CPSC 5530/4530 Project 5
Introducing the Project Environment – The Final Next Step

Seattle University - Departments of Computer Science and Electrical Engineering
James K. Peckol

Project Objectives:

This project is the final phase in the development of a low cost, portable, medical monitoring system. In the previous phases of this project, we have built a simple kernel and utilized a non-preemptive schedule to manage the selection and execution of the series of tasks comprising our system. We then moved those tasks to a real-time operating system (RTOS) called FreeRTOS that utilizes a preemptive, priority based scheduler to do that job.

Most processors designed for embedded applications are supported either by a simple kernel or a full operating system. Through software modules called drivers, either supports a wide variety of peripheral devices such as timers, digital input and output channels, telecommunications, and analog to digital conversion to support such applications.

In an earlier phase of our project, we have worked with one such built-in device and its driver, the timer. We have also designed several others, one for the display and a second for the keypad, into our system. The goal of this phase of the project is to continue and to extend our development of the medical monitoring system. To that end, we'll work with some of the other built in capabilities on the Stellaris EKI-LM3S8962 / Cortex-M3 system and we will learn to design, develop, and debug drivers for such devices under an RTOS.

The final subsystem must be capable collecting data from several different types of sensors, processing the data from those sensors, displaying it locally, and then transmitting it over a local area network to a collection management station. In the final phase of the design life cycle for the project, we will now,

1. Add features and capabilities to an existing product.
2. Incorporate several additional simple tasks to our system.
3. Work with a hard real-time constraint on one of the tasks.
4. Learn and work with some digital signal processing tools.
5. Introduce additional peripheral devices and develop drivers for them.
6. Introduce and manage a formal communication protocol.
7. Incorporate bidirectional remote communication via a simple web server, handler, and network interface into the system.
8. Incorporate additional safety components into the system.
9. Amend the formal specifications to reflect the new features.
10. Amend existing UML diagrams to reflect the new features.

Prerequisites:

Familiarity with C programming, the Texas Instruments Stellaris EKI-LM3S8962 implementation of the ARM Cortex-M3 v7M microcomputer, and the IAR Systems Embedded Workbench integrated C / Assembler development environment. A wee bit of patience.

Background Information:

Did incredibly well on Project 4; tired and even more anxious to relax. Getting ready to go party in a few weeks….but don't want to go outside with the current temps and the rain dropping.

Relevant chapters from the text: Chapters 5, 8, 9, 11, 12, and 16.

Real-time Operating System

We will continue working an RTOS – a real-time operating system called FreeRTOS. This is an operating system with an attitude…a pirate operating system …sorry, Seafair is long gone…r..yeah, but freertos' got rr's…and at least one nasty patch…rr…this ain't freetos, matey…rrr. Please check out the FreeRTOS web site. You can find this and related documentation at….

http://www.freertos.org/

Check out the getting started and advanced information here or directly at:

http://www.freertos.org/FreeRTOS-quick-start-guide.html

http://www.freertos.org/RTOS_ports.html

http://www.freertos.org/a00090.html#TI

In this project, we're going to continue to improve on the capabilities in our previous designs…this is the real world and we'll add more features to our system as well. We have to make money selling people things that we first convince them that they need…yes, we'll make modifications to Version 2.0 of our earlier system….and raise the price, of course. We have to support the continually flagging economy.

Cautions, Warnings, and Other Musings:

How can a telephone call originating in a small pub in rural Bettyhill (no relation to Benny Hill the olde British comedian), Scotland find its way to a patron's cell phone in the Big Time Ale House in Seattle? How does it know? …and why doesn't it need a visa or a master card at least? …and what if the guy moves at just the last minute? Does the call miss him? Where does the call go if it does?

In the U.S., always look to the left before crossing the street. In the U.K., always look right. In France, look both ways, then, don't cross the street…because they really are trying to kill you. In Italy…well….forget it, you'll never get across the street – don't even try. Viet Nam…that's a story for another day….however, Moses and the Red sea thing was nothing compared to watching 100 cars, 500 motorcycles and scooters, 250 bicycles, and 150 pedestrians go straight through an intersection from all four directions and have subsets making left and right hand turns at the same time and never touch one another. It's pure art.

Try to keep your Stellaris board level to prevent the machine code from collecting in one corner of the memory. This will prevent bits from sticking and causing a memory block. With a memory block, sometimes the Stellaris system will forget to download.

Never try to run your system with the power turned off. Under such circumstances, the results are generally less than satisfying.

Since current is dq/dt, if you are running low on current, raise your Stellaris board to about the same level as the USB connection on the PC and use short leads. This has the effect of reducing the dt in the denominator and giving you more current. You could also hold it out the window hoping that the OLED is really a solar panel.

If the IAR IDE is downloading your binaries too slowly, lower your Stellaris board so that it is substantially below the USB connection on the PC and put the IAR IDE window at the top of the PC screen. This enables any downloads to get a running start before coming into your board. It will now program much faster. Be careful not to get the download process going too fast, or the code will overshoot the Stellaris board and land in a pile of bits on the floor. This can be partially mitigated by downloading over a bit bucket. Note that local software stores stock several varieties of bit bucket, so make certain that you get the proper one. These are not reusable, so also please discard properly. Please note that the farther through the project that you are, the larger the bucket that you must have.… You can recycle old bits if necessary, however, watch when you are recycling that you don't get into an endless loop.

Throwing your completed but malfunctioning design on the floor, stomping on it, and screaming 'why don't you work you stupid fool' is typically not the most effective debugging technique although it is perhaps one of the more satisfying. The debugging commands, *step into* or *step over*, is referring to your code, not the system you just smashed on the floor. Further, *breakpoint* is referring to a point set in your code to stop the high-level flow through your program to allow more detailed debugging…it's not referencing how many bits you can cram into the Stellaris processor's memory before you destroy it.

When you are debugging your code, writing it, throwing it away, and rewriting again several dozen times does little to fix what is most likely a design error. Such an approach is not highly recommended, but can keep you entertained for hours….particularly if you can convince your partner to do it.

