SELECTING A MICROCONTROLLER DEVELOPMENT SYSTEM FOR A LABORATORY IN A REAL-TIME INTERFACING COURSE

Witold Kinsner

Department of Electrical and Computer Engineering
University of Manitoba, Winnipeg, MB, Canada R3T 5V6
and Telecommunications Research Laboratories, TRILabs
witold.kinsner@umanitoba.ca

Abstract: *The Canadian Engineering Accreditation Board (CEAB) requires Engineering Schools to update their laboratories on a regular basis. Selecting an appropriate hardware and software development environment for the lab is a difficult task because it triggers a redevelopment of the laboratory experiments, a process that requires solutions to complex technical and pedagogical intertwined problems. This paper presents an overview of the selection process of a specific microcontroller development board and a software development environment for a laboratory in an undergraduate course on interfacing of microcontrollers, microprocessors and microcomputers for real-time systems.*

Keywords: Microprocessor and microcontroller interfacing, real-time systems, new laboratory environment, processor selection process.

1. INTRODUCTION

This paper provides an overview of the selection process of a specific microcontroller development board and a software development environment for a laboratory in an undergraduate course on interfacing of microcontrollers, microprocessors and microcomputers for real-time systems [14], [15], [4], [28]. The Canadian Engineering Accreditation Board (CEAB) requires Engineering Schools to update their laboratories on a regular basis. Selecting an appropriate hardware and software development environment for the lab is a difficult task because it triggers a redevelopment of the laboratory experiments. Such a process is expensive and poses complex technical and pedagogical intertwined problems.

The selection is further complicated by the number of development systems available today, from the inexpensive Arduino and Raspberry Pi boards to more expensive Freescale HCS12 boards. Which one could serve the students best?

The difficulty of selecting a proper lab board is further compounded by the diversity of topics included in such a course on microcontroller interfacing. The course must cover several classes of topics, including (i) an introduction to real-time computing, architectures, processors, and

technologies, (ii) bus architectures, (iii) digital I/O synchronization (e.g., different classes of interrupts, *direct memory access* (DMA), context switching, and the major buffering techniques), (iv) *digital-to-analog* (D/A) and *analog-to-digital* (A/D) signal conversions and converters (e.g., parallel flash, serial counting, single ramp, tracking, *successive approximation* (SA), integrating (dual slope, quad slope), *voltage-to-frequency* (V/F)-based converters, and delta-sigma converters), and (v) interfacing aspects in data communications (e.g., major wired and wireless data communications protocols, data encoding, signal conditioning, forward error detection and correction). The previous implementations of our lab experiments in this course utilized various microprocessors and microcontrollers, including the 6800, 68000, 6805, HC08, and the HC11, all in boards designed and implemented by us in house [17].

The main criteria for selecting the new processor and the associated development environment included (i) modern architecture that covers many other processors, (ii) richness and flexibility for diverse interfacing schemes, (iii) features for real-time interfacing, (iv) richness of the instruction set architecture and implementation, and (v) availability of a reliable the development environment. To search for the proper system, we have reviewed most of the major microcontrollers (including Freescale, Microchip, Cypress, Texas Instruments, Atmel, and Parallax Semiconductor) and development boards (including Arduino Uno, Leonardo, Due, Mega, and Mint Duino, as well as Raspberry Pi, Beagle Bone, Texas Instruments, and iNEMO).

For the new labs, we have selected the HCS12 microcontroller development board. New laboratories have been developed and presented to the students once, followed by numerous improvements and corrections of the lab manual. A new tutorial on the HCS12 has also been developed [16]. Selection of topics in the tutorial, their presentation and structure are all quite novel, and should be helpful in reinforcing the knowledge of this important pipelined microcontroller that is compatible with many legacy microprocessors and microcontrollers. This paper compares the HCS12 to many other popular microcontrollers available today. Furthermore,

embodiments of various available educational and development boards today are also discussed.

2. HCS12 OVERVIEW

2.1 Background on the HCS12

The HCS12 (MC9S12C128) single-chip *microcontroller* (μ c) is part of the 48/52/80-pin flash-based MCU family for a wide range of cost and space sensitive general-purpose industrial and automotive network applications, and was released in 2001. It is a direct successor to the first 16-bit 8-MHz HC12 μ c with a flash memory, introduced by Motorola in 1996, which in turn was a direct successor to the HC11 μ c, as well as the 6801 and 6800 μ Ps. Its instruction set makes it one of the most efficient processors for high-level language programming, including C. The name HC11 and HC12 comes from the technology used to build the chip (i.e., *high-density complementary metal-oxide semiconductor*, HCMOS, integrated circuit). The original HC12 was designed to run at up to 8 MHz. The main difference between the HC12 and the HCS12 is that the latter uses a smaller 0.25- μ m and 0.18- μ m line-widths, and could operate not only at a higher speed (25 and 50 MHz, respectively), but also at a lower power. Both the HC12 and HCS12 have the same instruction set.

The HCS12 μ c family includes several chips with different sizes of flash, *electrically-erasable programmable read-only memory* (EEPROM), and *static random-access memory* (SRAM). For example, the MC9S12C128 has 128 KiB (binary kilo, pronounced as kibe) flash and 4 KiB SRAM; MC9S12C96 has 96 KiB flash and 4 KiB SRAM; MC9S12C64 has 64 KiB flash and 4 KiB SRAM; MC9S12C32 has 32 KiB flash and 2 KiB SRAM. Notice that the “C” version of the HCS12 (as used in the lab) has no internal EEPROM.

