Contents

1 Introduction

2 Abstract

3 Disaggregation Algorithm

3.1 Learning New Devices

3.2 Detecting Changes in Steady State

3.3 Device Checking

4 Graphical User Interface

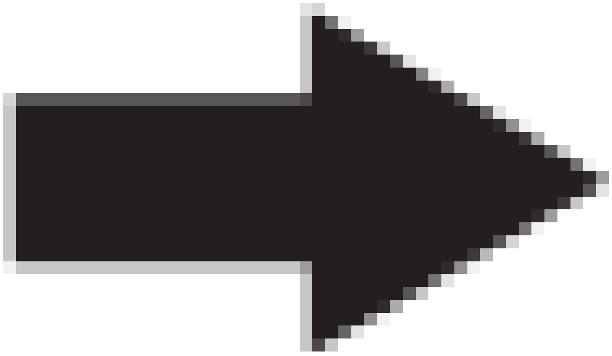

Figure 1: *Phase-correct PWM generation simulation*

4.1 Windows

4.2 Layouts

4.3 Animations

4.4 Graphs

5 Web Server

5.1 Storing Data as XML

5.2 JavaScript User Interface

6 Discussion

6.1 Problems

6.2 Further Enhancements

7 Conclusion

A C-code

A.1 gui.h

```c
#include "includes.h"
#include <string.h>

#define PICTUREWINDOW 0
#define RECTANGLEWINDOW 1
#define PROGRESSBAR 2
#define GRAPH 3

#ifndef __GUI_H
#define __GUI_H

extern FontType_t Terminal_9_12_6;

typedef struct{
  char type;
  short left, right, top, bottom;
  char clickable;
  void (*onClick)();
} Window;

typedef struct{
  char type;
  short left, right, top, bottom;
  char clickable;
  void (*onClick)();
  Bmp_t * picture;
} PictureWindow;

typedef struct{
  char type;
  short left, right, top, bottom;
  char clickable;
  char hidden;
  void (*onClick)();
  int backgroundColor, borderColor;
  char * text;
} RectangleWindow;

PictureWindow * GUI_initPictureWindow(int left, int top, int right, int bottom, Bmp_t * pic);
RectangleWindow * GUI_initRectangleWindow(int left, int top, int right, int bottom, int color, int bordercolor);
void GUI_setText(void * window, char * text);
void GUI_setOnClick(void * window, void (*function)());
void GUI_drawWindow(void * window);
void GUI_setHidden(void * window, char hide);
char GUI_onTouch(void * window, int x, int y);

#endif // __GUI_H
```

A.2 gui.c

```c
#include "gui.h"
#include "includes.h"

