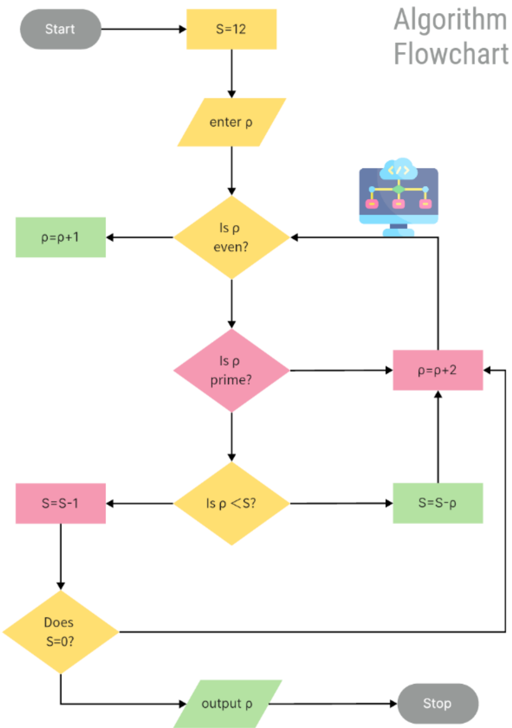
1. **States**: Represent the different conditions or modes the system can be in. Each state may have specific behaviors or actions associated with it.

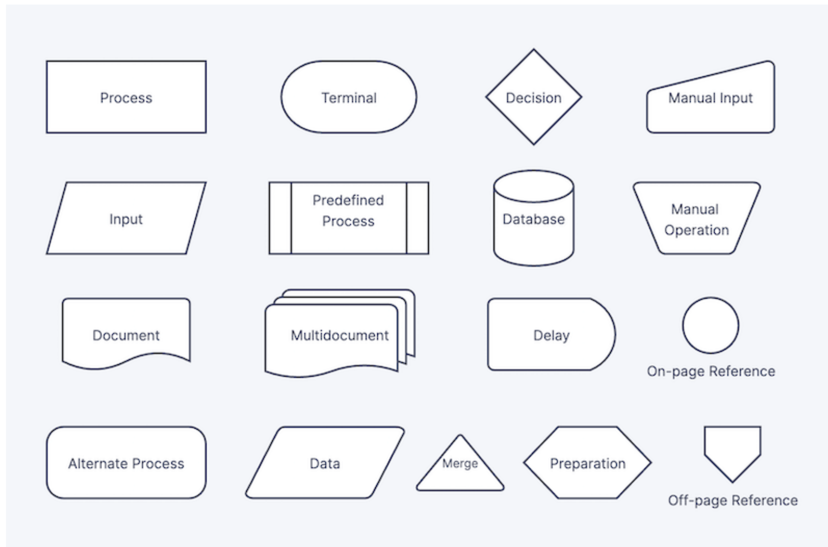
2. **Transitions**: Define how the system moves from one state to another based on certain conditions or inputs.

3. **Events**: Trigger transitions between states. Events are typically inputs or conditions that cause the system to change its state.

Flowchart symbols

A flowchart is a diagrammatic representation used to visualize the sequence of steps and decisions in a process or system. It helps in understanding, designing, and analyzing processes by breaking them down into individual components and illustrating the flow of control between them.





Key Features of Flowcharts

1. **Visual Representation**: Flowcharts use symbols and shapes to represent different types of actions, decisions, and processes, making complex workflows easier to understand.

2. **Step-by-Step Process**: They illustrate each step in a process, from the start to the end, showing how inputs are transformed into outputs.

3. **Decision Points**: Flowcharts highlight decision points where different paths can be taken based on certain conditions.

4. **Flow Direction**: Arrows or lines indicate the direction of flow between different steps or actions.

