OVERVIEW

The Frontend-Controller replaces about half a dozen ICs one usually needs for human-machine-interaction and the lower control layers. A simple, but extremely powerful structure provides a variety of unexpected possibilities.

APPLICATION OPERATION

The Frontend Controller (Figure 1) provides two hold-and-repeat functional debounced button inputs, which drive a binary up/down counter. Using a lookup table, the counter value can be converted to decimal or gray code, for example. Alternatively, an individual pattern sequence may be released on the output for each counter stage, generating pulses or control signals for succeeding logic.

Let us go into detail first before demonstrating the power of this structure:

FIGURE 1: FRONTEND CONTROLLER

Note: Either 2 inputs and 4 outputs or 1 input and 5 outputs possible.

Both button inputs are debounced as well as protected against noise. Upon pressing one of the buttons, the counter will change its state. Holding the button pressed for a certain time will initiate a repeat function, simulating a repeated button pressing (similar to the functionality of a PC keyboard).

The counters range is adjustable and not forced to multiples of two (e.g. 256), as the counters from the TTL series are. Upon reaching the counters maximum or minimum, it may either rollover or ignore any counting.
pulses in the corresponding direction. This way it is even possible to implement a modulo-5 counter (or frequency divider), for example.

Using a lookup-table, the counter value may be converted. Therefore it is possible to have the Frontend-Controller count binary, decimal or gray-code. The lookup-table may also contain bit patterns to drive succeeding components. This is even surpassed by the pattern player. It may release a whole sequence of patterns, individual for each counter stage, to generate pulses and control sequences for succeeding logic. Each sequence may be repeated upon reaching its end or it may be played just once.

Of course, all timings and frequencies are adjustable.

**EXAMPLE APPLICATIONS**

**Binary Modulo-7 Counter with Hold-and-Repeat**

Set the counter to seven stages, enable rollover feature and store binary values 0…6 into the lookup-table. A discrete solution would need button debouncing, two timers for hold and repeat, a counter, logic to detect counter stage 7 and reset counter: at least five integrated circuits.

**Eight Stage PWM Generator with LED Display**

Set the counter to eight stages, optionally activate rollover feature and write eight sequences (each one eight patterns long) for the pattern player. Bit 0…2 of each sequence holds the binary value of the sequence number (which corresponds to the counter value). Connect a 7-segment LED driver to these outputs. Bit 3 will provide the PWM output: For sequence #n, it will be high in n patterns, and low in the remaining (8-n) patterns. Enable the loop feature for all eight sequences.

**Six Stage Counter with (RS232) Serial Output**

Set the counter to six stages and write six sequences. Bit 0.2 always contains the binary equivalent of the stage number, bit 3 is used to release the serial bit stream (therefore, the sequence length has to be about 10). Disable loop feature.

You see, the Frontend-Controllers structure is simple but extremely powerful!

**GRAPHICAL HARDWARE REPRESENTATION**

![Diagram](image)

**MICROCHIP TOOLS USED**

Picstart Plus V1.3

**ASSEMBLER/COMPILER VERSION**

MPASM V1.5
Notes on the software: A routine called Interrupt is permanently called from the main program. The routine will check the TMR0 for an overflow (which will happen regularly) and execute a kind of software interrupt routine then. This will debounce the inputs (to make the main program get rid of that), and it will handle the pattern player to make it run independent from the main program.

Furthermore, the software listing contains some sample applications.

```assembly
;****************************************************************************
;* DESIGNING FOR DOLLARS entry: FRONTEND-CONTROLLER                        *
;* by Marc Hoffknecht & Gabi Borgards                           *
;*****************************************************************************
processor 12c508
radix dec
include "p12c508.inc"

#define __12C508
#define _config _WDT_OFF & _IntRC_OSC & _MCLRE_OFF & _CP_OFF

CBLOCK 0x0C ; start of RAM
ENDC

GOTO Main

;****************************************************************************
;* macros & standard definitions                                            *
;*****************************************************************************
#define zero STATUS, 2
#define carry STATUS, 0
#define SZ BTFSS zero ; (s)kip if (z)ero
#define SNZ BTFSC zero ; (s)kip if (n)ot (z)ero
#define RET RETLW 0 ; (ret)urn
MOVLF MACRO literal, file ; (mov)e (l)iteral to (f)ile
MOVLW literal
MOVF file
ENDM
#define TRUE -1
#define FALSE 0
STrue MACRO ; (s)kip if (true)
IORLW 0
BTFSC zero
ENDM
SFalse MACRO
IORLW 0
BTFSS zero
ENDM

;****************************************************************************
#define TMR0overflow 256 ; TMR0 will overrun every 256 us
#define us 1/TMR0overflow
```
#define ms 1000/TMR0overrun

EOS EQU 128 ; marks (e)n(d) (o)f (s)equence
RS EQU 128+64 ; marks (r)epeat (s)equence