Sometimes - but only in the most dire of situations – sacrificing small animals to the code elf living in your Stellaris board does occasionally work. However, these critters are not included in your lab kit and must be purchased separately from an outside vendor. Also, be aware that most of the time, code elves are not affected by such sacrifices. They simply laugh in your face…bwa ha ha…

Alternately, blaming your lab partner can work for a short time…until everyone finds out that you are really to blame.

Always keep only a single copy of your source code. This ensures that you will always have a maximum amount of disk space available for games, email, and some interesting pictures. If a code eating gremlin or elf happens to destroy your only copy, not to worry, you can always retype and debug it again.

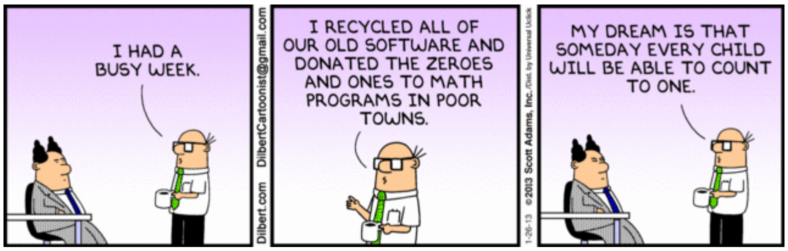
Always make certain that the cables connecting the PC to the Stellaris board are not twisted or have no knots. If they are twisted or tangled, the compiled instructions might get reversed as they are downloaded into the target and your program will run backwards.

Do not connect the digital outputs to the digital inputs of the Stellaris. Doing so has the potential of introducing excess 0's or 1's into the board and causing, like an over inflated child's balloon, small popping sounds leading to potential rupture of the space-time fabric within the Cortex-M3 interconnection scheme such that not even Dijkstra or the thumb mode will be able to stop the bit leaks.

In this part of the lab, when you need to upload from the Stellaris system, be certain to turn the cables around.

Always practice safe software engineering…don't leave unused bits laying around the lab or as Scott Adams writes in the Dilbert strip.

©Dilbert by Scott Adams – 26 January 2013



Project:

We will use this project to continue working with the formal development life cycle of an embedded system. Specifically, we will continue to move inside the system to implement the software modules (the *how* – the system internal view) that was reflected in the use cases (the *what* – the system external view) of the medical monitoring system. To this end, we will migrate the design from the simple kernel and scheduler developed earlier to a real time operating system. Inter task communication will still be implemented utilizing shared variables. We will continue to work with the Eclipse IDE development tool to edit and build the software then download and debug the code in the target environment.

In this final phase of the project we will continue to work with FreeRTOS and the IAR IDE development environment as we explore more of its capabilities. We will also continue to work with interrupts as the underlying timing mechanism for a time base as well as to signal events from the external world. We will continue to develop drivers for peripheral devices that will interact with the outside world. We will support communication over a simple local area network to a remote computer.

As we complete the development of the system, we will….

- ✓ Continue working with the FreeRTOS operating system,
- ✓ Continue working with interrupts, interrupt service routines, and hardware timing functions,
- ✓ Develop and incorporate additional coronary measurement capabilities.
- ✓ Incorporate a remote communications system and a simple web server, handler, and network interface into the system,
- ✓ Implement a command and control interface,
- ✓ Extend the warning and alarm subsystem,
- ✓ Improve the safety of the system,
- ✓ Provide support for several optional features,
- ✓ Utilize UML diagrams to model some of the dynamic aspects of the system.

This lab, lab report, and program are to be done as a team – play nice, share the equipment, and no fighting.

Software Project – Developing Basic Tasks

The economy is getting better but still lingers in the toilet…some engineering jobs are starting to appear, and you have just been given a once in a lifetime opportunity to join an exciting new start up. Some of the top venture people, working with *CrossLoop, Inc*. recent startup in the Valley have just tracked you down and are considering you for a position as an embedded systems designer on a new medical electronics device that they are funding. They have put together a set of preliminary requirements for a small medical product based on IPhone, Pre, Blackberry, and Google concepts, ideas, and technologies that is intended to serve as a peripheral to the CrossLoop system.

The product, *Doctor at Your Fingertips*, will have the ability to perform many of the essential measurements needed in an emergency situation or to support routine basic measurements of bodily functions that people with chronic medical problems need to make. The collected data can then be sent to a doctor or hospital where it can be analyzed and appropriate actions taken.

The system must be portable, lightweight, inexpensive, and Internet enabled. It must have the ability to make such fundamental measurements as pulse rate, blood pressure, temperature, blood glucose level, perform simple computations such as trending, and log historical data, or track medication regimen and prompt for compliance. It must also issue appropriate alarms when any of the measurements or trends exceeds normal ranges or there is a failure to follow a prescribed medication regimen.

Initial deliverables for the system include the display and alarm portion of the monitoring system as well demonstrated ability to handle pulse rate, blood pressure, and temperature measurements. Other measurements and capabilities follow in subsequent phases.

All of the sensors that will provide input to the system and any of the peripheral devices with which the system will be able to interact will not be available at present, so, we will simulate those signals for the first prototype.

You have now successfully delivered an alpha and beta working prototypes of that system and have now been awarded the development contract for the final stage of the project. Similar to the earlier phases, the tasks during the current phase include modifications to the design to improve performance as well as to incorporate additional features and capabilities.

We will now add the Phase IV features and capabilities are given in the requirements and design specifications that follow.

Phase IV requirements supersede earlier specifications wherever there may be a conflict.

Phase IV Additions

1. The system will incorporate a messaging system and the protocol to support bi-directional communication with an external computer.
2. The system will also incorporate an interface to a local area network (LAN) and will support the display of measurement, status, and alarm information on a web browser. Please see Appendix A.
3. The system will accept, interpret, and respond to commands from a remote computer. Please see Appendix B.
4. Control of the blood pressure cuff and initiation of the associated BP measurements will be completed.

5. The system will support EKG measurements.
6. The system will support the processing of raw EKG data.
7. EKG measured data will be displayed on the local OLED, the local console, and sent over the network to the remote system.
8. The system will facilitate access to an emergency site in the event that the path is blocked.
9. The overall system safety will be improved.

