Microcontroller: CPU and Memory

Amarjeet Singh

January 15, 2013

Partly adopted from EE202A, UCLA Slides by Mani Srivastava
Logistics

Programming EVK1100

- Guidelines for programming using AVR32 Studio on the course website
- You can use AVR Studio 6.0 – will share the guidelines on getting started with AVR Studio 6.0 soon as well
- Whatever you need to show for this week should be easily doable with the references on the webpage
- How many of you have got yourself the board issued?
- How many of you have played with the software/hardware?

Register to Piazza course page
- All discussions will happen on Piazza only

Groups for lab assignments
- Create your group of 2 students for lab assignments – will announce a link to fill in your group details on Piazza
- Submission through Google code repository only
Expectations from your first assignment:
- Be aware of different components on EVK1100
- Be aware of different utilities in whichever tool you use – What are different directories; How do you build a program; Where are resultant files stored etc.
- Know your code – what registers to set, how to communicate with LCD etc.
- Give the demo for the program to the TAs – Demo may be at a later stage but we will use the code committed in the repository for demo (so no commit after the deadline)

TA office hours:
Nipun: Tuesday 10 AM – 1 PM

Using Google Code Repository: Discuss during the break
Embedded Programming in C: Almost everyone at “1”

Different modules inside a microcontroller like Timers, Counters, ADC, memories: Quite a few at 2

What different interfaces exist for a microcontroller to talk to outside world e.g. SPI, UART: Quite a few at 2

Hardware development and processes therein e.g. soldering: Several people seem to have done it at least partially

Hardware debugging: A good mix of those who know a bit and those who have not done it

Optimizing system level constraints like power consumption, memory, synchronization: Everyone at 1 or 2 level
Development using Real Time Operating System (RTOS): Almost everyone at 1

Experience of deploying (making necessary connections, basic programming and setup to get it working) your own hardware system (other than a computer/laptop) from scratch: Very few have done it
Revision from last class

- What is a major discriminator between embedded and general purpose computing systems?
- What are different dimensions in embedded systems design space?
- What are typical components of a microcontroller?
- For each of 8-bit, 16-bit and 32-bit microcontrollers, what are the typical
  - Processing speed
  - Memory Capacity

<table>
<thead>
<tr>
<th>Bus Width</th>
<th>CPU Speeds</th>
<th>RAM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit</td>
<td>1-8 MHz</td>
<td>128-1K</td>
<td>512 to 10K</td>
</tr>
<tr>
<td>16-bit</td>
<td>4-25 Mhz</td>
<td>1K to 10K</td>
<td>10K to 128K</td>
</tr>
<tr>
<td>32-bit</td>
<td>10-100 Mhz</td>
<td>10K to 64M</td>
<td>128K to 512M</td>
</tr>
</tbody>
</table>
## MCU Comparison

### 8/16 bit

<table>
<thead>
<tr>
<th>Mfg</th>
<th>Device</th>
<th>Year</th>
<th>RAM (kB)</th>
<th>Flash (kB)</th>
<th>Active (mA)</th>
<th>Sleep (µA)</th>
<th>Wake (µs)</th>
<th>DMA (y/n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel</td>
<td>ATmega1281</td>
<td>2005</td>
<td>8</td>
<td>128</td>
<td>0.9</td>
<td>1</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>ATmega165P</td>
<td>2007</td>
<td>1</td>
<td>16</td>
<td>0.33</td>
<td>0.65</td>
<td>n/a</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>ATtiny13A</td>
<td>2008</td>
<td>0.064</td>
<td>1</td>
<td>0.19</td>
<td>0.15</td>
<td>n/a</td>
<td>no</td>
</tr>
<tr>
<td>TI</td>
<td>MSP430F1611</td>
<td>2004</td>
<td>10</td>
<td>48</td>
<td>0.5</td>
<td>2.6</td>
<td>6</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>MSP430F437</td>
<td>2004</td>
<td>1</td>
<td>32</td>
<td>0.3</td>
<td>1.1</td>
<td>6</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>MSP430F412</td>
<td>2001</td>
<td>0.256</td>
<td>4</td>
<td>0.2</td>
<td>0.7</td>
<td>6</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>MSP430F2002</td>
<td>2005</td>
<td>0.128</td>
<td>1</td>
<td>0.22</td>
<td>0.5</td>
<td>1</td>
<td>no</td>
</tr>
<tr>
<td>Microchip</td>
<td>PIC24F16KA102</td>
<td>2009</td>
<td>1.536</td>
<td>8</td>
<td>0.195</td>
<td>0.54</td>
<td>1</td>
<td>no</td>
</tr>
</tbody>
</table>

