Introduction to Real Time Programming

The Real Time Loop
Introduction

• Any system that needs a timely response to applied input can be considered "Real Time".

Examples: ABS systems, remote controls, and vending machines
Introduction

• Real Time applications are typically composed of several tasks or processes. For example:
  - motor control
  - solenoid control
  - pushbutton input
  - time-of-day scheduling
  - IR input
  - display refresh

could be combined to form a single application: the VCR.
Introduction

• Embedded systems invariably need to
  • perform a task at a certain rate
  • perform a task at a certain time
  • respond to an input condition within a certain time.

• One or more of these performance requirements may apply to a given application.
Solution Frameworks

• There are two commonly used software real-time software frameworks:
  
  • The Real Time Loop (RTL) – a very deterministic structure used in time-critical and simple systems.
  
  • The Real Time Multitasking Executive (RTX) – usually used in more complex systems. Unless developed in-house, the RTX can cost quite a bit.
Solution Frameworks

• Both the RTL and RTX are considered to be multitasking operating systems as they are the software foundation through which system resources can be accessed.
Solution Frameworks

• Why not just write your own program which adheres to the Real Time requirements?
• Using an RTL or RTX helps
  – provide structure for overall system software design;
  – break the application down into a set of manageable modules (tasks); and
  – shorten development time through creation of simple (independently tested) tasks.
System Resources

• Resources that a CPU has available to it to do its work (in this case, Real Time work) include:
  – CPU Time
  – Memory
  – Peripherals

Allocation of- and access to these resources are dictated by the chosen framework.
The Real Time Loop (RTL)

**Foreground**

1. Initialization
2. Task 1
3. Task 2
4. Task N

**Background**

1. Real-time Clock Interrupt
2. Interrupt Acknowledge
3. Update Real Time Clock Variables:
   - Ticks
   - Seconds
   - Minutes
   - Hours
   - This could also be implemented in a task.
4. retfie instruction

Stop Processing
Wait for Interrupt
The Real Time Loop (RTL)

• RTL nomenclature:

  **tick time** – the period of execution of the background (interrupt) and therefore the foreground.

  **foreground** (task/application level):
  - this is a polling loop that checks all tasks and performs those required
  - no task may hog the CPU
  - execution pauses at the end with a wait for the next interrupt
The Real Time Loop (RTL)

- RTL nomenclature (continued):

  background (interrupt level):
  - a periodic (timer) interrupt
  - usually very simple
Real Time Loop Fundamentals

• It is a cooperative system: for every tick, all tasks must be given attention.
  – Each task has the option to run every tick but does not necessarily choose to do so.
  – If it does run, it does so only for a small percentage of the tick time.
  – Long processes must be distributed over many ticks.
Real Time Loop Fundamentals

• The tick must occur often enough to service the most demanding task.
• The RTL is a failure if a required operation is not performed at the correct time or rate.
• Peripherals should interface with only one task.
  – Other tasks should communicate with this single task in order to utilize the resources offered by the peripheral.
Real Time Loop Fundamentals

• Software must be written in a way that does **not** tie-up the CPU by:
  - waiting for a hardware signal that may never arrive (i.e. no spin waits).
  - executing software delays (i.e. no delay routines).
Real Time Loop Execution

Tick Interval

Back-Ground

Foreground

Idle Time

ISR  Task 1  Task 2  ...  Task N  Waiting for Interrupt

ISR  Task 1  2  ...  N

Interrupt Occurs

Begin Waiting for Interrupt

Time
Flowchart of a Generic RTL Task

Task N

- Usually involves testing a flag.
- Anything to do?
  - Yes
    - Able to take action?
      - Yes
        - Conditional Task Function
          - Clear flag indicating action was required.
          - End of Task
      - No
    - No
      - End of Task
The RTL and Peripherals

• If a task’s responsibility is to interface with a peripheral, it will need to control access to this peripheral (when requested by other tasks).
  • Queues
  • Arbitration (priorities) – similar to a queue that is one entry deep.
The RTL and Peripherals

• The following is a flowchart for a task used to interface to the UART transmit circuitry.

• This task, TxTask, is the “gateway” for other tasks to access the UART transmit circuit.

• In addition to interfacing with a peripheral, transmission of even a single character is very slow: non-blocking execution is required!
UART TxTask

TxTask

TxPend=0?

Yes

No

UART Tx Ready?

Yes

No

Get Byte @ TxPointer
Increment TxPointer

Byte=NULL?

Yes

Clear TxPend

No

Transmit the Byte

End of Task
The RTL and Peripherals

• To make use of the UART's transmission circuit, all “producer” tasks must make use of TxTask.

• These producer tasks will ask:
  – Do I have anything to send?
  – If so, is the Transmit service available?

If the answer to both questions is “yes”, then the task reserves the TxTask (through a variable $TxPend$) and tells it what to send (through a variable $TxPointer$).
The RTL and Peripherals

If the transmit task is not available, then the producer task will have an opportunity to run during the next tick interval.

Once the producer task successfully queues (or arbitrates) its “job” for TxTask, it will likely clear its own flag to indicate “mission accomplished”

• Inter-task communication is implemented through use of variables, as demonstrated here.
Generic Producer Task

Task N

Unconditional Task Function

Tx Service required?

Yes

Tx Service available?

Yes

Set TxPend and TxPointer

A portion of the task may be dependent on UART transmission. (i.e. Building a string to send.)

Conditional Task Function

End of Task
RTL Tips and Caveats

• Dividing the overall application into separate tasks gets simpler with experience.
• Arbitration of peripheral access is simple to create if you understand what is going on.
• Write each task as a subroutine. This modularity allows simpler debugging.
RTL Tips and Caveats

• If you are using the PIC16F87XA, see the course website for the RTL framework!
• If you are using the Atmel AVR, see the course website for the RTL framework.
• Modifications may be necessary: particular peripherals have been used to generate the periodic interrupts – peripherals you may already have reserved for something else!