System Requirements Specification

1.0 General Description

A low cost, state of the art medical monitoring and analysis system is to be designed and developed. The following specification elaborates the requirements for the display and alarm portion of the system.

The display and alarm management subsystem must accept inputs from a number of sensors used to collect data from various parts of the human body and signal a warning if the data falls outside of pre-specified bounds. Some analysis of the data will be performed by other parts of the system to be designed later.

The outputs of the sensors, that are measuring a variety of natural phenomenon, comprise a number of different data types such as integers, strings or high precision numbers. The system must be designed to accommodate each of these different types.

2.0 Medical Monitoring System

Phase IV Description Modified

Displayed messages comprise three major categories: annunciation, status and warning / alarm. Such information is to be presented on a local display, on a series of lights on the front panel, on demand on a local terminal, and over a LAN to a remote computer.

Sensor signals are to be continuously monitored against predetermined limits. If such limits are exceeded, a visible and aural indication is to be given and shall continue until acknowledged.

Acknowledgement shall terminate the aural indication but the visible indication shall continue until the aberrant value has returned to its proper range. If the signal value remains out of range for more than five measurements, the aural annunciation shall resume.

The local and remote display function and remote command I/O will be fully incorporated during this phase.

System Inputs

The display and measurement component of the system in the first prototype must track and support the measurement and display of the following signals:

Measurements

Blood Pressure

Body temperature

Pulse rate

Phase IV Addition

EKG signals

Keypad Data

Measurement Selection

Scroll

Select

Alarm Acknowledge

System Outputs

The display component of the system in the second prototype must track and support the display of the following signals and data:

Measurements

Blood Pressure

Body temperature

Pulse rate

Phase IV Addition

EKG

The status, alarm, and warning management portion of the system must monitor and annunciate the following signals:

Status

Battery state

Alarms

Temperature, blood pressure, or pulse rate too high

Warning

Temperature, blood pressure, or pulse rate out of range

Phase IV Additions and Modifications

1. The design and implementation of the interface to a LAN and remote computer will be completed.
2. Control of and access to the monitoring system from a remote computer will be incorporated.
3. Control of the blood pressure cuff and initiation of the associated BP measurements will be completed.
4. EKG measured data will be displayed on the local OLED, the console, and sent over the network to the remote system.
5. The system will facilitate access to an emergency site in the event that the path is blocked.
6. The overall system safety will be improved.

2.1 Use Cases

The following use cases express the external view of the system.

Phase IV

(To be updated as necessary)

Software Design Specifications

1.0 Software Overview

A state of the art medical monitoring and analysis system is to be designed and developed.

The high-level design is to be implemented as a set of tasks that are executed, in turn, forever.

The measurement subsystem, display, and alarm management subsystem must accept inputs from a number of sensors that can be used to collect data from various parts of the human body and signal a warning if the data falls outside of pre-specified bounds. Some analysis of the data will be performed by other parts of the system to be designed later.

The outputs of the sensors that are measuring a variety of natural phenomenon comprise a number of different data types such as integers, strings or high precision numbers. The system must be designed to accommodate each of these different types.

The prototype will be implemented using the Stellaris development board and the prototype software will control the LED's on the board through the processor's output ports.

In addition, you must determine the execution time of each task empirically.

The following elaborates the specifications for the preliminary measurement, display, and alarm portions of the system.

2.0 Functional Decomposition – Task Diagrams

The system is decomposed into the major functional blocks: *Initialization, Schedule, Measure, EKG, Compute, Display, Communications, Annunciate, Status,* and *Traffic Management*. Based upon the System Requirements and use case diagrams, a functional decomposition diagram for the system is given as,

Phase IV

(To be updated as necessary)

The dynamic behaviour of each task is given, as appropriate, in the following activity diagrams

Phase IV

(To be updated as necessary)

2.1 System Architecture

The medical monitoring and analysis system is to be designed as a collection of tasks that execute continuously, on a periodic schedule, following power ON. The system tasks will all have equal priority and will be not be preemptable.

Information within the system will be exchanged utilizing a number of shared variables.

Phase IV Additions and Modifications

To implement the required additions and modifications to the product, the following new capabilities must be incorporated.

a. ***Communications Support***: The system is to provide data to and receive commands from a remote computer via a local area network connection.
b. ***Remote Command and Control***: The system must support the display of measurement data and warning and alarm information on and accept commands from a remote computer.
c. ***Command Management***: The task will receive and interpret commands from the *Remote Communications* task then direct the local subsystem(s) to perform the requested tasks.

   It will format any requested data to be sent by the *Remote Communications* task over the network to the remote system.
d. ***EKG Measurement –*** The system must provide the ability to make EKG measurements on the patient. Eventually, the design must support 12 lead capability; however, for this prototype, a single channel will suffice to prove the ability to make such time critical measurements. To that end, the system must collect a set of isochronously spaced samples of a sinusoidal analogue input signal, convert the them to digital form, store the samples for further processing, then signal when the collection is complete.
e. ***EKG Processing –*** The system must convert the sampled data from the time domain to the frequency domain, and then send the component frequency values to each of the display channels. The frequency content of such signals often contains tell tale spectra that can foreshadow potential problems like heart failure.
f. ***Traffic Management System***: *Optional task* that will provide the system with the capability to clear a path for the mobile medical unit as it tries to get through any traffic blocking its path to an emergency situation. It will first release a blast of phasor fire then follow with a barrage photon torpedoes thereby effectively clearing the path.
g. ***Safety***: Overall system safety must be improved.

2.1.1 Tasks and Task Control Blocks

The medical monitoring and analysis system is to be designed as a collection of tasks that execute continuously following power ON. The system tasks will be assigned a priority and will be pre-emptable. Information within the system will be exchanged utilizing a number of shared variables.

TCB creation will be done under freeRTOS utilizing a task create wrapper function. Details of the function are given in Appendix A.

