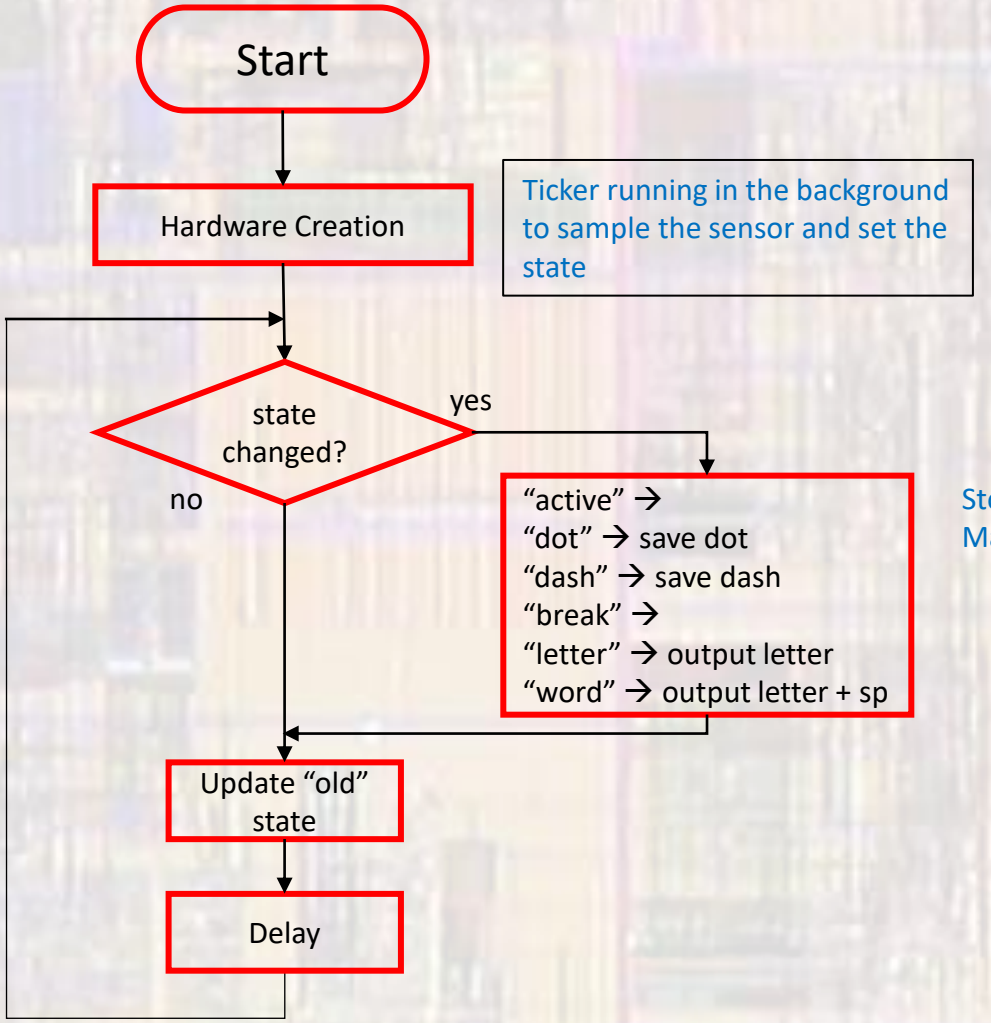
Whole Class Project Programming

Last updated 8/2/21

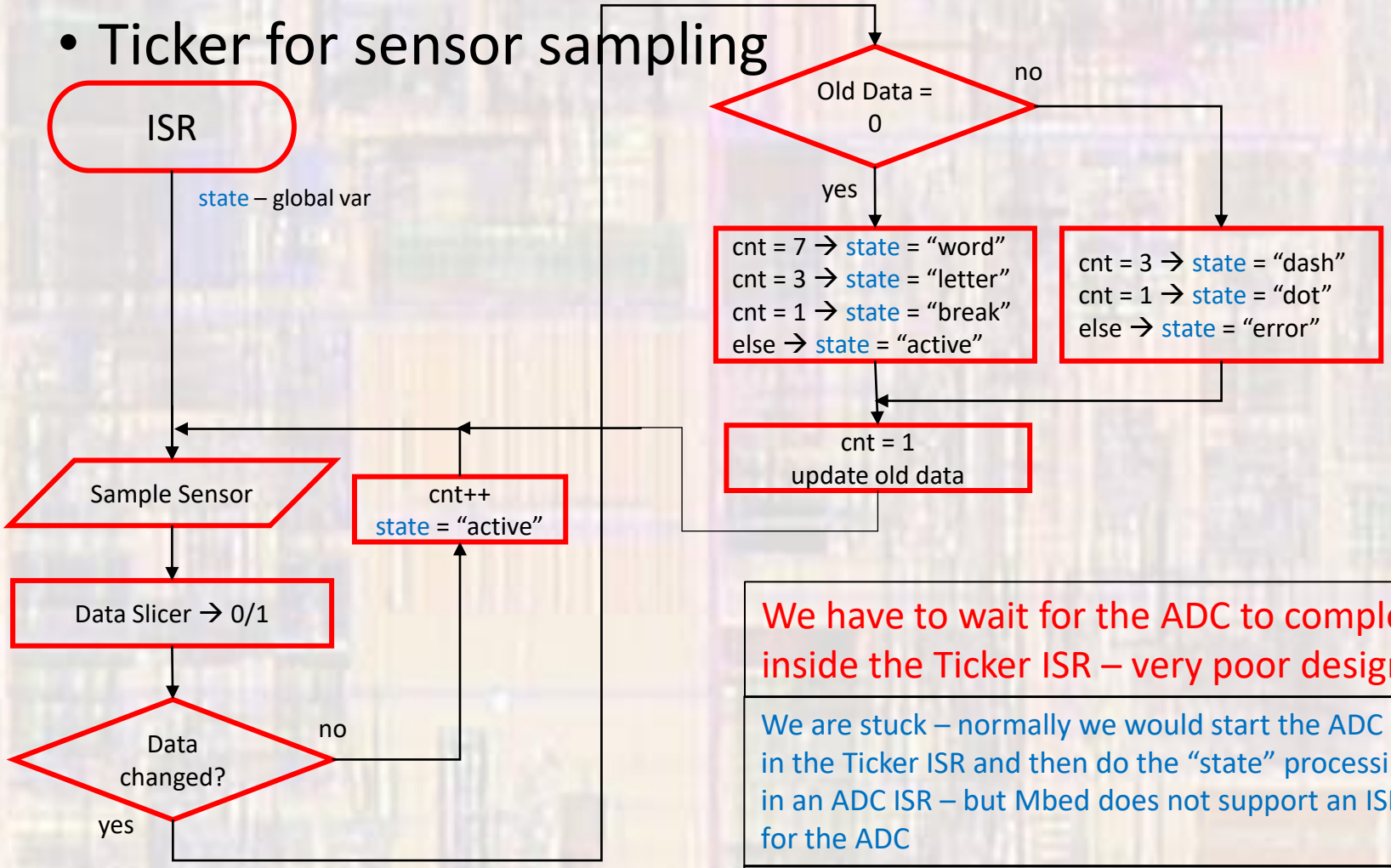
Whole Class Project - Programming

- Main



Whole Class Project - Programming

- Ticker for sensor sampling



We have to wait for the ADC to complete inside the Ticker ISR – very poor design!

