Module Descriptions

Team Dirac

12/9/2014

Abstract

This document follows on from the Module Descriptions document, modified to run in a RTOS such as salvo. There is actually very little difference to the modules, except the Integration module. There are however some small changes such as in the temperature module we can now have a task to update the temp every five seconds or so (instead of reading it every time we measure the range), and there will be some changes to the ISR functions in the modules.
In effect the modules are designed to be task specific functionality that is unchanged by the way in which they are called. Thus almost all RTOS related 'stuff' is generally confined to the main file in the integration module.

1 Tracking Task:

1.1 Function:

This task will track or search for the target by calling the Tracking module functionality periodically.

1.2 Priority:

This task is central to the normal operation of the system, but does not require precise timing. Thus it will be a moderate priority.

1.3 Eligibility:

This task will delay for 10ms (or similar time) and will also wait the Ultrasonic sensor so that the servo's are not running at the same time as the ultrasonic sensor and drawing too much current. This task will also wait the range measurement to ensure that there is at least one range calculation at the new position.

2 Range Measurement:

2.1 Function:

This task will call functions in the range measuring function to measure the range of the target at regular intervals as described in the specifications.

2.2 Priority:

This task will have medium priority (slightly higher than the tracking task), as it is also required for normal operation, but the timing is required to be more precise as the user sets the measurement rate and expects this frequency.

2.3 Eligibility:

This task will have a delay to make it run at a given frequency, but the main eligibility will be whether the IR and ultrasonic measurements are complete. It will also wait the servos so that the ultrasonic sensor and servos are not used at the same time and draw too much current.

2.4 Additional Notes:

This task may very well be split into two tasks: oe for the IR sensing and one for the Ultrasonic, or at least call the IR and ultrasonic read functions from this task (instead of just having a range measurement public interface function in the range module). The reason for this is that Salvo does not have a stack and

therefore cannot handle delays and task yields within function calls. So if we want the range function to be able to wait the return of the echo signal, then that cannot be hidden in private module functions.

3 User Input Task:

3.1 Function:

This task simply waits any user input (which sets a binary semaphore via an interrupt), and then handles the specified input. This essentially just removes the computation from the ISR.

3.2 Eligibility:

This task will wait any external user input and pass that to the user interface module.

3.3 Priority:

This task will have high priority as the user expects any input they make to be handled

4 Temperature Task:

4.1 Function:

This task will simply read the temperature from the temperature sensor, and store the value in the temperature module. Then a call to get temperature will simply return this variable as the last temperature reading sampled.

4.2 Eligibility:

This task will delay for 5 seconds, and will also wait the ADC resource.

4.3 Priority:

Low, there will be very little change in temperature, which will have an equally small change in the speed of sound. Thus not only do we not have to measure it quickly, it also won't matter if it is delayed by higher priority tasks.

5 Serial receive Task:

5.1 Function:

This task is responsible for handling any user instructions that come over serial (e.g. really just the User input task over serial). This task is separate from the user input task as they wait different events. Otherwise they are really the same, and will call the same functions in the user interface module.

5.2 Eligibility:

This task waits the serial line and passes the received data to the user interface module.

5.3 Priority:

Again, like the user input task this task will have high priority (for the same reasons).

6 Serial transmit task:

6.1 Function:

This task will call the transmit function in the serial module, which places a string into the transmit buffer, which is then transmitted over serial via interrupts.

6.2 Eligibility:

This task waits a message which contains the data to transmit, and also a counting semaphore which indicates the number of free places in the buffer. This means that if the buffer is filled for whatever reason the task will simply wait until it is emptied. A queue of messages could also be used to ensure that part of a message is not lost if two messages are signalled quickly.

6.3 Priority:

This function will have medium priority.

7 Range Finding Module

7.1 Description:

This module mainly concerns the usage of the ultrasonic range finding sensor, and its dependencies. The source code for the Range Finding module will be contained in the file RangeFinding.c and the public header will be RangeFinding.h. The ultrasonic module may use the Common.h which will contain include files, definitions and macros common to the entire program, but should otherwise be completely contained in the RangeFinding.c file.