The following function prototypes are given for the tasks are defined for the application

(To be updated as necessary)

The stack size for each of the tasks is given as follows:
(To be updated as necessary)

The priority for each of the tasks is given as follows:
(To be updated as necessary)

2.1.2 Intertask Data Exchange

All system shared variables will have global scope. Based upon the requirements specification, the following shared variables are defined to hold the measurement data, display, status, and alarm information.
The initial state of each of the variables is specified as follows:

Measurements

Data

Type unsigned int – the buffers are not initialized

Type unsigned int
- temperatureRawBuf[8] — Declare as a 8 measurement temperature buffer, initial raw value 75

Type unsigned int
- bloodPressRawBuf[16] — Declare as a 16 measurement blood pressure buffer, initial raw value 80

Type unsigned int
- pulseRateRawBuf [8] — Declare as a 8 measurement pulse rate buffer, initial raw value 0

> 1. The systolic pressure measurements are to be stored in the first half (positions 0..7) of the blood pressure buffer and the diastolic stored in the second half of the buffer (positions 8..15).

Phase IV addition

Type unsigned int
- EKGRawBuf[256] — Declare a 256 measurement EKG buffer

Type unsigned int
- EKGFreqBuf[16] — Declare a 16 measurement EKG result buffer

Display

Type unsigned char
- tempCorrected Buf[8] — Declare as an 8 measurement temperature buffer
- bloodPressCorrectedBuf[16] — Declare as a 16 measurement blood pressure buffer
- pulseRateRawBuf[8] — Declare as an 8 measurement pulse rate buffer

Phase IV Addition

Type unsigned int

- EKGFreqBuf[16] — Declare a 16 measurement EKG result buffer

Keypad

Type unsigned short

- Mode — initial value 0
- Measurement Selection — initial value 0
- Scroll — initial value 0
- Select — initial value 0
- Alarm Acknowledge — initial value 0

Phase IV Addition

Remote Communication:

(To be supplied by engineering)

Command Management:

(To be supplied by engineering)

Traffic Management:

(To be supplied by engineering)

Status

Type unsigned short

- batteryState — initial value 200

Alarms

Type unsigned char

- bpOutOfRange — initial value 0
- tempOutOfRange — initial value 0
- pulseOutOfRange — initial value 0

Warning

Type Bool[2]

- bpHigh — initial value FALSE
- tempHigh — initial value FALSE
- pulseLow — initial value FALSE

2. Although an explicit Boolean type was added to the ANSI standard in March 2000, the compiler we're using doesn't recognize it as an intrinsic or native type. (See http://en.wikipedia.org/wiki/C_programming_language#C99 if interested)

We can emulate the Boolean type as follows:

```
enum _myBool { FALSE = 0, TRUE = 1 };
typedef enum _myBool Bool;
```

Put the code snippet in an include file and include it as necessary.

2.1.3 Data Structures

The TCB member, taskDataPtr, will reference a struct containing references to all data utilized by task. Each data struct will contain pointers to data required/modified by the target task as given in the following representative example, where "data" would be an integer required by myTask.

```
Global
    int data;
    struct DataStruct
    {
        int* taskDataPtr;
    };
```

```
typedef struct DataStruct TaskData;
TaskData myTaskData;
myTaskData.taskDataPtr = &data;
```

The data that will be held in the structs associated with each task are given as follows.

MeasureData – Holds pointers to the variables:

- temperatureRawBuf
- bloodPressRawBuf[3]
- pulseRateRawBuf
- measurementSelection

3. The systolic pressure measurements are to be stored in the first half (positions 0..7) of the blood pressure buffer and the diastolic stored in the second half of the buffer (positions 8..15).

Phase IV addition

EKGData – Holds pointer to the variables:

- EKGRawBuf
- EKGFreqBuf

ComputeData – Holds pointers to the variables:

- temperatureRawBuf
- bloodPressRawBuf
- pulseRateRawBuf
- tempCorrectedBuf
- bloodPressCorrectedBuf
- prCorrectedBuf
- measurementSelection

DisplayData – Holds pointers to the variables:

- mode
- tempCorrectedBuf
- bloodPressCorrectedBuf
- prCorrectedBuf
- batteryState

Phase IV Addition

- EKGFreqBuf

WarningAlarmData – Holds pointers to the variables:
- temperatureRawBuf
- bloodPressRawBuf
- pulseRateRawBuf
- batteryState

Status – Holds pointers to the variables:
- batteryState

KeypadData – Holds pointer to the variables:
- mode
- scroll
- select
- measurementSelection
- alarmAcknowledge

CommunicationsData – Holds pointer to the variables:
- tempCorrectedBuf
- bloodPressCorrectedBuf
- prCorrectedBuf

Phase IV Addition

RemoteCommunicationData – Holds pointers to the variables:
(To be supplied by engineering)

CommandManagementData – Holds pointers to the variables:
(To be supplied by engineering)

TrafficManagementData – Holds pointers to the variables:
(To be supplied by engineering)

The following data structs are defined for the application,

Phase IV

(To be updated as necessary)

2.1.4 Task Queue

Task queue creation will be done under the FreeRTOS OS and will contain ten tasks corresponding to those identified in Section 2.2. Once created, it will not require direct management by the user. Details of the OS and TBCs are given in Appendix A of Project 4.

Tasks will be pre-emptable; if a task has nothing to do, it will exit immediately.

The data pointer of each TCB should be initialized to point to the proper data structure used by that task. For example, if "*MeasureData*" is the data structure for the *Measure* task, then the data pointer of the TCB should point to *MeasureData*.

2.2 Task Definitions

Phase IV *Modify the task definition*

As identified in the functional decomposition in Section 2.0 and system architecture in Section 2.1, the system is decomposed into the major functional blocks: *Initialization, Keypad, Measure, EKG Capture, EKG Processing, Compute, Display, Annunciation, Warning and Alarm, Status, Local and Remote Communications, Traffic Management (optional)*, and *Schedule*.