Together with many other μ cs, the HCS12 is well suited for embedded applications (in which the presence of the μ c is transparent to the user) [29].

2.1 HCS12 Functionality

The block diagram of the HCS12 (e.g., [6]–[12]) contains all the functional units and data ports on a single chip. The HCS12 used in our lab (MC9S12C128) includes a single 16-bit core with a CPU12, 128 KB of flash EEPROM, 4 KB of RAM, as well as a few standard on-chip peripherals, including an asynchronous *serial communications interface* (SCI), a *synchronous serial peripheral interface* (SPI), an 8-channel 16-bit *timer module* (TIM), a 6-channel 8-bit *pulse-width modulator* (PWM), an 8-channel 10-bit *analog-to-digital converter* (ADC) (labelled as the *analog-to-digital*, ATD, module), and a *controller area network* (CAN 2.0A/B) labelled as Motorola scalable CAN (MSCAN). All HCS12

microcontrollers have 16-bit data path throughout. To operate, the HCS12 also has a voltage regulator, phase-locked loop clock generator, and other utilities.

3. HCS12 FUNCTIONAL BLOCKS

The HCS12 blocks include: (i) dual-output voltage regulator, (ii) CPU, (iii) memories and memory space map, (iv) data communications, (v) timer module, (vi) pulse width modulator, (vii) analog/digital conversions, (viii) interrupts, (ix) clock and reset generator, and (x) ports and systems integration module.

3.1 Dual-Output Voltage Regulator in the HCS12

The HCS12 requires an external supply voltage (VDDR, VDDX) in the range from 2.97 to 5.5 V. Its internal band-gap-based voltage regulator provides two separate VDD voltages (2.5 V), differing in the amount of current that can be sourced. The regulator has *low-voltage detect* (LVD) with *low-voltage interrupt* (LVI), *power-on reset* (POR), and a *low-voltage reset* (LVR). For more details, see [7], [Cody08; Ch. 11].

3.2 CPU in the HCS12

The HCS12 *central processing unit* (CPU) includes a single *arithmetic-logic unit* (ALU), two 8-bit accumulators (ACCA, ACCB) that can be combined into one 16-bit double accumulator D (ACCD), two 16-bit index registers (IX, IY), a 16-bit stack pointer (SP), and a condition-code register (CCR). Those elements are all available to the programmer (see [16, Fig. 3.1]).

The *condition code* (CC) register stores the standard flags (i) arithmetic carry (C) for extended-precision calculations, (ii) two's-complement overflow (V) to recover from overflows by saturation or other remedies, (iii) zero (Z) to finish loops and other operations, (iv) negative (N), (v) I interrupt mask (I) to disable all interrupt requests, (vi) half-carry (H) for decimal arithmetic, (vii) X interrupt mask (X) to disable the conditional XIRQ_L and other external interrupts, and (viii) stop instruction disable (S).

Although not available to the programmer, the *program counter* (PC) is also included in the model to show the extent of its address space. The other elements not available to the programmer, but essential to the operation of a CPU are (i) the *address register* (AR) for storing all the physical addresses during instruction execution, (ii) *instruction register* (IR) to store the current instruction during its decoding and execution, (iii) instruction decoder, and (iv) sequencer controller to generate all the signals required to execute an instruction. For more details, see [7], [Cody08; Ch. 4].

3.3 Memories and Memory Space Map of HCS12

The standard memory space for the HCS12 is 64 KiB (16-bit addresses) [16, Fig. 9a]. It is logically subdivided into four 16-KB blocks. The SRAM is placed within the bottom block of the memory because the 1,024 registers occupy the space from \$0000 to \$03FF. The remaining 3 KiB of the SRAM are used for variables and the stack. The remaining three 4-KiB segments of the first 16-KiB block are not used [16, Fig. 3.2]. The next three 16-KiB blocks may be used for the flash memory. The top 16-KiB block contains 256 **interrupt vectors** located at \$FF00 to \$FFFF.

If the flash memory exceeds the available address space, the flash has to be paged. The third 16-KiB block from \$8000 to \$BFFF is used for paging. The paging is done through an extension register PPAGE containing values \$38 to \$3F, as shown in (see [16, Fig. 3.3b]). The memory management is done through a *memory management centre* (MMC) in the CPU by translating and manipulation of the logical addresses into physical addresses [5].

The 128-KiB physical flash memory is arranged in 1,024 rows, each containing 128 bytes [16, Fig 3.3]. A set of eight (8) rows constitutes a sector. Each sector can be erased independently. The memory can also be erased. When erased, a bit reads 1. When programmed, a bit reads 0. The read operation can be done on a byte, aligned words, and misaligned words. The erase and program is done with a single voltage.

3.4 Data Communications in the HCS12

The HCS12 provides three means of communicating data between two devices: (i) a single asynchronous serial communications interface (SCI), (ii) a single synchronous serial peripheral interface (SPI), and controller-area network (CAN 2.0A/B) labelled as *Motorola scalable CAN* (MSCAN) [16, Fig. 2.1]. The first technique is similar to the well-known universal asynchronous receiver-transmitter (UART). The second technique was developed by Motorola in 1980 for high speed communications (10 Mbps) over short distances, which is faster than the I2C protocol (3.4 Mbps), and is summarized in Chapter 2, Part 1 of the class notes [Kins13a]. The third technique was introduced by Robert Bosch to the Society of Automotive Engineers (SAE) in 1986 to carry short packets (of up to 8 bytes) in harsh environments such as automobiles [4; Ch. 16]. The first two techniques are used in the lab, and will be described next.