7.2 Public Interface:

This module will contain the following public access functions:

7.2.1 Configuration Description:

The configuration function will perform all necessary actions in order to configure the range finding module, including a delay period if necessary, such that after this function returns the range finding module is ready to begin full operation.

7.2.2 Configuration Prototype:

The configuration function will take no arguments and return nothing. It will be prototyped as so:

void configureRange(void);

7.2.3 Range Description:

The RangeFinding module will use both the IR and Ultrasonic sensors to calculate the (calibrated) range.

7.2.4 Range Prototype:

The Range function will take no arguments and return an unsigned int representing the measured distance in mm. The function will take the following prototype:

unsigned int range();

7.2.5 IR Range Description:

Calculates the range from the IR sensor.

7.2.6 IR Range Prototype:

The IR Range function will take no arguments and will return an unsigned int representing the measured distance in mm. The function will take the following prototype:

unsigned int IRrange();

7.2.7 Begin Ultrasonic Description:

The begin ultrasonic function will begin an ultrasonic scan

7.2.8 Begin Ultrasonic Prototype:

The begin ultrasonic function will take no arguments and will return nothing. It will use the following prototype:

void beginUS(void);

7.2.9 Ultrasonic Range Description:

Returns the range of the last ultrasonic scan. When the echo is returned it will be stored in a register, and this function just uses that value (which will be from the last scan) to calculate the distance. Note: This function does not include polling because it is designed to run in a RTOS system where it will wait for the echo to return.

7.2.10 Ultrasonic Range Prototype:

The ultrasonic range function will take no arguments and will return an unsigned int representing the measured distance in mm. The function will take the following prototype:

unsigned int USrange();

7.2.11 ISR Function:

The ISR function acts as an Interrupt Service Routine for the range Finding module. It handles any interrupt driven functions.

7.2.12 ISR prototype:

The ISR function takes no arguments and returns nothing. It will have the following prototype:

void rangeISR(void);

7.2.13 Calibration Function:

The calibration function will change a calibration variable in the rangefinding module that gets added to every range calculation. This will set the distance returned by the current target to the passed argument.

7.2.14 Calibration prototype:

The calibration function will take a single argument of the distance to set the measurement to as a signed int in mm. It will then return nothing. The function will use the following prototype:

void calibrateRange(signed int distance);

7.2.15 Raw Range:

This function returns the calibrated range.

7.2.16 Return Calibration offset prototype:

The return calibration offset function will take no arguments and will return a single signed int equal to the measured distance in mm. The function will use the following prototype:

signed int rawRange(void);

7.3 Functions:

In addition to the public interface functions described above, the ultrasonic module should also contain the following private functions:

- Output Capture interrupt to act as a timeout, resetting the measurement flag and clearing the CCP input register.
- Speed of sound Calculator

7.4 Hardware:

The ultrasonic module will make use of the following hardware:

- Polaroid 6500 series ultrasonic range finder sensor
- 5V power supply

7.5 Processor Resources:

- Input Capture module
- Timer0
- Output Compare (using interrupts)
- Analogue to Digital Converter
- 1 Digital output pin
- 1 Digital input pin
- 1 analogue input pin

8 User Interface Module

8.1 Description:

This module concerns the interface to the system, and how the operator will be able to use it. Any source code for this module will be contained in the User-Interface.c file, with the public interface in UserInterface.h. The module may use the Common.h header, but otherwise all functionality should be contained within the UserInterface.c file.

8.2 Public Interface:

Much of the functionality of the user interface module will be interrupt driven and thus not have any public access functions. Thus the user interface module will only have one public function to display the target information. There may be more function in the future depending on how the user interface evolves. It will also have a public ISR function which is called whenever a User Interface related Interrupt is thrown.

8.2.1 Configure Description:

Configures the interface for use, e.g. will set everything to their default values, and put the device in a 'startup mode'.