Such capabilities are implemented in the following class (task) diagrams,

(To be updated as necessary)

Phase IV

The dynamic behaviour of each task is given, as appropriate, in the following activity diagrams

(To be updated as necessary)

The functionality of each of the tasks is specified as follows:

Startup

The *startup* task shall run one time at startup and is to be the first task to run. It shall not be part of the task queue. The task shall,

- ✓ Configure and activate the system time base that is to be the reference for timing all warning and alarm durations.

  The time base will utilize one of the Cortex-M3's hardware timers as the basis for scheduling the execution of the remaining tasks (as necessary) every five seconds and the warning and alarms as required.

  The following sequence diagram gives the flow of control algorithm for the system

  (To be updated as necessary)

- ✓ Configure and initialize all hardware subsystems.
- ✓ Create and initialize all statically scheduled tasks.
- ✓ Enable all necessary interrupts.
- ✓ Start the system.
- ✓ Exit.

The static tasks are to be assigned the following priorities:

(To be updated as necessary)

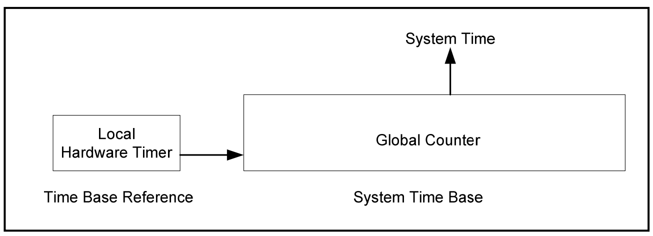
Schedule

The FreeRTOS scheduler manages the execution order and period of the tasks in the system. However, the scheduler is not in the task queue.

The system's task schedule comprises a major cycle and a series of minor cycles. The period of the major cycle is 5 seconds. The duration of the minor cycle is specified by the designer.

Following each cycle major cycle, task activity will be suspended except for the operation of the warning and error annunciation, for five seconds. In between major cycles, there shall be a number of minor cycles to support functionality that must execute on a shorter period.

The following block diagram illustrates the design of the system time base. The Global Counter is incremented every time the Local Delay expires. If the Local Delay is 100 ms, for example, then 10 counts on the Global Counter represent 1 sec.



All tasks have access to the System Time Base and, thus, utilize it as a reference upon which to base their timing.

Note, all timing in the system must be derived from the System Time Base. The individual tasks cannot implement their own delay functions. Further, the system cannot block for five seconds.

The system scheduler will examine all *addTask* or *deleteTask* type flags and add or remove all flagged tasks to or from the task queue.

The following state chart gives the flow of control algorithm for the system

(To be supplied – by engineering)

Measure

The Measure function shall accept a pointer to void with a return of void.

The pointer in the task argument will be re-cast as a pointer to the *Measure* task's data structure type before it can be dereferenced.

During task execution, only the measurements selected by the user are to be performed.

The various parameters must be simulated because the necessary sensors are unavailable.

To simulate the parameters, the following operations are to be performed on each of the raw data variables referenced in *MeasureData*.

temperatureRaw

In the final product, temperature data will be collected from an external temperature sensor. For the current prototype, the patient's temperature will be modeled using an internal analogue signal source.

The Cortex-M3 processor supports an on-chip temperature sensor. The patient temperature will be modeled by utilizing the on-chip sensor then scaling the measured value appropriately to the human temperature range.

The temperature data is to be held in a circular eight reading buffer.

systolicPressRaw and diastolicPressRaw

The analog signals for the systolic and diastolic pressures are to be read at a fixed time interval of 5 ms following an externally generated signal from the blood pressure cuff. When the signal occurs, the blood pressure measurement *addTask* flag is to be set to signal the scheduler to add the blood pressure measurement tasks to the task queue.

For the current prototype, the cuff is to be inflated and deflated manually using two pushbuttons on the Stellaris board. One button will increment the pressure by 10% for each press and the other will decrement it by 10% for each press.

The cuff and patient will be modeled by a simple counter. The counter is to be incremented by one step as the cuff is inflated and decremented by one step as the cuff is deflated. To model the range of different systolic and diastolic blood pressures for different patients, the counter output will be compared against preset limits.

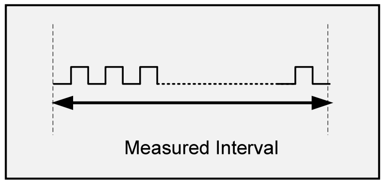
At a blood pressure in the range of 110 to 150 mm HG, an interrupt is to be generated signaling that the systolic measurement is to be made. At a blood pressure of 50 to 80 mm HG, an interrupt is to be generated signaling that the diastolic measurement is to be made.

When the blood pressure measurements are complete, the scheduler shall be signaled to remove the pressure measurement tasks from the task queue.

pulseRateRaw

Pulse rate is measured by a pulse rate transducer. The output of the transducer is an analog signal with a range of 0 to + 50 mV DC.

The analog signal from the pulse rate sensor outputs is to be amplified into the range of 0 to + 3.3 V DC and converted into a digital signal that appears as a series of successive pulses. In the final product, such pulses will be detected using an external event interrupt. For the current prototype, they can be modeled using an internal interrupt or an external interrupt generated by a GPIO signal.



The number of pulses or beats occurring during the measurement interval must be determined and stored in a buffer as a binary value. The measured values must be stored in a circular 8 reading buffer if the current reading is more than 15% different from the previously stored reading – note this is above or below the previously stored reading.

The pulse rate transducers are currently under development. One of the objectives of the present phase is to obtain some field data on a beta version of the transducers. To this end, the upper frequency limit of the incoming signal shall be empirically determined. The upper limit will correspond to 200 beats per minute and the lower to 10 beats per minute.

When the task has completed a new set of measurements, the *addTask* flag for the *Compute* task is to be set.

Phase IV additions

EKG Capture

The *EKG Capture* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the *EKG Capture* task's data structure type before it can be dereferenced.

The EKG capture task will accept the output of an analogue scanner used to collect EKG signals from a patient. The design must eventually support 12 lead measurements.