3.4.1 Asynchronous Serial Communications Interface (SCI): The SCI provides full-duplex, asynchronous, serial data communications between two devices on two wires: the serial transmit data (TxD), and serial receive data 8- or

9-bit data (RxD). Each line has a common reference [16, Fig. 3.4a].

The SCI has its own programmable baud generator and double buffer (to allow for slower read times). The baud rate clock is synchronized with the bus clock, and drives the receiver. The receiver samples each bit 16 times (to provide a good immunity to noise). The format of the character sent has 11 bits, with a start bit (LOW), 8-bit data, one parity bit, and one stop bit (HI). When transmitting 9-bit data, bit T8 in *SCI data register high* (SCIDRH) is the ninth bit (bit 8). The SCI can also operate with a single wire (on the TxD).

3.4.2 Synchronous Serial Peripheral Interface (SPI):

The SPI provides full-duplex, synchronous, serial data communications between two devices on two wires: the *master output slave input* (MOSI) and the *master input slave output* (MISO) [16, Fig. 3.4b]. The synchronization is done through a common local clock line, the *serial clock* (SCK). The polarity and phase of the clock can be programmed. The slave is selected by the signal slave select, (SS_L). The SPI also has a double buffer [16, Fig. 3.5]. Other examples of SCI, SPI and CAN are shown in [16, Fig 3.6].

3.5 Timer Module in the HCS12

The timer module (TIM) provides many timing signals needed for interfacing. It is based on 16-bit counter driven by the system clock, called the bus clock, and provides signals on 8 channels. The channels can be configured as either output to produce desired output waveforms, or input captures to capture the time when an event occurs on the channel. It appears to be the most complex module on the HCS12.

The basic function of TIM is to provide an interfacing clock that is a scaled down version of the bus clock (with possible divisions by 2, 4, 8, 16, 64, and 128). The timer overflow can be used for periodic interrupts (215 clock cycles, or 4.096 ms at 8 MHz bus clock). For more precise timing of interrupts, the output compare can be used. TIM can also generate pulses of any desired width and aspect ratio (i.e., the ratio between the pulse duration to the period). For more information, see [7], [4; Ch. 14].

3.6 Pulse Width Modulator in the HCS12

The pulse width modulation (PWM) module has six channels with independent control of left- and center-aligned outputs on each channel. Each of the six PWM channels has a programmable period and duty cycle, as well as a dedicated counter. It allows four different clock sources to be used with the counters. Each of the channels can create independent continuous waveforms with

software-selectable duty rates from 0% to 100%. For more information, see [7], [4; Sec. 14.10].

3.7 Analog/Digital Conversions in the HCS12

The analog-to-digital (ATD) module has a single 8-bit or 10-bit successive-approximation (SA) analog-to-digital converter (ADC) with ± 1 least-significant bit (LSb) accuracy. The ADC guarantees linear conversion over the full temperature range, with no missing codes. Both the conversion time and aperture time are programmable. At 2-MHz ADC clock, the 8-bit conversion is done in 6 μ s, and the 10-bit conversion is done in 7 μ s. The eight input channels are multiplexed to a single sample-and-hold (S&H) analog memory capable of holding the input voltage for the duration of each conversion. The definitions of all the terms used here can be found in this course [14]. For more information, see [7], [4; Ch. 17].

3.8 Interrupts in the HCS12

Real-time systems use interrupts to service their various asynchronous events. “Asynchronous” implies that the event can happen at any time, without any regard to the internal operation of the μ c. This subject is one of the most important in this course on interfacing. The course classifies an interrupt as (i) conditional (serviced when the μ c can service it), and unconditional (serviced as soon as possible in the system). Priorities must also be assigned to interrupts when more than one interrupt can happen simultaneously. Priority arbitration techniques require enabling and disabling of interrupts. Nesting of interrupts is also required in most systems. Interrupts also require interrupt vectors, located at the top of the HCS12 memory space. An example of vector assignment table is given in [4; Appendix F]. Those and other topics are discussed in depth in this course. Cady also discusses the HCS12 interrupt handling [4; Ch. 12].

3.9 Clock and Reset Generator in the HCS12

The clock and reset generator (CRG) provides critical signals to the HCS12, including: (i) the system clock system reset generation on special conditions, (ii) real-time interrupts, and (iii) the *computer-operating-properly* (COP) watchdog timer interrupts. The system clock is derived from either an external crystal oscillator, or an internal phase-locked loop (PLL).

The COP (or in general, a watchdog timer) is used to monitor software execution an HCS12-based system. When the COP counter is enabled, it starts counting down to zero. When zero is reached, it means that something has gone wrong with the execution, and that the program should be reset. If the software runs correctly, it can rest the watchdog before it reaches zero (it is the “kicking the dog” action to keep it awake).

The watchdog must have its own clock, and be independent of the μ c it is designed to monitor. However, since the COP must first be set and also disabled (to test individual modules without the watchdog), μ cs have procedures to prevent the watchdog from becoming disabled accidentally. For example, the HCS12 requires the program to write \$55 followed by \$AA to reset it (pattern-sensitive protection). The ATmega16 sets two 1s into a register, followed by a reset within four cycles (time-sensitive protection). For more information, see [4; Chs. 21 and 20].