```

```
PictureWindow * GUI_initPictureWindow(int left, int top, int right, int bottom, Bmp_t * pic){
  // Initializes a window specified by absolute coordinates and a bitmap
  PictureWindow * temp;
  temp = (PictureWindow*)malloc(sizeof(*temp));
  temp->left = left;
  temp->right = right;
  temp->top = top;
  temp->bottom = bottom;
  temp->picture = (Bmp_t*) pic;
  temp->type = 0;

  temp->clickable = 0;
  return temp;
}

RectangleWindow * GUI_initRectangleWindow(int left, int top, int right, int bottom, int backgroundColor, int borderColor){
  // Initializes a window specified by absolute coordinates, a background- and a bordercolor
  RectangleWindow * temp;
  temp = (RectangleWindow*)malloc(sizeof(*temp));
  temp->left = left;
  temp->right = right;
  temp->top = top;
  temp->bottom = bottom;
  temp->type = 1;
  temp->text = "\0";

  temp->backgroundColor = backgroundColor;
  temp->borderColor = borderColor;

  temp->clickable = 0;
  temp->hidden = 0;
  return temp;
}

void GUI_setText(void * window, char * text){
  // Sets the text to Draw_ within a given window
  ((RectangleWindow*)window)->text = text;
}

void GUI_setOnClick(void * window, void (*function)()){
  // Sets a function to be called when this window is clicked
  ((Window*)window)->onClick = function;

  // Setting an Onclick will make the Window clickable
  ((Window*)window)->clickable = 1;
}

void GUI_drawWindow(void * window){
  // Draw the window based on its type. The type is contained within the first
  // byte of all the window structs
  Window * temp = (Window*)window;

  RectangleWindow * tempRect;

  switch (temp->type){
  case PICTUREWINDOW:
  // This is a picturewindow, Draw_ the bitmap using the GLCD command
  GLCD_LoadPic(temp->left,temp->top, ((PictureWindow*)temp)->picture, 0);
  break;
  case RECTANGLEWINDOW:
  // This is a rectanglewindow, Draw_ a rectangle using the user made graphics lib
  tempRect = (RectangleWindow*)window;

  // Should this window be Draw_n?
  if (tempRect->hidden == 1) return;

  Draw_FilledRectangle(tempRect->left, tempRect->top, tempRect->right - tempRect->left,
          tempRect->bottom - tempRect->top, tempRect->backgroundColor, tempRect->borderColor, 1);

  // Draw the text by computing the coordinates required to position the text in the
  // center of the window
  if (tempRect->text != "\0"){
    // x-position can be found as the middle of the window minus half the size of the string(in pixels)
    int xpos = (tempRect->right-tempRect->left-Terminal_9_12_6.H_Size*strlen(tempRect->text))/2;

    // y-position can be found as the middle of the window minues half the size of the font(in pixels)
    int ypos = (tempRect->bottom-tempRect->top-Terminal_9_12_6.V_Size)/2;

    // Set the window position
    GLCD_SetWindow(tempRect->left+xpos, tempRect->top+ypos,
          tempRect->left+xpos*2+Terminal_9_12_6.H_Size*strlen(tempRect->text),
          tempRect->top+ypos*2+Terminal_9_12_6.V_Size);

    // Draw the text that has been set
    GLCD_TextSetPos(0, 0);
    GLCD_SetFont(&Terminal_9_12_6, 0xFFFFFF, tempRect->backgroundColor);
    GLCD_print("%s", tempRect->text);
  }
  break;
  case PROGRESSBAR:
  // This is a progressbar, Draw_ it using the specifically designed progressbar method
  ProgressBar_DrawFull((ProgressBar*)window);
  break;
  case GRAPH:
  // This is a graph, Draw_ it using the specifically designed graph method
  Graph_draw((Graph*)window);
  break;
  }
}

void GUI_setHidden(void * window, char hide){
  // This only applies to rectanglewindows sofar
  if (((Window*)window)->type != RECTANGLEWINDOW) return;

  RectangleWindow * tempRect = (RectangleWindow*)window;

  // Hide or show?
  tempRect->hidden = hide;

  if (hide){
```

```
  Draw_FilledRectangle(tempRect->left, tempRect->top, tempRect->right - tempRect->left,
            tempRect->bottom - tempRect->top, 0x0, 0x0, 1);
  }
  else{
  GUI_drawWindow(window);
  }
}

char GUI_onTouch(void * window, int x, int y){
  // Checks wether the touch is within the bounds of this particular window
  Window * temp = (Window*)window;

  // Only clickable windows are eligible for receiving touches
  if (!temp->clickable) return 0;

  // Handle click
  if (temp->left <= x && x <= temp->right &&
    temp->top <= y && y <= temp->bottom){
    // Call onClick function if one has been set
    if (temp->onClick != NULL){
      temp->onClick();
    }
    return 1;
    }

  return 0;
}



```

A.3 layout.h

```
#include "includes.h"

#ifndef __LAYOUT_H
#define __LAYOUT_H

typedef struct{
  char size;
  Window * windows[10];
} Layout;

Layout * Layout_initLayout();
void Layout_addWindow(Layout * layout, void * window);
void * Layout_getWindow(Layout * layout, char pos);
void Layout_removeWindow(Layout * layout);
void Layout_drawWindows(Layout * layout);
char Layout_dispatchTouch(Layout * layout, int x, int y);

#endif // __LAYOUT_H
```

A.4 layout.c

```
#include "layout.h"

```

```
Layout * Layout_initLayout(){
    // Initialize a layout. This is basically a struct holding pointers to a
    // number of windows
  Layout * temp;
  temp = (Layout*)malloc(sizeof(*temp));

  temp->size = 0;
  return temp;
}

void Layout_addWindow(Layout * layout, void * window){
  // Add a window to the given layout
  layout->windows[layout->size] = (Window*)window;
  layout->size++;
}

void Layout_removeWindow(Layout * layout){
  // Pop the topmost window from the layout
  layout->windows[layout->size] = NULL;
  layout->size--;
}

void * Layout_getWindow(Layout * layout, char pos){
  // Returns the window at a given position in the layout
  return layout->windows[pos];
}

void Layout_drawWindows(Layout * layout){
  // Calls the appropriate Draw_ing methods, hereby Draw_ing all windows within
  // the given layout
  for(int i=0; i<layout->size; i++){
  GUI_drawWindow(layout->windows[i]);
  }
}

char Layout_dispatchTouch(Layout * layout, int x, int y){
  // Run onTouch checks for all windows in the specified layout. A successfull
  // reading returns 1, if no windows accept the touch, 0 is returned
  for(int i=0; i<layout->size; i++){
  if (GUI_onTouch(layout->windows[i], x, y)){
    return 1;
  }
  }
  return 0;
}
```

