This section requires advanced programming skills. The regular reader could skip this section and have no trouble with the rest of the class. This section uses a function variable. For example we could define a function variable as such

void (*functionPt)(void);

This variable points to a function with no inputs or outputs. Assume we have this function

```
void test(void){
  printf("hello");
}
```

I can execute this code to set the value of the function variable to equal the function **test**.

```
 functionPt = &test;
```

I can execute the function defined by the function variable using this syntax

```
 (*functionPt)();
```

If you do not understand this brief introduction to function variables, I suggest you skip this section and go on to the stepper motor section.

Example 10.2. Design vending machine with two outputs (soda, change) and two inputs (dime, nickel).

Solution: This vending machine example illustrates additional flexibility that we can build into our FSM implementations. In particular, rather than simple digital inputs, we will create an input function that returns the current values of the inputs. Similarly, rather than simple digital outputs, we will implement general functions for each state. We could have solved this particular vending machine using the approach in the previous example, but this approach provides an alternative mechanism when the input and/or output operations become complex. Our simple vending machine has two coin sensors, one for dimes and one for nickels, see Figure 10.8. When a coin falls through a slot in the front of the machine, light from the QEB1134 sensor reflects off the coin and is recognized back at the sensor. An op amp (OPA2350) creates a digital high at the Port B input whenever a coin is reflecting light. So as the coin passes the sensor, a pulse (V2) is created. The two coin sensors will be inputs to the FSM. If the digital input is high (1), this means there is a coin currently falling through the slot. When a coin is inserted into the machine, the sensor goes high, then low. Because of the nature

of vending machines we will assume there cannot be both a nickel and a dime at the same time. This means the FSM input can be 0, 1, or 2. To implement the soda and change dispensers, we will interface two solenoids to Port E. The coil current of the solenoids is less than 40 mA, so we can use the 7406 open collector driver. If the software makes PE0 high, waits 10ms, then makes PE0 low, one soda will be dispensed. If the software makes PE1 high, waits 10ms, then makes PE1 low, one nickel will be returned.

