Finite State Machines

Lino Marques

2023

Outline

- Fundamentals of FSM
- Implementation
- Example
- Proposal

Fundamentals

- A finite-state machine (FSM) is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some inputs; the change from one state to another is called a transition.
- FSM and variants: **Moore** (output depends only on the state), **Mealy** (depends on the state and input variables), UML statecharts, Petri nets, Sequential function charts (SFC)…

C language implementation Definition of states

```
typedef enum{
    STATE1,
    STATE2,
    STATE3,
    STATE4,
} states_t;

states_t state;
```

FSM implementation

- Implement the finite state machine using switch/case or if/else constructs

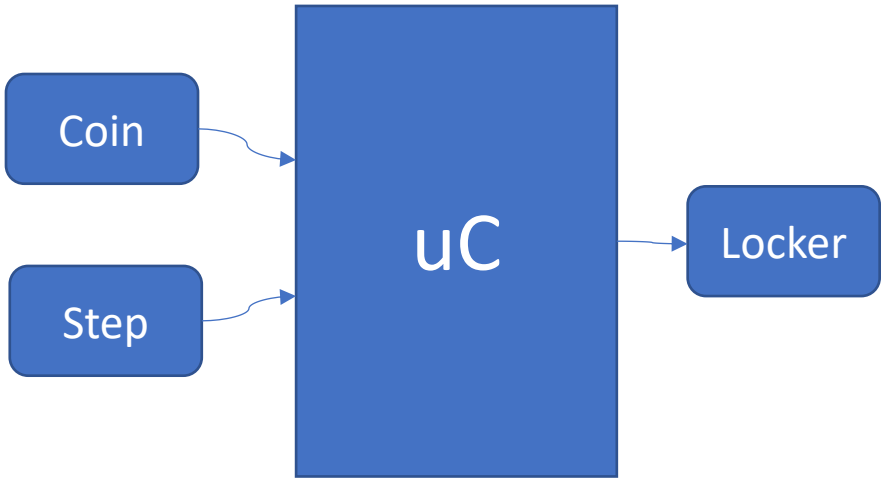
```
while (1) {
   switch(state){
      case STATE1:
         // Do something
         State1Task();
         if(TransitionCondition1())
            state = State_i;
         else if(TransitionCondition2())
            state = State_j;
         break;
      case STATE2:
         State2Task();
         if(TransitionCondition2())
            state = State_m;
         break;
      default: state = State_k;
   }
```

Example: coin-operated turnstile

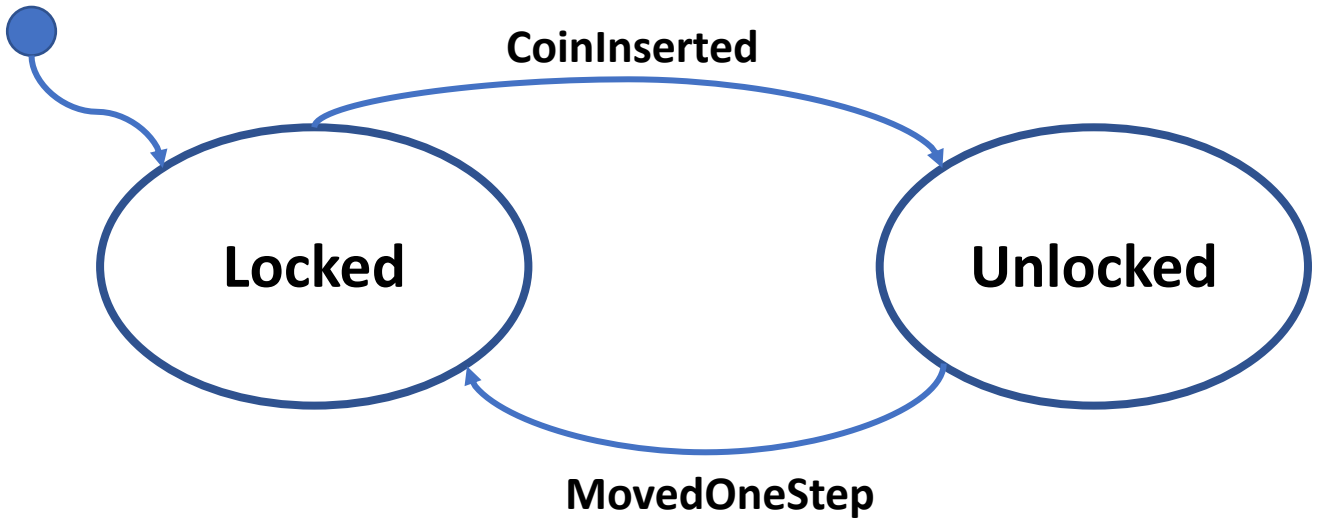
- A person willing to go through, needs to insert a coin in order to unlock the turnstile.
- After moving the turnstile until the next locking position, the system changes again to the locked state.