A.5 draw.h

```
#ifndef DRAW_H_
#define DRAW_H_

#include "drv_glcd.h"

void Draw_VerticalLine(int x0, int y0, int length, int lineColor);
void Draw_HorizontalLine(int x0, int y0, int length, int lineColor);
void Draw_Line(int x0, int y0, int x1, int y1, int lineColor);

void Draw_Circle(int x0, int y0, int radius, int borderColor);
void Draw_FilledCircle(int x0, int y0, int radius,int backgroundColor,
            int borderColor, int Draw_Border);

void plot8points(int x0, int y0, int x, int y, int borderColor);
void plot4points(int x0, int y0, int x, int y, int borderColor);


void Draw_Rectangle(int x0, int y0, int width, int height, int borderColor);
void Draw_FilledRectangle(int x0, int y0, int width, int height,
            int backgroundColor, int borderColor, int Draw_Border);

#endif
```

A.6 draw.c

```
#include "includes.h"
#include <math.h>

void Draw_HorizontalLine(int x0, int y0, int length, int lineColor) {
  for (int i = 0; i < length; i++) {
    DRAW_PIXEL(i + x0, y0, lineColor);
  }
}

void Draw_VerticalLine(int x0, int y0, int length, int lineColor){
  for (int i = 0; i < length; i++) {
    DRAW_PIXEL(x0, y0 + i, lineColor);
  }
}


// Implemented with Bresenham's algorithm (taken from the site
// http://rosettacode.org/wiki/Bitmap/Bresenham's_line_algorithm
void Draw_Line(int x0, int y0, int x1, int y1, int lineColor) {
  int dx = abs(x1-x0), sx = x0<x1 ? 1 : -1;
  int dy = abs(y1-y0), sy = y0<y1 ? 1 : -1;
  int err = (dx>dy ? dx : -dy)/2, e2;

  for(;;){
    DRAW_PIXEL(x0, y0, lineColor);
    if (x0==x1 && y0==y1) break;
    e2 = err;
    if (e2 >-dx) { err -= dy; x0 += sx; }
    if (e2 < dy) { err += dx; y0 += sy; }
  }
}


// Draw circle using Bresenham's midpoint circle algorithm
// From http://en.wikipedia.org/wiki/Midpoint_circle_algorithm
void Draw_Circle(int x0, int y0, int radius, int borderColor) {
  int error = -radius;
  int x = radius;
  int y = 0;

  while (x >= y) {
    plot8points(x0, y0, x, y, borderColor);
    error += y;
    ++y;
    error += y;
    if (error >= 0) {
      error -= x;
      --x;
      error -= x;
    }
  }
}

void plot8points(int x0, int y0, int x, int y, int borderColor) {
  plot4points(x0, y0, x, y, borderColor);
  if (x != y) plot4points(x0, y0, y, x, borderColor);
}

void plot4points(int x0, int y0, int x, int y, int borderColor) {
  DRAW_PIXEL(x0 + x, y0 + y, borderColor);
  if (x != 0) DRAW_PIXEL(x0 - x, y0 + y, borderColor);
  if (y != 0) DRAW_PIXEL(x0 + x, y0 - y, borderColor);
  if (x != 0 && y != 0) DRAW_PIXEL(x0 - x, y0 - y, borderColor);
}



/*


// Circle rasterization algorithm (modified from version from the following site)
// http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture6/circle.html
void Draw_Circle(int x0, int y0, int radius, int borderColor){
  int x, y, r2;

  r2 = radius * radius;
  for (x = -radius; x <= radius; x++) {
    y = (int) (sqrt(r2 - x*x) + 0.5);
    DRAW_PIXEL(x0 + x, y0 + y, borderColor);
    DRAW_PIXEL(x0 + x, y0 - y, borderColor);
  }
}

*/

void Draw_FilledCircle(int x0, int y0, int r, int backgroundColor, int borderColor, int Draw_Border) {

  for (int x = -r; x <= r; x++) {
    int dy = (int)(sqrt(r*r - x*x));
    for (int y = -dy; y <= dy; y++) {
      DRAW_PIXEL(x0+x, y0+y, backgroundColor);
    }
  }

  if (Draw_Border) Draw_Circle(x0,y0,r,borderColor);
}


void Draw_Rectangle(int x0, int y0, int width, int height, int borderColor) {
  Draw_HorizontalLine(x0, y0, width, borderColor);
  Draw_HorizontalLine(x0, y0 + height - 1, width, borderColor);
  Draw_VerticalLine(x0, y0, height, borderColor);
  Draw_VerticalLine(x0 + width - 1, y0, height, borderColor);
}
```

```

void Draw_FilledRectangle(int x0, int y0, int width, int height,
            int backgroundColor, int borderColor, int Draw_Border) {

  // Draw background
  for (int x = 0; x < width; x++) {
    for (int y = 0; y < height; y++) {
      DRAW_PIXEL(x0 + x, y0 + y, backgroundColor);
    }
  }

  // Draw border
  if (Draw_Border) Draw_Rectangle(x0, y0, width, height, borderColor);
}