Copyright Released: Madhero88

To demonstrate feasibility of the concept, the task must accept a time varying sinusoidal analogue data signal from a single strategically placed transducer, isochronously collect 256 samples, convert the sampled values to digital form, then store the converted samples in a 256 measurement buffer for further processing. When a capture is complete, the task must signal the *EKG Processing* task.

This task is scheduled on demand.

Normally, the time constraints on the FFT, dictate that the driver be written at the register level rather than invoked through a wrapper function.

Because we will not have the external signal available to us, as we did with pulse rate and temperature, we will model the signal and the sampling operation. To do so, begin with a sine wave with a peak amplitude of 3.0 and compute 256 temporally equally spaced samples with inter sample temporal spacing to model a sampling rate at two and a half to three times the maximum specified frequency.

EKG Processing

The *EKG Processing* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the *EKG Processing* task's data structure type before it can be dereferenced.

The EKG processing task must perform an FFT on the EKG samples collected by the capture task. When the FFT is complete, the task must indicate that new data is available to permit local display and transmission to a remote site for analysis. The frequency content of signals often contains tell tale spectra that can foreshadow potential problems such as death.

The sampled data shall be converted to the frequency domain using an FFT algorithm (*fftHelp.doc* Appendix C) provided by *Bwang Software, Ltd.*.

Signal frequencies shall be used to classify the input signal.

You'll find an integer algorithm developed by Brent Plump in the Project 4 folder.

The signal characteristics and frequency ranges are specified as follows:

- EKG data: 35 Hz ≤ Frequency ≤ 3.75K Hz
- Peak Amplitude: 3.3 volts
- Signal starting phase Variable
- Signal: Sinusoid

The 16 most recent values shall be tagged (to identify the EKG) and retained in a buffer for transmission on command from the main office.

This task is scheduled on demand.

Compute

The *Compute* task is to be scheduled only if new data is available from the measurement task.

The *Compute* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the *Compute* task's data structure type before it can be dereferenced.

The *Compute* task is to be scheduled only if new data is available from the measurement task. When scheduled, the *Compute* task will take the data acquired by the *Measure* task, perform any necessary transformations or corrections; when the computations are complete, the *Compute* task is to be deleted from the task queue.

The following relationships are defined between the raw data and the converted values.

1. Temperature in Celsius: tempCorrected = 5 + 0.75•tempRaw
2. Systolic Pressure in mm Hg: sysPressCorrected = 9 + 2•systolicRaw
3. Diastolic Pressure in mm Hg: diasPressCorrected = 6 + 1.5•diastolicRaw
4. Pulse Rate in beats per minute: prCorrected = 8 + 3•bpRaw

Keypad Task

The keypad task function shall accept a pointer to void with a return of void.

The pointer in the task argument will be re-cast as a pointer to the *Keypad* task's data structure type before it can be dereferenced.

The keypad is used to enter user information as well as to acknowledge alarm signals associated with the current design and in future enhancements to the system.

The following functions are defined for the keypad:

1. Mode Select
   - Menu
   - Annunciation
2. Menu
   - Blood Pressure
   - Temperature
   - Pulse Rate

   Phase IV addition
   - EKG
3. Annunciation
   - Measurement and Alarm information
4. Scrolling and Selection
   - In the menu mode, scroll to menu choice
   - Select menu choice

The keypad shall be scanned for new key presses on a two-second cycle or as needed.

Display

The *Display* function shall accept a pointer to void with a return of void.

The pointer in the task argument will be re-cast as a pointer to the *Display* task's data structure type before it can be dereferenced.

The *Display* task is charged with the responsibility of retrieving the results of the *Compute* task, formatting the data so that it may be displayed on the instrument front panel display, and presenting the information to the user.

The *Display* task is charged with the responsibility of retrieving the results of the latest EKG measurement from the *EKG Processing* task, formatting the data so that it may be displayed on the instrument front panel display, and finally presenting the information.

The *Display* task is also charged with the responsibility of annunciating the state of the system battery.

The *Display* task will support two modes: Menu and Annunciation.

In the Menu mode, the following will be displayed

Menu

- Blood Pressure
- Temperature
- Pulse Rate

Phase IV addition

- EKG

In the Annunciation mode, the following will be displayed

Annunciation

- Measurement and Alarm information

The ASCII encoded Measurement and Alarm information shall be presented on the OLED display in the Annunciation mode. Blood pressure shall be presented on the top row and temperature, pulse rate, and battery status shall be presented on the second row of the display.

1. Temperature: <temperature> C
2. Systolic pressure: <systolic pressure> mm Hg
3. Diastolic pressure: <diastolic pressure> mm Hg
4. Pulse rate: <pulse rate> BPM>
5. Battery: <charge remaining>

Phase IV addition

6. EKG: <frequency> Hz

The display on the Stellaris board should appear as illustrated in the following front panel diagram,

Phase IV

(To be updated as necessary)

Warning-Alarm

The *Warning-Alarm* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the *Warning-Alarm* task's data structure type before it can be dereferenced.

The normal range for the measurements is specified as follows:

1. Temperature: 36.1 C to 37.8 C
2. Systolic pressure: 120 to 130 mm Hg
3. Diastolic pressure: 70 to 80 mm Hg
4. Pulse rate: 60 to 100 beats per minute
5. Battery: Greater than 20% charge remaining

The *normal* state shall exist under the following conditions:

1. If all measurements are within their specified range, the LED-0 on the front panel shall be illuminated and solid.
2. If the state of the battery is within specified limits.

A *warning* shall be issued under the following conditions:

1. If the pulse rate measurement is out of range, the LED-0 on the front panel shall be illuminated and flashing with a two second period.
2. If the temperature measurement are out of range, the LED-0 on the front panel shall be illuminated and flashing with a 1 second period.
3. If either of the blood pressure measurements is out of range, the LED-1 on the front panel shall be illuminated and flashing with a 0.5 second period.

An alarm shall be issued under the following conditions

1. If the systolic blood pressure measurement is more than 20 percent above the specified limit, the aural alarm shall sound with a series of one second tones.
2. If *Acknowledge* key on the front panel is depressed, the aural alarm shall cease. If the signal value remains out of range for more than five measurements, the aural annunciation shall resume.

