Computational hardware

- Digital logic (CSE370)
  - gates and flip-flops: glue logic, simple FSMs, registers
- Programmable logic devices (CSE370, CSE467)
  - two-level PLDs: FSMs, muxes, decoders
  - field-programmable gate arrays: FSMs, basic data-paths
- Microprocessors (CSE378)
  - general-purpose computer
  - instructions can implement complex control structures
  - supports computations/manipulations of data in memory

Microprocessors

- Arbitrary computations
  - arbitrary control structures
  - arbitrary data structures
  - specify function at high-level and exploit compilers and debuggers
- To save hardware
  - if function requires too much logic when implemented with gates/FFs
    - operations are too complex, better broken down as instructions
    - lots of data manipulation (memory)
  - if function does not require higher performance of customized logic
    - ever-increasing performance of processors puts more and more applications
      in this category
    - minimize the amount of external logic

Microprocessor basics

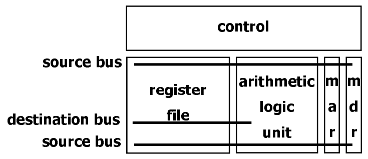
- Composed of three parts
  - data-path: data manipulation and storage
  - control: determines sequence of actions executed in data-path and interactions to be had with environment
  - interface: signals seen by the environment of the processor
- Instruction execution engine: fetch/execute cycle
  - flow of control determined by modifications to program counter
  - instruction classes:
    - data: move, arithmetic and logical operations
    - control: branch, loop, subroutine call
    - interface: load, store from external memory

Microprocessor basics (cont'd)

- Can implement arbitrary state machine with auxiliary data-path
  - control instructions implement state diagram
  - registers and ALUs act as data storage and manipulation
  - interaction with the environment through memory interface
  - how are individual signal wires sensed and controlled?

Microprocessor organization

- Controller
  - inputs: from ALU (conditions), instruction read from memory
  - outputs: select inputs for registers, ALU operations, read/write to memory



- Data-path
  - register file to hold data
  - arithmetic logic unit to manipulate data
  - program counter (to implement relative jumps and increments)
- Interface
  - data to/from memory (address and data registers in data path)
  - read/write signals to memory (from control)

General-purpose processor

- Programmed by user
- New applications are developed routinely
- General-purpose
  - must handle a wide ranging variety of applications
- Interacts with environment through memory
  - all devices communicate through memory data
  - DMA operations between disk and I/O devices
  - dual-ported memory (e.g., display screen)
  - oblivious to passage of time (takes all the time it needs)

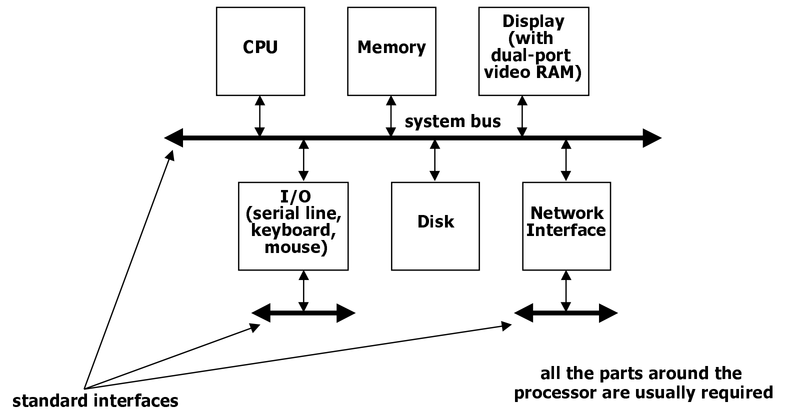
Embedded processor

- Programmed once by manufacturer of system
- Executes a single program (or a limited suite) with few parameters
- Task-specific
  - can be optimized for specific application
- Interacts with environment in many ways
  - direct sensing and control of signal wires
  - communication protocols to environment and other devices
  - real-time interactions and constraints

Why embedded processors?

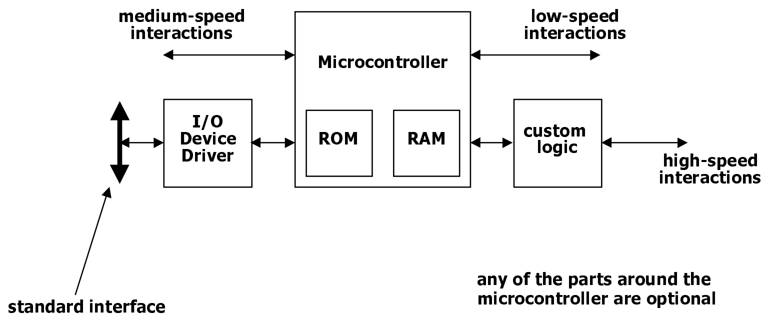
- High overhead in building a general-purpose system
  - storing/loading programs
  - operating system manages running of programs and access to data
  - shared system resources (e.g., system bus, large memory)
  - many parts
    - communication through shared memory/bus
    - each I/O device requires its own hardware
- Optimization opportunities
  - as much hardware as necessary for application
    - cheaper, portable, lower-power systems
  - as much software as necessary for application
    - no complete OS requirement
  - can integrate processor, memory, and I/O devices onto a single-chip

Typical general-purpose architecture


CPU
Memory
Display (with dual-port video RAM)
system bus
I/O (serial line, keyboard, mouse)
Disk
Network Interface
standard interfaces

all the parts around the processor are usually required

Typical task-specific architecture



any of the parts around the microcontroller are optional

How does this change things?

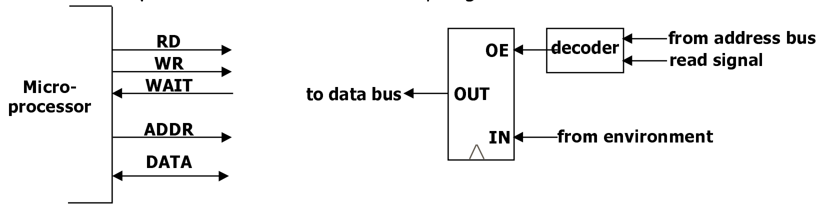
- Sense and control of environment
  - processor must be able to "read" and "write" individual signal wires
  - controls I/O devices directly
- Measurement of time
  - many applications require precise spacing of events
  - reaction times to external stimuli may be constrained
- Communication
  - protocols must be implemented by processor
  - integrate I/O device or emulate in software
  - capability of using external device if necessary

Interactions with the environment

- Basic processor only has address and data busses to memory
- Inputs are read from memory
- Outputs are written to memory
- Thus, for a processor to sense/control signal wires in the environment they must be made to appear as memory bits
  - how do we make wires look like memory?

