AvrX

https://github.com/kororos/AvrX

Yousef Ebrahimi
Professor Ryan Robucci

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Introduction

- AvrX is a Real Time Multitasking Kernel written for the Atmel AVR series of micro controllers.
- The Kernel is written in assembly. Total kernel size varies from ~500 to 700 words depending upon which version is being used.
- Since the kernel is provided as a library of routines, practical applications take up less space because not all functions are used.
- AvrX (RTOS) maintains state information for the programmer.
- Instead of a complicated state machine for each task, running off an interrupt timer, the designer can write linear code (do this, wait for something, then do that... etc). In general the linear code is much easier to design, debug, understand and it is almost always smaller.

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Versions

- There are two versions of AvrX available:
- AvrX v2.3 for the IAR assembler. This is a tiny version that is suitable for Assembly programming only. It makes very efficient use of RAM and takes very few cycles to service interrupts.
- AvrX 2.6 for the IAR Systems and GCC C compiler. This version of AvrX is written for a small memory model (16 bit pointers) and a native C interface. The code size is larger than the 2.3 version (~700 words vs. 500) and all registers need to be swapped with each context change, so it is somewhat slower and uses more SRAM. As for speed, processing a system timer tick took 211 cycles in AvrX 2.3 but takes 234 in AvrX 2.6

UMBC
AN HONORS UNIVERSITY IN MARYLAND

OS Interrupt / System Tick

- To do scheduling, a HARDWARE timer with an interrupt is needed!
- Can also use it to allow having software timers.
- One of the available hardware timers can be used to cause an interrupt as the base timer for the AvrX RTOS.
- To choose the hardware timer, its tick rate and more we need to modify "avrx_hardware_custom.h" header file
  - I have changed the header file to use Timer2
  - Its tick rate will be 1 millisecond
  - Add following code to have the timer setup

```
AVRX_SIGINT(AVRX_HARDWARE_TIMER_SIGNAL){
    IntProlog();          // Switch to kernel stack/context
    AvrXTimerHandler();   // Call Time queue manager
    Epilog();             // Return to tasks
}
```

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Structure of Interrupt Handler

- AvrX completely handles the saving and restoring of the interrupted context

```
AVRX_SIGINT(INTERRUPT_NAME){
    IntProlog();
    ///handling code
    Epilog();
}
```

- Get the INTERRUPT_NAME from avrx\avrx-signal.h file

UMBC
AN HONORS UNIVERSITY IN MARYLAND

System Tick setup

- In main you need to run following macro

```
AVRX_HARDWARE_SETUP_COMMANDS;
```

- Macro has been defined at “avrx_hardware_custom.h”
- It simply sets the registers for Timer2 to generate the interrupt every 1ms --- ISR in previous slide
- **IntProlog():** Pushes entire register context onto the stack, returning a frame pointer to the saved context.
  - Usage: Internal use of AvrX and ISRs.
- **Epilog**(): Restore previous context (kernel or user).
  - Usage: Internal use of AvrX and ISRs.
- **AvrXTimerHandler():** internal function from AvrX to handle the scheduling of tasks/setting timers and …