The state of the battery display, LED-3, shall flash with a one second period when the state of the battery drops below 20% remaining.

Serial Communications

The serial communication task shall accept a pointer to void with a return of void.

In the implementation of the function, this pointer will be re-cast as a pointer to the *serial communications* task's data structure type.

The *serial communications* task shall run whenever a warning occurs.

The *serial communications* task will format the data to be displayed and send that data to the serial port for display on either Hyperterm™. Once finished, it will delete itself from the queue.

The data presented at the remote site must be expressed as strings as follows:

1. Temperature: <temperature> C
2. Systolic pressure: <systolic pressure> mm Hg
3. Diastolic pressure: <diastolic pressure> mm Hg
4. Pulse rate: <pulse rate> BPM
5. Battery: <charge remaining>

Phase IV addition

6. EKG: <frequency> Hz

Phase IV Addition

Command

The *Command* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the c*ommand* task's data structure type before it can be dereferenced.

The task shall be scheduled whenever a command has been received by the system or when an outgoing message must be formatted in preparation for transmission to the remote computer.

Receive

When a command has been received by the system, the task must verify that the received message is valid. If valid, it is acted upon; if invalid, an error response must be sent to the Remote Communications task.

The legal commands and their interpretation are specified in Appendix B.

Transmit

When a message is to be transmitted, the Command task must build the message body. The message body is then sent to the Remote Communications task for transmission.

After the message has been interpreted and verified as correct or an outgoing message has been built and forwarded to the Remote Communications task, the Command task shall be deleted.

Phase IV Addition

Remote Communications

The *Remote Communication* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the Remote Communication task's data structure type.

The Remote Communications task shall be started following power ON then:

- Initialize the network interface
- Connect to and configure a local area network (LAN)
- Set up a web server and handler to communicate with a remote browser
- Format the data to be displayed and send the formatted data over the network for display on the browser.
- Continually update the displayed data at a 5 second rate.

The corrected patient data presented at the remote site must be expressed as five strings as follows:

1. Temperature: <temperature> C
2. Systolic pressure: <systolic pressure> mm Hg
3. Diastolic pressure: <diastolic pressure> mm Hg
4. Pulse rate: <pulse rate> BPM
5. EKG < Measured Frequency reading > Hz
6. Battery: <charge remaining>

The measured value must flash whenever a warning occurs. Annotation must also accompany each displayed measurement indicating how many times during an eight hour interval that a new warning has occurred.

The remote display shall also present the following information:

- The name of the product
- The patient's name
- The doctor's name.

The following diagram gives the layout of the remote web page.

(To be supplied – by engineering)

Status

The *Status* function shall accept a pointer to void with a return of void.

In the implementation of the task, this pointer will be re-cast as a pointer to the *Status* task's data structure type before it can be dereferenced.

The battery state shall be decremented by 1 each time the *Status* task is entered.

Phase IV

2.3 Data and Control Flow

The system inputs and outputs and the data and control flow through the system are specified as shown in the following data flow diagram.

(To be updated as necessary)

2.3 Performance

The execution time of each task is to be determined empirically. (You need to accurately measure it and document your results.)

2.5 General

Once each cycle through the task queue, one of the digital output lines must be toggled.

- All the structures and variables are declared as globals although they must be accessed as locals.

Note: We declare the variables as globals to permit their access at run time.

- The flow of control for the system will be implemented using a construct of the form

```
while(1)
{
    myStuff;
}
```

The program should walk through the queue you defined above and call each of the functions in turn. Be sure to implement your queue so that when it gets to the last element, it wraps back around to the head of the queue.

3.0 Recommended Design Approach

This project involves designing, developing, and integrating a number of software components. On any such project, the approach one takes can greatly affect the ease at which the project comes together and the quality of the final product. To this end, we strongly encourage you to follow these guidelines:

1. Develop all of your UML diagrams first. This will give you both the static and dynamic structure of the system.
2. Block out the functionality of each module. This analysis should be based upon your use cases.

   This will give you a chance think through how you want each module to work and what you want it to do.
3. Do a preliminary design of the tasks and associated data structures. This will give you a chance to look at the big picture and to think about how you want your design to work before writing any code.

   This analysis should be based upon your UML class/task diagrams.
4. Write the pseudo code for the system and for each of the constituent modules.
5. Develop the high-level flow of control in your system. This analysis should be based upon your activity and sequence diagrams. Then code the top-level structure of your system with the bodies of each module stubbed out.

   This will enable you to verify the flow of control within your system works and that you are able to invoke each of your procedures and have them return the expected results in the expected place.
6. When you are ready to create the project in the IAR IDE. It is strongly recommended that you follow these steps:

   a. Build your project.
   b. Correct any compile errors and warnings.
   c. Test your code.
   d. Repeat steps a-c as necessary.
   e. Write your report
   f. Demo your project.
   g. Go have a beer.

***Caution*: Interchanging step g with any other step can significantly affect the successful completion of your design / project.**

Project Report

Write up your Project report following the guideline on the class web page – please check the web site to make certain that you have covered all of the requirements stipulated there.

You are welcomed and encouraged to use any of the example code on the system either directly or as a guide. For any such code you use, you must cite the source**…you will be given a failing mark on the Project if you do not cite your sources in your listing - this is not something to be hand written in after the fact, it must be included in your source code…** This is an easy step that you should get in the habit of doing.

Do not forget to use proper coding style; including proper comments. Please see the coding standard on the class web page under documentation.

Please include in your lab report an estimate of the number of hours you spent working on each of the following:

Design

Test / Debug

Documentation

Please include the items listed below in your project report:

1. Hard copy of your pseudo code
2. Hard copy of your source code.
3. Empirically measured individual task execution time.
4. Include a high-level block diagram with your report.
5. If you were not able to get your design to work, include a contingency section describing the problem you are having, an explanation of possible causes, a discussion of what you did to try to solve the problem, and why such attempts failed.
6. The final report must be signed by team members attesting to the fact that the work contained therein is their own and each must identify which portion(s) of the project she or he worked on.
7. If a stealth submersible sinks, how do they find it?
8. Does a helium filled balloon fall or rise south of the equator?