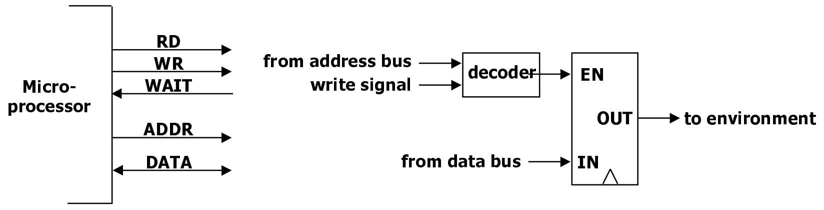
Sensing external signals

- Map external wire to a bit in the address space of the processor
- External register or latch buffers values coming from environment
  - map register into address space
  - decoder needed to select register for reading
  - output enable (OE) so that many registers can use the same data bus



Controlling external signals

- Map external wire to a bit in the address space of the processor
- Connect output of memory-mapped register to environment
  - map register into address space
  - decoder need to select register for writing (holds value indefinitely)
  - input enable (EN) so that many registers can use the same data bus



Time and instruction execution

- Keep track of detailed timing of each instruction's execution
  - highly dependent on code
  - hard to use compilers
  - not enough control over code generation
  - interactions with caches/instruction-buffers
- Loops to implement delays
  - keep track of time in counters
  - keeps processor busy counting and not doing other useful things
- Real-time clock
  - take differences at different points in the program

Time measurement via parallel timers

- Separate and parallel counting unit(s)
  - co-processor to microprocessor
  - does not require microprocessor intervention
  - in simple case, like a real-time clock
  - set timer, interrupt generated when expired
- More interesting timer units
  - self reloading timers for regular interrupts
  - pre-scaling for measuring larger times
  - started by external events

Input/output events

- Input capture
  - record time when input event occured
  - to be used in later handling of event
- Output compare
  - set output to happen at a point in the future
  - reactive outputs – set output to happen a pre-defined time after some input
  - processor can go on to do other things in the meantime

System bus based communication

- Extend address/data bus to outside
- Use specialized devices to implement communication protocol
- Map devices and their registers to memory locations
- Read/write data to receive/send in buffers in device or shared memory
- Poll register in device for status
- Wait for interrupt from device on interesting events
  - send completed
  - receive occured

Support for communication protocols

- Built-in device drivers
  - for common communication protocols
  - serial-line protocols most common as they require few pins
- Serial-line controller
  - special register in memory space for interaction
  - may use timer unit(s) to generate timing events
    - for spacing of bits on signal wire
    - for sampling rate
- Increase level of integration
  - no external devices
  - may further eliminate need for shared memory or system bus

Microcontrollers

- Embedded processor with much more integrated on same chip
  - processor core + co-processors + memory
  - ROM for program memory, RAM for data memory, special registers
  - parallel I/O ports to sense and control wires
  - timer units to measure time in various ways
  - communication subsystems to permit direct link

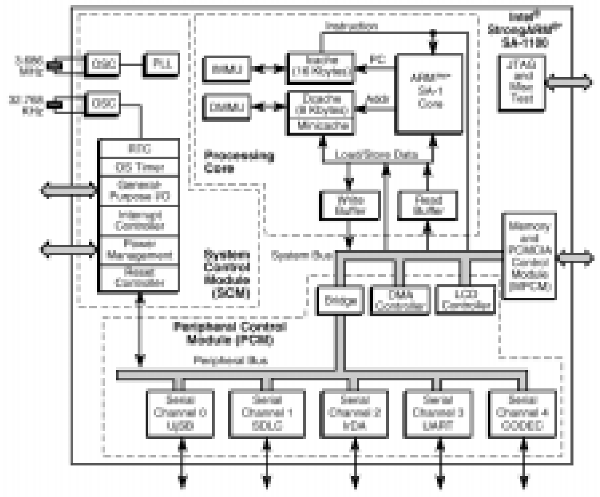
Microcontrollers (cont'd)

- Other features not usually found in general-purpose CPUs
  - expanded interrupt handling capabilities
    - multiple interrupts with priority and selective enable/disable
    - automatic saving of context before handling interrupt
    - interrupt vectoring to quickly jump to handlers
  - more instructions for bit manipulations
    - support operations on bits (signal wires) rather than just words
- Integrated memory and support functions for cheaper system cost
  - built-in EPROM, Flash, and/or RAM
  - DRAM controller to handle refresh
  - page-mode support for faster block transfers

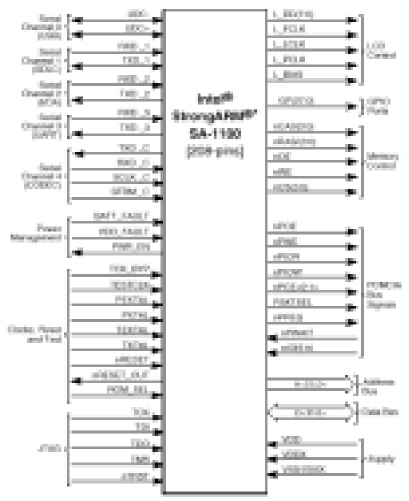
Microcontroller we will be using

- Intel StrongARM (SA1100)
  - 32-bit microcontroller - ARM RISC-processor core (SA-1)
  - instruction (16KB) and data (8KB) caches
  - real-time clock (32.768 MHz)
  - general-purpose I/O (with interrupt support)
  - serial communication interfaces
    - can emulate v.34 modem in software
    - supports 4MB/sec IrDA
    - USB slave
  - display interface (1024x1024)
  - external memory controller
  - extensive low-power modes
  - rich software support (OSes, app. libraries)
  - designed for portable applications