UMBC
AN HONORS UNIVERSITY IN MARYLAND

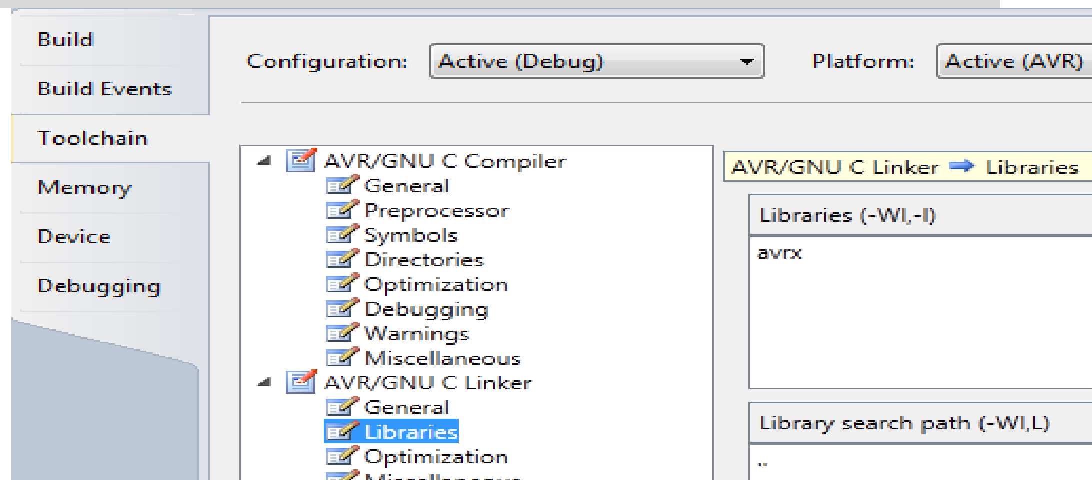
Toggling LED using Timer1

```
#include <avr/io.h>
#include "avrx.h"
#include "avrx_hardware_custom.h"
AVRX_SIGINT(AVRX_HARDWARE_TIMER_SIGNAL){  -- system tick
   IntProlog();                    // Switch to kernel stack/context
   AvrXTimerHandler();       // Call Time queue manager
   Epilog();                         // Return to tasks
}

void setupTimer1() {
  TCCR1B  = 0b100;  //select prescaler
  TCCR1B |= (1<<WGM12); TCCR1A = 0; //set to CTC mode
  //Setting the compare match value
  OCR1A = 15625 ;  // OCR1AH = 15625 >> 8; OCR1AL = 15265;
  TIMSK1 = 2; //only set interrupt for compare match on A
}
```

Toggling LED – Cont'd

```
AVRX_SIGINT(SIG_OUTPUT_COMPARE1A){  //ISR
    IntProlog();
    PORTB ^=(0x02);
    Epilog();
}

int main(void){
   AvrXSetKernelStack(0);   //next slide
   AVRX_HARDWARE_SETUP_COMMANDS;  //setups system Timer
   setupTimer1();
   DDRB = 0xff; // set PORTB as output
   Epilog();//starts scheduling of tasks
   while(1) ;}
```

Don't forget to include libavrx.a when building the project



UMBC
AN HONORS UNIVERSITY IN MARYLAND

Base of AvrX projects

```
#include <avr/io.h>
#include "avrx.h"
#include "avrx_hardware_custom.h"
AVRX_SIGINT(AVRX_HARDWARE_TIMER_SIGNAL){ // system tick
   IntProlog();        // Switch to kernel stack/context
   AvrXTimerHandler();// Call Time queue manager
   Epilog();          // Return to tasks
}

int main(void){
   AvrXSetKernelStack(0);
   AVRX_HARDWARE_SETUP_COMMANDS;
   //TODO::
   Epilog();
   while(1);
}
```

- AvrXSetKernelStack: Sets AvrX Stack to "newstack" or, if NULL then to the current stack.

UMBC
AN HONORS UNIVERSITY IN MARYLAND

AvrX Task

- To define a task (process) – [this like applications that OS will schedule and run]

```
AVRX_GCC_TASKDEF(start, c_stack, priority)
```

  - ***start***: name of the task
  - ***c_stack***: the additional stack required above the 35 bytes used for the standard context
  - ***priority***: priority of the task (lower value means more important)
    - 16 levels*
- Example:

```
AVRX_GCC_TASKDEF(firstTask,0,1) {
  int i = 0;
  while(1){
    ++i;
    if (i % 1000 == 0){
      LED = LED ^ 0x04;
    }
  }
}
```

*http://www.barello.net/avrx/overview.htm

UMBC
AN HONORS UNIVERSITY IN MARYLAND

AvrX Task – Cont'd

- AvrXRunTask: To initialize and run the task.

```
int main(void){
  AvrXSetKernelStack(0);
  AVRX_HARDWARE_SETUP_COMMANDS;
  DDRB = 0xff; // set PORTB as output

  AvrXRunTask(TCB(firstTask));

  Epilog();
  while(1);
}
```

- TCB : Returns the pointer to the task control block allocated for the passed in task (firstTask).

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Adding 2nd Task

- Creating 2nd task -- First task is same as before

```
AVRX_GCC_TASKDEF(Task2,0,1){
  int i = 0;
  while(1){
    ++i;
    if (i % 1000 == 0){
      PORTB ^= 0x08;
    }
  }
}
```

```
AVRX_GCC_TASKDEF(firstTask,0,1)
{}
```

Adding following line to main function

```
AvrXRunTask(TCB(firstTask));
AvrXRunTask(TCB(Task2));
```

- In scheduling the task, Higher priority task will take over the CPU and will not release it since we have forever loop!!!
- Same priority task must cooperate – they must yield the CPU to other processes.