3.10. Ports and Systems Integration Module in the HCS12

The HCS12 functional blocks communicate with the outside world through ports that are coordinated by the *system integration module* (SIM) [4; Fig. 11.1]. The coordination is done through 1,024 registers located at the bottom of the SRAM (locations \$0000 to \$03FF). Since the registers are located in the SRAM, they must be initialized before any operations commence. The nine ports in the HCS12 include:

1. Port A, B, and E are related to the HCS12 core. Ports A (MSB) and B (LSB) are the multiplexed address and data buses. Port E carries the main control and interrupt signals;
2. Port T carries the TIM signals (the PWM module may also be routed to Port T);
3. Port S carries the SCI signals;
4. Port M carries the MSCAN and SPI signals;
5. Port P carries the PWM signals;
6. Port J pins can be used as external interrupt sources and standard I/O signals; and
7. Port AD is dedicated to analog signals.

Each port is controlled through a data direction register (DDR) [16, Fig. 2.1]. The I/O pins can provide 5 V (with two drive strengths to reduce electromagnetic interference at low drives), and selectable pull-up or pull-down capabilities (to reduce the latch-up conditions in the CMOS circuits).

The pin interpretation on Ports A and B in the HCS12C128 is illustrated in [16, Fig. 3.7]. Both ports are used to transfer 16-bit addresses, or 16-bit data. Narrow data are transferred through Port A.

The data direction register (DDR) controls the direction of the port, as shown in [16, Fig. 3.8]. The buffers W and R in the data path are three-state buffers, controlled by a phase splitter PS (to eliminate data skew). When $DDR_{B0} = 1$, the write buffer (W) is active, and the port bit $PTB0$ is set for output. On the other hand, when $DDR_{B0} = 0$, the read buffer (R) is active, and the port bit $PTB0$ is set for input.

4. HCS12 OPERATIONS, INSTRUCTION SET, AND ADDRESSING MODES

As any processor, the HCS12 has several (7) operating modes, a large instruction set (over 1 k), and a number (8) of addressing modes.

4.1 HCS12 Operating Modes

The HCS12 can function in seven (7) operating modes. A particular mode is selected by external signals on three I/O pins on the rising edge of the RESET signal. The main modes include (e.g., [7], [4; Ch. 20]):

1. **Normal single-chip mode.** All the internal functional units can be used, except for the power-on-reset (POR) and the external RESET. All the I/O pins can be used for interfacing, as no external addresses or data are generated in this mode.
2. **Normal expanded mode.** If the internal RAM is too small, this mode allows for a larger external RAM to be used outside the μc , at the expense of losing the Port a and B, and a part of Port E for addresses, data, and control signals such as the read/write (R/W) signal.
3. **Special single-chip mode.** In this mode, the background debugger mode (PDM) is activated, and uses special hardware and firmware inside the HCS12 for user debugging.

4.2 HCS12 Instruction Set

The HCS12 has over 1,000 instructions that can be grouped into 17 categories of 188 distinct operations. Learning all the instructions are facilitated greatly by understanding the categories of instructions because they also appear on other processors. The categories are:

(1) Load registers, (2) Store registers, (3) Transfer / exchange registers, (4) Move memory contents operations, (5) Decrement/increment operations, (6) Clear/set operations, (7) Arithmetic operations, (8) Logic operations, (9) Rotate/shift operations, (10) Data test operations, (11) Conditional branching, (12) Loop primitives, (13) Jump/branch operations, (14) Conditional code, (15) Interrupt instructions, (16) Miscellaneous operations (e.g., NOP, GGND), (17) Fuzzy-logic-oriented and special math instructions.

4.3 HCS12 Addressing Modes

The HCS12 has all the standard addressing modes, and some additional modes. They include: (1) **inherent** (modification of bits in registers, implied by the name of the instruction; no operand), (2) **immediate** (the operand

carries the data), (3) **direct** (the operand carries the short 0-page address of the data), (4) **extended** (the operand carries a full pointer to the full address of the data), (5) **indexed** (the operand carries an offset that is added to the index register to form the physical address of the data, with pre- or post incrementation/decrementation), (6) **indexed indirect** (the operand carries a pointer to the offset that is then added to the index register to form the physical address of the data), and (7) **relative** (the operand carries either a short offset -128 to $+127$, or long offset $-32,768$ to $+32,767$, relative to the current location that calculates to the effective address to branch to).

4.4 HCS12 Instruction Formats

This section describes the HCS12 instruction format in the context of other formats possible. The zero-address instruction format is the simplest. The instruction has the OPCODE field, and no operand field at all [16, Fig. 4.1a]. The OPCODE tells the processor what to do with the operands stored on the system stack. For example, an ADD instruction takes two operands from the stack, adds them in the ALU, and the result is placed back on the stack. This CPU architecture is called the stack architecture. The simplicity of this architecture has a price in the number of operations to be performed to finish a task, as shown later in this section.

The HCS12 instructions have a single-address format (i.e., there is a single operations code, OPCODE, which is followed by a single operand that could be either a data, or a direct address, or an indirect address, or an offset for indexing, or a branch) [16, Fig. 4.1b].

This format requires that the first operand be stored in an accumulator prior to the instruction referencing the second operand. This format is also destructive because the destination of the result is not specified in the instruction, and must be assumed to go to the accumulator, thus destroying its previous contents. This CPU architecture is called the accumulator architecture.

In the two-address instruction format [16, Fig. 4.1c], the operands carry the data or addresses of the two operands, thus bringing them to the ALU faster. However, the result must still be assumed to go to the one of the sources, thus destroying it. The 68000 μP uses the second address for the destination, while ATmega16 μc uses the first address for the destination. The cost of this faster operation is a dual internal bus, as discussed in class [Kins12].

