1. Course overview
2. Intro to PICOBLAZE, C and Number systems and Boolean Algebra
3. Course overview with microprocessor MU0 (I)
4. Course overview with microprocessor MU0 (II)
5. Verilog HDL
6. Digital system components using schematics and Verilog
7. Combinational logic standard forms. Karnaugh maps
8. Combinational ccts and configurable logic devices
9. Simple Sequential circuits, flip flops
10. Sequential circuits, counters, registers, memories
11. Non-ideal effects in digital circuits
12. Finite State Machines
13. Design of FSMs
14. Design of FSMs
15. Datapaths

16. An introduction to Processor Design
17. The ARM Architecture
18. ARM Assembley Language Programming
19. Programming in C
20. Conclusion
4. For testing purposes we do not require your bit files. We will need a folder containing the VERILOG modules in a suitable form for loading directly into ISE WebPACK 9.2i. The design flow should execute continuously and free of errors. We should not have to do any PIN assignments. Consequently you must provide a suitable User Constraints File (UCF) in each of your source folders. You must check that the design flow works from scratch in each new project in ISE WePACK 9.2i BEFORE you upload your code.

5. Hand in separate project source code and UCF files in separate folders for each of levels 1, 2 and 3 ready for design flow implementation. Zip these up into the one file and name as UXXXXXX-NAME-RPC-CODE-ENGN3213-2010.zip.

6. Upload to WATTLE the report in PDF format and the code in ZIP format. Use the naming convention for the report UXXXXXX-NAME-RPC-REPORT-ENGN3213-2010.pdf

7. Hand in the 20 pages (no appendices) of your written report to the student admin in double sided and stapled hard copy.

8. **You may not use any third party code. All VERILOG code is to be the original work of the student save code offered for general use in the course.** You cannot use PICOBLAZE. If you think you need a Xilinx IP-core then check with me first.
Example: ADD Instruction (L16-17: RTL systems)

ADD

Fetch
Data Out  Address  Data In
ACC  PC
ALU
Timing and Control

Decode/Execute
Data Out  Address  Data In
ACC  PC
ALU
Timing and Control
Electronic Device Abstractions: Combinational Devices (L7, L8)

- A **combinational** circuit is a device that takes a digital input and produces a unique **instantaneous digital output** (save propagation delays etc).
- A combinational circuit is like a function $y = f(x)$ and we must allow sufficient time for its outputs to settle.

$\begin{array}{c}
X \\
\downarrow \\
Y
\end{array}$

\[ Y = f(X) \]
Electronic Device Abstractions: Sequential Devices (L9, L10)

In a **sequential** device the result is only transferred to the output at the active edge of a clock (e.g. D-type flip flop)

![Sequential Device Diagram]
Intro to Verilog I: Verilog RTL Program Structure (L5)

IO entry point

inputs/outputs

Test bench
Verilog module

module name
declarations

Control path
Finite state machine

Data path

Includes simulation code as well

The testbench does NOT appear in hardware

Synthesisable code only
Intro to Verilog III: Modelling sequential and combinational circuits (L5)

always @(posedge clk or A) begin
  //Non-blocking IO
  X <= Y;
  Y <= A;
end

always @(A or B) begin
  //Blocking IO
  D = B;
  C = A;
end

Sequential code assignment at the posedge clk

Combinational code
All values settle, clock period >> settling time
State Machine Schematically: Moore Machine

State-machine structure (Moore)

- Inputs
- Next-state Logic $F$
- Exciation
- State Memory
- Current state
- Output Logic $G$
- Outputs
- Clock signal
- Typically edge-triggered D flip-flops
- Output depends on state only
State Machine Schematically: Mealy Machine

State-machine structure (Mealy)

- Outputs depend on state and input
- Next-state logic $F$
- State memory
- Output logic $G$
- Clock input
- Clock signal
- Typically edge-triggered D flip-flops

ENGN3213: Digital Systems and Microprocessors L#12-13
Example: Pedx Controller: Circuit (CLOCK not shown)

- Input: Push-button W
- Car outputs: GYR light
- Pedestrian outputs: HALT/WALK
State Diagrams

Moore Machine

Mealy Machine
Pedestrian Traffic Controller State Transition Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 if ( \overline{W} ) else s1 if W</td>
<td>G, HALT</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>Y, HALT</td>
</tr>
<tr>
<td>s2</td>
<td>s3</td>
<td>R, WALK</td>
</tr>
<tr>
<td>s3</td>
<td>s0</td>
<td>R, HALT</td>
</tr>
</tbody>
</table>
State Machine Electronic Representation

Inputs

Logic

F(inputs, State)

Next State

Sequential

D Q

D Q

D Q

Output

Logic

G(state, input?)