Architecture

- Actuators (uC outputs)
  - Locking mechanism
- Detectors (uC inputs)
  - Coin detector
  - Position detector



State diagram



Implementation

- Inputs:
  - CoinDetector: PC13 B1 (Botão azul)
  - StepDetector: PC11
- Output:
  - Locker: PB7 LD2 (LED azul)

```
typedef enum{
    LOCKED,
    UNLOCKED,
} states_t;

states_t state=LOCKED;
GPIO_PinState lastStep = GPIO_PIN_SET, currStep;
```

Main loop

```
while (1)
   {
   switch(state){
      case LOCKED:
         // Do something (if necessary)
         HAL_GPIO_WritePin(Locker_GPIO_Port, Locker_Pin, GPIO_PIN_RESET);
         if(HAL_GPIO_ReadPin(CoinDetector_GPIO_Port, CoinDetector_Pin))
            state=UNLOCKED;
         break;
      case UNLOCKED:
         HAL_GPIO_WritePin(Locker_GPIO_Port, Locker_Pin, GPIO_PIN_SET);
         // Wait for rising edge of StepDetector
         currStep = HAL_GPIO_ReadPin(StepDetector_GPIO_Port, StepDetector_Pin);
         if(currStep && !lastStep)
            state=LOCKED;
         lastStep = currStep;
         break;
      default: state=LOCKED;
   }
```

Challenge: Vending machine

- Assume a vending machine containing 4 types of snacks. Each snack costs 1€ and the machine contains a sensor to detect the insertion of 1€ coins. After receiving a coin, the machine activates a green LED and lets the user select the desired snack by pressing one of four available buttons. The machine contains a sensor to detect the release of a product.

The system

- For inspiration, you should consider the machine described in the following link: https://blog.arduino.cc/2016/06/29/venduino-is-a-diy-arduino-vending-machine/

Architecture

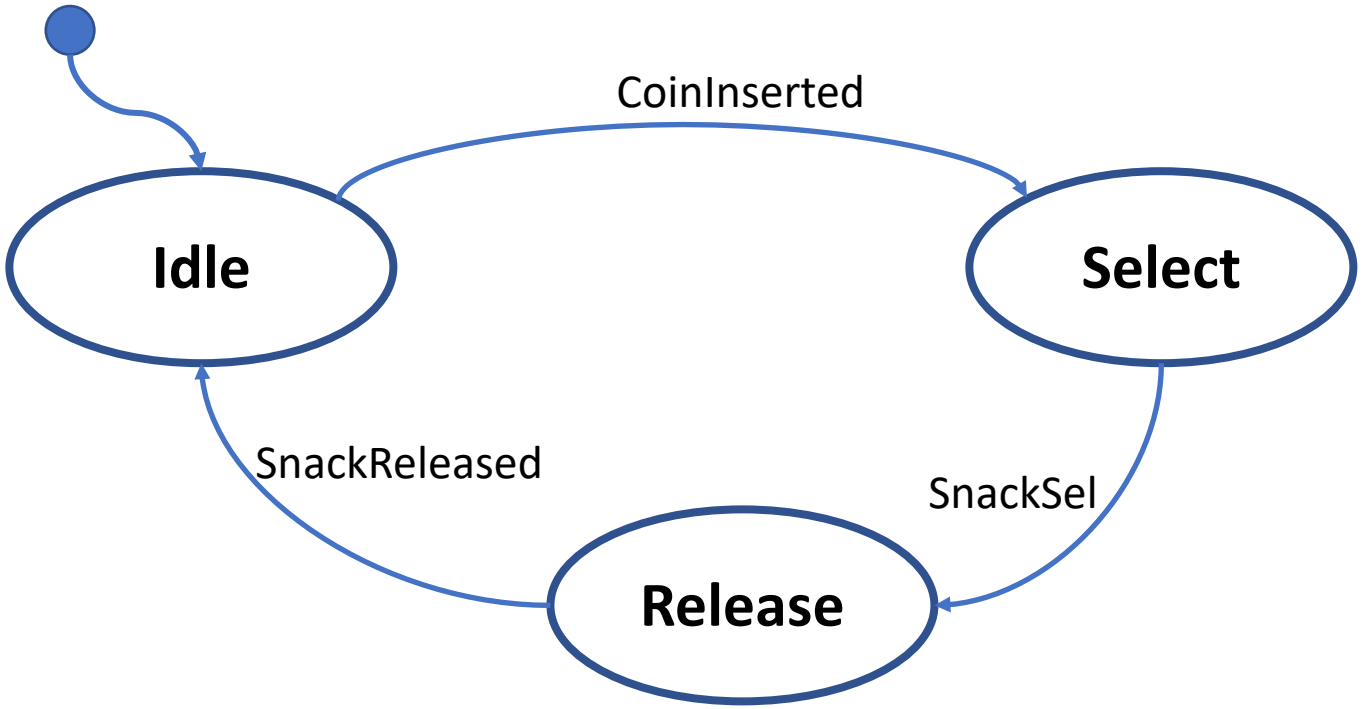
Inputs:

- CoinDetector: PC13
- Snack1: PC6
- Snack2: PB15
- Snack3: PB13
- Snack4: PB12
- ProdReleased: PA4

Outputs:

- LED: PB7
- Motor1: PB4
- Motor2: PB5
- Motor3: PB8
- Motor4: PB9

State diagram



Implementation (draft)

```
while (1) {
 switch(state){
 case IDLE:
      //Actions for state Idle
      if(HAL_GPIO_ReadPin(CoinDetector_GPIO_Port, CoinDetector_Pin)){
           state = SELECT;
           }
      break;
 case SELECT:
      //Actions for state Select
      tecla = readKeypad();
      if(tecla){
           state = RELEASE;
           }
      break;
 case RELEASE:
       //Actions for state Release
      if(HAL_GPIO_ReadPin(ProdReleased_GPIO_Port, ProdReleased_Pin)){
           state=IDLE;
           }
 }
}
```