In the three-address instruction format [16, Fig. 4.1d], both operands can be brought to the ALU simultaneously, and the results goes to the independent location. This processing is not destructive. The cost of this faster and non-destructive operation is a triple internal bus, as discussed in class [Kins12].

4.5 Why Pipelined Architecture in HCS12?

In the past, von Neumann microprocessor (μ P) architectures such as the MC6800 processed instructions in a sequential manner, as shown in [16, Fig. 4.2].

An instruction had to be fetched from an external ROM, and brought into the CPU, decoded, and executed there. Since the fetch was done by the controller / sequencer in a constant time, this time was called the *machine cycle* (MC), and was independent of the clock frequency; that is, the operation was completed in n machine cycles. On the MC6800, MC = 1 clock cycle.

In the above environment, the instruction execution time is variable, depending on the instruction type and its addressing mode selected. For example, the inherent addressing mode requires 1 MC to execute, while the direct addressing mode requires 2 MC, and an interrupt requires 14 MC [22].

An improvement in the execution speed of instructions can come from three main changes: (a) Harvard architecture because both the instructions and operands are brought to the CPU simultaneously, (b) RISC architectures because the instructions have a constant size, (c) Pipelining which allows overlap of the fetch and execute phases.

The HCS12 cannot use the first two solutions (Harvard and RISC) because it has to be compatible with the older generations of μ Ps such as the MC6800, 6801, 6808, 68HC11, and HC12. So, pipelining is the only option. Pipelining is the overlapping of the execute phase of the current instruction, while fetching of the next instruction, as shown in [16, Fig. 4.3].

In the simplified example of [16, Fig. 4.3] (the fetch time is the same as the execution time), three pipelined instructions execute within the time of two non-pipelined instructions.

To achieve pipelining in some of the legacy processors, an instruction queue (a buffer) was placed in the CPU to prefetch a block of instructions. Since the μ cs have the RAM and ROM memories on the same chip, pipelining is easier to implement. Since the fetch phase is no longer as prominent as in the legacy μ Ps, and since the CPU clock cycle governs the pipelining, the time measure is relegated to the CPU clock cycle. Notice that the CPU clock cycle does not have to be the same as the μ c clock cycle. In fact, the HCS12 CPU clock frequency is half of its chip clock frequency. For example if the μ c clock frequency is 10 MHz, the CPU clock frequency is 5 MHz.

Pipelining provides a clear advantage over the sequential processing if the program executes successive instructions. However, if a displacement instruction is executed (such as a short branch or long jump), the pipe must be reloaded. This is called a branch penalty. Conditional branching (e.g., BRSET) still executes in one CPU clock cycle, if the condition is not met.

5. HCS12 PROGRAM DESIGN AND PROGRAMMING ENVIRONMENT

5.1 Development Cycle

Developing embedded products follows a development cycle. Such a cycle often includes the following steps: (i) product specification, (ii) partitioning of the design into its software and hardware components, (iii) iterative refinement of the partitioning, (iv) interdependent hardware and software design tasks, (v) integration of the hardware and software components, (vi) product testing and release, and (vii) product maintenance and improvements. Although there are many models for the development cycle (for hardware, software, and codesign), there are some patterns that good designs use [18].

Berger [2] provided an illustrative diagram (e.g., [16, Fig. 5.1]) linking the time spent on the development and the cost of fixing a bug or defect. The cost increases exponentially. This good practice requires identifying such defects early in the cycle.

5.1 Codesign

Codesign of hardware and software is the hallmark approach in the courses provided to our students in Electrical and Computer Engineering. Problem specifications must be understood first. It should be followed by design specifications. As hardware must be designed before implementing it, a program must be designed before coding it. Design tools (i) must be easy to use, (ii) must support structured programming, (iii) should make the design transparent to the designer at many levels, and (iv) should facilitate good documentation, and should facilitate debugging and testing.

5.2 Programming Environment: Compiler, Assembler, Linker, and Debugger

In the past, programmers developed programs at the lowest **machine level** (1s and 0s). It was quickly augmented by **assembly programming** whose mnemonics and symbolic addresses could be read by many more programmers. **Assemblers** translate instructions into the machine level, according to the mnemonics, as well as pseudo-operations and directives. An absolute assembler performs the translation exactly as intended by the programmer. However, when the location of the final code cannot be known in advance, the intermediate code has to be relocatable to an appropriate final physical location. Such **relocatable assemblers** and **linkers** have been developed to help in making the code relocatable and linking it to other modules optimally. A linker combines relocatable object files to produce a target absolute file designated for the microcontroller's memory.

Macroassemblers have also been developed to increase reusability of frequently-used assembly instructions.

CodeWarrior has developed a Development Studio for the Freescale HCS12 μ c. The Studio includes an HCS12 assembler that can run on different platforms, cross-assembles the code into a format that can load into the HCS12 memory. One of the formats is the S-Record. For more information, see [4; Ch. 5 (assembler) and Ch. 6 (Linker)]. Figure 5.2 [16] shows common steps in software translation for the HCS12.

If the source code is in C or some other language, it has to be compiled to the assembly code (.asm). The assembler translates the files to an object code (.o) that is linkable. All the available object-code files are linked together by a linker. The resulting code in a loadable format (e.g., .S19) is loaded by a loader into the target machine. The diagram also shows the cycles of fixing bugs, if they are uncovered during the process.