```

A.7 animation.h

```
#include <inttypes.h>
#include "includes.h"
#include "graphics/ProgressBar.h"

#ifndef __ANIMATION_H
#define __ANIMATION_H


typedef struct{
    short increment;
  short value;
  void * object;
  char (*animate)(void *, int);
} Animation;


void Animation_init();
void Animation_post(void * object, int increment, int value, char (*animate)(
    void *, int));
void Timer2IntrHandler(void);
char Animation_isRunning();

#endif
```

A.8 animation.c

```
#include "includes.h"
#include "graphics/ProgressBar.h"

static Animation * animationHolder;
static char mAnimating = 0;

// All animations run on a 20ms interrupt-based routine
void Animation_init(){
  // enable clock for this peripheral (timer2)
  PCONP_bit.PCTIM2 = 1;

  // Set prescaler to 4
  PCLKSEL1_bit.PCLK_TIMER2 = 0;

  // Set interrupt flag on compare-match with MR0
  T2MCR_bit.MR0I = 1;

  // Reset MR0 on compare-match
  T2MCR_bit.MR0R = 1;

  // This value yields a 20ms interrupt time
  T2MR0 = 0x57E40;

  // Init timer 2 interrupt
  T2IR_bit.MR0INT = 1;
  VIC_SetVectoredIRQ(Timer2IntrHandler,0,VIC_TIMER2);
  VICINTENABLE |= 1UL << VIC_TIMER2;

  // Allocate memory for the animationHolder
  animationHolder = (Animation*)malloc(sizeof(*animationHolder));
  mAnimating = 0;
}

void Animation_post(void * object, int increment, int value, char (*animatecall)(void *, int)){
  // Posts an animation to be run using timer2. This call requires a starting value and a value
  // that is used as increment between frames. The object that is to be animated must itself
  // implement an update-function that returns 1 if the animation is finished, and 0 if the animation
  // is not yet done. This update-function must also be passed into this function (animatecall)
  if (!mAnimating){
  animationHolder->animate = animatecall;
  animationHolder->increment = increment;
  animationHolder->value = value;
  animationHolder->object = object;

  // Start animating
  mAnimating = 1;
  // Start the timer (enable counting)
  T2TCR_bit.CE = 1;
  }
}

void Timer2IntrHandler(void){
  // Update current animation if applicable
  if (mAnimating){
  // Add increment to current value
  animationHolder->value += animationHolder->increment;
  if (animationHolder->animate(animationHolder->object, animationHolder->value)){
    // The update-function returned 1 - the animation is done. Remove it
    mAnimating = 0;

    // Stop the timer (disable counting)
    T2TCR_bit.CE = 0;
  }
  }

  // clear interrupt
  T2IR_bit.MR0INT = 1;
  VICADDRESS = 0;
```

```
}

char Animation_isRunning(){
  // Returns animation state
  return mAnimating;
}
```

A.9 parsing.h

```
#include <inttypes.h>

#ifndef __PARSING_H
#define __PARSING_H

#define VOLTAGE_GAIN 95.929
#define CURRENT_GAIN 6.7242
#define P_POWER_GAIN 0.001244
#define Q_POWER_GAIN 0.001
#define H_POWER_GAIN 0.01

#define VOLTAGE_PARSE_OFFSET 1
#define CURRENT_PARSE_OFFSET 9
#define P_POWER_PARSE_OFFSET 17
#define Q_POWER_PARSE_OFFSET 25
#define H_POWER_PARSE_OFFSET 33

typedef struct{
  double voltage;
  double current;
  double P_power;
  double Q_power;
  double H_power;
} Measurement;

void Parsing_parse(char * Buffer, Measurement * measurement);
static double convertVoltage(int Vrms);
static double convertCurrent(int Irms);
static double convertP_power(int P_power);
static double convertQ_power(int Q_power);
static double convertH_power(int H_power);
#endif
```