; reorder bits from pattern
; definition to suit pinout:
Pattern MACRO x
RETLW (((x)&3)<<1)|(((x)&12)<<2)|(((x)&16)>>4)|(((x)&192))
ENDM

Value MACRO x
RETLW (((x)&3)<<1)|(((x)&12)<<2)|(((x)&16)>>4)|(EOS)
ENDM

;****************************************************************************
;* configuration                                                            *
;****************************************************************************

;     ;   range:  purpose:
OutputWidth EQU 5     ;   4-5 one input only if set to 5
DebounceTime EQU 50*ms  ; .25-65ms duration of input bouncing
RepeatDelay EQU 500*ms  ; -16 s delay till automatic button repeat
RepeatTime EQU 500*ms  ; -16 s speed of automatic button repeat
Pattern.Delay EQU 50*ms  ; .25-65ms speed of playing patterns
PullUps EQU TRUE    ;  activate internal pullups on inputs
RollOver  EQU TRUE    ; counter rollover feature

; sample application: five stage PWM generator
; bit 2.0 output stage number ( 0 ,  1 ,  2 ,  3 ,   4 ),
; bit 3 outputs a PWM signal of  0% , 25% , 50% , 75% , 100%
Pattern.Length EQU 4

Pattern 0+0 ; use the 'pattern' macro to define
Pattern 0+0 ; sequences of patterns ...
Pattern 0+0 ; and mark each sequence's end with either
Pattern 0+0+RS ; RS = repeat sequence or
Pattern 1+8
Pattern 1+0
Pattern 1+0
Pattern 1+0+RS
Pattern 2+8
Pattern 2+8
Pattern 2+0
Pattern 2+0+RS
Pattern 3+8
Pattern 3+8
Pattern 3+8
Pattern 3+0+RS
Pattern 4+8
Pattern 4+8
Pattern 4+8
Pattern 4+8+RS

; 2nd sample application: five stage decimal counter
; OutputWidth EQU 5
; Pattern.Length EQU 1; no sequence of patterns
; D4...D0
Discrete Logic Replacement

; Value b'00001' ; use the 'value' macro to define
; Value b'00010' ; patterns only ( no sequence )
; Value b'00100' ; ( in fact 'value' defines a one pattern
; Value b'01000' ; long sequence and automatically marks
; Value b'10000' ; the EOS ).

Pattern.Max EQU (($-1)/Pattern.Length)-1 ; calculate number of sequences

****************************************************************************
CountUp GOTO CountUp_ ; since CALL can only access the
CountDown GOTO CountDown_ ; first 256 byte of memory, we need
; to place these jump vectors here.
****************************************************************************

;* button debouncing                                                        *
****************************************************************************

; This section handles the button input(s). The variable 'Buttons' holds the
; valid (debounced) state of the corresponding pin. When the button state
; is changing, a timer will start running down. If it reaches zero, the button
; was validly pressed, otherwise it had to be noise.

CBLOCK
Buttons ; holds the valid button states
TimerUp
TimerDown
ENDC

#define Button.Up Buttons, 3
#define Button.DownButtons, 0

; InitButtons MACRO
MOVLF DebounceTime, TimerUp
MOVLF DebounceTime, TimerDown
MOVLF 255, Buttons ; button inputs are active low
ENDM

; HandleButtons MACRO
HandleUp MOVF GPIO, W ; mark all pins that differ between
XORWF Buttons, W ; 'GPIO' and 'Buttons' with a '1',
ANDLW b'001000' ; and mask out button (bit 3).
SNZ ; 'w' will be zero, if there are no
GOTO ReloadTimerUp ; changes,
DECF TimerUp ; otherwise, a timer has to reach
GOTO HandleDown ; zero first.
XORWF Buttons

ReloadTimerUp MOVLF DebounceTime, TimerUp

HandleDown IF OutputWidth==4
MOVF GPIO, W
XORWF Buttons, W
ANDLW b'000001'
SNZ ; 'w' will be zero, if there are no
GOTO ReloadTimerDown ; changes,
DECF TimerUp ; otherwise, a timer has to reach
GOTO Buttons.done
XORWF Buttons

ReloadTimerDown MOVLF DebounceTime, TimerDown
;;; This section provides routines for an interrupt driven pattern player.
;; The 'SetPattern' macro is used to arrange everything for a new pattern
;; sequence. Therefore, load 'w' with the memory location of the first pattern
;; in the sequence. Bits 6 & 7 are used as a repeat-flag & end-of-pattern
;; flag.