Active power consumption @ 1MHz

### 8/16/32 bit

<table>
<thead>
<tr>
<th>Mfg</th>
<th>Device</th>
<th>Arch (bit)</th>
<th>VCC (V)</th>
<th>RAM (kB)</th>
<th>Flash (kB)</th>
<th>Active (mA)</th>
<th>Stop (µA)</th>
<th>Standby (µA)</th>
<th>Wake (µs)</th>
<th>Standby</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel</td>
<td>ATmega1281</td>
<td>8</td>
<td>1.8-5.5</td>
<td>8</td>
<td>128</td>
<td>n/a</td>
<td>10 @ 8MHz</td>
<td>n/a</td>
<td>7.5</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>3.0-3.6</td>
<td>32</td>
<td>128</td>
<td>23.5 @ 60MHz</td>
<td>5.5 @ 12MHz</td>
<td>21.5</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>1.6-3.6</td>
<td>52</td>
<td>2x128</td>
<td>44 @ 84MHz</td>
<td>18.3 @ 24MHz</td>
<td>9.2</td>
<td>0.37</td>
<td>&lt;10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;500</td>
</tr>
<tr>
<td>STM</td>
<td>STM32F103RC</td>
<td>32</td>
<td>2.0-3.6</td>
<td>48</td>
<td>256</td>
<td>51 @ 72MHz</td>
<td>7.2 @ 8MHz</td>
<td>24</td>
<td>3.4</td>
<td>5.4</td>
</tr>
<tr>
<td>TI</td>
<td>MSP430F1611</td>
<td>16</td>
<td>1.8-3.6</td>
<td>10</td>
<td>48</td>
<td>n/a</td>
<td>0.5 @ 8MHz</td>
<td>n/a</td>
<td>2.6</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>1.8-3.6</td>
<td>16</td>
<td>256</td>
<td>n/a</td>
<td>1.3 @ 8MHz</td>
<td>8</td>
<td>1.7</td>
<td>5</td>
</tr>
</tbody>
</table>
Cypress PSoC

- Programmable System on Chip Core
  - M8C, 8051, Cortex-M3
  - Flash Memory, SRAM
  - Watchdog, multiple clocks
- Configurable Analog and Digital blocks
  - Similar to CPLD/FPGA
  - Blocks can be combined for ADC, Counters, Amplifiers
- Programmable routing and interconnects
- TiVo, Capacitive sensing of iPoD
Digital Signal Processors

- Similar to MCUs in architecture

- CPU Core optimized for complex numeric tasks
  - Basic execution unit: Multiply + Accumulate
  - Data intensive operations
  - Usually used as co-processor
  - Signal filtering
  - Video compression/Decompression

- Example: Analog Devices Blackfin
  - Runs embedded OS (uCLinux) - Does not need host CPU
  - 600 MHz and below

- Example: TI C64X
  - Used primarily for data encoding/decoding
  - Coupled with ARM Cortex-A8 (as host CPU)
  - 1 GHz and below
Communication Interfaces in Embedded Systems

- Communication is an important aspect of Embedded Systems
- Often contain specialized communication chips
- Wired
  - Interfacing with sensors and other system components
    - CAN - Controller Area Network
    - I²C - Inter-Integrated Circuit
    - SPI - Serial Peripheral Interface
    - UART - Universal Asynchronous Receiver Transmitter
    - USB - Universal Serial Bus
  - Communication with other embedded systems
    - Ethernet
- Wireless
  - Ever more useful due to lower cost and ease of installation
  - Many standards for short, mid, long range communication
Wireless Technologies

Short range
- IEEE 802.15.4, ZigBee Alliance
  - Home Automation
  - Sensor Networks
- z-Wave
  - Home Automation
- Bluetooth
  - Short range
  - Audio headsets

Mid range
- 802.11
- WiMax

Long range
- Cellular - GSM/CDMA
- Satellite
- Proprietary point to point links

Infrared
- Remote control
- Proprietary transceivers
Radio Choices for Wireless Embedded Nodes

\[ P(\text{in dBm}) = 10 \log \left( \frac{P(\text{in mW})}{1 \text{mW}} \right) \]

+20 dBm = 100 mW; 0 dBm = 1 mW
-3 dBm = 0.5 mW; -10 dBm = 0.1 mW

<table>
<thead>
<tr>
<th></th>
<th>802.11g</th>
<th>802.15.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chipset</td>
<td>Atheros 5006XS</td>
<td>CC2420</td>
</tr>
<tr>
<td>Output Power</td>
<td>16 dbm</td>
<td>0 dbm</td>
</tr>
<tr>
<td>Rx Sensitivity</td>
<td>-78 dbm@36 Mbps</td>
<td>-90 dbm@250 Kbps</td>
</tr>
<tr>
<td>Tx Power (Max Output)</td>
<td>1320 mW</td>
<td>57.42 mW</td>
</tr>
<tr>
<td>Rx Power</td>
<td>924 mW</td>
<td>65.01 mW</td>
</tr>
<tr>
<td>Total Power</td>
<td>2.24 W</td>
<td>122.43 mW</td>
</tr>
<tr>
<td>Effective Throughput</td>
<td>20 Mbps</td>
<td>125 Kbps</td>
</tr>
<tr>
<td>Efficiency (nJ/bit)</td>
<td>112</td>
<td>979</td>
</tr>
</tbody>
</table>
Peripherals in Embedded Systems