8.2.2 Configure Prototype:

The configure function will take no arguments, and will return nothing. It will use the following prototype:

void configUSER(void);

8.2.3 Display Description:

This function will display the current known target information. Depending on the current operation mode this may be on a computer through the UART, or on the LCD or both. If there is no target detected the function will still display the current tracking azimuth and inclination (as indications of where it is no) but will display "No Target" or "-" in lieu of a range, or something similar.

8.2.4 Display Prototype:

The display function will take a single argument of type TrackingData which is a typedef'ed struct defined in Common.h. It will return nothing. The prototype for this function will be:

void display(TrackingData data);

8.2.5 ISR Function:

This function is the equivalent of an ISR function for the User Interface module. It handles any interrupt driven functions in the User Interface module.

8.2.6 ISR prototype:

The prototype for this function will be:

void userISR(void);

8.3 Functions:

In addition to the public access functions the user interface module will also contain the following functions:

- Interrupt driven user input handling
- Interrupt driven UART receiver and input handling

8.4 Hardware:

The user interface module will make use of the following hardware:

- TTL to RS-232 adapter and serial port
- LCD display
- Value cycle button
- Operation mode Switch
- Potentiometer or encoder or similar
- SPI interfaced numpad?

8.5 Processor Resources:

The user interface module will make use of the following processor resources:

- External interrupts
- UART module (interrupt driven)
- SPI module (interrupt driven) - If numpad used
- ADC (if Pot used)

9 Tracking Module

9.1 Description:

This module concerns the movement of the base, the reading of the IR sensor and searching/tracking the target.

9.2 Public Interface:

The tracking module will have two public access functions: a search function and a track function. It also has a public interrupt function.

9.2.1 Search Description:

The search function moves the base in an incremental search pattern whenever called. The search function will need to check that it is not already at the limits of the base before it starts moving. It will then start moving the base and wait until it reaches the desired increment.

9.2.2 Search Prototype:

The search function will take no arguments and return nothing. The tracking module will have a static variable which stores the direction to move in the next function call. The search function will use the following prototype:

void search(void);

9.2.3 Edge Description:

This function moves the base (very precisely) until it finds the edge of the target (e.g. when it can no longer 'see' the target) in both the azimuth and inclination axis. It then calculates the offset to the middle of the target (either through the target diameter and distance, or by finding the opposite edge), and then directs the sensors to the centre of the target.

9.2.4 Edge Prototype:

The Edge function will take no arguments and return the tracking data - in the format of a struct defined in Common.h. The Edge function will take the following prototype:

TrackingData edge(void);

9.2.5 ISR Function:

This function is the equivalent of an ISR function for the Tracking module. It handles any interrupt driven functions in the Tracking module.

9.2.6 ISR Interface:

The ISR function takes no arguments and returns nothing. The ISR function uses the following prototype:

void trackingISR(void);

9.3 Functions:

In addition to the public functions outlined above the Tracking module 4 should also include the following functionality:

- a Static variable to store the next direction to search

10 Integration Module

10.1 Description:

This module contains the integration of all modules into a functioning system. The code for this module will be contained in the main.c file with no public interface header. It will also use the Common.h file, but all functionality will be contained in the main.c file, which will also include the entry point of the program. The main file is the file which contains the entry point for the program, and non-module specific state infrastructure - it stores the current system state, calls state functions (via a switch case) and handles transitions based on the returned data from modules.

10.2 Public Interface:

The Integration module will have public access functions for use of the user interface module.

10.3 Functions:

There are entry and action functions for each of the system states, as well as main(). Main() is essentially an infinite loop which stores the current state and calls the entry and action functions as necessary. The functions alter the current state, changing the function called in the next iteration.

10.4 Hardware:

This module includes hardware necessary for interfacing the hardware of the other modules examples include:

- 5V power supply and regulator (stepped down from 9V)

10.5 Processor Resources:

This module has no allocated processor resources as such. The interrupt vectors functions and ISR's are included, but they simply call other module functions to handle the cases.