We are stuck – normally we would start the ADC in the Ticker ISR and then do the "state" processing in an ADC ISR – but Mbed does not support an ISR for the ADC

Also note – the RTOS version of Mbed does not allow access to the ADC inside an ISR → Bare Metal Profile

Whole Class Project - Programming

- Ticker for sensor sampling
  - Setup

```
//////////////////////////
// Global Area
//////////////////////////

#define TICKER_PERIOD 5ms       // 5 samples per dot/break (25ms)

// ISR function prototypes
void tick_isr(void);


// Global HARDWARE Objects
// Create an ADC object, attached to A3
AnalogIn Photocell(A3);
// Create Ticker object to make measurements
Ticker Tk_1;


// global variable for ISR
uint8_t read_status = ACTIVE;


    ////////////////////////////
    // Inside main
    ////////////////////////////

    // attach ISR and start ticker
    Tk_1.attach(&tick_isr, TICKER_PERIOD);
```

Whole Class Project - Programming

- Ticker for sensor sampling
  - ISR

```
//////////////////////
// Global area
//////////////////////
#define THRESHOLD   0.52     // ADC 0/1 reference from detector.cpp
#define DOT_TICKS   5        // timing values - 5:1 vs dot timing
#define DASH_TICKS  15
#define BREAK_TICKS 6
#define LETTER_TICKS 15
#define WORD_TICKS  35
#define ERROR       0        // state values
#define ACTIVE      1
#define DOT         2
#define DASH        3
#define LETTER      4
#define SPACE       5
#define WORD        6


void tick_isr(void){
    //////////////////
    // ISR to cause a periodic ADC read
    //
    // Must be run in Bare Metal mode due to MUTEX issues
    // in an ISR (the ADC read)
    //////////////////

    // local variables
    float sensor_val;                   // 0 - 1
    uint8_t read_val = 0;               // 0 or 1
    static uint8_t read_val_old = 0;
    static uint8_t num_ticks = 0;

    // read and evaluate the sensor
    sensor_val = Photocell.read();
    if(sensor_val > THRESHOLD)
        read_val = 0;
    else
        read_val = 1;
```

```
    //////////////////////////////////////
    //
    // In a normal system the ADC read would be used to
    // generate a second interrupt and do this processing
    // but mbed does not support ADC interrupts
    //
    //////////////////////////////////////

    // determine the 'value' of the waveform
    if(read_val != read_val_old){       // transition
        if(read_val_old == 0){
            if(num_ticks == WORD_TICKS){
                read_status = WORD;
            } else if(num_ticks == LETTER_TICKS){
                read_status = LETTER;
            } else if(num_ticks == BREAK_TICKS){
                read_status = BREAK;
            } else {
                read_status = ACTIVE;
            }// end if
        } else {                                // read_val_old = 1
            if(num_ticks == DASH_TICKS){
                read_status = DASH;
            } else if(num_ticks == DOT_TICKS){
                read_status = DOT;
            } else {
                read_status = ERROR;
            }// end if
        }// end if
        num_ticks = 1;
    } else {                                    // no transition
        read_status = ACTIVE;
        num_ticks++;
    }// end if

    // update the "old" value
    read_val_old = read_val;

    return;
}// end tick_isr
```

Whole Class Project - Programming

- Saving a Dot or Dash
  - Maximum # of elements in code = 4
    - 4 element array
  - Only 2 values
    - Dot = 2, dash = 3, nothing = 9

N → —• → {9,9,2,3}

V → •••— → {3,2,2,2}

Whole Class Project - Programming

- Outputting a letter
  - Wait for the end of letter indicator
    - 000 from the sensor
    - State = “letter” from ISR
  - Or the end of word indicator
    - 0000000 from the sensor
    - State = “word” from ISR
  - Lots of options to convert the 4 element array to a letter
    - Big if/else
    - Big switch
    - Index into an array

Individual letters are separated
By 3 0’s in a row → “letter”

Individual words are separated
By 7 0’s in a row → “word”

Baud rate = 40Hz
Each 1 or 0 lasts for 25ms

Whole Class Project - Programming

A	10111	•—	J	1011101110111	•———	S	10101	•••
B	111010101	—•••	K	111010111	—•—	T	111	—
C	11101011101	—•—•	L	101110101	•—••	U	1010111	••—
D	1110101	—••	M	1110111	——	V	101010111	•••—
E	1	•	N	11101	—•	W	101110111	•——
F	101011101	••—•	O	11101110111	———	X	11101010111	—••—
G	111011101	——•	P	10111011101	•——•	Y	1110101110111	—•——
H	1010101	••••	Q	1110111010111	——•—	Z	11101110101	——••
I	101	••	R	1011101	•—•			