5.3 Programming Environment: Embedded Programming in C

The knowledge of assembly programming is essential to develop optimal system. However, the program development can be improved by using a high-level language (C or C++, or C#) to develop the code. Various functional modules can be developed independently. A compiler/assembler produces the corresponding object files that can be linked to build the final target application. The *Code Warrior Integrated Development Environment* (IDE) allows for such a development. For more information, see [4; Ch. 6 (Linker) and Ch. 10 (C development)], [19].

5.4 Python and Its Libraries

Programming in Python has been gaining many practitioners lately. The Python libraries (v.2.x and 3.x) are very extensive, and can help in rapid development of applications. Some practitioners say that MatLab functionality can be repeated in the Python environment.

6. THE LAB/PROJECT BOARD

6.1 Project Development Board Overview

The Freescale PBMCUSLK project board ([16, Fig. 6.1], [25]) is designed as a common multi-course platform to speed the learning curve of users. It uses application modules ([16, Fig. 6.2], [25]) that can be plugged in on the left side of the board. The application modules include the HCS12 (APS12C128SLK [1], as used in our class), the HCS08, the Radio Frequency transceiver (AP13192USLK [1], [3]), and many others. One can also build a custom application module. The modules are fully functional with or without this 8.5" x 11" project board.

Learning the project board does not have to be difficult. We should start from the short quickstart guide [20]. The technical specifications of the board are provided in the MCU Project Board document [21]. It is important to study the feature of the board [21; pp. 5-6], including on board voltage regulators (5 V @ 500 mA; 3.3 V @ 500 mA; ± 15 V @ 50 mA; with the input voltage of 9 V @ 1.2 A). It has a dual-row header sockets placed around the prototyping area to provide convenient access to all on-board features. The Getting Started application note [13] is also critical in learning the details of interfacing with the board, including setting up the board. Other applications notes are related to the HCS12 (e.g., the MC9S12C32 [23]). Engineering Bulletins are also helpful (e.g., switching between μ cs with different characteristics, [26]).

Within a single type of an application module, we may have several of its versions, depending on the size of the μ c. For example, Figs. 6.3 and 6.4a [16] show the newer HCS12C128 module (with a 128 KiB flash) that is much larger than the older HCS12C32 module (with a 32 KiB flash), as shown in Fig. 6.4b. In the past, our lab used the latter module.

7. OTHER POPULAR MICROCONTROLLER FAMILIES

7.1 Other Microcontrollers (μ cs)

In addition to **Freescale** with numerous microprocessors and μ cs (e.g., 68HC05, HC11, HC12, HCS12), there are many companies that make microcontrollers, with each family having different properties such as architecture, speed, low power, and performance. The companies include very large companies such as Microchip (PIC, dsPIC), Cypress (PSoC), TI (MSP430), Atmel (AVR), and smaller such as Parallax (BasicStamp, Propeller). The microcontrollers have been popularized through the introduction of small affordable boards such as Arduino and Raspberry Pi, as described next.

Microchip makes a wide range of the PIC and dsPIC μ cs. The cores have Harvard architectures. The PIC18 family includes 8-bit microcontrollers, running in the range from 1 to 40 MHz. The PIC24 family includes 16-bit microcontrollers. The dsPIC30 family includes 16-bit μ cs optimized for DSP operations involving multiplications and divisions. Such operations take 40 cycles on a PIC18, but only 4 on a dsPIC. These μ cs are taught in our programs.

Cypress makes the PSoC (Programmable System on Chip). They have programmable digital and analog blocks that can be linked to form entire systems such as common peripherals. They are useful for rapid prototyping. They have three lines: PSoC 1 (for basic circuits), 3 and 5 (very complex systems).

Texas Instruments (TI) makes the MSP430, a 16-bit low-power microcontroller family. They're also very

commonly used in industry, especially in the aerospace applications. Their inexpensive LaunchPad development board (\$4.30) has helped many new users in developing their skills. We have offered many workshops on the Arduino and TI boards through the University of Manitoba IEEE Student Branch.

Atmel makes the AVR microcontrollers [30] and the ARM microcontrollers [31]. The original 8-bit Harvard-architecture processor was developed at the Norwegian Institute of Technology in Trondheim, Norway, in 1996, and implemented by the Nordic VLSI there. The name AVR stands for Alf (Egil Bogen) and Vegard (Wollan)'s RISC processor. Nordic VLSI was acquired by Atmel. The μ cs are grouped into six classes: (i) tinyAVR (or ATtiny), (ii) megaAVR (or ATmega), (iii) XMEGA (or ATxmega), (v) application-specific AVR, (v) FPSLIC (or AVR with FPGA), and (vi) AVR32. The 8-bit tinyAVR family has been used in many projects (e.g., the AVR Programmer is available from SparkFun [27]). The megaAVR family is used in Arduino Uno and Leonardo, as well in automotive applications such as security, safety, powertrain and entertainment systems (e.g., BMW, Daimler-Chrysler and TRW).

Atmel's ARM μ cs have a different genealogy. The original 32-bit ARM (Acorn RISC Machine, and now Advanced RISC Machine) microprocessor was introduced in 1983, and was used in the popular BBC microcomputer and Apple's Newton personal digital assistant (PDA). The newer versions of the 32-bit machines (ARMv6 and ARMv7) and the 64-bit ARMv8 are used in many computers such as iTVs, iPods, iPads and iPhones. For example, the iPhone 5S uses the ARMv8. The ARM architecture is also licensed to other companies to develop their versions. Examples include the Qualcomm's Snapdragon (using their Krait or Scorpion CPUs), Nvidia's Tegra, Marvell's XScale, Freescale's i.MX and Texas Instruments's OMAP. Atmel offers many popular open-source toolchains and compilers. In 2010 alone, 6.1 billion ARM-based processors were shipped, thus representing 95% of smartphones, 35% of digital televisions and set-top boxes, and 10% of mobile computers. Fig. 6.1 [16] shows a die of the Cortex-M3 microcontroller with 1 Megabyte flash memory by STMicroelectronics.