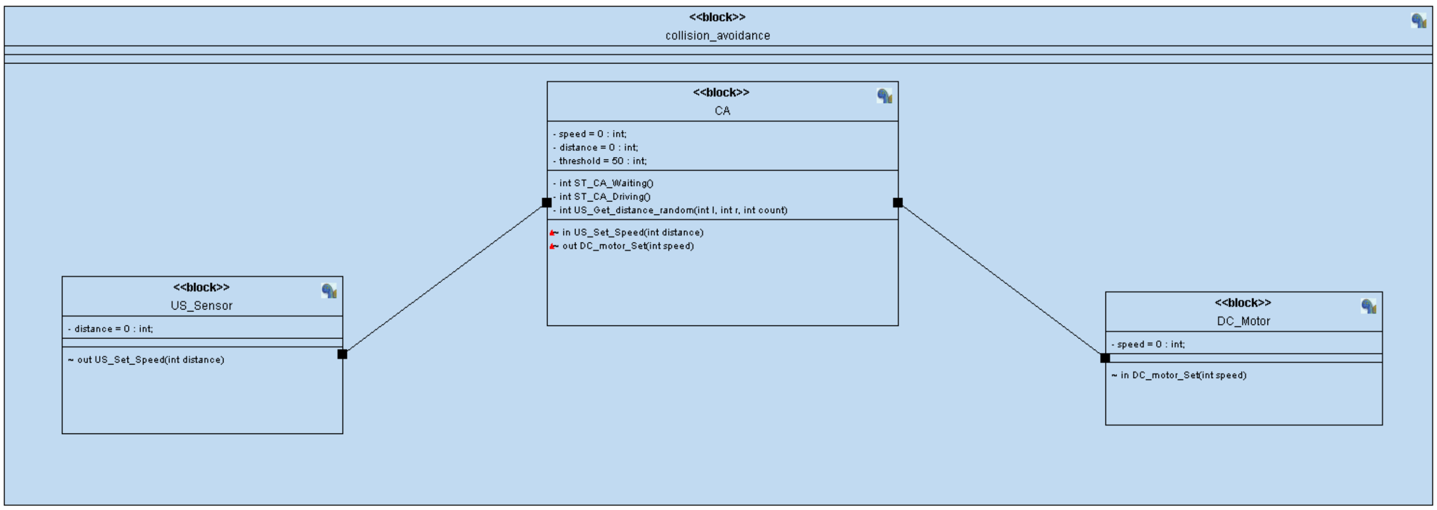
State Machine vs Flowchart

Comparison	State Machines	Flowcharts
Definition	A model used to represent and manage the different states of a system and transitions between those states based on events.	A diagram that represents the sequence of steps and decisions in a process.
Primary Focus	Focuses on different states of a system and transitions between them.	Focuses on the sequence of steps and decisions in a process.
Key Symbols	- **States**: Represents different conditions. - **Transitions**: Shows how the system moves between states based on events. - **Events**: Triggers transitions between states.	- **Start/End (Oval)**: Represents the start or end. - **Process (Rectangle)**: Represents a process step. - **Decision (Diamond)**: Represents a decision point. - **Input/Output (Parallelogram)**: Represents input or output operations.
Applications	- Control systems like traffic lights. - Software state management (e.g., user interfaces). - Embedded systems with state-dependent behavior.	- Documenting and standardizing processes. - Analyzing and troubleshooting workflows. - Designing algorithms and processes.
Complexity	Can be more complex due to multiple states and transitions.	Typically simpler, focusing on the sequence of steps and decisions.
Representation	- **State Diagram**: Uses nodes (states) and directed edges (transitions). - **State Table**: Tabular representation of states, events, and transitions.	- **Flowchart Symbols**: Arranged in sequence with arrows showing the flow. - **Flow Diagram**: Represents the logical sequence of steps.

types of state machines

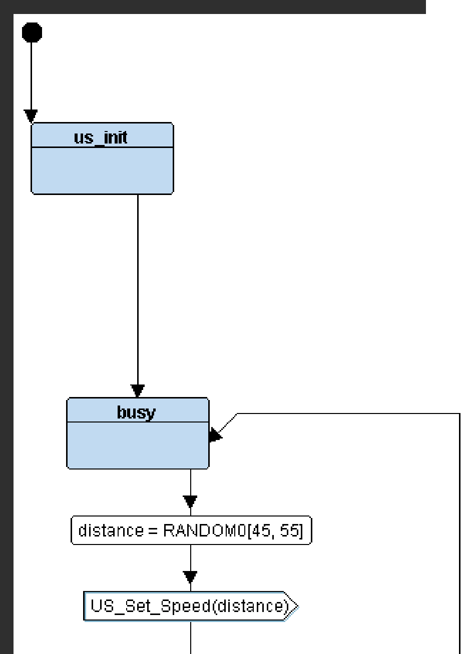
- **Finite State Machine (FSM)**: Basic state machine with a finite number of states.
- **Moore Machine**: Output depends only on the current state.
- **Mealy Machine**: Output depends on both the current state and input.
- **Hierarchical State Machine**: States are nested within other states.
- **Extended State Machine**: Includes additional variables affecting transitions.
- **Probabilistic State Machine**: Uses probabilities for state transitions.
- **Timed State Machine**: Transitions are based on time constraints.