- Outputting a letter
  - Index into an array
    - Need to get a binary index value
    - Consider the original data format
      - 1, 2, 3, 4 element values
      - If dot is treated as a 0 – no difference between •, ••, and •••
      - If dash is treated as a 0 – no difference between —, ——, and ———
    - Solve the problem by adding a 1 to the MSB location

      • → 10 •• → 100 ••• → 1000 •••• → 1 0000

      — → 11 —— → 111 ——— → 1111
      - Biggest index is ——•— → 1 1101 = 29, just a little bigger than 26

    - Dot/dash to letter conversion array

      U = ••— → 1001 → index 9

```
char letters[] = { '*','*','E','T','I','A','N','M','S','U',
                   'R','W','D','K','G','O','H','V','F','*','L',
                   '*','P','J','B','X','C','Y','Z','Q','*','*'};
```

* = invalid

Whole Class Project - Programming

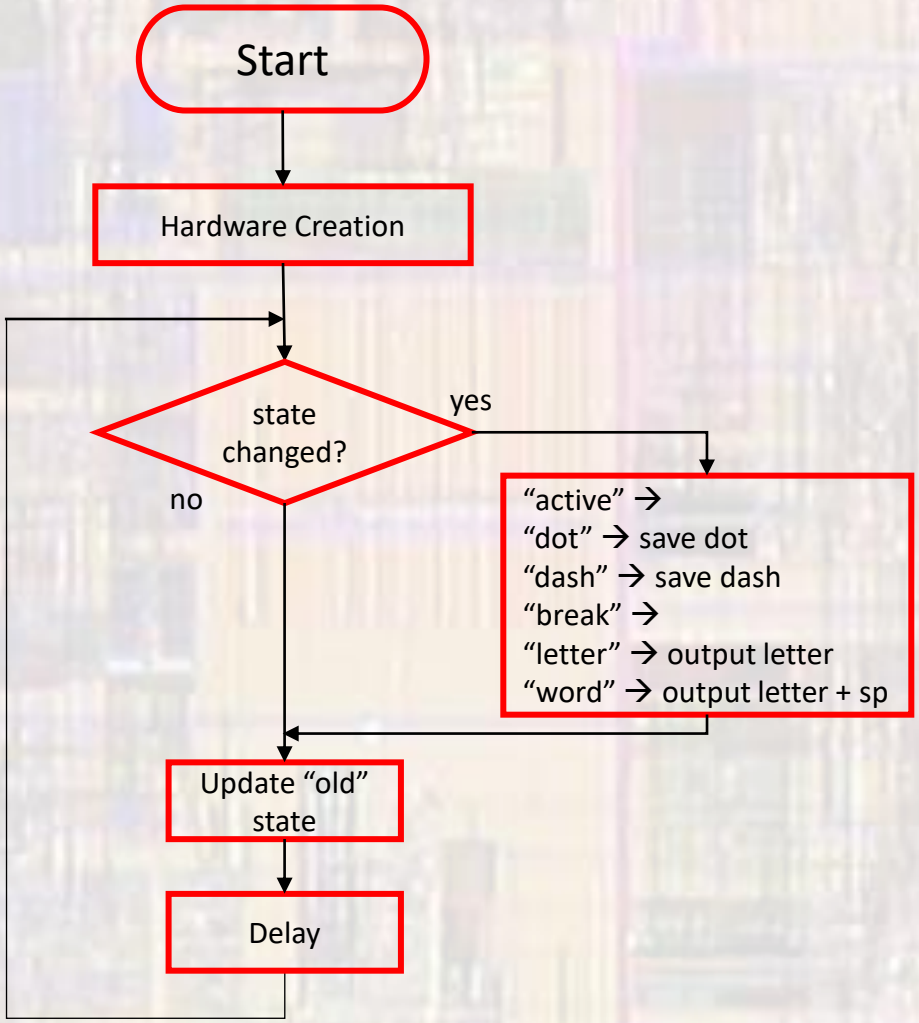
- Outputting a letter

```
uint8_t modify_format(uint8_t array[]){
    ////////////////////
    // this function converts from dot/dash format to a single
    // binary value with a leading 1
    // dot(0) -> 10,    dot dash(01) -> 1 0 1,    dash dash dot dash(1101) -> 1 1 1 0 1
    //
    // it then clears the array by filling with NOVAL (9)
    ////////////////////

    uint8_t i;
    uint8_t bin_value;
    bin_value = 1;                          // initialize leading 1

    // do the conversion
    for(i = 0; i < 4; i++){
        switch(array[i]){
            case DOT:                       // shift with 0 fill
                bin_value = bin_value << 1;
                break;
            case DASH:                      // shift with 1 fill
                bin_value = bin_value << 1;
                bin_value++;
                break;
            case NOVAL:
                break;
            default:
                break;
        } //end switch
    }// end for

    // clear the array
    for(i = 0; i < 4; i++){
        array[i] = NOVAL;
    }

    return bin_value;
}// end modify_format
```

```
char bin_to_ascii(uint8_t binval, const char ary[]){
    ////////////////////////
    // this function uses the modified binary value
    // to convert to ASCII using a pre-defined
    // ASCII array organized to match the modified
    // binary input
    //
    // dot is E which is encoded as 10(modified) so E is located at index 2
    // dash dot dash is R which is encoded as 1101(modified) so R is
    // located at index 13
    ////////////////////////
    char charval;
    charval = ary[binval];

    return charval;
}// end bin_to_ascii
```