Parallax Semiconductor makes the Propeller which is a 32-bit symmetric multicore (8 cores), high-speed (80 MHz), low-power microcontroller, with shared memory and a built-in interpreter for programming in a high-level object-based language, called SpinTM, and low-level (assembly) language [24]. With the set of pre-built Parallax "objects" for video, mice, keyboards, NTSC/VGA displays, LCDs, and sensors, your application is a matter of high-level integration with Propeller microcontrollers. The Propeller is designed for high-speed embedded processing while maintaining low power, low current consumption and a small physical footprint.

7.2 Other Development System Boards

While the HCS12 μ c-based boards are the development systems that any computer and electrical engineer must know for industrial-grade applications, there are other boards that are less expensive for experimentation and light-grade applications. They include [16]:

1. ArduinoTM Single-Board Computers and Shields;
2. Uno, Leonardo, Due, Mega, MintDuino;
3. Raspberry Pi Single-Board Computers;
4. BeagleBone Black Single-Board Computers;
5. TI Single-Board Computers (TI LaunchPadTM MSP430, SchmartBoard); and
6. Navigational Single-Board Computers (iNEMO).

8. CONCLUDING REMARKS

Upgrading laboratories in interfacing courses for real-time systems in computer engineering and electrical engineering programs is mandated by our accreditation bodies such as the Canadian Engineering Accreditation Board or the US Accreditation Board for Engineering and Technology (ABET). Selecting a microprocessor (μ P) or a microprocessor (μ c) for such a laboratory is a difficult task. The selected processor must then be incorporated in the labs—a laborious, time-consuming, and expensive process.

In the past we had groups of processors designed well from the pedagogical point of view. The current μ cs have many more architectures, instruction sets, input/output (I/O) capabilities, I/O protocols, and programming environments to choose from. Since our students must be prepared for those options in a work-place well, the selected processor must satisfy many of the requirements.

This paper summarizes a μ c selection process based on technical knowledge and capabilities of various μ cs. The process has been undertaken in our department, and lead to an update of the lab.

Acknowledgements

We would like to thank the Department of Electrical and Computer Engineering at the University of Manitoba for partial financial support of this project. Freescale is also acknowledged for helping us acquire the boards.

References

- [1] *Application Module Student Learning Kit Featuring Freescale HCS12: User Guide*. Chandler, AZ: Freescale Semiconductor, February 2008, 20 pp. Available as of September 30, 2013 from: http://www.ece.cmu.edu/~ece348/labs/docs/APS12C128_module/SLKS12UG_module_user_guide.pdf