Implement Simple state machine in C using multiple Modules

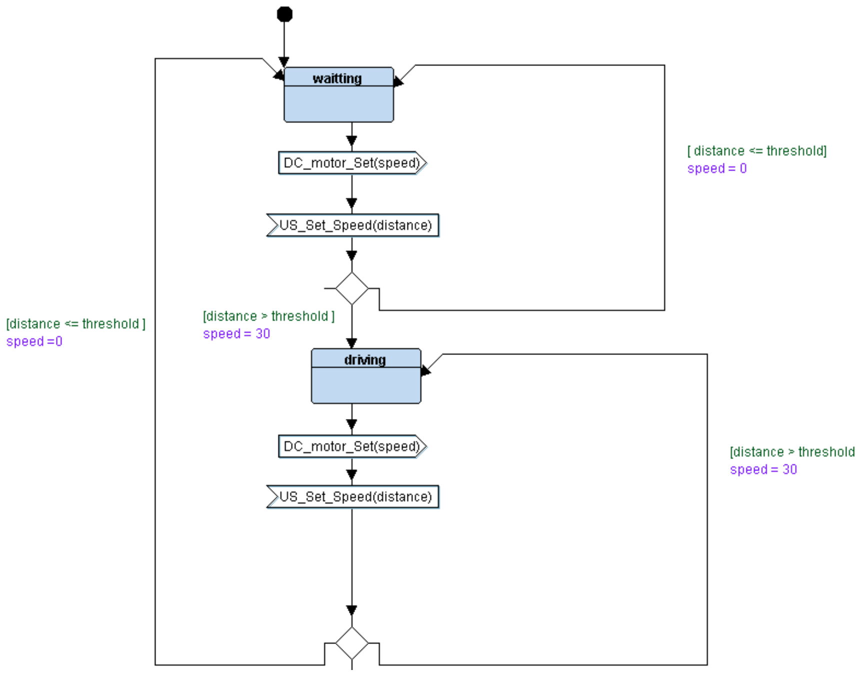


US_Sensor

```c
/*
 * CA.c
 *
 *  Created on: Aug 4, 2024
 *      Author: Abdallah Ghazy
 */

#include "US.h"

int US_distance = 0;


int US_Get_distance_random(int l, int r, int count) {
    return (rand() % (r - l + 1)) + l;
}

void (*US_state)();

US_init(){
        printf("\nUS_init......");
}



STATE_define(US_busy) {
        US_state_id = US_busy;

    US_distance = US_Get_distance_random(45, 55, 1);
    printf("\n CA Waiting State: distance = %d Speed = %d \n", US_distance);
    US_set_distance(US_distance);
    US_state = STATE(US_busy);
}
```