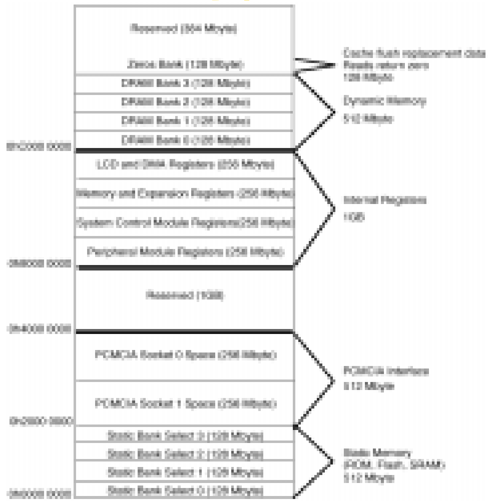
SA-1100 Architecture



SA-1100 Pins

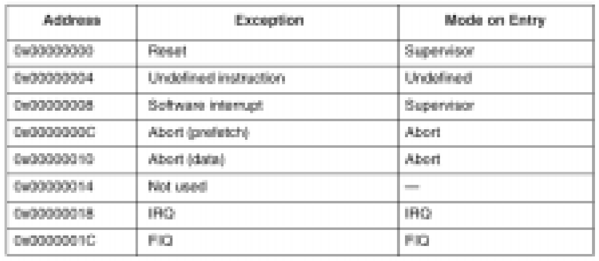


SA-1100 Memory Map



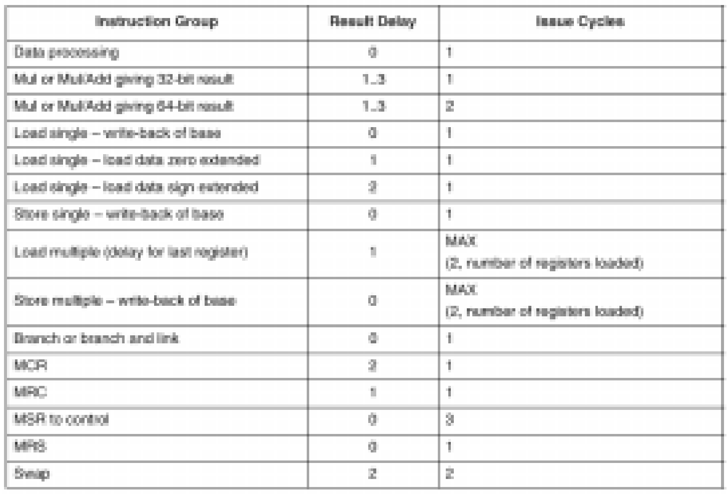
Reserved (384 Mbyte)
Zeros Bank (128 Mbyte)
DRAM Bank 3 (128 Mbyte)
DRAM Bank 2 (128 Mbyte)
DRAM Bank 1 (128 Mbyte)
DRAM Bank 0 (128 Mbyte)
0xC000 0000
LCD and DMA Registers (256 Mbyte)
Memory and Expansion Registers (256 Mbyte)
System Control Module Registers(256 Mbyte)
Peripheral Module Registers (256 Mbyte)
0x8000 0000
Reserved (1GB)
0x4000 0000
PCMCIA Socket 0 Space (256 Mbyte)
PCMCIA Socket 1 Space (256 Mbyte)
0x2000 0000
Static Bank Select 3 (128 Mbyte)
Static Bank Select 2 (128 Mbyte)
Static Bank Select 1 (128 Mbyte)
Static Bank Select 0 (128 Mbyte)
0x0000 0000
Cache flush replacement data
Reads return zero
128 Mbyte
Dynamic Memory
512 Mbyte
Internal Registers
1GB
PCMCIA Interface
512 Mbyte
Static Memory
(ROM, Flash, SRAM)
512 Mbyte

SA-1100 Vector Map



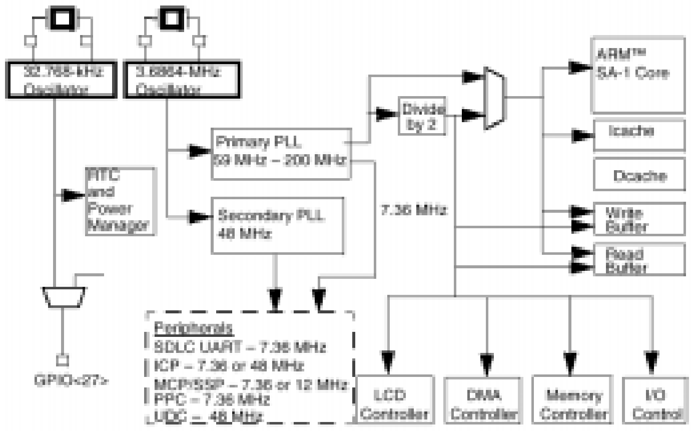
Address	Exception	Mode on Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Not used	—
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

SA-1100 Instruction Timings

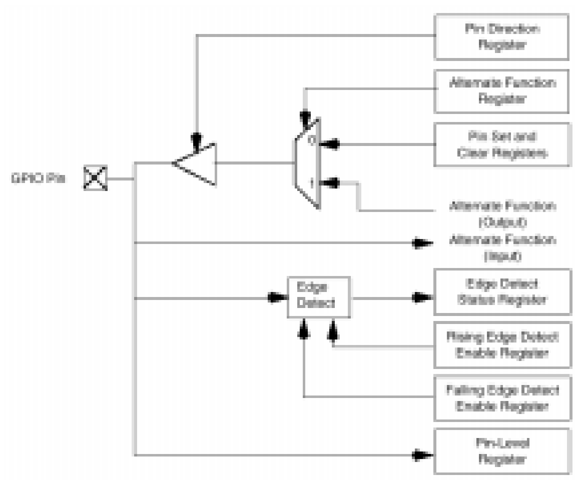


Instruction Group	Result Delay	Issue Cycles
Data processing	0	1
Mul or Mul/Add giving 32-bit result	1..3	1
Mul or Mul/Add giving 64-bit result	1..3	2
Load single – write-back of base	0	1
Load single – load data zero extended	1	1
Load single – load data sign extended	2	1
Store single – write-back of base	0	1
Load multiple (delay for last register)	1	MAX (2, number of registers loaded)
Store multiple – write-back of base	0	MAX (2, number of registers loaded)
Branch or branch and link	0	1
MCR	2	1
MRC	1	1
MSR to control	0	3
MRS	0	1
Swap	2	2

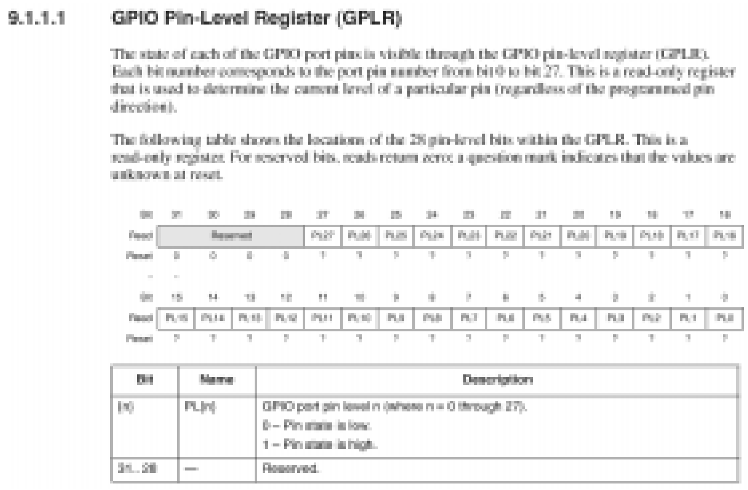
SA-1100 Clocks



SA-1100 General-Purpose I/O Pins (28)


Pin Direction Register
Alternate Function Register
Pin Set and Clear Registers
Alternate Function (Output)
Alternate Function (Input)
GPIO Pin
0
1
Edge Detect
Edge Detect Status Register
Rising Edge Detect Enable Register
Falling Edge Detect Enable Register
Pin-Level Register

SA-1100 GPIO Pin-Level Register (GPLR)



9.1.1.1 GPIO Pin-Level Register (GPLR)

The state of each of the GPIO port pins is visible through the GPIO pin-level register (GPLR). Each bit number corresponds to the port pin number from bit 0 to bit 27. This is a read-only register that is used to determine the current level of a particular pin (regardless of the programmed pin direction).