- [2] Arnold S. Berger, *Embedded Systems Design: An Introduction to Processes, Tools, and Techniques*. Lawrence, KS: CMP Books, 2002, 237 pp. {ISBN: 978-1-57820073-3}
- [3] Juan Czares Blanco, *Interfacing API3192USLK with Microcontroller Student Learning Kits*. Chandler, AZ: Freescale Semiconductor, January 2007, 17 pp. Available as of September 30, 2013 from: http://www.freescale.com/files/student_learning_kits/doc/app_note/SLK0106AN.pdf
- [4] Fredrick M. Cady, *Software and Hardware Engineering: Assembly and C Programming for the Freescale HCS12 Microcontroller*. Oxford, UK: Oxford University, 2008 (2nd ed.), 750 pp. Includes a CDROM with CodeWarrior Studio for HCS12, v. 4.5.
- [5] John Doyle & Daniel McKenna, "HCS12X family memory organization," *Application Note AN2734*. Denver, CO: Freescale Semiconductor, Revision 1, Feb 2007, 12 pp.
- [6] Freescale, *S12C: 16-Bit Automotive Microcontroller Application Notes*. Denver, CO: Freescale Semiconductor, 2013. Available as of September 2013 from http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12C (Press the Application Notes tab at this Web page.)
- [7] Freescale, *MC9S12C128V1: MC9S12C/MC9S12GC Family Reference Manual*. Denver, CO: Freescale Semiconductor, Revision 1.24, May 2010, 690 pp. http://cache.freescale.com/files/microcontrollers/doc/ata_sheet/MC9S12C128V1.pdf?fp=1
- [8] Freescale, *Reference Manual: HCS12 Microcontrollers, S12CPUV2*. Denver, CO: Freescale Semiconductor, Revision 4, March 2006, 452 pp. http://cache.freescale.com/files/microcontrollers/doc/ref_manual/S12CPUV2.pdf?fp=1
- [9] Freescale, *MC9S12CFAMPB: MC9S12C-Family 16-Bit Microcontroller Brief*. Denver, CO: Freescale Semiconductor, Revision 5, March 2006, 26 pp. http://cache.freescale.com/files/microcontrollers/doc/prod_brief/MC9S12CFAMPB.pdf?fp=1
- [10] Freescale, *MC9S12C Family Device User Guide*. Denver, CO: Freescale Semiconductor, Revision 1.05, 11 Feb 2004, 136 pp. <http://www.datasheetarchive.com/9S12C128DGV1%2FD-datasheet.html> or <http://www.si.ic-datasheet.com/datasheets/pdf/9/9S/9S1/9S12C128DGV1.html>
- [11] *Grayhill Standard Keypads: Series 96*. Datasheet, 4 pp. http://www.grayhill.com/assets/1/7/Keypads_96.pdf
- [12] Günther Gridling and Bettina Weiss, *Introduction to Microcontrollers*. Vienna, AT: Vienna University of Technology, February 2, 2007 (Version 1.4), 175 pp. Available as of September 2013 from <http://wwwold.ecs.tuwien.ac.at/lehre/Microcontroller/Docs/Lecture/Microcontroller.pdf>
- [13] *Getting Started with the Student Learning Kit Featuring the Freescale HCS12 Microcontroller Application Module*. Chandler, AZ: Freescale Semiconductor, February 2008, 23 pp. Available as of September 30, 2013 from: http://cache.freescale.com/files/student_learning_kits/doc/user_guide/S12QSG.pdf?fp=1
- [14] Witold Kinsner, *Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems*. Lecture Notes. Winnipeg, MB: University of Manitoba, 2013, 643 pp.
- [15] Witold Kinsner, *Laboratories for Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems*. Lab Notes; 2013, 102 pp.
- [16] Witold Kinsner, *The HCS12 Microcontroller: A Tutorial*. Winnipeg, MB: University of Manitoba, 2013, 66 pp.
- [17] Witold Kinsner, *Laboratories for Microcontroller, Microprocessor, and Microcomputer Interfacing for Real-Time Systems*. Lab Notes. Winnipeg MB: University of Manitoba, 2010, 92 pp.
- [18] Witold Kinsner, "Challenges in the design of adaptive, intelligent and cognitive systems," *Intern. J. Software Science & Computational Intelligence*, vol. 1, no. 3, pp. 16-35, July-Sept. 2009
- [19] Muhammad Ali Mazidi and Danny Causey, *HCS12 Microcontroller and Embedded Systems using Assembly and C with Code Warrior*. Upper Saddle River, NJ: Pearson/ Prentice Hall, 2009, 738 pp. {ISBN-13: 978-0-13-607229-4; \$135.83}
- [20] *MCU Module, Educational Module, Quickstart Guide*. Garland, TX: Axiom Manufacturing, April 7, 2005, 3 pp. Available as of September 30, 2013 from: http://www.coe.montana.edu/ee/courses/ee/ee371/pdf/files/Module_QuickStart.pdf
- [21] *MCU Project Board 2: Prototyping Board with Microcontroller Interface*. Garland, TX: Axiom Manufacturing, June 15, 2005. Available as of September 30, 2013 from: http://www.async.ece.utah.edu/~myers/nobackup/ece5780/misc/mcu_project_board.pdf
- [22] Grant M. More and Daniel Malik, "A software interrupt priority scheme for HCS12 microcontrollers," *AN2617 Application Note*. Denver, CO: Freescale Semiconductor, Feb 2004. 12 pp. http://cache.freescale.com/files/microcontrollers/doc/app_note/AN2617.pdf?fp=1&WT_TYPE=Application%20Notes&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation
- [23] Eduardo Montanez and John McLellan, *Getting Started with the Microcontroller Student Learning Kit Project Board Using the MC9S12C32 Microcontroller Application Module*. Austin, TX: Freescale

- Semiconductor. Sep 2006, 18 pp. Available as of September 30, 2013 from:
<http://myweb.wit.edu/johnsont/Courses/462/PBS12C32SLKQSUG.pdf>
- [24] Parallax Semiconductor, "Propeller: Our multicore concept and architecture," 2013. Available as of September 30, 2013 from:
<http://www.parallaxsemiconductor.com/>
- [25] *Project Board (PBMCU32SLK)*. Chandler, AZ: Freescale Semiconductor, 2008, 2 pp. Available as of September 30, 2013 from:
http://www.freescale.com/files/student_learning_kits/doc/fact_sheet/BRMCSTDNTLEARN_low.pdf
- [26] *SLK Migration from the C32 to the C128: For use with APS12C32SLK & APS12C128SLK*, Sept 2008, 6 pp. Available as of September 30, 2013 from:
http://www.freescale.com/files/student_learning_kits/doc/errata/SLKS12MIGEB.pdf
- [27] SparkFun, "Tiny AVR programmer," 2013. Available as of September 2013 from
<https://www.sparkfun.com/products/11460>
- [28] Jonathan W. Valvano, *Embedded Microcomputer Systems: Real Time Interfacing*. Stamford, CT: CENGAGE Learning, 2012 (3rd ed.), 793 pp. & CD-ROM. This book covers design methodologies with examples, using the Motorola 9S12 machine, with many practical examples. The CD-ROM includes a simulator to show internal and I/O activities, and many links to data sheets. {ISBN 978-1-111-42625-5}
- [29] Han-Way Wang, *HCS12/9S12: An Introduction to Software and Hardware Interfacing*. New York, NY: Delmar Cengage Learning, 2009 (2nd ed.), 880 pp.
- [30] Wikipedia, "AVR architecture," 2013. Available as of September 2013 from
http://en.wikipedia.org/wiki/Atmel_AVR
<http://www.atmel.com/products/microcontrollers/avr/tinyavr.aspx>
<http://www.atmel.ca/products/microcontrollers/avr/default.aspx>
- [31] Wikipedia, "ARM architecture," 2013. Available as of September 2013 from
http://en.wikipedia.org/wiki/ARM_architecture