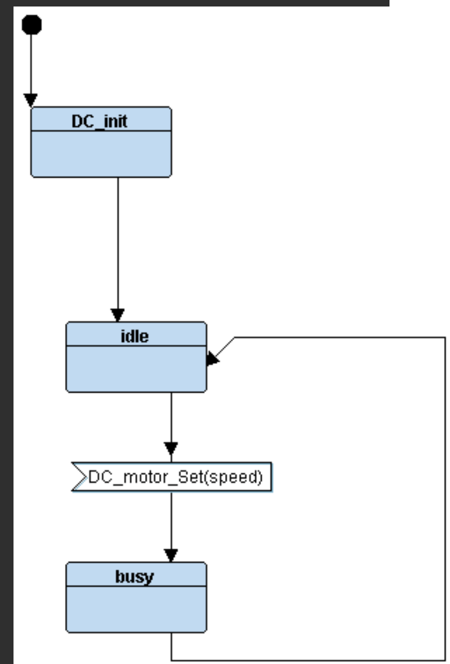
Collision avoiding

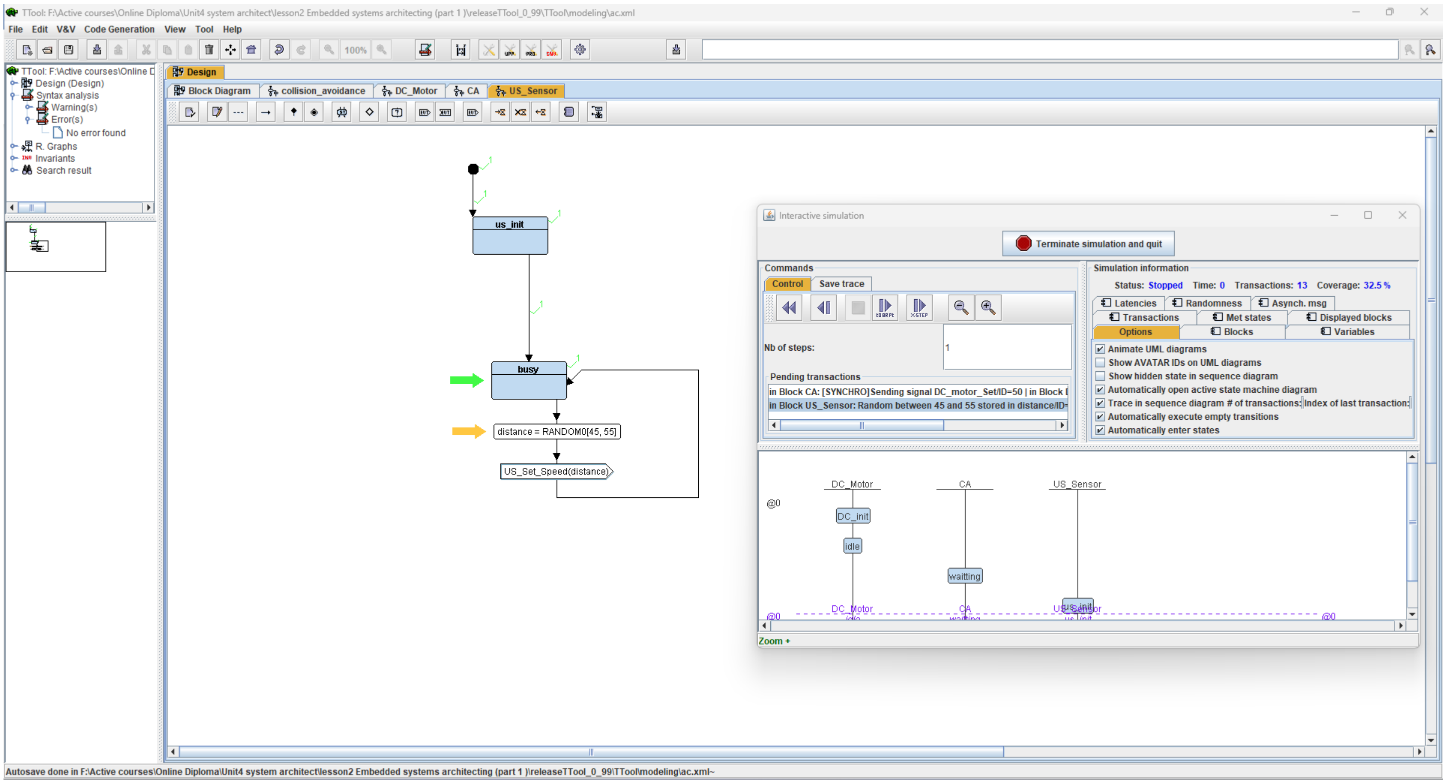


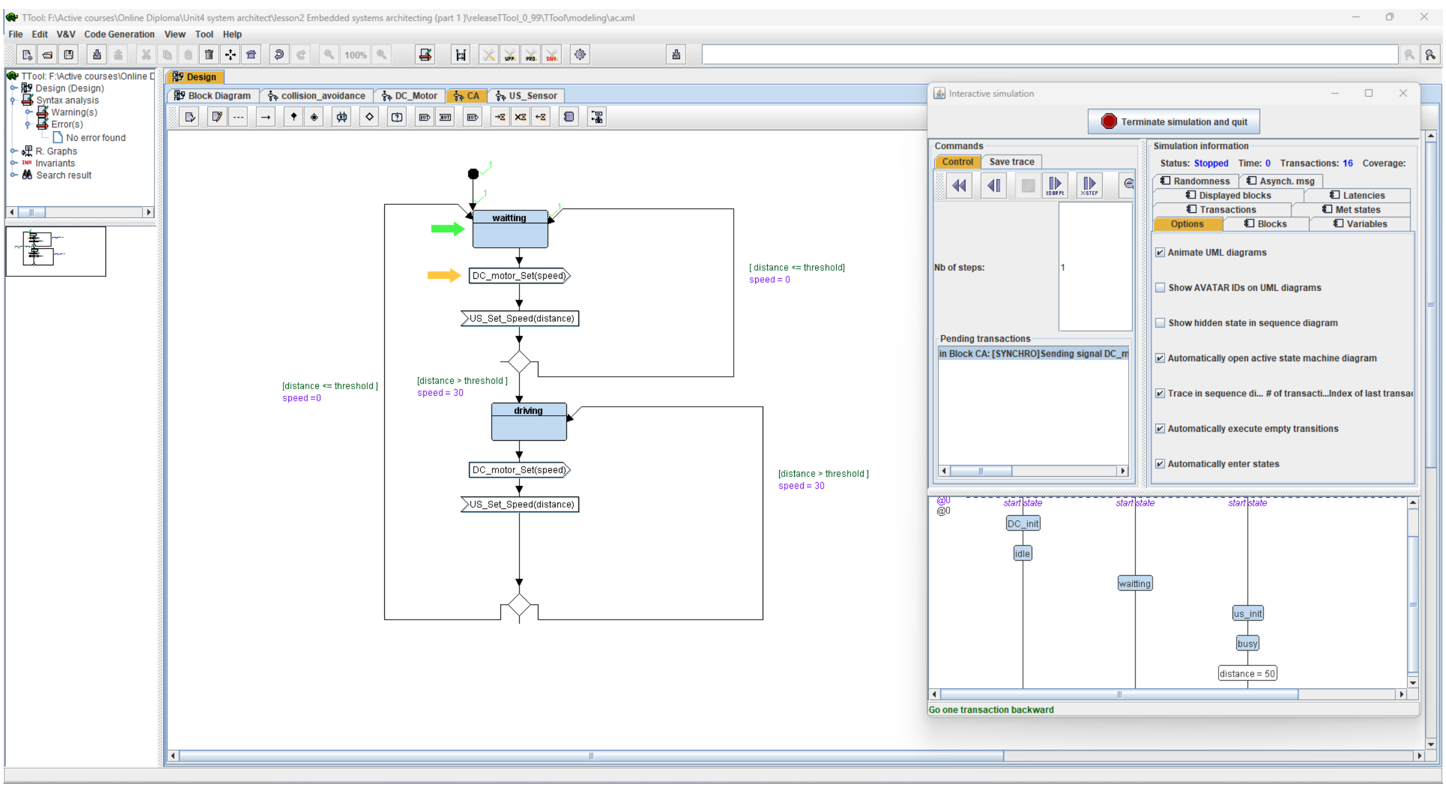

```c
/*
 * CA.c
 *
 *  Created on: Aug 4, 2024
 *      Author: Abdallah Ghazy
 */

#include "CA.h"

int CA_speed = 0;
int CA_distance = 0;
int CA_threshold = 50;

void (*CA_state)();

void US_set_distance(int d){
       CA_distance = d;
       CA_state = (CA_distance <= CA_threshold) ? STATE(CA_waiting) : STATE(CA_driving);

       printf("US-------------- Distance = %d -------------------------- \n",CA_distance);
}



STATE_define(CA_waiting) {
    CA_state_id = CA_waiting;
    printf("\n CA_Waiting State: distance = %d Speed = %d \n", CA_distance, CA_speed);
    CA_speed = 0;
    DC_motor(CA_speed);
}

STATE_define(CA_driving) {
    CA_state_id = CA_driving;
    CA_speed = 30;
    printf("\n CA_Driving State: distance = %d Speed = %d \n", CA_distance, CA_speed);


    DC_motor(CA_speed);
}
```

DC_Motor

```c
/*
 * CA.c
 *
 *  Created on: Aug 4, 2024
 *      Author: Abdallah Ghazy
 */

#include "DC.h"
#include "state.h"
int DC_speed = 0;
void (*DC_state)();

void DC_init() {
	printf("DC_init............");
}

void DC_motor(int s) {
	DC_speed = s;
	DC_state = STATE(DC_busy);
	printf("US-------------- DC_speed = %d -------------------------- \n",
			DC_speed);
}

STATE_define(DC_idle) {
	DC_state_id = DC_idle;

	printf("\n DC_idle State: Speed = %d \n", DC_speed);
}

STATE_define(DC_busy) {
	DC_state_id = DC_busy;
	printf("US-------------- DC_speed = %d -------------------------- \n",
			DC_speed);
}
```