Whole Class Project - Programming

- Main



Store Dots and Dashes in an array
Max elements in the array is 4

Whole Class Project - Programming

- Main

```
////////////////////////////////////
//
// project_whole_class project
//
// created 7/26/21 by tj
// rev 0
//
////////////////////////////////////
//
// Whole class project
//
// Bare Metal Profile  (set in json)
//
// Read a code using the photo-sensor
// Code timing - 25ms / dot time
// Ticker timing - 5 samples / dot -> 5ms
//
////////////////////////////////////

#include "mbed.h"
#include <stdio.h>
#include "tims_library.h"

#define THRESHOLD   0.35 //0.52    // ADC 0/1 reference from detector.cpp
#define TICKER_PERIOD   5ms // 5 samples / dot
#define DOT_TICKS   5       // timing values - 5:1 vs dot timing
#define DASH_TICKS  15
#define BREAK_TICKS 5
#define LETTER_TICKS 15
#define WORD_TICKS  35
#define ERROR       0       // state values
#define ACTIVE      1
#define DOT         2
#define DASH        3
#define LETTER      4
#define BRK         5
#define WORD        6
#define NOVAL       9
```

```
// ISR function prototypes
void tick_isr(void);

// function prototypes
char output_letter(uint8_t input_array[], char ref_array[]);
uint8_t modify_format(uint8_t array[]);
char bin_to_ascii(uint8_t binval, const char ary[]);
void print_ary(const uint8_t ary[]);


// Global HARDWARE Objects
// Create an ADC object, attached to A3
AnalogIn Photocell(A3);
// Create Ticker object to make measurements
Ticker Tk_1;
// Bus output to drive 7-segment display
BusOut Sseg(D8, D9, D10, D11, D12, D13, D14, D15);

// global variable for ISR
// note it is defined as volatile since it can
// change without main knowing it - volatile
// forces it to be read from memory each time
// instead of from a CPU register
volatile uint8_t read_status = ACTIVE;
```