A.10 parsing.c

```
#include "parsing.h"
#include <stdlib.h>
#include <math.h>

void Parsing_parse(char * Buffer, Measurement * measurement){
  // All parameters are received as hexidecimal strings. These are converted
  // to doubles using the strtol function and multiplying by factors obtained through
  // measurements.

  // Holder for the hexidecimal string
  char temp[9];

  // Get voltage
  for (int j=0; j<9; j++){
  temp[j] = Buffer[j+VOLTAGE_PARSE_OFFSET];
  }
  temp[8] = '\0';

  // Convert from hex to double
  int voltage = strtol(temp, NULL, 16);
  measurement->voltage = convertVoltage(voltage);

  // Compute normalizing factor
  double scale = pow(measurement->voltage / 235.0, 2);

  // Get current
  for (int j=0; j<9; j++){
  temp[j] = Buffer[j+CURRENT_PARSE_OFFSET];
  }
  temp[8] = '\0';

  // Convert from hex to double
  int current = strtol(temp, NULL, 16);
  measurement->current = convertCurrent(current);

  // Get P Power
  for (int j=0; j<9; j++){
  temp[j] = Buffer[j+P_POWER_PARSE_OFFSET];
  }
  temp[8] = '\0';

  // Convert from hex to double and normalize
  int power = strtol(temp, NULL, 16);
  double pACT = convertP_power(power);
  measurement->P_power = scale * pACT;

  // Get Reactive Power
  for (int j=0; j<9; j++){
  temp[j] = Buffer[j+Q_POWER_PARSE_OFFSET];
  }
  temp[8] = '\0';

  // Convert from hex to double and normalize
  power = strtol(temp, NULL, 16);
  double pREAC = convertQ_power(power);
  measurement->Q_power = scale * pREAC;

  // Get Harmonic Power
  for (int j=0; j<9; j++){
  temp[j] = Buffer[j+H_POWER_PARSE_OFFSET];
  }
  temp[8] = '\0';

  // Convert from hex to double and normalize
  power = strtol(temp, NULL, 16);
  double pHAR = convertH_power(power);
  measurement->H_power = scale * pHAR;
}

static double convertVoltage(int Vrms){
  // Applies the appropriate gain to obtain a proper reading
  return Vrms * VOLTAGE_GAIN * pow(2,-22);
}
```

```c

static double convertCurrent(int Irms) {
  // Applies the appropriate gain to obtain a proper reading
  return Irms * CURRENT_GAIN * pow(2,-22);
}

static double convertP_power(int P_power){
  // Converts from two's-complement and applies the appropriate gain to obtain a proper reading
  if (P_power >= 0x800000) {
  P_power ^= 0xFFFFFF;
  P_power += 1;
  }
  if (P_power < 0) P_power = 0;

  return P_power * P_POWER_GAIN;
}

static double convertQ_power(int Q_power){
  // Converts from two's-complement and applies the appropriate gain to obtain a proper reading
  if (Q_power >= 0x800000) {
  Q_power ^= 0xFFFFFF;
  Q_power += 1;
  }

  return Q_power * Q_POWER_GAIN;
}

static double convertH_power(int H_power){
  // Converts from two's-complement and applies the appropriate gain to obtain a proper reading
  if (H_power >= 0x800000) {
  H_power ^= 0xFFFFFF;
  H_power += 1;
  }

  // Normalize and invert
  double H_power_d = - H_power*H_POWER_GAIN + 6.0;
  if (H_power_d < 0) H_power_d = 0;

  return H_power_d;
}
```

A.11 rtc.h

```c
#include "includes.h"

#ifndef __RTC_H
#define __RTC_H

void RTC_init();

char RTC_getSeconds();
char RTC_getMinutes();
char RTC_getHours();
void RTC_setTime(char seconds, char minutes, char hours);

#endif // __RTC_H
```

A.12 rtc.c

```c
#include "includes.h"


void RTC_init(){
  // Enable Clock for the RTC
  PCONP_bit.PCRTC = 1;

  // Use 32kHz internal oscillator
  CCR = 0;
  CCR_bit.CLKSRC = 1;

  // Disable ALL interrupts
  CIIR = 0;
  CISS = 0;

  // Reset current time
  RTC_setTime(0,0,0);

  // Enable counting
  CCR_bit.CLKEN = 1;
}

char RTC_getSeconds(){
  // Return bits 0-5 of CTIME0 register
  return SEC;
}

char RTC_getMinutes(){
  // Return bits 8-13 of CTIME0 register
  return MIN;
}

char RTC_getHours(){
  // Return bits 16-20 of CTIME0 register
  return HOUR;
}

void RTC_setTime(char hours, char minutes, char seconds){
  HOUR = hours;
  MIN = minutes;
  SEC = seconds;
}
```