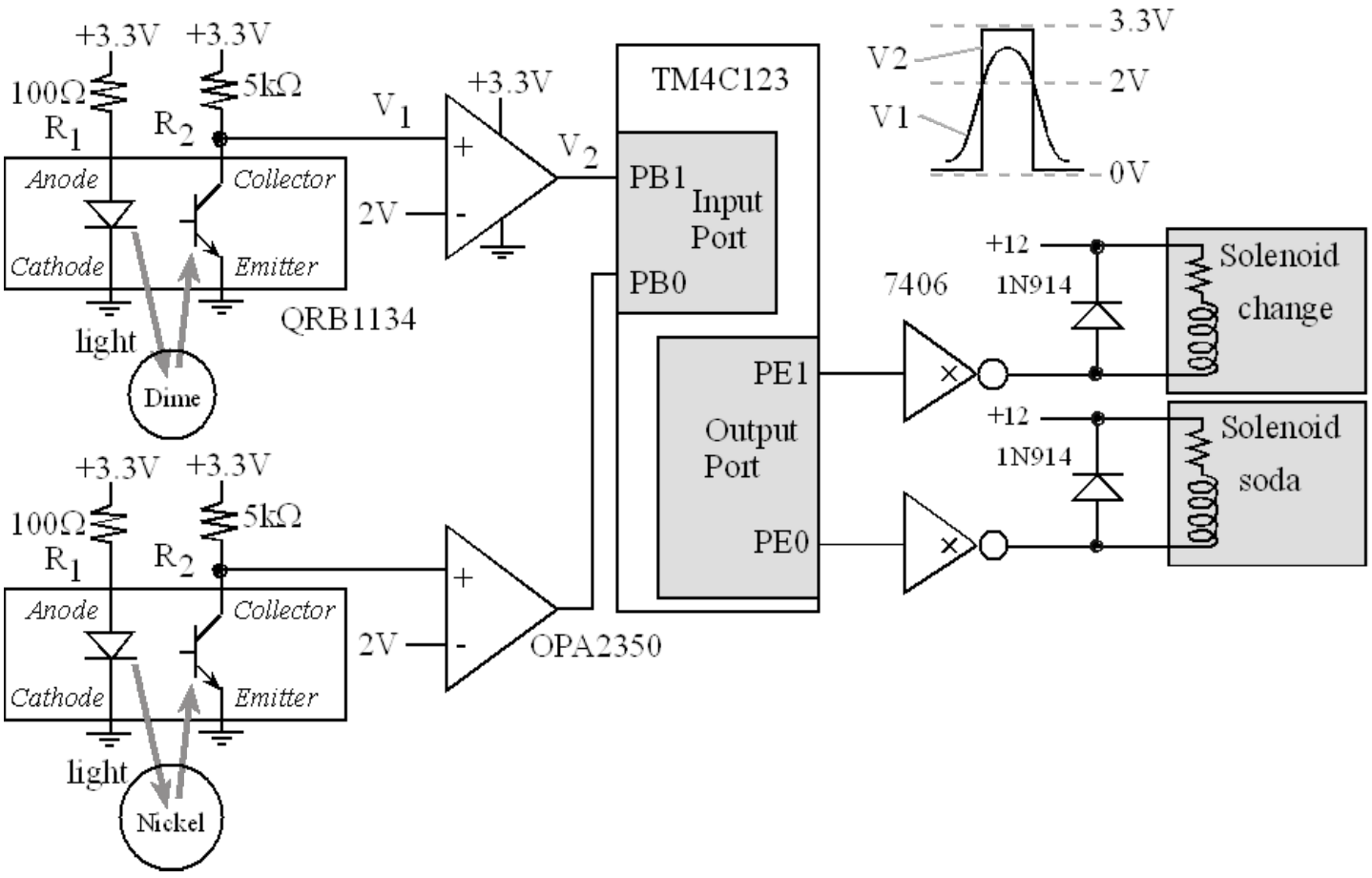

Figure 10.8. A vending machine interfaced to a microcontroller.

We need to decide on the sequence of operations before we draw the state graph.

1) Initialize timer and directions registers
2) Specify initial state
3) Perform FSM controller
    a) Call an output function, which depends on the state
    b) Delay, which depends on the state
    c) Call an input function to get the status of the coin sensors
    d) Change states, which depends on the state and the input.

Figure 10.9 shows the Moore FSM that implements the vending machine. A soda costs 15 cents, and the machine accepts nickels (5 cents) and dimes (10 cents). We have an input sensor to detect nickels (bit 0) and an input sensor to detect dimes (bit 1.) We choose the wait time in each state to be 20ms, which smaller than the time it takes the coin to pass by the sensor. Waiting in each state will debounce the sensor, preventing multiple counting of a single event. Note that we

wait in all states, because the sensor may bounce both on touch and release. Each state also has a function to execute. The function **Soda** will trigger the Port E output so that a soda is dispensed. Similarly, the function **Change** will trigger the Port E output so that a nickel is returned. The **M** states refer to the amount of collected money. When we are in a **W** state, we have collected that much money, but we're still waiting for the last coin to pass the sensor. For example, we start with no money in state **M0**. If we insert a dime, the input will go 10_2, and our state machine will jump to state **W10**. We will stay in state **W10** until the dime passes by the coin sensor. In particular when the input goes to 00, then we go to state **M10**. If we insert a second dime, the input will go 10_2, and our state machine will jump to state **W20**. Again, we will stay in state **W20** until this dime passes. When the input goes to 00, then we go to state **M20**. Now we call the function **change** and jump to state **M15**. Lastly, we call the function **Soda** and jump back to state **M0**.

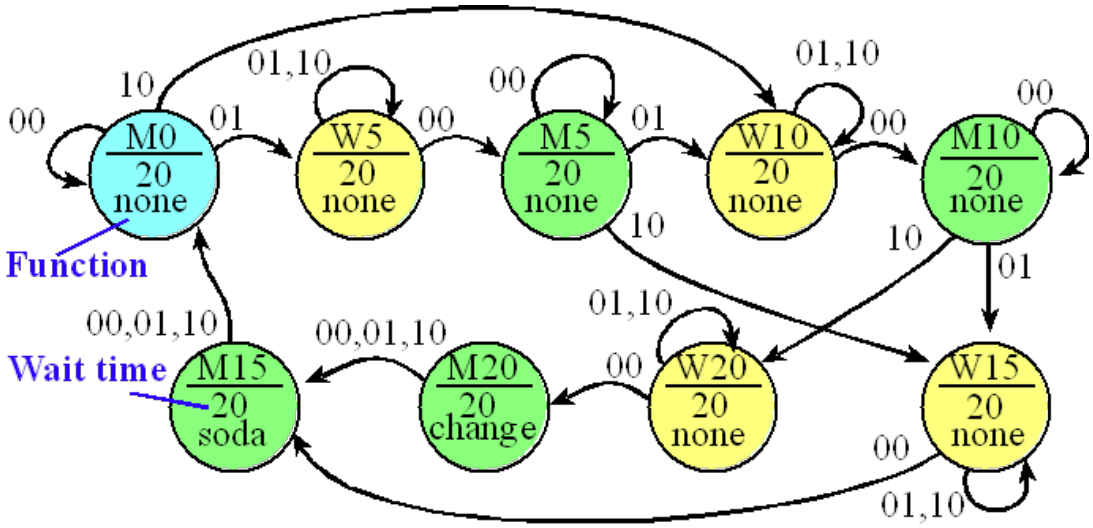

Figure 10.9. This Moore FSM implements a vending machine.

