MCF5307 ColdFire Janus Development Board

Irene Huang

Winter 2010

Electrical and Computer Engineering

University of Waterloo

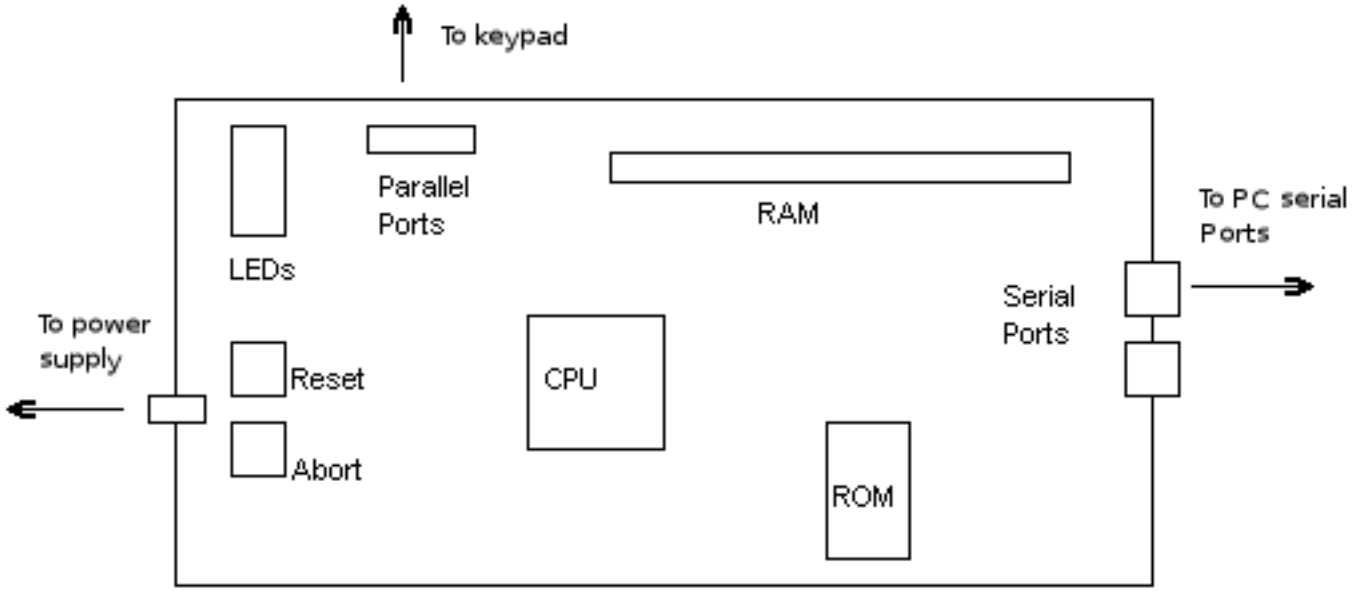
ECE354 Lab Page

- Lab Instructor's page and important documents
  - http://www.ece.uwaterloo.ca/~yqhuang/labs/se350
  - http://www.ece.uwaterloo.ca/~yqhuang/labs/se350/document.html
- SBC5307 Manual and References
  - http://www.ece.uwaterloo.ca/~yqhuang/labs/se350/doc/manual_ece354_lab_s06.pdf
- Sample source code in the lab manual
  - http://www.ece.uwaterloo.ca/~yqhuang/labs/ece354/doc/manual_5307v1a.zip

Outline

- Overview of the 5307 ColdFire Board
- MCF5307 ColdFire Processor Core
- MCF5307 ColdFire Janus Board Details
- Timer Module
- UART Module
- Programming

Overview: MCF5307 ColdFire Janus Development Board



- A ColdFire 5307 processor
- A ROM for monitor program
- 32 MB RAM (or more)
- Two serial ports (UARTs)
- Two timers/counters
- Parallel Ports (not used in ece354)
- LEDs (not used in ece354)
- Reset: the red button
- Abort: the black button

MCF5307 ColdFire Core

- 32-bit CPU, 90 MHz core, RISC
- 32-bit addresses and 32-bit data
- 8 32-bit address registers (a0-a7)
- 8 32 bit-data registers (d0-d7)
- 32-bit program counter (pc)
- 16-bit status register (sr)
- One stack pointer for both user and supervisor modes
- Vector base register (vbr) to relocate exception-vector table

MCF5307 Janus Board

- The monitor program: janusROM in flash memory
- COM1 setting
  - 9600 baud, 8 bit, no parity, 1 stop bit, 25ms line delay
- MBAR at 0xF0 000 000
  - Module Base Address Register
  - Logical base address for the memory-mapped space containing control registers for the on-chip peripherals
- RAM starts at 0x10 000 000
  - PC100 SDRAM
- The VBR starts 0x10 000 000
  - Starts at RAM, no need to be relocated
- Programs start at 0x10 100 000

Janus Board ROM Services

- TRAP #15 functions
  - OUT_CHAR
    - `#0x13 -> d0, Out_char -> d1`
  - IN_CHAR
    - `#0x10 -> d0, In_char <- d1`
  - CHAR_PRESENT
    - NOT Implemented
  - EXIT_TO_DEBUG
    - `#0 -> d0`
  - *Reference: section 2.5 in SBC5206 User's Manual*
- Command set
  - Type "help" at janusROM prompt to see the command list
  - Frequently used commands
    - bf, br, cont, help[?], md, mm ,peek, rc, rd, tftp, tr[dl]
    - -h for command specific details

