Embedded Rice Cooker System

An Embedded Systems Project Using the MSP430FR2355

IECE 334: Programming Hardware Systems

Irving Becerril

Date: 04/26/2024

COLLEGE OF
NANOTECHNOLOGY,
SCIENCE, AND ENGINEERING
UNIVERSITY AT ALBANY | STATE UNIVERSITY OF NEW YORK

Agenda

- System Overview
- Topics Integrated
- FSM Design
- Library Design
- Conclusion
- Questions?

System Overview: Goal

- Designed to emulate a rice cooker's actions

System Overview: Functionality

- States
  - OFF
  - SET_COOK_TIME
  - COOK
  - KEEP_WARM

Topics Integrated

- Libraries
  - Digital I/O
  - Port Interrupts
  - Timer Interrupts
- Synchronous State Machine

Topics Integrated Continued

- Analog-to-Digital Conversion (ADC)
- Pulse-Width Modulation (PWM)
- Power Modes (LPM0)

Topics Integrated: ADC

- ADC using potentiometer

```
    configADC();    // Configures ADC
```

```
    case SET_COOK_TIME:
        readADC_Value();
        updateCookTime(60); // Updates time_in_10ms
        break;
```

```
void updateCookTime(int time_in_sec){
    time_in_10ms = getADC_Value()/4095.0 * 100.0 * time_in_sec; // 0-60 seconds
}
```

Topics Integrated: ADC Continued

```
void configADC(){
    P1SEL1 |= BIT2;
    P1SEL0 |= BIT2;
    // Configuring ADC
    ADCCTL0 &= ~ADCSHT; //Clear ADCSHY from def. of ADCSHT = 01
    ADCCTL0 |= ADCSHT_2;    // Conversion Cycles = 16 (ADCSHT = 10)
    ADCCTL0 |= ADCON;   // Turn ADC ON
    ADCCTL1 |= ADCSSEL_2; // ADC Clock Source = SMCLK
    ADCCTL1 |= ADCSHP;    // Sample signal source = sampling timer
    ADCCTL2 &= ~ADCRES;   // Clear ADCRES from def. of ADCRES = 01
    ADCCTL2 |= ADCRES_2;  // Resolution = 12-bit (ADCRES=10)
    ADCMCTL0 |= ADCINCH_2; // ADC Input Channel = A2 (P1.2)
}
```

```
void readADC_Value(){
    // Perform ADC (using polling)
    ADCCTL0 |= ADCENC | ADCSC;        // Enable and Start ADC conversion
    while(( ADCIFG & ADCIFG0) == 0);  // Busy wait for ADC to complete
    adcValue = ADCMEM0;               // Read ADC result
}
```

```
double getADC_Value(){
    return adcValue;
}
```

Topics Integrated: Low Power Modes

- LPM0

```
    case OFF:
        setLPM0(ENTER); // Go into LPM0
        break;
```

```
void startBtn(void){
    setLPM0(EXIT);  // Exit LPM0
    startFlag = HIGH;
}
```

Topics Integrated: Low Power Modes Continued

```
typedef enum{EXIT, ENTER, NEUTRAL} EE;
```

```
EE readLPM0(){
    return _LPM0;
}

void setLPM0(EE enter_exit){
    _LPM0 = enter_exit;

    if(_LPM0 == ENTER){ // Enter Low Power Mode 0
        _LPM0 = NEUTRAL;    // Reset _LPM0
        __bis_SR_register(GIE | LPM0_bits); // Goes into LPM0 state
    }
}
```

Topics Integrated: Low Power Modes Continued

```
#pragma vector = PORT2_VECTOR;
__interrupt void ISR_Port2(void){
    int i;
    for(i = 0; i < 8; i++){ // Finds which bit is HIGH and clears it's interrupt flag
        if(P2IFG & (1 << i)){
            P2IFG &= ~(1 << i); // Clear interrupt flag
            (*port2_ISRs[i])();

            if(port2_ISRs[i] != NULL){
                if(readLPM0() == EXIT){
                    __bic_SR_register_on_exit(LPM0_bits); //Wake up CPU
                    setLPM0(NEUTRAL);
                }
            }
            break;  // Exit loop after finding bit
        }
    }
}
```

Topics Integrated: PWM

- PWM using LED

```
attachCCR0(10, &timerCCR0, &timerCCR1); //10ms clock using CCR0
```

```
case COOK:
    if(countTime == time_in_10ms){
        state = KEEP_WARM;
        countTime = 0;  // Reset countTime
        digitalWrite(P6_0, LOW);    // turn off
        set_CCR1(5);    // Set the 2nd timer for PWM
    }else{
        countTime++;
    }
    break;
```