- Diverse set of peripherals to provide specific functionality
- Timers and Counters
  - Generate events at specific time or determine duration between two external events
  - Count pulses on some input signals
  - Watchdog timer - Failure to receive a signal in time will produce system reset
- Pulse Width Modulators (PWM)
  - Allows controlling time period and duty cycle
- Analog to Digital Converters (ADC)
  - Physical environment around the system produces analog values while the system understands digital values
  - Number of encoding bits and maximum voltage of input signal decides the resolution ($V_{max}/(2^n - 1)$)
Memory Interface in Embedded Systems

- Typically small (few KBs) on-chip memory
  - PIC24F Family – RAM up to 8KB, Flash up to 128KB
  - AVR xMega Family – Ram up to 32KB; Flash up to 384KB, EEPROM up to 4KB
  - AVR Atmega family - RAM up to 2 KB; Flash up to 32 KB; EEPROM up to 1 KB;
  - AVR 32UC Family – RAM up to 64 KB; Flash up to 512 KB; SDRAM Interface

- DMA Controller allows movement of data from one memory address to another without CPU intervention
  - Increases the throughput of peripheral modules
  - Reduce system power consumption (CPU can be in sleep mode while the data moves to/from peripheral)
Other Platforms We Have

- **uLEAP**: Developed at UCLA
  - MSP430 Microcontroller
  - Energy sensing circuitry on board
  - Bluetooth communication
  - Programming in uC Linux

- **TelsoB Motes** (one of the most commonly used platform for sensor network applications)
  - MSP430 microcontroller
  - Sensors on board

- **OpenPicus (Flyport)**
  - PIC microcontroller
  - Comes in Ethernet-WiFi version (working on GSM/GPS version as well)
  - To be used in course project
  - Programming in freeRTOS environment
Often for smaller code it is easier to just do the programming and test the code on the actual hardware itself.
Programming Languages and Compilers

- Many possible languages
  - Assembler (MCU specific)
  - C/C++, NesC
  - Java
  - Verilog / VHDL
  - Python

- Cross compiler translates high level code to MCU specific code instructions
  - Cross compiler usually runs on your development (host) system
  - Generated binary does not usually execute on host system
  - Needs an emulator/simulator to interpret
Programming Embedded Devices

Each class of devices is programmed slightly differently

8/16-bit class devices: ATiny, ATmega, AT xMega, PIC24F,
- Have a small on-chip hardware unit called a bootstrap loader (UISP, BSL)
- An external programming device communicates with the BSL through multiplexed pins, eg. the serial interface, USB
- These devices can also be programmed via JTAG pins
- Typically, the BSL will erase and overwrite the entire flash memory

32-bit class devices: Atmel 32 UC, Cortex-M3, ARM7, PXA27x
- Have a software bootloader that is programmed once using JTAG
- Bootloader has board specific drivers that can communicate with the host machine through a serial port or network connection
- Typically, only parts of the flash are erased and overwritten
- Once OS is booted, can use utilities like NFS to load applications into RAM
We will broadly discuss the details common across 8/16-bit class devices, specifically targeting AVR family (common in boards such as Arduino, xMegaA1) and PIC24F Family used in flyport boards you will be using for the project.

Many of the things overlap with 32 bit devices as well.
Wherever required, I will cover specifics of 32 devices that will be useful for you to learn.

References:
- AVR32 Family datasheet from Atmel
- PIC24F Family datasheet from Microchip
- AVR xMegaA1 datasheet from Atmel
Useful Resources

- Websites of original vendors

- Datasheets for the microcontrollers

- Application Notes from the vendor explaining how to do a specific task

- User guides
  - EVK1100 user guide
AVR CPU

- xmega A1 uses 8/16-bit AVR CPU
- Main purpose is program execution:
  - Access memory
  - Perform calculations
  - Control peripherals
- Harvard Architecture
  - Separate buses and memories for program and data
- Single level pipeline
  - Allows instruction to be executed in every cycle
**AVR CPU**

- Register file containing 32x8-bit registers
  - 6 registers can be used as 3 16-bit registers for data space addressing
- ALU supports arithmetic and logic operations
  - Supports both 8- and 16-bit arithmetic
- Program memory: In-system programmable flash
  - Up to 384 KB in our case
- Data Memory
  - Up to 32 KB SRAM
  - UP to 4 KB EEPROM
AVR CPU