11 Azimuth/Inclination Module

11.1 Description:

This module concerns the operation of the pan tilt actuators to point the sensors in the correct direction (e.g. actually point the sensors at the target, and track it). The functionality for the Azimuth/Inclination module will be completely contained in the PanTilt.c file, with the public interface in the PanTilt.h file.

11.2 Public Interface:

The Azimuth/Inclination module will have the following public access functions:

11.2.1 Move:

This function will simple move the pan tilt mechanism to the specified Azimuth and Inclination.

11.2.2 Move Prototype:

The move function will take a struct Direction defined in Common.h which contains the destination azimuth and inclination. The function will return nothing. The function will use the following prototype:

void move(Direction destination);

11.2.3 Return Direction:

This function will return the current position (direction) of the pan tilt mechanism.

11.2.4 Return Direction prototype:

The return direction function will take no arguments and will return a struct Direction defined in Common.h which contains the current azimuth and inclination that the pan tilt is currently pointing. The function will use the following prototype:

Direction getDir(void);

11.2.5 Calibrate:

This function will modify the calibration offset in the Pan Tilt module so that any direction to move to the specified direction will move the mechanism to the current direction.

11.2.6 Calibrate Prototype:

The calibrate function will take a struct Direction (defined in Common.h) which contains the Azimuth and Inclination to calibrate the system to. The function will return nothing, and will use the following prototype:

void calibratePanTilt(Direction reference);

11.2.7 Raw Direction:

This function will return the raw direction (without any calibration)

11.2.8 Raw Direction Prototype:

The function will take no arguments and return a struct Direction (defined in Common.h) that contains the azimuth and inclination. The function will use the following prototype:

Direction rawDir(void);

11.2.9 Configuration:

This function configures the pan tilt mechanism, beginning PWM generation for the servos and moving the pan tilt to its start position.

11.2.10 Configuration Prototype:

This function will take no arguments and return nothing. The function will use the following prototype:

void configPanTilt(void);

11.2.11 Pan Tilt ISR:

The pan tilt ISR is called whenever an interrupt service routine concerning the pan tilt module is called. It will then act as the ISR for this module.

11.2.12 Pan Tilt ISR Prototype:

The pan tilt ISR will take no arguments and return nothing. It will have the following prototype:

void panTiltISR(void);

11.3 Functions:

In addition to the public access functions defined above the Azimuth/Inclination module should also implement the following hidden functionality.

-

11.4 Hardware:

The Azimuth/Inclination module will make use of the following hardware:

- Pan Tilt mechanism

11.5 Processor Resources:

The Azimuth/Inclination module will make use of the following processor resources:

-

12 Temp Module

12.1 Description:

This module contains all the functionality associated with the temperature sensor.

12.2 Public Interface:

The temp module will contain the following public access functions:

12.2.1 Calibrate:

This function will calibrate the temperature sensor, so that the current value read by the sensor will be interpreted as the value passed to the function.

12.2.2 Calibrate Prototype:

The calibrate function will take an unsigned character representing the temperature in degrees Celsius and return nothing. The function will take the following prototype:

void calibrate(unsigned char reference);

12.2.3 Get Temperature Description:

The get temperature function simply return the current temperature stored in the Temperature module by the last temperature read function call. If the temperature read function has not been called then it should call it, or just initialise the recorded temp to standard room temperature or something.

12.2.4 Get Temperature Prototype:

The get temperature function will take no arguments and will return an unsigned char representing the temperature in degrees Celsius. The function will take the following prototype:

unsigned char getTemp(void);

12.2.5 Temperature Read Description:

The temperature read function will multiplex the ADC onto the analogue input pin connected to the temperature probe, perform a conversion, and then convert that into a temperature in Celsius. Again polling should be sufficient. The temperature may be returned in increments of 0.5^0C to improve resolution (e.g. just double temp in Celsius). If this is the case two separate interface functions should be used one specifically labelled x2, which would thus be able to read the temperature from 0-128 deg. Note: The second function may use the exact same

functionality, and possibly even call the original one to do so, simply modifying the output.