A.13 xml.h

```c
#include "includes.h"

#ifndef __XML_H
#define __XML_H


#define XML_DATA_SIZE 3650
#define XML_DEVICES_SIZE 300
#define XML_ENTRY_SIZE 15
#define TAG_START_LENGTH 3
#define TAG_END_LENGTH 4

```

```
#define XML_DATA 0
#define XML_DEVICES 1

static void XML_copyString(const char * string, char * array, int * length);
static void XML_startTag(char id, char * array, int * length);
static void XML_endTag(char id, char * array, int * length);
static void XML_addDouble(double value, char * array, int * length);
static void XML_addNode(double value, char id, char * array, int * length);
static void XML_addTime();
static void XML_addStates(char deviceStates);
void XML_addMeasurement(Measurement * m, char deviceStates);
void XML_addDevice(char * name);
void XML_clearDevices();
char * XML_getContent(char which);
char XML_getLength(char which);

#endif // __XML_H
```

A.14 xml.c

```
#include "includes.h"
#include "xml.h"


static char XML_Data[XML_DATA_SIZE];
static int XML_Data_Length = 0;

static char XML_Devices[XML_DEVICES_SIZE];
static int XML_Devices_Length = 0;

const char * xml_header = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";

// Copy a string into current position in xml buffer
static void XML_copyString(const char * string, char * array, int * length){
  while (*string != '\0') {
  array[(*length)++] = *(string++);
  }
}

// Add start tag of an xml node
static void XML_startTag(char id, char * array, int * length){
  array[(*length)++] = '<';
  array[(*length)++] = id;
  array[(*length)++] = '>';
}

// Add end tag of an xml node
static void XML_endTag(char id, char * array, int * length){
  array[(*length)++] = '<';
  array[(*length)++] = '/';
  array[(*length)++] = id;
  array[(*length)++] = '>';
}

// Convert double to string and add to buffer
static void XML_addDouble(double value, char * array, int * length){
  char temp[8];

  // handle sign
  if (value < 0) {
  temp[0] = '-';
  value *= -1;
  } else {
  temp[0] = ' ';
  }

  // Convert to string
  int integers = (int)value;
  int decimals = (int)((value-integers)*100);
  snprintf(temp, 8, "%03d.%02d", integers, decimals);

  // Add to xml_buffer
  XML_copyString(temp, array, length);
}

// Add single child node to the xml file
static void XML_addNode(double value, char id, char * array, int * length){
  XML_startTag(id, array, length);
  XML_addDouble(value, array, length);
  XML_endTag(id, array, length);
}

static void XML_addTime(){
  XML_startTag('T', XML_Data, &XML_Data_Length);

  // Get current time
  char seconds = RTC_getSeconds();
  char minutes = RTC_getMinutes();
  char hours = RTC_getHours();

  // Format as HHMMSS and add to data
  char temp[7];
  snprintf(temp, 7, "%02d%02d%02d", hours, minutes, seconds);
  XML_copyString(temp, XML_Data, &XML_Data_Length);

   XML_endTag('T', XML_Data, &XML_Data_Length);
}

static void XML_addStates(char deviceStates){
  XML_startTag('S', XML_Data, &XML_Data_Length);
  XML_Data[XML_Data_Length++] = deviceStates+0x30;
  XML_endTag('S', XML_Data, &XML_Data_Length);
}


// Add one measurement to the xml file
void XML_addMeasurement(Measurement * m, char deviceStates) {

  // Check for free space in the xml file
  if (XML_Data_Length + XML_ENTRY_SIZE >= XML_DATA_SIZE) return;

  // If this is first measurement to be added, add xml header and root element
  if (XML_Data_Length == 0) {
  XML_copyString(xml_header, XML_Data, &XML_Data_Length);
  XML_startTag('D', XML_Data, &XML_Data_Length);
  } else {
  // If this is not the first element to be added,
  // overwrite the end tag of the root element
  XML_Data_Length -= TAG_END_LENGTH;
  }
```

```c

  // Add measurement nodes
  XML_startTag('M', XML_Data, &XML_Data_Length);
  XML_addNode(m->voltage, 'V', XML_Data, &XML_Data_Length);
  XML_addNode(m->current, 'I', XML_Data, &XML_Data_Length);
  XML_addNode(m->P_power, 'P', XML_Data, &XML_Data_Length);
  XML_addNode(m->Q_power, 'Q', XML_Data, &XML_Data_Length);
  XML_addNode(m->H_power, 'H', XML_Data, &XML_Data_Length);
  XML_addTime();
  XML_addStates(deviceStates);
  XML_endTag('M', XML_Data, &XML_Data_Length);

  // End root element
  XML_endTag('D', XML_Data, &XML_Data_Length);
}

// Add a new device to the XML file
void XML_addDevice(char * name){

  // If this is first measurement to be added, add xml header and root element
  if (XML_Devices_Length == 0) {
  XML_copyString(xml_header, XML_Devices, &XML_Devices_Length);
  XML_startTag('D', XML_Devices, &XML_Devices_Length);
  } else {
  // If this is not the first element to be added,
  // overwrite the end tag of the root element
  XML_Devices_Length -= TAG_END_LENGTH;
  }

  // Name is always 5 bytes long
  XML_startTag('N', XML_Devices, &XML_Devices_Length);
  XML_copyString(name, XML_Devices, &XML_Devices_Length);
  XML_endTag('N', XML_Devices, &XML_Devices_Length);

  // End root element
  XML_endTag('D', XML_Devices, &XML_Devices_Length);
}

void XML_clearDevices(){
  // Reset XML file for devices
  XML_Devices_Length = 0;
}

char * XML_getContent(char which){
  if (which == XML_DATA) return XML_Data;
  if (which == XML_DEVICES) return XML_Devices;

  // Unknown
  return NULL;
}

char XML_getLength(char which){
  if (which == XML_DATA) return XML_Data_Length;
  if (which == XML_DEVICES) return XML_Devices_Length;

  // Unknown
  return 0;
}
```