Whole Class Project

- Main

```
int main(void){
    setbuf(stdout, NULL);

    // splash
    printf("project_whole_class - example for EE2905\n");
    printf("Using Mbed OS version %d.%d.%d\n\n",
           MBED_MAJOR_VERSION, MBED_MINOR_VERSION, MBED_PATCH_VERSION);

    // local variables
    uint8_t i;                      // tmp index
    uint8_t old_read_status;
    uint8_t letter_ary[4];          // array to hold dot/dash pattern
    uint8_t idx;                    // array index
    char char_val;                  // char version of letter
    char letters[] = {'*','+','E','T','I','A','N','M','S','U',
                      'R','W','D','K','G','O','H','V','F','-','L',
                      '/','P','J','B','X','C','Y','Z','Q','&','#'};
                                    // modified letter array
    old_read_status = 0;
    idx = 0;
    // clear the letter array
    for(i = 0; i < 4; i++){
        letter_ary[i] = NOVAL;
    }

    // attach ISR and start ticker
    Tk_1.attach(&tick_isr, TICKER_PERIOD);
```

```
    // continuously check the status and provide output
    while(1){
        if(read_status != old_read_status){
            switch(read_status){
                case ACTIVE:
                    // no idx change
                    break;
                case DOT:
                    // store DOT and increment index
                    letter_ary[idx] = DOT;
                    idx++;
                    break;
                case DASH:
                    // store DASH and increment index
                    letter_ary[idx] = DASH;
                    idx++;
                    break;
                case BRK:
                    // no idx change
                    break;
                case LETTER:
                    // convert and output the letter
                    char_val = output_letter(letter_ary, letters);
                    printf("%c", char_val);
                    idx = 0;
                    break;
                case WORD:
                    // convert and output the letter + space
                    char_val = output_letter(letter_ary, letters);
                    printf("%c ", char_val);
                    idx = 0;
                    break;
                default:
                    idx = 0;
                    break;
            }// end switch
            if(idx > 3)                 //out of bounds check
                idx = 0;
        } else {
            // Nothing to do
        }// end if

        // update the 'old' value
        old_read_status = read_status;

        // mange loop frequency - 40Hz Baud rate
        wait_us(1000);                  // 1 ms loop
    }// end while

    return 0;
}// end main
```

Whole Class Project - Programming

- Helper Functions

```
uint8_t modify_format(uint8_t array[]){
    ////////////////////
    // this function converts from dot/dash format to a single
    // binary value with a leading 1
    // dot(0) -> 10,    dot dash(01) -> 1 0 1,    dash dash dot dash(1101) -> 1 1 1 0 1
    //
    // it then clears the array by filling with NOVAL (9)
    ////////////////////

    uint8_t i;
    uint8_t bin_value;
    bin_value = 1;                          // initialize leading 1

    // do the conversion
    for(i = 0; i < 4; i++){
        switch(array[i]){
            case DOT:                       // shift with 0 fill
                bin_value = bin_value << 1;
                break;
            case DASH:                      // shift with 1 fill
                bin_value = bin_value << 1;
                bin_value++;
                break;
            case NOVAL:
                break;
            default:
                break;
        } //end switch
    }// end for

    // clear the array
    for(i = 0; i < 4; i++){
        array[i] = NOVAL;
    }

    return bin_value;
}// end modify_format
```

```
char output_letter(uint8_t input_array[], char ref_array[]){
    uint8_t bin_val;                // modified binary version of letter
    char char_val;                  // char version of letter

    // modify the format to add a leading 1
    bin_val = modify_format(input_array);

    // convert the binary to a character using the special array
    char_val = bin_to_ascii(bin_val, ref_array);

    // output to the sseg
    display_sseg(&Sseg, char_val);

    // note: the console display is seperate

    return char_val;
}// end output_letter
```

Whole Class Project - Programming

- Helper Functions

```
char bin_to_ascii(uint8_t binval, const char ary[]){
    //////////////////////////
    // this function uses the modified binary value
    // to convert to ASCII using a pre-defined
    // ASCII array organized to match the modified
    // binary input
    //
    // dot is E which is encoded as 10(modified) so E is located at index 2
    // dash dot dash is R which is encoded as 1101(modified) so R is
    // located at index 13
    //////////////////////////
    char charval;
    charval = ary[binval];

    return charval;
}// end bin_to_ascii
```

```
void print_ary(const uint8_t ary[]){
    // convenience function
    uint8_t i;
    for(i = 0; i < 4; i++){
       printf("%i ", ary[i]);
   }
   printf("\n");
   return;
}
```