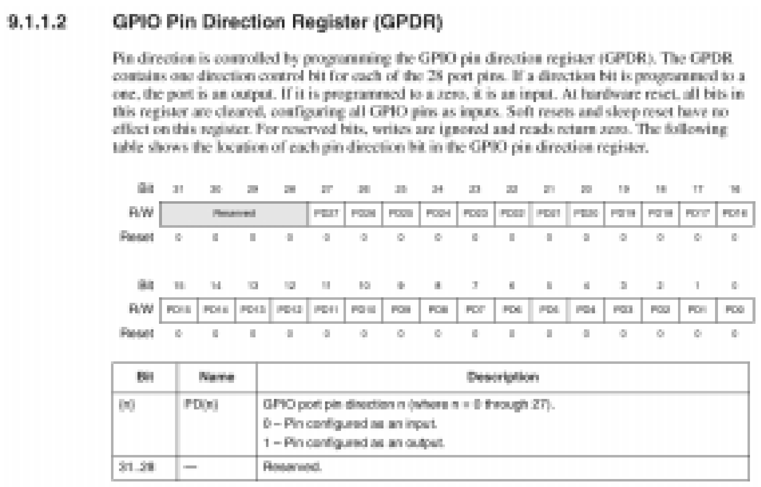
The following table shows the locations of the 28 pin-level bits within the GPLR. This is a read-only register. For reserved bits, reads return zero; a question mark indicates that the values are unknown at reset.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Read	Reserved				PL27	PL26	PL25	PL24	PL23	PL22	PL21	PL20	PL19	PL18	PL17	PL16
Reset	0	0	0	0	?	?	?	?	?	?	?	?	?	?	?	?

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read	PL15	PL14	PL13	PL12	PL11	PL10	PL9	PL8	PL7	PL6	PL5	PL4	PL3	PL2	PL1	PL0
Reset	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Bit	Name	Description
(n)	PL(n)	GPIO port pin level n (where n = 0 through 27). 0 – Pin state is low. 1 – Pin state is high.
31..28	—	Reserved.

SA-1100 GPIO Pin Direction Register (GPDR)



9.1.1.2 GPIO Pin Direction Register (GPDR)

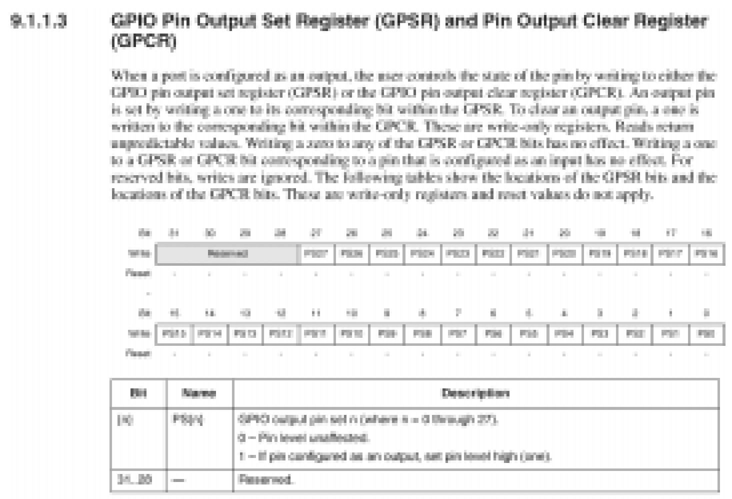
Pin direction is controlled by programming the GPIO pin direction register (GPDR). The GPDR contains one direction control bit for each of the 28 port pins. If a direction bit is programmed to a one, the port is an output. If it is programmed to a zero, it is an input. At hardware reset, all bits in this register are cleared, configuring all GPIO pins as inputs. Soft resets and sleep reset have no effect on this register. For reserved bits, writes are ignored and reads return zero. The following table shows the location of each pin direction bit in the GPIO pin direction register.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R/W	Reserved				PD27	PD26	PD25	PD24	PD23	PD22	PD21	PD20	PD19	PD18	PD17	PD16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	PD15	PD14	PD13	PD12	PD11	PD10	PD9	PD8	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	Name	Description
(n)	PD(n)	GPIO port pin direction n (where n = 0 through 27). 0 – Pin configured as an input. 1 – Pin configured as an output.
31..28	—	Reserved.

SA-1100 GPIO Output Set/Clear Registers



9.1.1.3 GPIO Pin Output Set Register (GPSR) and Pin Output Clear Register (GPCR)

When a port is configured as an output, the user controls the state of the pin by writing to either the GPIO pin output set register (GPSR) or the GPIO pin output clear register (GPCR). An output pin is set by writing a one to its corresponding bit within the GPSR. To clear an output pin, a one is written to the corresponding bit within the GPCR. These are write-only registers. Reads return unpredictable values. Writing a zero to any of the GPSR or GPCR bits has no effect. Writing a one to a GPSR or GPCR bit corresponding to a pin that is configured as an input has no effect. For reserved bits, writes are ignored. The following tables show the locations of the GPSR bits and the locations of the GPCR bits. These are write-only registers and reset values do not apply.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Write	Reserved				PS27	PS26	PS25	PS24	PS23	PS22	PS21	PS20	PS19	PS18	PS17	PS16
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	PS15	PS14	PS13	PS12	PS11	PS10	PS9	PS8	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Bit	Name	Description
(n)	PS(n)	GPIO output pin set n (where n = 0 through 27). 0 – Pin level unaffected. 1 – If pin configured as an output, set pin level high (one).
31..28	—	Reserved.