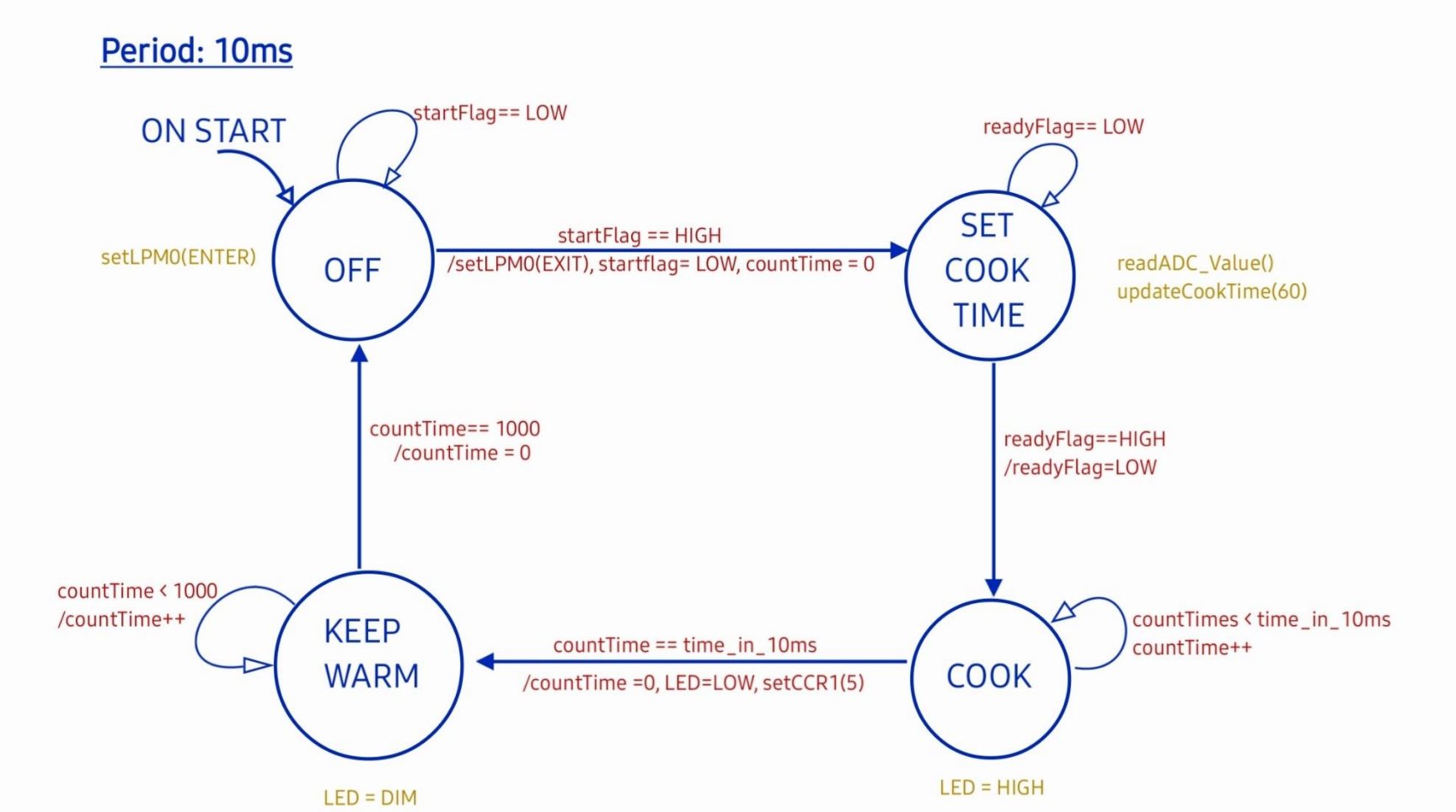
Topics Integrated: PWM Continued

```
void attachCCR0(unsigned int time_ms, void (*funPtr)(void), void(*funPtr2)(void)){
    timerISR = funPtr;
    timerISR_1 = funPtr2;

    TB0CTL |= TBCLR;
    TB0CTL |= TBSSEL__ACLK;
    TB0CTL |= MC__UP;

    TB0CCR0 = time_ms * (32768.0/1000);  // 10ms timer

    //Setup Timer Compare IRQ for CCR0 and CCR1
    TB0CCTL0 |= CCIE;   // Enable TB0 CCR0 Overflow IRQ
    TB0CCTL0 &= ~CCIFG; // Clear CCR0 Flag
    TB0CCTL1 |= CCIE;   // Enable TB0 CCR1 Overflow IRQ
    TB0CCTL1 &= ~CCIFG; // Clear CCR1 Flag
}
```

```
void set_CCR1(unsigned int time_ms){
    TB0CCR1 = time_ms * (32768.0/1000);
}
```

FSM Design

- Synchronous Finite State Machine



Library Design

- Reusability
  - Designed to perform specific, self-contained-tasks
  - Functions are general/abstract
- Manipulate Files
  - Files can work together in order to create different results using different functions

Questions?

- Questions?