Since this is a layered system, we will begin by designing the low-level input/output functions that handle the operation of the sensors and solenoid, see Program 10.5. The bit-specific addressing **COINS** provides friendly access to PB1 and PB0, **CHANGE** provides friendly access to PE1, and **SODA** provides friendly access to PE0. The initialization specifies Port B bits 1 and 0 to be input and the Port E bits 1 and 0 to be outputs. The PLL and SysTick are also initialized.

```
#define T10ms 800000
#define T20ms 1600000
#define COINS   (*((volatile unsigned long *)0x4002400C))
#define SODA    (*((volatile unsigned long *)0x40005004))
#define CHANGE  (*((volatile unsigned long *)0x40005008))
void FSM_Init(void){ volatile unsigned long delay;
  PLL_Init();       // 80 MHz, Program 10.1
  SysTick_Init();   // Program 10.2
  SYSCTL_RCGC2_R |= 0x12;      // 1) B E
  delay = SYSCTL_RCGC2_R;      // 2) no need to unlock
  GPIO_PORTE_AMSEL_R &= ~0x03; // 3) disable analog function on PE1-0
  GPIO_PORTE_PCTL_R &= ~0x000000FF; // 4) enable regular GPIO
```

```
  GPIO_PORTE_DIR_R &= ~0x03;   // 5) inputs on PE1-0
  GPIO_PORTE_AFSEL_R &= ~0x03; // 6) regular function on PE1-0
  GPIO_PORTE_DEN_R |= 0x03;    // 7) enable digital on PE1-0
  GPIO_PORTB_AMSEL_R &= ~0x3F; // 3) disable analog function on PB5-0
  GPIO_PORTB_PCTL_R &= ~0x000000FF; // 4) enable regular GPIO
  GPIO_PORTB_DIR_R |= 0x03;    // 5) outputs on PB1-0
  GPIO_PORTB_AFSEL_R &= ~0x03; // 6) regular function on PB1-0
  GPIO_PORTB_DEN_R |= 0x03;    // 7) enable digital on PB1-0
  SODA = 0; CHANGE = 0;
}
unsigned long Coin_Input(void){
  return COINS;  // PB1,0 can be 0, 1, or 2
}
void Solenoid_None(void){
};
void Solenoid_Soda(void){
  SODA = 0x01;          // activate solenoid
  SysTick_Wait(T10ms);  // 10 msec, dispenses a delicious soda
  SODA = 0x00;          // deactivate
}
void Solenoid_Change(void){
  CHANGE = 0x02;        // activate solenoid
  SysTick_Wait(T10ms);  // 10 msec, return 5 cents
  CHANGE = 0x00;        // deactivate
}
```

Program 10.5. Low-level input/output functions for the vending machine.

The initial state is defined as **M0**. Our controller software first calls the function for this state, waits for the specified amount of time, reads the sensor inputs from Port B, then switches to the next state depending on the input data. Notice again the 1-to-1 correspondence between the state graph in Figure 10.9 and the data structure in Program 10.6.

```
struct State {
  void (*CmdPt)(void);   // output function
  unsigned long Time;    // wait time, 12.5ns units
  unsigned long Next[3];};
typedef const struct State StateType;
#define M0  0
#define W5  1
#define M5  2
#define W10 3
#define M10 4
#define W15 5
#define M15 6
#define W20 7
```

```
#define M20 8
StateType FSM[9]={
  {&Solenoid_None,  T20ms,{M0,W5,W10}},      // M0, no money
  {&Solenoid_None,  T20ms,{M5,W5,W5}},       // W5, seeing a nickel
  {&Solenoid_None,  T20ms,{M5,W10,W15}},     // M5, have 5 cents
  {&Solenoid_None,  T20ms,{M10,W10,W10}},    // W10, seeing a dime
  {&Solenoid_None,  T20ms,{M10,W15,W20}},    // M10, have 10 cents
  {&Solenoid_None,  T20ms,{M15,W15,W15}},    // W15, seeing something
  {&Solenoid_Soda,  T20ms,{M0,M0,M0}},       // M15, have 15 cents
  {&Solenoid_None,  T20ms,{M20,W20,W20}},    // W20, seeing dime
  {&Solenoid_Change,T20ms,{M15,M15,M15}}};   // M20, have 20 cents
unsigned long S; // index into current state
unsigned long Input;
int main(void){
  FSM_Init();
  S = M0;       // Initial State
  while(1){
    (FSM[S].CmdPt)();           // call output function
    SysTick_Wait(FSM[S].Time);  // wait Program 10.2
    Input = Coin_Input();       // input can be 0,1,2
    S = FSM[S].Next[Input];     // next
  }
}
```

Program 10.6. Vending machine controller.

Help

We're currently aware of some issues with this interactive. Please be patient as we fix this as soon as possible. Thank you.

Deposit 5¢ **Deposit 10¢**

Current input: 00

Total amount inserted: 5 ¢