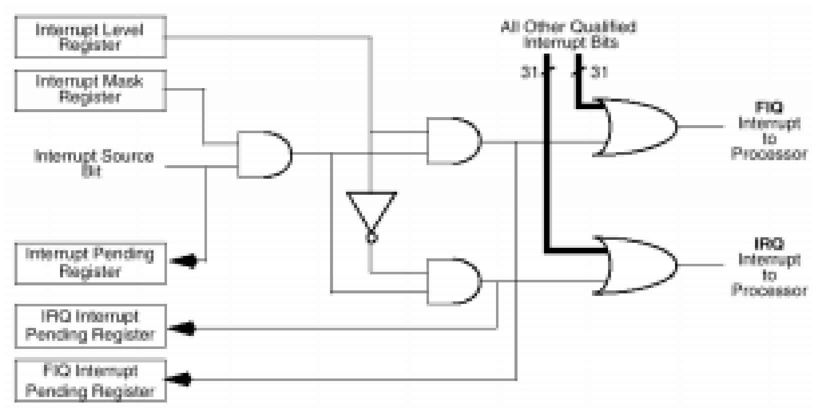
SA-1100 GPIO Alternate Functions

Pin	Alternate Function	Direction	Unit	Signal Description
GP<27>	32KHZ_OUT	Output	Clocks	Raw 32.768-kHz oscillator output
GP<26>	RCLK_OUT	Output	Clocks	Internal clock/2
GP<25>	RTC clock	Output	RTC	Trimmed 1-Hz clock
GP<24>	Reserved	—	—	—
GP<23>	TREQB	Input	Test controller	TIC request B
GP<22>	TREQA/MBREQ	Input	Test controller	Either TIC request A or MBREQ
GP<21>	TIC_ACK/MBGNT	Output	Test controller	Either TIC acknowledge or MBGNT
GP<21>	MCP_CLK	Input	Serial port 4	MCP clock in
GP<20>	UART_SCLK3	Input	Serial port 3:UART	Sample clock input
GP<19>	SSP_CLK	Input	Serial port 2:SSP	Sample clock input
GP<18>	UART_SCLK1	Input	Serial port 1:UART	Sample clock input
GP<17>	SDLC_AAF	Output	Serial port 1:SDLC	Abort after frame control
GP<16>	SDLC_SCLK	I/O	Serial port 1:SDLC	Geoport clock out
GP<15>	UART_RXD	Input	Serial port 1:UART	UART receive
GP<14>	UART_TXD	Output	Serial port 1:UART	UART transmit
GP<13>	SSP_SFRM	Output	Serial Port 4:SSP	SSP frame clock
GP<12>	SSP_SCLK	Output	Serial port 4:SSP	SSP serial clock
GP<11>	SSP_RXD	Input	Serial port 4:SSP	SSP receive
GP<10>	SSP_TXD	Output	Serial port 4:SSP	SSP transmit
GP<2..9>	LDD<8..15>	Output	LCD controller	High-order data pins for split-screen color LCD support
GP<1>	Reserved	—	—	No alternate function
GP<0>	Reserved	—	—	No alternate function

SA-1100 GPIO Register Locations

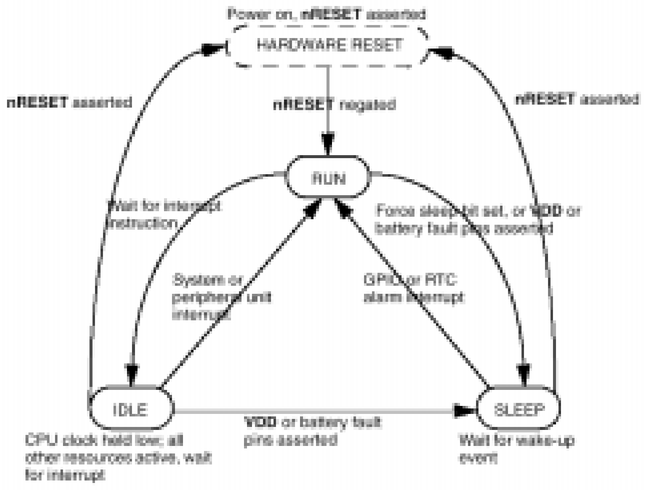
Address	Name	Description
0h 9004 0000	GPLR	GPIO pin-level register
0h 9004 0004	GPDR	GPIO pin direction register
0h 9004 0008	GPSR	GPIO pin output set register
0h 9004 000C	GPCR	GPIO pin output clear register
0h 9004 0010	GRER	GPIO rising-edge detect register
0h 9004 0014	GFER	GPIO falling-edge detect register
0h 9004 0018	GEDR	GPIO edge detect status register
0h 9004 001C	GAFR	GPIO alternate function register

SA-1100 Interrupt Logic



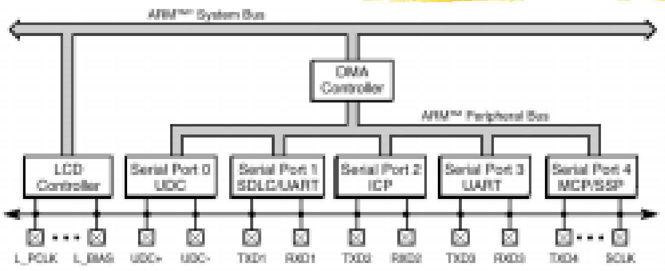
Interrupt Level Register
Interrupt Mask Register
Interrupt Source Bit
Interrupt Pending Register
IRQ Interrupt Pending Register
FIQ Interrupt Pending Register
All Other Qualified Interrupt Bits
31
31
FIQ Interrupt to Processor
IRQ Interrupt to Processor

SA-1100 Sleep Modes



Power on, nRESET asserted
HARDWARE RESET
nRESET asserted
nRESET negated
nRESET asserted
RUN
Wait for interrupt instruction
Force sleep bit set, or VDD or battery fault pins asserted
System or peripheral unit interrupt
GPIO or RTC alarm interrupt
IDLE
VDD or battery fault pins asserted
SLEEP
CPU clock held low; all other resources active, wait for interrupt
Wait for wake-up event

SA-1100 Peripheral Control Module



Peripheral	Serial Protocol	Base Address
LCD Controller		0h B010 0000
Serial Port 0	USB	0h 8000 0000
Serial Port 1	UART	0h 8001 0000
	SDLC	0h 8002 0000
Serial Port 2 (ICP)	UART	0h 8003 0000
	HSSP	0h 8004 0000
Serial Port 3	UART	0h 8005 0000
Serial Port 4	MPC	0h 8006 0000
	SSP	0h 8007 0000
Peripheral Pin Controller (PPC)[1]		0h 9006 0000

SA-1100 Peripheral Pins

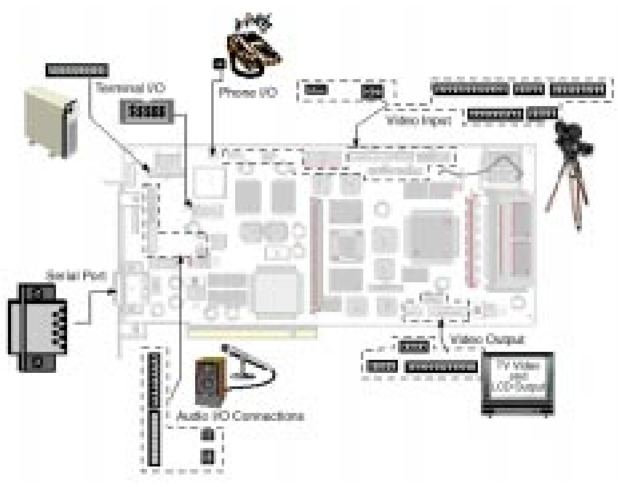
Peripheral	GPIO Pin	Function
LCD Controller	GPIO<2>	LDD<8> pin for dual panel color mode.
	GPIO<3>	LDD<9> pin for dual panel color mode.
	GPIO<4>	LDD<10> pin for dual panel color mode.
	GPIO<5>	LDD<11> pin for dual panel color mode.
	GPIO<6>	LDD<12> pin for dual panel color mode.
	GPIO<7>	LDD<13> pin for dual panel color mode.
	GPIO<8>	LDD<14> pin for dual panel color mode.
	GPIO<9>	LDD<15> pin for dual panel color mode.
Serial port 0 USB	N/A	None.
Serial port 1 SDLC/UART	GPIO<14>	Transmit pin for UART when SDLC and UART both needed.
	GPIO<15>	Receive pin for UART when SDLC and UART both needed.
	GPIO<16>	Sample clock input/output to SDLC.
	GPIO<17>	Toggle to drive external Enable for SDLC transmit packets.
	GPIO<18>	Sample clock input to UART.
Serial port 2 ICP	N/A	None.
Serial port 3 UART	GPIO<20>	Sample clock input to UART.
Serial port 4 MPC/SSP	GPIO<10>	Transmit pin for SSP when MCP and SSP both needed.
	GPIO<11>	Receive pin for SSP when MCP and SSP both needed.
	GPIO<12>	SCLK pin for SSP when MCP and SSP both needed.
	GPIO<13>	SFRM pin for SSP when MCP and SSP both needed.
	GPIO<19>	Clock input pin for SSP to drive the frame and sample rates when other than nonmultiple of 3.6864 MHz needed.
	GPIO<21>	Clock input pin for MCP to drive the frame and sample rates when other than 12 Mbps needed.