CBLOCK
  Pattern.Timer
  Pattern.Begin
  Pattern.Pointer
  Pattern.Current
ENDC

ReadMemory
  MOVWF PCL

SetPattern MACRO
  ; load pointer to next patt.
  ; into 'w' and execute this
  ;
  MOVWF Pattern.Pointer ; save pointer
  CALL ReadMemory ; load word at w from memory
  MOVWF Pattern.Current ; and save it
  MOVWF GPIO
  MOVLF Pattern.Delay, Pattern.Timer
ENDM

HandlePatternPlayer MACRO
  LOCAL done

  DECFSZ Pattern.Timer
  GOTO done

  BTFSC Pattern.Current, 7 ; bit 7 marks end of sequence
  GOTO EndOfSequence ; jump, if we are at the end

  INCF Pattern.Pointer, W ; no, we are within the seq.
  SetPattern
  GOTO done

EndOfSequence
  BTFSS Pattern.Current, 6 ; end of sequence reached
  GOTO done ; exit, if no 'RepeatFlag'
  MOVF Pattern.Begin, W
  SetPattern

  done
ENDM

;; Execute 'CALL Interrupt' every now and then. It will check the TMRO for
;; an overflow. When an overflow is detected, a kind of 'software-interrupt'
;; is executed. The oftener 'Interrupt' is called, the more accurate will the
;; timing be ...
; The interrupt routine will provide a timer which is incremented about every 256 us. TRUE will be returned in 'w' when this timer hits zero.

CBLOCK
OldTMR0
TimerL
TimerH
ENDC

; Interrupt  MOVF OldTMR0, W ; overflow occured if OldTMR0 > TMR0
    SUBWF TMR0, W
    BTFSC carry
    GOTO Interrupt.done
    ADDWF OldTMR0

; ***********************************************************
; software-interrupt: program enters here about every 256 us
HandleButtons
HandlePatternPlayer
    INCFSZ TimerL
    RETLW FALSE
    INCFSZ TimerH
    RETLW FALSE
    RETLW TRUE ; return TRUE upon hitting zero
; ***********************************************************

Interrupt.done ADDWF OldTMR0
    RETLW FALSE

;*****************************************************************************
LoadTimer MACRO Value
    MOVLW low(-Value)
    MOVWF TimerL
    MOVLW high(-Value)
    MOVWF TimerH
ENDM

;*****************************************************************************
;* multiply macro                                                          *
;*****************************************************************************

; (MUL)tiply(L)iteral(W)    performs w * literal -> w
CBLOCK
mul
ENDC

MULLW MACRO literal
    VARIABLE i

    MOVWF mul
    CLRW
    BCF carry

    i=0
    WHILE ( literal >>= (1<<i) )
    IF ( literal & (1<<i) )
        ADDWF mul, W
    ENDF
RLF mul
i += 1
ENDW
ENDM

;*****************************************************************************
;* main program                                                           *
;*****************************************************************************

CBLOCK
Counter
ENDC

Main MOVWF OSCCAL

IF PullUps
  MOVLW b’10011000’
ELSE
  MOVLW b’11011000’
ENDIF
OPTION ; -> TMR0overrun every 256 us
MOVLW 1 ; initialize pattern player
MOVWF Pattern.Begin ; set first pattern ( sequence )
SetPattern
CLRF Counter

IF OutputWidth==4
  MOVLW b’001001’ ; two button inputs
ELSE
  MOVLW b’001000’ ; only one button input
ENDIF
TRIS GPIO

InitButtons

;*****************************************************************************
MainLoop CALL Interrupt

BTFSS Button.Up ; button inputs are active low
  GOTO UpPressed
BTFSS Button.Down
  GOTO DownPressed
  GOTO MainLoop

; UpPressed CALL CountUp ; Upon pressing button, count one
LoadTimer RepeatDelay ; stage up. When holding the button
UpLoop

UpPressed CALL CountDown ; exit if button is no more pressed
LoadTimer RepeatDelay

DownPressed CALL CountUp
LoadTimer RepeatDelay

DownLoop BTFSC Button.Down
  GOTO MainLoop
  GOTO UpLoop
GOTO DownLoop
GOTO DownLoop ; hitting zero in the timer.
CALL CountDown
LoadTimer RepeatTime
GOTO DownLoop

; CountDown_ TSTF Counter ; check counter boundaries and count
; SZ ; one stage down here.
GOTO Decrease
IF RollOver
MOVLF Pattern.Max, Counter
GOTO CalcPatternBegin
ELSE
RET
ENDIF
Decrease DECF Counter
GOTO CalcPatternBegin

; CountUp_ MOVLW Pattern.Max; check counter boundaries and count
; SUBWF Counter, W ; one stage up here.
BTFSS carry
GOTO Increase
IF RollOver
CLR Counter
ELSE
MOVLF Pattern.Max, Counter
ENDIF
GOTO CalcPatternBegin
Increase INC Counter
GOTO CalcPatternBegin

; CalcPatternBegin ; Calculate the beginning of the
MOV Counter, W ; pattern for the new 'Counter'
MULLW Pattern.Length ; value...
MOVWF Pattern.Begin
INCF Pattern.Begin ; pattern start at memory loc. 1
MOV Pattern.Begin, W ; prepare to play this pattern
SetPattern
RET

;*******************************************************

END