Combination

Outputs
Example: Pedx Controller: Circuit

Input combinational logic
D-type flip-flops state transitions
Output combinational logic
Two State Moore Machine in Verilog

module FSM(clk,.....)

parameter S0 = 1'b0;
parameter S1 = 1'b1;

always @(posedge clk) state <= Snext; //STATE MEMORY -> SEQUENTIAL

always @(*) begin //NEXT STATE LOGIC
    case(state)
        S0:
            if(something) Snext = S1;
            else Snext = S0;
        S1:
            if(something_else) Snext = S0;
            else Snext = S1;
        default:
            Snext = S0;
    endcase
end

always @(state) //OUTPUT LOGIC
    case(state)
        S0:
            Output = f(state);
        S1:
            Output = g(state);
        default:
            Output = h(state);
    endcase
endmodule
Design II: Register Transfer Level Abstraction (ctd)

Control path

Data path

Hardware blocks

sequential

group sequential blocks

Clk
Figure 24: Datapath with 3-port register file [Gajski].
In designing a datapath we assume that a control word plus inputs is provided as an input to the system c.f. MU0

In a pure algorithmic datapath we have to control somehow → add a counter or clocked stack or FSM (RTL?)

Maybe we should consider algorithmic datapaths for simple tasks. More complex datapaths are promoted to RTLs.

But RTLs have a datapath anyway

The line is blurred....we are just learning new approaches to design
Figure 5-1: Character LCD Interface
Power-On Initialization

The initialization sequence first establishes that the FPGA application wishes to use the four-bit data interface to the LCD as follows:

- Wait 15 ms or longer, although the display is generally ready when the FPGA finishes configuration. The 15 ms interval is 750,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 4.1 ms or longer, which is 205,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 100 μs or longer, which is 5,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x3, pulse LCD_E High for 12 clock cycles.
- Wait 40 μs or longer, which is 2,000 clock cycles at 50 MHz.
- Write SF_D<11:8> = 0x2, pulse LCD_E High for 12 clock cycles.
- Wait 40 μs or longer, which is 2,000 clock cycles at 50 MHz.
always @(posedge sys_clk) begin
    SF_CE0 <= 1'b1;
    if(rst) begin
        //---------------------------------------------------------------
        if(~lcd_init_done) begin
            cnt <= cnt + 1'b1;
            //---------------------------------------------------------------
            if(~cnt[init_width]) begin // Enables a Flip-Flop
                if(&{cnt[{init_width-4}:5]}) begin // Enables a FF
                    if(~(cnt[init_width-1]&cnt[init_width-2])) // Does not need to enable FF
                        SF_D <= 4'b0011; else SF_D <= 4'b0010; // Decides SF_D (multiplexer)
                end
                //---------------------------------------------------------------
                if(&{cnt[{init_width-3}:4]}) begin // Enables a FF
                    if(~(cnt[3] & cnt[2])) begin // Enables a FF
                        LCD_RS <= 1'b1;
                        LCD_RW <= 1'b0;
                        LCD_E <= 1'b1; end else begin LCD_E <= 1'b0; // Decides LCD_E (multiplexer)
                    end
                    //---------------------------------------------------------------
                end else begin
                    lcd_init_done <= 1'b1;
                end
                //---------------------------------------------------------------
            end else begin
                //---------------------------------------------------------------
            end else begin
                lcd_init_done <= 1'b0;
                cnt <= 1'b0;
            end
            end // always @(posedge sys_clk)
The flip-flops remember states across clock ticks when they are disabled.
The C Programming Language

- C is a high level language... Looks more like VERILOG than assembly language but easier to learn and implement in most situations... even SystemC can output VERILOG

- Dennis Ritchie (Circa 1973)

- LINUX and WINDOWS are written in C

- Can use C to program micros

- C++ = Object oriented C

- UNIX and C developed together
The C Versions...  Why learn C?

- C has an ANSI standard form “ANSI Standard C”.