A.15 uart.h

```
#include "includes.h"

#ifndef __UART_H
#define __UART_H

#define UART_MAX_BAUD_RATE     256000
#define BUFFER_SIZE 42

// Define UARTs
typedef enum _UartNum_t
{
  UART_0 = 0, UART_1, UART_2, UART_3,
} UartNum_t;

typedef enum _UartMode_t
{
  NORM = 0, IRDA
} UartMode_t;

typedef enum _UartParity_t
{
  UART_ODD_PARITY = 0, UART_EVEN_PARITY,
  UART_FORCE_1_PARITY, UART_FORCE_0_PARITY,
  UART_NO_PARITY
} UartParity_t;

typedef enum _UartStopBits_t
{
  UART_ONE_STOP_BIT = 0, UART_TWO_STOP_BIT,
} UartStopBits_t;

typedef enum _UartWordWidth_t
{
  UART_WORD_WIDTH_5 = 0, UART_WORD_WIDTH_6,
  UART_WORD_WIDTH_7, UART_WORD_WIDTH_8
} UartWordWidth_t;

typedef struct _UartLineCoding_t
{
  Int32U           dwDTERate;
  UartStopBits_t   bStopBitsFormat;
  UartParity_t     bParityType;
  UartWordWidth_t  bDataBits;
} UartLineCoding_t, * pUartLineCoding_t;

extern Int32U SYS_GetFpclk(Int32U Periphery);

void UART_Check(char * externUART_Buffer);
void Uart0Isr(void);
Boolean UART_init(UartNum_t Uart,Int32U IrqSlot, UartMode_t UartMode);
void UartSetLineCoding(UartNum_t Uart,UartLineCoding_t UartCoding);
void UartCalcDivider(Int32U Freq, Int32U Baud, pInt32U pDiv, pInt32U pAddDiv,
    pInt32U pMul);

#endif // __UART_H
```

A.16 uart.c

```
#define UART_GLOBAL
#include "uart.h"

static char UART_Buffer[BUFFER_SIZE];
static int iterator = 0;
char RxFlag = 0;

void UART_Check(char * externUART_Buffer){
    if (RxFlag){
    // Flag is high, copy the contents of the internal UART buffer to the main
    // Buffer
    for (int i=0; i<BUFFER_SIZE; i++){
        externUART_Buffer[i] = UART_Buffer[i];

        // Clear the internal UART buffer
        UART_Buffer[i] = 0;
    }
    RxFlag = 0;
    }
    else{
    // A full command has not been received yet.
    externUART_Buffer[0] = ’E’;
    }
}

static void Uart0Isr(void){
    //Int32U UartIntId = U0IIR, LineStatus, Counter;

    // RxFlag must be low and first character must equal
    // Z for the transmission to be valid
    if (!RxFlag){
    if (iterator == 0){
        if (U0RBR == ’Z’){
        // Starting character received
        UART_Buffer[iterator] = ’Z’;
        iterator = 1;
        }
    }
    else{
        // Transmission has been started by a valid character, just continue streaming
        UART_Buffer[iterator] = U0RBR;
        iterator++;

        // Check for carriage return. This signified the end of a transmission
        if (iterator >= BUFFER_SIZE || UART_Buffer[iterator-1] == ’\r’){
        iterator = 0;

        // Set the flag high to allow the buffer to be passed onto the main program
        RxFlag = 1;
        }
    }
    }
    else{
    // U0RBR must always be read
    if (U0RBR == 0);
    }
    VICADDRESS = 0;  // Clear interrupt in VIC.
}