Development board we will be using

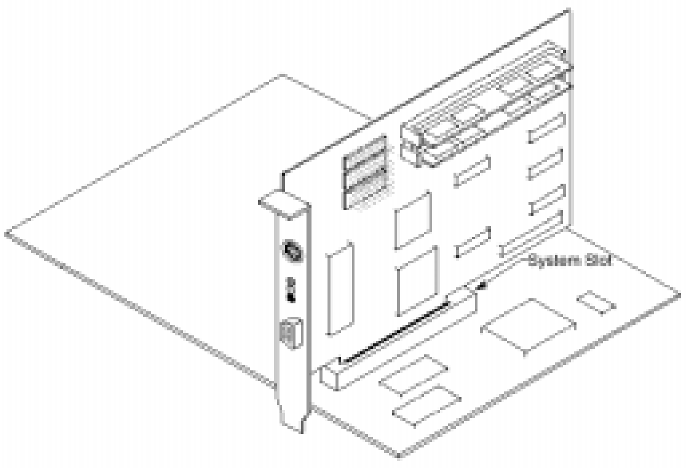
- Multimedia development board for StrongARM (MDB)
  - SA-1100 reference design
  - monitor program to load and debug program (Angel)
  - 128KB monitor (Angel) flash memory, 4MB application flash memory
  - 16MB of DRAM

  - 16-button keypad, 8 LEDs, 2 7-segment digit displays, touch screen
  - IrDA, RS-232, audio I/O, video I/O, phone (POTS) interface]
  - DSP co-processor for audio/video applications
  - CPLD to map all the I/O device on-board (re-programmable)
  - XBus interface - to connect other I/O devices on other boards
  - daughter card interface (see SA-1101 companion board)

StrongARM MDB Connections



MDB and PCI Backplane

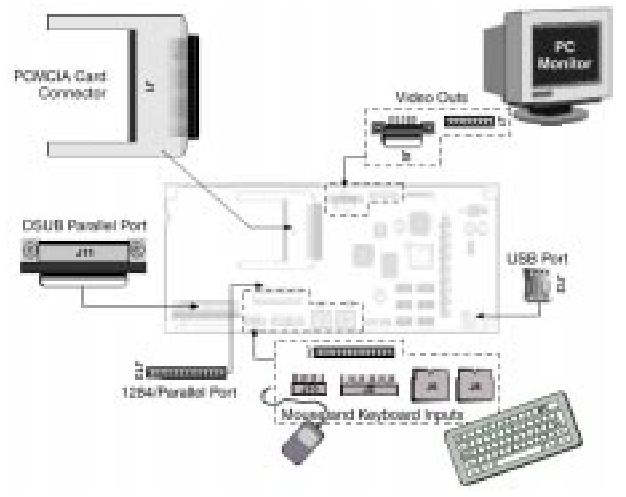


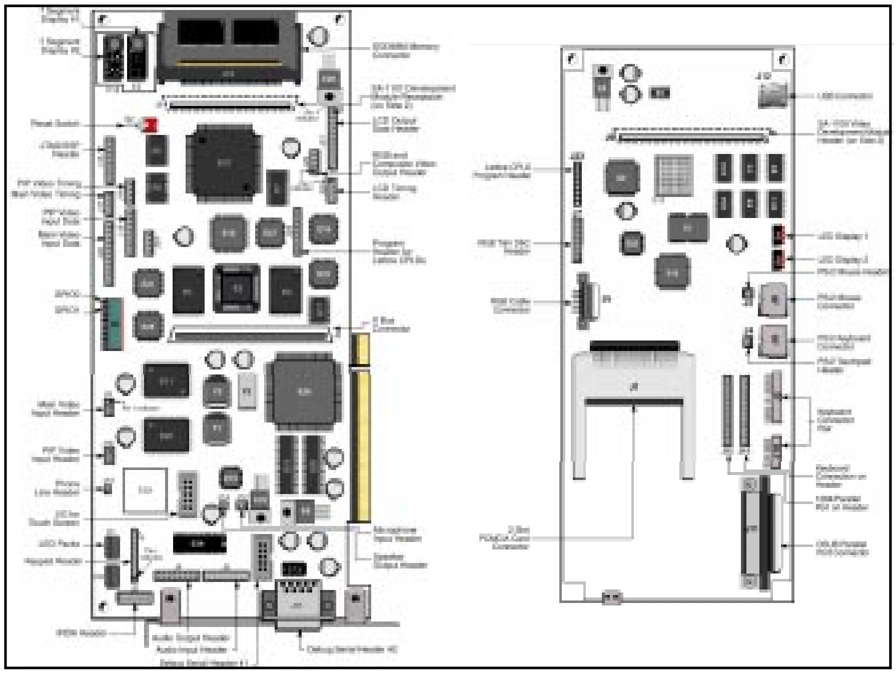
Companion board to MDB

- Daughter board for StrongARM (MDB)
  - SA-1101 reference design
  - connected to same memory bus as SA-1100 board

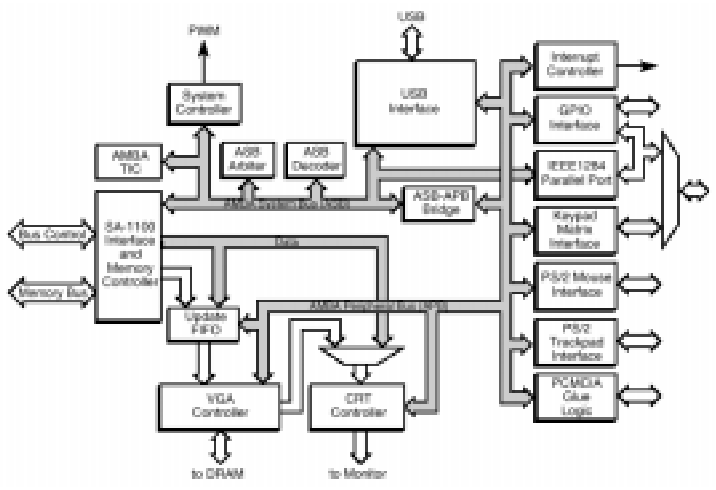
  - VGA output (1024x768 with 8bits/color)
  - USB host controller
  - Glue logic for two PCMCIA slots
  - Two PS/2 ports (keyboard and mouse)
  - 16x8 matrix keyboard interface
  - IEEE1284 parallel port