- Particular versions of C for different applications:
  - Microsoft Visual C, (Borland, Watcom), … ,
  - and MINGW for Intel, GNU tools,
  - Keil, uCLINUX and SDCC for embedded devices:
    ARM, MIPS, 8081, Motorola 68000 and even PICs
    - All recognise ANSI standard C

- Can use C to program micros but also... to write complex software tools (I used it write software for in-class SMS quizzes) and Graphical User apps (GUIs)

- Useful skill to learn
Proverbial 'Hello World' main module to do system call printf()

```c
#include <stdio.h>

int main()
{
    printf("Hello World!!\n");
    return 1;
}
```

classical 'Hello World' main module preprocessor directive

printf.. the most useful debugging tool

3213: Digital Systems & Microprocessors: L#22
C modules, type, control loops, return values

```c
int b2b( int btst[8] )
{
    int theByte = 0;
    int index = 1;
    int i;

    for(i=0; i < 8; i++)
    {
        theByte = theByte + index * btst[i];
        index = 2 * index;
    }
    return theByte;
}

int main()
{
    int bts[8], i;
    int TheByte;

    for(i=7; i >= 0; i--)
    {
        bts[i] = i % 2;
        printf("%d", bts[i]);
    }
    printf("\n");
    TheByte = b2b(bts);
    printf("TheByte = %d\n", TheByte);
    return 1;
}
```
More data types

strings are pointers to chars

floating point and double precision

short 16 bit
int, long 32 bit

char = 8 bit (the byte!)

int main()
{
    char c = 'A';  // A character
    char * name = "ALFRED"; // A string

    float f = 32.33;
    double g = 32.33;
    double d = 1.2e-13;

    short sshort = 32767;
    int sint = 32767;
    long slong = 32767;

    unsigned short usshort = 32767;
    unsigned int usint = 32767;
    unsigned long uslong = 32767;

    char sc = 255;
    char tc = 0x0;  // Hex
    char yc;

    unsigned char uc = 255;
    unsigned char vc = 0x0;  // Hex
    unsigned char wc;
    return 1;
}
Math header

Preprocessor directive #define

cosine, sine, log, exp etc
Read files
headers
Command Line Arguments

return;

tclose(firtitle);  
{
flag = fscanf(firtile, "%c\n", next-char);
fprintf(outtile, "%c\n", next-char);
}

while (EOF || flag)
{
   for i = flag
      flag = fscanf(firtile, "%c\n", next-char);

   int tile = fopen(tileename, "r");

   if argc > 2
      int flg;
      char next-char;
      FILE *inttile;
      char tileename[10], *name;

      main(argc, argv)

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
Pointers

Pointers point to memory occupied by a variable.
Use the * operator

q and j passed by reference
k passed only by value

A pointer

Assigning a value to a pointer

The & gives the address of the variable following it

Question:
What happens to p, i and k?

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

void squareme( int * q, int * j, int k ) {
    *q = (*q)*(*q);
    *j = (*j)*(*j);
    k = k*k;
}

int main(void)
{
    int *p = malloc(sizeof(int));
    int i,k;

    *p = 10;
    i = 10;
    k = 10;
    printf("*p = %d\n",*p);
    printf("i = %d\n",i);
    printf("k = %d\n",k);

    squareme( p, &i, k);
    printf("*p = %d\n",*p);
    printf("i = %d\n",i);
    printf("k = %d\n",k);

    return 1;
}
```
What output does this code produce?

```c
void swap(int a, int *b)
{
    int temp;
    temp = a;
    a = *b;
    *b = temp;
}
int main()
{
    int x = 2, y = 3;
    swap(x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 1;
}
```
More fun with pointers: **KEEP IT SIMPLE STUPID PROTOCOL**

```c
static int kiss_esc(unsigned char *s, unsigned char *d, int len)
{
    unsigned char *ptr = d;
    unsigned char c;
    /*
       * Send an initial END character to flush out any
       * data that may have accumulated in the receiver
       * due to line noise.
       */
    *ptr++ = END;
    while (len-- > 0) {
        switch (c = *s++) {
            case END:
                *ptr++ = ESC;
                *ptr++ = ESC_END;
                break;
            case ESC:
                *ptr++ = ESC;
                *ptr++ = ESC_ESC;
                break;
            default:
                *ptr++ = c;
                break;
        }
    }
    *ptr++ = END;
}
```