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Software Timers

- Having a system tick allows AvrX to support software timers
  - Not as accurate as hardware timers though. – OS manages them.

- TimerControlBlock
  - A structure to handle timer
  - It has 16 bit counter – our timers are 16-bit wide
  - Following line defines 2 timers as global variables,

```
TimerControlBlock timer1,timer2;
or use macro
AVRX_TIMER(timer1);
```

- AvrXStartTimer(TimerControlBlock* timeControlBlockPtr, unsigned count);
  - non-blocking API
  - timeControlBlockPtr : pointer to the timer to start.
  - 2nd parameter is timeout tick count.

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Timer

- AvrXWaitTimer(&TimerControlBlock);
  - Waits on a timer to expire. This suspends the task and takes it out of ready queue. (allows same/lower priority tasks get the CPU)

Fixing Task 1 and 2

```
AVRX_GCC_TASKDEF(firstTask,4,1){
  while(1){
    AvrXStartTimer(&timer1, 1000);
    AvrXWaitTimer(&timer1);
    PORTB ^= 0x01;
  }
}
```

```
AVRX_GCC_TASKDEF(Task2,4,1){
  while(1){
    AvrXDelay(&timer2, 2000);
    PORTB ^= 0x08;
  }
}
```

- AvrXDelay(& TimerControlBlock, unsigned);
  - Calls AvrXStartTimer and then AvrXWaitTimer
- Stack size has increase to 4; Allow at least 2 bytes for level of internal function calls!!

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Mutex

- AvrX supports Mutex Semaphores.
  - They can be unlocked by a process different than the one that locked them.
- Define a Mutex

```
Mutex timeOut;
Or use macro
AVRX_MUTEX(timeOut);
```

- Wait for it:

```
AVRX311_WAIT_P(timeOut);
```

- Signal it:

```
AVRX311_SIGNAL_V(timeOut);
```

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Example

```
AVRX_GCC_TASKDEF(firstTask, 2 ,1){
    while(1){
        AVRX311_WAIT_P(timeOut);
        PORTB ^= 0x01;
     }
}
AVRX_GCC_TASKDEF(Task3, 2 , 1)
{    while(1){
        AVRX311_WAIT_P(timeOut);
        PORTB ^= 0x10;
    }
}
```

```
AVRX_GCC_TASKDEF(Task2, 4 ,1){
    while(1){
       AvrXDelay(&timer1,1000);
       AVRX311_SIGNAL_V(timeOut);
       AVRX311_SIGNAL_V(timeOut);
       PORTB ^= 0x08;
    }
}
```

- Remove one of the "AVRX311_SIGNAL_V " and observe the behavior.
- Mutex Semaphore is not binary in AvrX

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Creating Tasks

Objective action	Code
Define a task with code: Declare task data structure and the top level C declaration (AVRX_TASK + C function declaration)	AVRX_TASKDEF(start, stacksz, priority)
Declare a task: Declare task data structures and forward reference to task	AVRX_TASK(start, stacksz, priority)
Declare external task: Declare external task data structures	AVRX_EXTERNTASK(start)

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Basic Task Control

- `Assume taskXYZ declared with AVRX_GCC_TASKDEF(taskXYZ,…`

Objective action	Code
Run/Start a task	AvrXRunTask(TCB(taskXYZ));
Terminate a Task	AvrXTerminate(PID(taskXYZ));
Suspend a task	AvrXSuspend(PID(taskXYZ));
Resume a task	AvrXResume(PID(taskXYZ));
Cause task itself to give up CPU to next ready task	AvrXYield(); or any wait like AVRXDelay(&timerXYZ, 1); Use AVRXDelay if any problems found with AvrXYield
Cause task itself terminate	AvrXTaskExit(void);

UMBC
AN HONORS UNIVERSITY IN MARYLAND

Interrupt Handler

- AvrX has its own API for creating interrupt handlers

Objective action	Code
Declare the top level C declaration for an interrupt handler	AVRX_SIGINT(vector);

Task Referencing

- `Assume taskXYZ declared with AVRX_GCC_TASKDEF(taskXYZ,…`

Objective action	Code
Get pointer to task’s PID	PID(taskXYZ);
Get pointer to task’s TCB	TCB(taskXYZ);
Get pointer to task’s own PID	AvrXSelf();

UMBC
AN HONORS UNIVERSITY IN MARYLAND