SA-1101 Companion Board Connections





SA-1101



Why this choice? (donated by Intel Corporation)

- StrongARM
  - basic 32-bit microcontroller with straightforward instruction set
  - full-featured (many I/O devices included on same chip)
  - good real-time capabilities
  - becoming very common (lots of resources available on web)
  - good development environment available (ARM SDT)
- Multimedia Development Board
  - fully integrated system (only need to add project-specific hardware)
  - debugging support (Angel monitor in ROM), serial communication to PC
  - supports a wide range of interfaces (audio, video, …) through main board and daughter card
  - MDB targets task-specific applications/devices
  - companion board allows connections to PC world

Programming the SA1100

- Subject of laboratory assignment #1
  - program some LEDs to flash on the MDB
  - do it three different ways
- Development environment
  - ARM's Software Development Toolkit (SDT)
- C programming
  - sa1100.h - important register definitions
  - sa1100.c - library of useful functions
  - sa1100asm.s - special function in assembly language
  - main.c - main program
  - lab1.apj - project file for SDT

A sample program (main.c)

```
#define DLEDR        0x18800006
```

```
#include <stdio.h>
#include <stdlib.h>

#include "sa1100.h"

void outport(DWORD *lPortAddress, DWORD lValue)
{
  *lPortAddress = lValue;
}

int main( void ) {
  int i;
  outport((DWORD *) DLEDR, 0x0);
  for( i = 0; i < 1000000; i++ );
  printf( "Hello World!" );
  outport((DWORD *) DLEDR, 0xF);
  return 0;
}
```

```
to console
(over serial
port to PC)
```

Busy wait function (sa1100.c)

```
void Wait( unsigned uSec ) {
  // waits the specified number of microseconds

  volatile unsigned *lTimerOSCRAddress= (unsigned *)OSCR;
  unsigned lValue, Time;

  Time = *lTimerOSCRAddress;

  lValue = Time + (uSec * TIMERTICK);
  if (lValue < Time) {
    while (Time < *lTimerOSCRAddress);
  }
  while (*lTimerOSCRAddress <= lValue);
}
```

```
#define OSCR      0x90000010 /* OS Timer Counter Registers   */
#define TIMERTICK 4      /* 1 microsecond is 3.7 clock ticks */
```

Timer interrupt functions (sa1100.c)

```
void TimerSetup( unsigned timerSel, unsigned cnt ) {
  struct os_timer_regs *OSTimerRegs = (struct os_timer_regs*)OSMR_0;

  DisableTimerIRQ( IRQ_OSTIMER0+timerSel );

  OSTimerRegs->oscr = 0;                   // initialize timer counter register
  OSTimerRegs->osmr[timerSel] = cnt;       // set timer max count
  OSTimerRegs->oier |= 1 << timerSel;      // unmask timer irq
  OSTimerRegs->ossr  = 1 << timerSel;      // reset interrupt source

  EnableTimerIRQ( IRQ_OSTIMER0+timerSel );
}

void initTimer( unsigned timerSel, unsigned us ) {
  TimerSetup( timerSel, 0x24*us );
}

void EnableTimerIRQ( unsigned timerSel ) {
  SetIrqLevel( timerSel, FALSE );
  EnableIrq( timerSel );
}

void DisableTimerIRQ( unsigned timerSel ) {
  DisableIRQ( timerSel );
}
```

Interrupt setup functions (sa1100.c)

```
void EnableIrq( unsigned uMask ) {
  struct scm_ic_regs * icreg = (struct scm_ic_regs *)IC_BASE;

  icreg->icmr |= uMask;
}

void DisableIRQ( unsigned uMask ) {
  struct scm_ic_regs * icreg = (struct scm_ic_regs *)IC_BASE;

  icreg->icmr &= ~uMask;
}

void DisableAllInts() {
  struct scm_ic_regs * icreg = (struct scm_ic_regs *)IC_BASE;

  icreg->icmr = 0;
}
```

```
#define IC_BASE     0x90050000 /*  SCM IC Base      */
```

```
struct scm_ic_regs {
  int icip;          // R
  int icmr;          // RW
  int iclr;          // RW
  int reserved0;     // --
  int icfp;          // R
  int reserved1;     // --
  int reserved2;     // --
  int reserved3;     // --
  int icpr;          // R
};
```

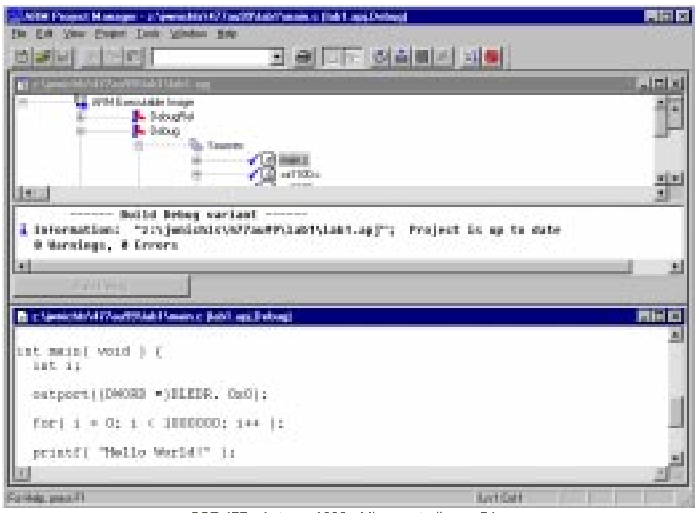
Installing and interrupt handler (sa1100.c)

```
unsigned Install_Handler( unsigned routine, unsigned *vector ) {
  unsigned vec, oldvec;

  vec = ( (routine - (unsigned)vector - 0x8) >> 2 );

  if (vec & 0xff000000) {
    return 0;
  }

  vec = 0xea000000 | vec;
  oldvec = *vector;
  *vector = vec;

  CleanAllCaches();

  return oldvec;
}
```

Including assembly code (SA1100asm.s)

```
CleanAllCaches
        STMFD r13!,{r0-r1,lr}             ; Store registers
        mov r0,#&e0000000
        add r1,r0,#8192
loop2
        ldr r2,[r0],#32
        teq r1,r0
        bne loop2
        MCR MMUCP,0,r0,MMUFlushTLBReg,c7,CP15_FLUSH_ALL ; Flush the TLB
        NOP
        NOP
        NOP
        MCR MMUCP,0,r0,MMUFlushIDCReg,c7,CP15_FLUSH_ALL ; Flush Data
        NOP
        NOP
        NOP
        LDMFD r13!,{r0-r1,lr}             ; Restore original registers
        mov pc,lr
```