TTool: F:\Active courses\Online Diploma\Unit4 system architect\lesson2 Embedded systems architecting (part 1)\releaseTTool_0_99\TTool\modeling\ac.xml
File Edit V&V Code Generation View Tool Help
Design
Block Diagram
collision_avoidance
DC_Motor
CA
US_Sensor
us_init
busy
distance = RANDOM0[45, 55]
US_Set_Speed(distance)
Interactive simulation
Terminate simulation and quit
Commands
Control
Save trace
Nb of steps:
Pending transactions
in Block CA: [SYNCHRO]Sending signal DC_motor_Set/ID=50 | in Block
in Block US_Sensor: Random between 45 and 55 stored in distance/ID=
Simulation information
Status: Stopped Time: 0 Transactions: 13 Coverage: 32.5 %
Latencies
Randomness
Asynch. msg
Transactions
Met states
Displayed blocks
Options
Blocks
Variables
Animate UML diagrams
Show AVATAR IDs on UML diagrams
Show hidden state in sequence diagram
Automatically open active state machine diagram
Trace in sequence diagram # of transactions: Index of last transaction:
Automatically execute empty transitions
Automatically enter states
DC_Motor
CA
US_Sensor
@0
DC_init
idle
waitting
Zoom +
Autosave done in F:\Active courses\Online Diploma\Unit4 system architect\lesson2 Embedded systems architecting (part 1)\releaseTTool_0_99\TTool\modeling\ac.xml~


TTool: F:\Active courses\Online Diploma\Unit4 system architect\lesson2 Embedded systems architecting (part 1)\releaseTTool_0_99\TTool\modeling\ac.xml
File Edit V&V Code Generation View Tool Help
Design
Block Diagram
collision_avoidance
DC_Motor
CA
US_Sensor
waitting
DC_motor_Set(speed)
US_Set_Speed(distance)
[distance <= threshold]
speed = 0
[distance <= threshold]
speed =0
[distance > threshold]
speed = 30
driving
DC_motor_Set(speed)
US_Set_Speed(distance)
[distance > threshold]
speed = 30
Interactive simulation
Terminate simulation and quit
Commands
Control
Save trace
Nb of steps:
Pending transactions
in Block CA: [SYNCHRO]Sending signal DC_m
Simulation information
Status: Stopped Time: 0 Transactions: 16 Coverage:
Randomness
Asynch. msg
Displayed blocks
Latencies
Transactions
Met states
Options
Blocks
Variables
Animate UML diagrams
Show AVATAR IDs on UML diagrams
Show hidden state in sequence diagram
Automatically open active state machine diagram
Trace in sequence di... # of transacti...Index of last transac
Automatically execute empty transitions
Automatically enter states
start state
DC_init
idle
waitting
us_init
busy
distance = 50
Go one transaction backward

```
init......DC_init............
 CA_Waiting State: distance = 53 Speed = 1
US-------------- Distance = 53 -------------------------

 CA_Driving State: distance = 53 Speed = 30
US-------------- DC_speed = 30 -------------------------
US-------------- DC_speed = 30 -------------------------

 CA_Waiting State: distance = 54 Speed = 1
US-------------- Distance = 54 -------------------------

 CA_Driving State: distance = 54 Speed = 30
US-------------- DC_speed = 30 -------------------------
US-------------- DC_speed = 30 -------------------------

 CA_Waiting State: distance = 54 Speed = 1
US-------------- Distance = 54 -------------------------

 CA_Driving State: distance = 54 Speed = 30
US-------------- DC_speed = 30 -------------------------
US-------------- DC_speed = 30 -------------------------

 CA_Waiting State: distance = 46 Speed = 1
US-------------- Distance = 46 -------------------------

 CA_Waiting State: distance = 46 Speed = 30
US-------------- DC_speed = 0 -------------------------
US-------------- DC_speed = 0 -------------------------

 CA_Waiting State: distance = 52 Speed = 1
US-------------- Distance = 52 -------------------------

 CA_Driving State: distance = 52 Speed = 30
US-------------- DC_speed = 30 -------------------------
US-------------- DC_speed = 30 -------------------------

 CA_Waiting State: distance = 50 Speed = 1
US-------------- Distance = 50 -------------------------

 CA_Waiting State: distance = 50 Speed = 30
US-------------- DC_speed = 0 -------------------------
US-------------- DC_speed = 0 -------------------------

 CA_Waiting State: distance = 50 Speed = 1
US-------------- Distance = 50 -------------------------

 CA_Waiting State: distance = 50 Speed = 0
US-------------- DC_speed = 0 -------------------------
US-------------- DC_speed = 0 -------------------------
```