Whole Class Project - Programming

- Ticker ISR

```
void tick_isr(void){
    //////////////////
    // ISR to cause a periodic ADC read
    //
    // Must be run in Bare Metal mode due to MUTEX issues
    // in an ISR (the ADC read)
    //////////////////

    // local variables
    float sensor_val;                   // 0 - 1
    uint8_t read_val = 0;               // 0 or 1
    static uint8_t read_val_old = 0;
    static uint8_t num_ticks = 0;

    // read and evaluate the sensor
    sensor_val = Photocell.read();
    if(sensor_val > THRESHOLD)
        read_val = 0;
    else
        read_val = 1;
```

```
    //////////////////////////////////////
    //
    // In a normal system the ADC read would be used to
    // generate a second interrupt and do this processing
    // but mbed does not support ADC interrupts
    //
    //////////////////////////////////////

    // determine the 'value' of the waveform
    if(read_val != read_val_old){       // transition
        if(read_val_old == 0){
            if(num_ticks == WORD_TICKS){
                read_status = WORD;
            } else if(num_ticks == LETTER_TICKS){
                read_status = LETTER;
            } else if(num_ticks == BREAK_TICKS){
                read_status = BREAK;
            } else {
                read_status = ACTIVE;
            }// end if
        } else {                        // read_val_old = 1
            if(num_ticks == DASH_TICKS){
                read_status = DASH;
            } else if(num_ticks == DOT_TICKS){
                read_status = DOT;
            } else {
                read_status = ERROR;
            }// end if
        }// end if
        num_ticks = 1;
    } else {                            // no transition
        read_status = ACTIVE;
        num_ticks++;
    }// end if

    // update the "old" value
    read_val_old = read_val;

    return;
}// end tick_isr
```

Whole Class Project - Programming

- SSEG

```
void display_sseg(char value){
    // Seven Segment Alphabet
    // uses active low led segments (common Anode)
    switch(value){
        case 'A': Sseg.write(0x08); break;      // 0 0001000
        case 'B': Sseg.write(0x03); break;      // 0 0000011
        case 'C': Sseg.write(0x46); break;      // 0 1000110
        case 'D': Sseg.write(0x21); break;      // 0 0100001
        case 'E': Sseg.write(0x06); break;      // 0 0000110
        case 'F': Sseg.write(0x0E); break;      // 0 0001110
        case 'G': Sseg.write(0x20); break;      // 0 0010000
        case 'H': Sseg.write(0x09); break;      // 0 0001001
        case 'I': Sseg.write(0x4F); break;      // 0 1001111
        case 'J': Sseg.write(0x61); break;      // 0 1100001
        case 'K': Sseg.write(0x0D); break;      // 0 0001101
        case 'L': Sseg.write(0x47); break;      // 0 1000111
        case 'M': Sseg.write(0x6A); break;      // 0 1101010
        case 'N': Sseg.write(0x2B); break;      // 0 0101011
        case 'O': Sseg.write(0x40); break;      // 0 1000000
        case 'P': Sseg.write(0x0C); break;      // 0 0001100
        case 'Q': Sseg.write(0x18); break;      // 0 0011000
        case 'R': Sseg.write(0x2F); break;      // 0 0101111
        case 'S': Sseg.write(0x12); break;      // 0 0010010
        case 'T': Sseg.write(0x07); break;      // 0 0000111
        case 'U': Sseg.write(0x41); break;      // 0 1000001
        case 'V': Sseg.write(0x63); break;      // 0 1100011
        case 'W': Sseg.write(0x55); break;      // 0 1010101
        case 'X': Sseg.write(0x49); break;      // 0 1001001
        case 'Y': Sseg.write(0x11); break;      // 0 0010001
        case 'Z': Sseg.write(0x24); break;      // 0 0100100

        default: Sseg.write(0x3F); break;       // 0 0111111
    }// end switch

    return;
}// end display_sseg
```