Interrupt Control

- Setup ISR in Vector Table
  - VBR + vector number offset
- ICR (Interrupt Control Registers)
  - To set interrupt level and priority
  - ICR1: Timer0, MBAR+0x4D
  - ICR5: UART1, MBAR+0x51
  - Timer1 and UART0 are Not Available for the students.
- SIM_IMR (Interrupt Mask Register)
  - MBAR+0x44

Timer Module

- Two general-purpose16-bit timers
  - Timer0, used by the RTX
  - Timer1, used by auto tests, N/A to students
  - System clock runs at 45 MHz
- Timer programming Model
  - Prescaler: divides the clock from 1 to 256
    - TMR: Timer Mode Register (+ 0x140)
  - Reference compare:
    - TRR: Timer Reference Register (+ 0x144)
    - TER: Timer Event Register (+ 0x151)
  - Current Value of the timer counter
    - TCN: Timer Counter (+ 0x14C)
- Reference chapter 13 in *MCF5307 Integrated Microprocessor User's Manual*

UART Module

- Two independent UARTs
  - UART0: used by janusROM, N/A to students
  - UART1: used by the RTX
  - Full-duplex asynchronous/synchronous
  - Receiver: polled or interrupt-driven
- UART Programming Model
  - UART registers are accessible only as BYTE
  - UCR :UART Command Register (+ 0x208)
  - UISR/UIMR:UART Interrupt Status/Mask Register (+ 0x214)
  - USR:UART Status Register(+ 0x204)
  - UMR:UART Mode Register (+ 0x200)
- Reference chapter 14 in *MCF5307 Integrated Microprocessor User's Manual*

ColdFire Cross Compiler

- On ecelinux
  - M68k-elf-gcc v4.3.2: /opt/gcc-coldfire/bin
- On eceunix
  - m68k-elf-gcc v3.4.6 : /opt/gcc-3.4.6-mcf5307/bin/
- Windows
  - Install your own David Fiddes's cross compiler http://sca.uwaterloo.ca/www.calm.hw.ac.uk/davidf/coldfire/download.htm
  - Sourcery G++ Lite http://www.codesourcery.com/sgpp/lite_edition.html
- Two important Commands
  - m68k-elf-gcc: to compile and link
  - m68k-elf-objcopy: to generate .s19 file
  - m68k-elf-objdump: to generate assembly code

Code Development Steps

- Edit source code
- Compile source code to .s19 file
  - Do not use standard C library
  - Use a make file
- Download the .s19 file to the board
  - Real board: tr -p terminal, dl -p terminal
  - Remote cf-sever: cfcp, tftp -f <filename>
- Run the code from the board
  - Type command “go 10100000”
- Reference FAQ (Q1-4) on Lab Instructor’s Page

Example start.s File

```
/* Allocate space for new stack and old stack pointer */
   .comm old_stack,4
   .comm main_stack,4096
   .even

   /* Install a new stack */
   move.l %a7,old_stack
   move.l #main_stack+4096, %a7

   /* Jump into main */
   jsr main

   /* Restore old stack */
   move.l old_stack,%a7

   /* Store return value from main */
   move.l %d0, %d7

   /* Get back to the monitor */
   move.l #0,%d0
   trap #15
```

Example hello.c File

```
// C Function wrapper for TRAP #15
// function to output a character
VOID rtx_dbug_out_char( CHAR c ){

  /* Store registers */
  asm( "move.l %d0, -(%a7)" );
  asm( "move.l %d1, -(%a7)" );

  /* Load CHAR c into d1 */
  asm( "move.l 8(%a6), %d1" );

  /* Setup trap function */
  asm( "move.l #0x13, %d0" );
  asm( "trap #15" );

  /* Restore registers  */
  asm(" move.l %d1, (%a7)+" );
  asm(" move.l %d0, (%a7)+" );
}
```

```
// Print a C-style null
// terminated string
SINT32 rtx_dbug_outs( CHAR* s ) {
  if ( s == NULL ) {
      return RTX_ERROR;
  }
  while ( *s != '\0' ) {
    rtx_dbug_out_char( *s++ );
  }
  return RTX_SUCCESS;
}
// required by gcc
int __main( void ) {
    return 0;
}

int main( void ) {
  rtx_dbug_outs( "Hello!\n\r" );
}
```

Example Make File

```
# make file for hello.c in manual_5307.zip file
CC=m68k-elf-gcc
CXX=m68k-elf-g++
CFLAGS=-O2 -Wall -m5200 -pipe -nostdlib
LD=m68k-elf-gcc
AS=m68k-elf-as
AR=m68k-elf-ar
ARFLAGS=
OBJCPY=m68k-elf-objcopy
ASM=../start.s
LDFLAGS = -T../mcf5307.ld -Wl,-Map=hello.map

# Note, GCC builds things in order, it's important to put the
# ASM first, so that it is located at the beginning of our program.
hello.s19: hello.c
    $(CC) $(CFLAGS) $(LDFLAGS) -o hello.o $(ASM) hello.c
    $(OBJCPY) --output-format=srec hello.o hello.s19

clean:
    rm -f *.s19 *.o *.map
```

Tips

- Study the serial and timer example code in manual_5307v1a.zip for interrupt handling and C-callable Assembly
- Make sure your rtx compiles by using ecelinux/eceunix supplied cross compilers
- Define milestones of the project for easy time and project management
- Start early