```

```

Boolean UART_init(UartNum_t Uart,Int32U IrqSlot, UartMode_t UartMode){
  volatile Int8U Tmp;
  // Init buffer
  for (int i=0; i<BUFFER_SIZE; i++){
  UART_Buffer[i] = 0;
  }
  RxFlag = 0;

  // Enable UART0
  PCONP_bit.PCUART0 = 1;
  // Assign Port 0,1 to UART0
  // TX
  PINSEL0_bit.P0_2 = 1;
  // RX
  PINSEL0_bit.P0_3 = 1;

  U0LCR = 0x03; // Word Length =8, no parity , 1 stop
  U0FCR = 0x7;  // Enable and Clear the UART0 FIFO, Set RX FIFO interrupt level - 1 char
  // Transmit enable
  U0TER_bit.TXEN = 1;
  Tmp = U0IER;  // Clear pending interrupts
  // enable RBR Interrupt
  U0IER = 0x01;

  VIC_SetVectoredIRQ(Uart0Isr,IrqSlot,VIC_UART0);
  VICINTENABLE |= 1<<VIC_UART0;

  // Set up Line Coding
  UartLineCoding_t UartLineCoding;
  UartLineCoding.dwDTERate = 115200;  // Update the baud rate
  UartLineCoding.bStopBitsFormat = UART_ONE_STOP_BIT; // Update the stop bits number
  UartLineCoding.bParityType = UART_NO_PARITY;  // Update the parity type
  UartLineCoding.bDataBits = UART_WORD_WIDTH_8; // Update the word width
  UartSetLineCoding(UART_0, UartLineCoding); // Set UART line coding

  return(TRUE);
}

/*************************************************************************
* Function Name: UartSetLineCoding
* Parameters:  UartNum_t Uart, UartLineCoding_t pUartCoding
*
* Return: None
*
* Description: Init UART Baud rate, Word width, Stop bits, Parity type
*
*************************************************************************/
void UartSetLineCoding(UartNum_t Uart, UartLineCoding_t UartCoding)
{
  Int32U Mul, Div, AddDiv, Freq;

  // Check parameters
  if ((UartCoding.dwDTERate == 0) || (UartCoding.dwDTERate > UART_MAX_BAUD_RATE))
  {
    return;
  }
  Freq = SYS_GetFpclk(UART0_PCLK_OFFSET);
  UartCalcDivider(Freq, UartCoding.dwDTERate,&Div,&AddDiv,&Mul);
  U0LCR_bit.WLS = UartCoding.bDataBits;
  U0LCR_bit.SBS = UartCoding.bStopBitsFormat;
  U0LCR_bit.PE  =(UartCoding.bParityType == UART_NO_PARITY)?0:1;
  U0LCR_bit.PS  = UartCoding.bParityType;
  U0LCR_bit.DLAB = 1;
  U0DLL = Div & 0xFF;
  U0DLM = (Div >> 8) & 0xFF;
  U0FDR = AddDiv + (Mul << 4);
  U0LCR_bit.DLAB = 0;
}

static void UartCalcDivider(Int32U Freq, Int32U Baud,
              pInt32U pDiv, pInt32U pAddDiv, pInt32U pMul)
{
  Int32U Temp, Error = (Int32U)-1;
  Int32U K1, K2, K3, Baudrate;
  Int32U DivTemp, MulTemp, AddDivTemp;

  //
  for(MulTemp = 1; MulTemp < 16; ++MulTemp)
  {
    K1 = Freq*MulTemp;
    for(AddDivTemp = 1; AddDivTemp < 16; ++AddDivTemp)
    {
      K3 = (MulTemp + AddDivTemp)<<4;
      K2 =  K3 * Baud;
      DivTemp = K1/K2;
      // if DIVADDVAL>0, UnDL must be UnDL >= 0x0002 or the UART will
      // not operate at the desired baud-rate!
      if(DivTemp < 2)
      {
        continue;
      }
      Baudrate = DivTemp * K3;
      Baudrate = K1/Baudrate;
      Temp = (Baudrate > Baud)? \
    (Baudrate - Baud): \
      (Baud - Baudrate);
      if (Temp < Error)
      {
      Error = Temp;
      *pDiv = DivTemp;
      *pMul = MulTemp;
      *pAddDiv = AddDivTemp;
      if(Error == 0)
      {
        return;
      }
      }
    }
  }
}

```