ARM Project Manager After Successful Build

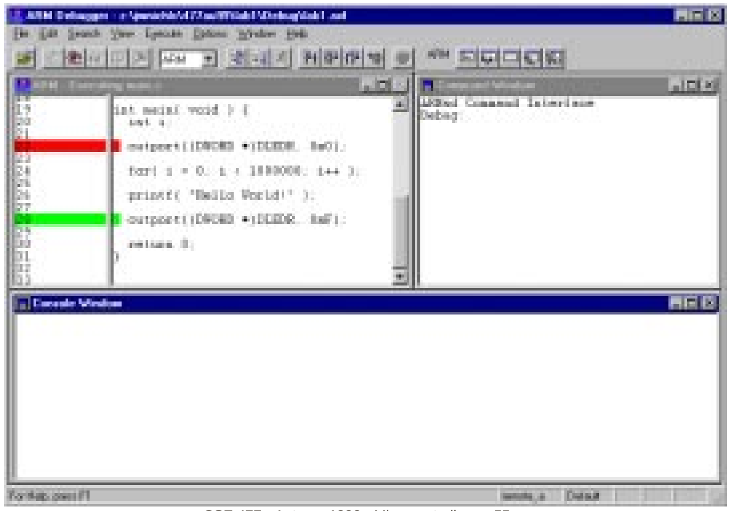


```
------ Build Debug variant ------
Information:  "z:\jmichi\cse477\lab1\lab1.apj";  Project is up to date
0 Warnings, 0 Errors
```

```
int main( void ) {
  int i;

  outport((DWORD *)LEDR, 0x0);

  for( i = 0; i < 1000000; i++ );

  printf( "Hello World!" );
```

Debugger In Progress

(breakpoint at red line, current exec. point at green)



ARM Debugger - c:\pmichk\477\sol99\lab1\Debug\lab1.axf
File Edit Search View Execute Options Window Help
ARM
ARM - Executing main()
Command Window
Console Window
For Help, press F1
remote_a
Default

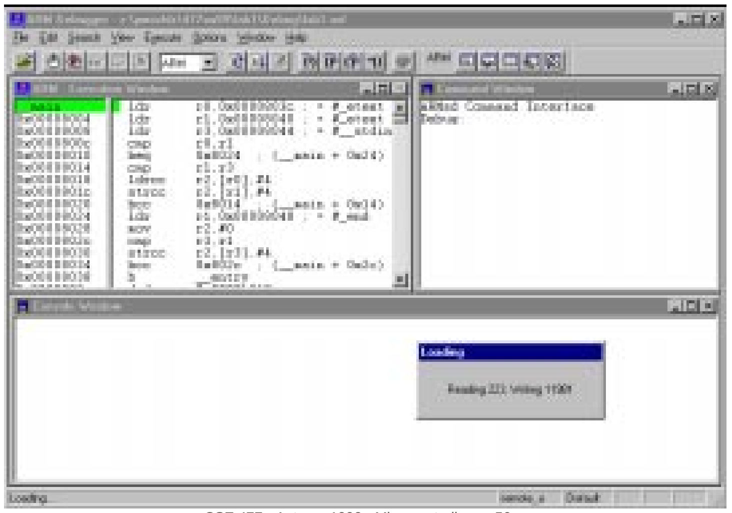
```
int main( void ) {
  int i;

  outport((DWORD *)DDEDR, 0x0);

  for( i = 0; i < 1000000; i++ );

  printf( "Hello World!" );

  outport((DWORD *)DDEDR, 0xF);

  return 0;
}
```

```
ARMsd Command Interface
Debug:
```

CSE 477 - Autumn 1999 - Microcontrollers - 55

ARM Debugger loading program to board

(with assembly shown in Execution Window)



ARM Debugger - c:\pmichk\477\sol99\lab1\Debug\lab1.axf
File Edit Search View Execute Options Window Help
ARM
ARM - Execution Window
Command Window
Console Window
Loading
Loading...
remote_a
Default

```
main        ldr     r0,0x0000803c ; = #_etext
0x00008004  ldr     r1,0x00008040 ; = #_etext
0x00008008  ldr     r3,0x00008044 ; = #__stdin
0x0000800c  cmp     r0,r1
0x00008010  beq     0x8024  ; (__main + 0x24)
0x00008014  cmp     r1,r3
0x00008018  ldrcc   r2,[r0],#4
0x0000801c  strcc   r2,[r1],#4
0x00008020  bcc     0x8014  ; (__main + 0x14)
0x00008024  ldr     r1,0x00008048 ; = #_end
0x00008028  mov     r2,#0
0x0000802c  cmp     r3,r1
0x00008030  strcc   r2,[r3],#4
0x00008034  bcc     0x802c  ; (__main + 0x2c)
0x00008038  b       __entry
```

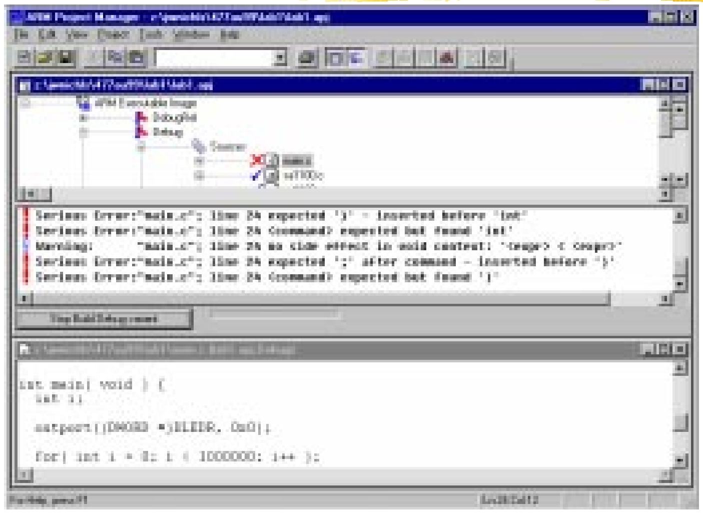
```
ARMsd Command Interface
Debug:
```

Reading 221 Writing 1596

CSE 477 - Autumn 1999 - Microcontrollers - 56

ARM Project Manager during build with errors

(error is variable declaration in for loop – last line of code window)



```
Serious Error:"main.c"; line 24 expected ')' - inserted before 'int'
Serious Error:"main.c"; line 24 <command> expected but found 'int'
Warning:       "main.c"; line 24 no side effect in void context: '<expr> < <expr>'
Serious Error:"main.c"; line 24 expected ';' after command - inserted before ')'
Serious Error:"main.c"; line 24 <command> expected but found ')'
```

```
int main( void ) {
  int i;

  outport((DWORD *)LEDR, 0x0);

  for( int i = 0; i < 1000000; i++ );
```

Ground rules for the lab (SIG 327)

- State-of-the-art facility we work hard to maintain
- Christopher Morgan is the manager of the lab
  - listen to him carefully
  - trust him to tell you the right thing
- TAs will hold their office hours there
- Make sure everything is always left the way you found it
  - hardware, software, and physical surroundings
  - neatness does count
- Keep your stuff in the bench drawers (don't take it home)
- No drinking, smoking, or other behavior that may harm
- Don't let others into the lab (especially off hours)