- I/O memory space - All peripherals and modules addressed through Peripheral Module Address Map

### Table 30-1. Peripheral Module Address Map

<table>
<thead>
<tr>
<th>Base Address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GPIO</td>
<td>General Purpose IO Registers</td>
</tr>
<tr>
<td>0x0010</td>
<td>VPOR0</td>
<td>Virtual Port 0</td>
</tr>
<tr>
<td>0x0014</td>
<td>VPOR1</td>
<td>Virtual Port 1</td>
</tr>
<tr>
<td>0x0018</td>
<td>VPOR2</td>
<td>Virtual Port 2</td>
</tr>
<tr>
<td>0x001C</td>
<td>VPOR3</td>
<td>Virtual Port 3</td>
</tr>
<tr>
<td>0x0030</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td>0x0040</td>
<td>CLK</td>
<td>Clock Control</td>
</tr>
<tr>
<td>0x0048</td>
<td>SLEEP</td>
<td>Sleep Controller</td>
</tr>
<tr>
<td>0x0050</td>
<td>OSC</td>
<td>Oscillator Control</td>
</tr>
</tbody>
</table>

Multiplier/DES
PIC 24F Core Architecture

- 16 16-bit working registers
- Harvard Architecture
- 16-bit ALU
  - Enhanced with multiplier and divide hardware
Microcontroller Memory

- Primarily three types of memory:
  - Program Flash Memory: Used for storing the program
  - Data Memory of SRAM: Used for temporary storage of data values
  - EEPROM Memory: Used for permanent storage of data values or initial parameters for the microcontroller

- AT xmega A1
  - Up to 384 KB Flash (128 KB in our board)
  - Up to 32 KB SRAM (8 KB on our case)
  - Up to 4 KB EEPROM (2 KB in our case)

- PIC 24F
  - Up to 128 KB Flash (maximum in our case)
  - Up to 8 KB SRAM (maximum in our case)
  - No internal EEPROM
Each register is also assigned a data memory address mapping them directly to the first 32 locations of user Data space.

R26 to R31 act as 16 bit address pointers for indirect addressing of user space.

64 I/O memory space can be accessed directly or Data space locations following register files i.e. from 0x20 - 0x5F.

Extended I/O space from 0x60 - 0xFF in SRAM.
Xmega A1 - Flash Program Memory

- In-system reprogrammable
- Each location is 16-bits wide
  - All instructions are 16/32 bits wide
- For software security, divided into Boot Flash Section and Application Flash Section
  - Boot Flash Section is for Boot Loader Program for programming the Application Flash Section using on-board UART
- Endurance of at least 10,000 write/erase cycles
- If no boot loader is needed, entire flash is available for application code
Xmega A1 – Programming Sequence

- Usually programmed through JTAG interface
  - Need specific devices for programming
  - Follows a certain programming sequence – generated by ISP device

- Without any JTAG interface, microcontroller executes the main code

- Main code can be programmed to have a boot loader
  - A piece of program that puts the microcontroller in the programming mode for a short time and waits for the programming signal (on a separate set of pins)
  - If no signal is received, then the actual program executes
Xmega A1 - Data Memory

- First 32 locations for register file
- Next approx. 4 KB for I/O Memory
- Next up to 4 KB for EEPROM
- Next locations for internal data SRAM
- Finally address space up to 16 MB of external memory
  - Supported through External Bus Interface (EBI)
- Direct addressing reaches the entire data space
- Timing:
  - Read/Write to I/O Memory takes 1 clock cycle
  - EEPROM page write – 1 clock; EEPROM page read – 3 clock cycles
  - Write to SRAM – 1 clock; Read from SRAM – 2 clock cycles
  - External memory – multi clock cycle
Instruction Execution Timing

Parallel Instruction Fetches and Instruction Executions

Single Cycle ALU Operation
Direct Memory Access Controller (DMAC) to move data between memories and peripherals
- Uses the same data bus as CPU to transfer the data
- Can read from memory mapped EEPROM but cannot write to it
- Cannot access Flash
- 4 channels that can be configured independently
  - Each channel can perform data transfer in blocks of 1 byte – 64 KB
- Since AVR CPU and DMAC use the same data bus, block transfer is divided into smaller burst transfers
  - Selectable to burst of 1, 2, 4, or 8 bytes
Different bus masters (CPU, DMAC write, DMAC read etc.) can access different memories at the same time.

E.g. CPU can access the external memory while DMAC is transferring data from internal SRAM to I/O memory (USB module acting as master).
In case several masters request access to the same bus:
- Bus master with ongoing access
- Bus master with ongoing burst
- Bus master requesting burst access (CPU has priority)
- Bus master requesting bus access (CPU has priority)
What did you learn today?

On a small sheet of paper, give me a short (2-3 sentences) description of what you learned today.

You can use it to give any broad comments on the class as well.