12.2.6 Temperature Read Prototype:

The temperature read function will take no arguments and will return an unsigned char representing the temperature in degrees Celsius. The function will take the following prototype:

unsigned char readTemp(void);

A second temperature read function may be implemented to return the temperature x2 , and will return an unsigned char representing twice the temperature in degrees Celsius. The function will take the following prototype:

unsigned char readTempx2(void);

12.2.7 Raw Temp:

The raw temp function will return the temperature without any calibration.

12.2.8 Raw Temp Prototype:

The raw temp function will take no arguments and return an unsigned char representing the temperature in deg Celsius. The function will take the following prototype:

unsigned char rawTemp(void);

12.3 Functions:

In addition to the public access functions the Pan Tilt module should also implement the following (private) functions:

- PWM generation

12.4 Hardware:

The Temperature module will make use of the following hardware:

- The TI_LM 35 temperature sensor

12.5 Processor Resources:

The Temperature module will make use of the following processor resources:

- ADC

13 Serial Module

13.1 Description:

The Serial module will control all serial communication to the external interface.

13.2 Public Interface:

The serial module will have the following public accessor functions:

13.2.1 Configure:

The configure function will configure the serial port for immediate use. It will set the baud rate to 9600, with 8 bit bytes and no parity on the UART. It will also enable the serial interrupts.

13.2.2 Configure Prototype:

The configure function will have no arguments and will return nothing. The configure function will take the following prototype:

void configureSerial(void);

13.2.3 Transmit:

This function will begin a serial transmission of a given string. It will also enable transmission interrupts so that the string will continue to be transmitted while the processor is free to do other things. It will continue to transmit until it reaches a null terminator.

13.2.4 Transmit prototype:

The transmission function will take an argument of a character pointer, which is equivalent to a string. It will return nothing, and have the following prototype:

void transmit(char *string);

13.2.5 Serial ISR:

This function is called whenever a serial related ISR is thrown. It will act as the interrupt service routine for the serial module.

13.2.6 Serial ISR Prototype:

This function will take no arguments and will return nothing. It will take the following prototype:

void serialISR(void);

13.2.7 Receive Empty:

This checks if there has been any data received over serial.

13.2.8 Receive Empty:

The receive empty function will take no arguments and will return a char (non-zero if empty, zero if not empty). The receive empty function will take the following prototype:

char receiveEmpty(void);

13.2.9 Receive Peek:

This function returns the next character received over the serial, without removing it from the buffer. This means the next peek or pop operation will return the same thing.

13.2.10 Receive Peek prototype:

The receive peek function will take no arguments and will return a character. It will take the following prototype:

char receivePeek(void);

13.2.11 Receive Pop:

This function returns the next character received over the serial, and removes it from the buffer.

13.2.12 Receive Pop prototype:

The receive pop function will take no arguments and will return a character. It will take the following prototype:

char receivePop(void);

13.3 Functions:

In addition to the public functions described above the Serial module should also implement the following hidden functionality:

- Circular Buffer types
- Peek, Push and Pop buffer operations

13.4 Hardware:

The Serial module will use the following hardware:

- Serial port
- TTL to RS-232 adapter

13.5 Processor Resources:

The Serial module will make use of the following processor resources:

- USART module
- Serial interrupts

14 LCD Module

14.1 Description:

This module contains all the functionality required to communicate and use the LCD hardware.

14.2 Public Interface:

The LCD module will contain the following public functions:

14.2.1 Configuration:

The configuration function will setup the LCD for use, and display the initial 'welcome' message, or simply put the LCD into it default state

14.2.2 Configuration Prototype:

The configuration function will take no arguments and will return nothing. It will use the following prototype:

void configLCD(void);

14.3 Functions:

In addition to the public functions described above the LCD module will also implement the following (hidden) functions.

-

14.4 Hardware:

The LCD module will make use of the following hardware:

- LCD screen

14.5 Processor Resources:

The LCD module will make use of the following processor resources:

-