NOTE: In a formal report, your pseudo code, source, numbers, raw data, etc. should go into an appendix. The body of the report is for the discussion, don't clutter it up with a bunch of other stuff. You can always refer to the information in the appendices, as you need to.

NOTE: If any of the above requirements is not clear, or you have any concerns or questions about you're required to do, please do not hesitate to ask us.

Appendix A: The Network Connection

The network connection will comprise a local web server and client handler on the Stellaris board and a remote client on a PC connected to the lab's local area network (LAN).

The IAR development IDE and Stellaris examples provide a good place to begin. It is recommended that you start with the project *enet_io* and proceed from there.

From the comments accompanying the file *enet_io.c* in that project,

> *The example application demonstrates web-based I/O control using the Stellaris Ethernet controller and the lwIP TCP/IP Stack. DHCP is used to obtain an Ethernet address.*

If the DHCP finds an address, that address will be displayed on the OLED. Connect the browser on the PC to that address.

> *If DHCP times out without obtaining an address, a static IP address will be chosen using AutoIP. The address that is selected will be shown on the OLED display allowing you to access the internal web pages served by the application via your normal web browser.*
>
> *Two different methods of controlling board peripherals via web pages are illustrated via pages labeled "IO Control Demo 1" and "IO Control Demo 2" in the navigation menu on the left of the application's home page.*

Modifying the existing website or creating your own requires working with several tools that come with the Stellaris environment. Under the *enet_io* project directory, there is file called *io_fsdata.h* and a directory, *fs*. All of the files that make up the website are stored as a file system image in the file *io_fsdata.h* that is linked into the application image in flash. The subdirectory, *fs*, contains all of the associated html files from which the image is built.

It is recommended that you start with the files in the *fs* directory *enet_io* under the project.

- ✓ To build the application image, first, it's recommended that you save a backup of the existing *io_fsdata.h* file.
- ✓ Next, make your changes to the files in the *fs* directory by adding images or modifying one of the other .html files as necessary. After making the appropriate changes, you must rebuild the file system image using command:

  makefsfile -i fs -o io_fsdata.h -h –r

This function is found in the *C:\StellarisWare\tools\bin* directory. You must call the function from the Microsoft Command Line…

Once you have a new image file, *io_fsdata.h*, add it to the project directory, replacing the previous one.

You should now have a new web site

Appendix B: Supported Commands and Responses

The Commands and Responses for the embedded application task are given as follows.

- I The I command initializes the network communications between your system and the browser page.
- E The E error response is given for incorrect commands or non-existent commands.
- S The S command indicates START mode. The command shall start the embedded tasks by directing the hardware to initiate the measurement tasks. In doing so, the command shall also enable all the interrupts.
- P The P command indicates STOP mode. This command shall stop the embedded tasks by terminating any running measurement tasks. Such an action shall disable any data collecting interrupts.
- D The D command enables or disables the local display.
- M The M command. The M command requests the return of the most recent value(s) of all measurement data.
- W The W command. The W command requests the return of the most recent value(s) of all warning and alarm data.

Appendix C: FFT Helper

A Real - Time Embedded Application

Using the Cortex-M3 Microcontroller

Bin Wang

Issues on A/D configuration for FFT application

1. Nyquist theory: sampling frequency must be greater than twice the signal frequency (in order to recover the original signal without aliasing).

   Fs >2*Fn

Samples from A/D must be equally spaced in time.

FFT generic function

X[k]= $\sum_{n=0}^{N-1} x[n] W_N^{nK}$, N is sequence length.

X[k], X[n] are supposed to be complex.

X[n] has real and imaginary parts. The samples you get from A/D are the real part of X[n]. The imaginary of X[n] are all zero.

2. X[k] has non-zero values of both real and imaginary parts.

   void fft(complex*x, complex*w, unsigned int m)

   Also you need to typedef{int *real, *imag} complex;
3. Here m is the FFT stage, N=2^m. Usually we use 128, 256 as the length of the sequence, then m will be 7,8 correspondingly. The longer the sequence, the more accurate the FFT. But we are constrained by the available memory space and the speed.
4. Here w is computed inside the FFT function, we do not care and we are not supposed to pass things into it.
5. Since the A/D samples the input signal from 0v-5v, you need shift all the samples by 2.5v to make the samples have positive and negative parts before doing FFT.

In Place Computations

Input array and output array share the same piece of memory space.

Note: each time you do the FFT, remember to initialize the input imaginary buffer to zero.

Determine frequency from the index:

$f = fs * m\_index / N$

f: freq of the measured signal

fs: sample freq

m_index: index which corresponds to the maximum magnitude

$Mag^2 = real^2 + imag^2$

N: sequence length or $N = 2^m$

Note: The range of FFT is $[0, 2\pi]$, and also symmetric about π.

[0,N] is mapped to this range. When you search for the maximum magnitude, half range searching is desired.

Appendix D: Measuring Blood Pressure

Blood Pressure

A blood pressure measurement is made in two major steps. The process begins with a blood pressure or sphygmomanometer cuff is wrapped around the patient's upper arm. An aural sensing device such as a stethoscope is placed over the brachial artery on the front side of the arm just over the elbow.

The pressure in the cuff is increased to a level of approximately 180mmHg. Such a pressure compresses the brachial artery causing it to collapse and prevent further blood flow. At this point, the pressure in the cuff is slowly decreased. When the artery opens, blood begins to flow again causing vibrations against the artery wall. The pressure at which this occurs is called the systolic pressure and is approximately 120mmHg in the normal case.

As the pressure on the cuff continues to decrease, the blood flow continues to increase. Vibrations against the artery wall also decrease until the blood flow through the artery returns to laminar. The point at which the vibrations cease is called the diastolic pressure. In the normal case, this will be approximately 80mmHg.

These sounds of the blood against the artery wall are called Korotkoff sounds after Dr. Nikolai Korotkoff, a Russian physician who first described them.