Using the Tap Tempo

The FXCore has a built in tap tempo function that provides the number of samples between two successive taps. In addition to this count it will debounce the tap input and can indicate if the tap button is being held down or what we call in a “sticky” state.

The following special function registers (SFRs) are available to the user, some are read only other can only be set in the program header, etc.:

TAPTEMPO : This is read only and it the count in samples between two successive taps.
MAXTEMPO : This is a value in samples that the user can read/write to set the maximum allowed number of samples between successive taps. This allows the user to set a “time out” value.
TAPSTKRLD : This value can only be set in the header of a program with a .sreg directive, this value sets the number of samples a user must hold down the tap button for it to be considered “sticky”
TAPDBRLD : This value can only be set in the header of a program with a .sreg directive, this value sets the number of samples to debounce the tap input for.

In addition to the above SFRs the following bits in the FLAGS register are relevant to the tap tempo mechanism:

FLAGS[5] : TB2nTB1	If equal to 0 then a TB1* event, if 1 then a TB2* event
FLAGS[4] = TapSTKY	1=TAP switch pushed longer than TAPSTKRLD
FLAGS[3] = NewTT	1=A new TAP Tempo value was measured ,valid for 1 sample period
FLAGS[2] = TapRE	1=The TAP was just released ,valid for 1 sample period
FLAGS[1] = TapPE	1=The TAP switch was just pushed, valid for 1 sample period
FLAGS[0] = TapDB	Debounced logic level of the TAP switch

* : TB1 is the first button push and TB2 is the second button push

```
// Example using the tap tempo - an-4_a.fxc
// This example creates an adjustable delay on channel 0 using
// the tap tempo to control the delay.

// First define a delay line of some length
.mem delay 32767

// Using tap tempo for a basic delay is very simple
// First read IN0 and write it to the delay line
cpy_cs r0, in0
wrdel delay, r0

// Next put the start address of the delay line in r1
wrdld r1, delay
```



```
// Since wrdld loads numbers into the upper 16-bits and
// we need them in the lower 16 do a logical shift right
sr r1, 16

// Result is in acc 32 so save it to r1 for now
cpy_cc r1, acc32

// Read the tap tempo count, count is in sample periods
cpy_cs r2, taptempo

// Now simply add the base address of the delay to the tap count
add r2, r1

// Use the result in acc32 as a pointer into the delay line and
// put the sample in r1
rddelx r1, acc32

// Add the dry signal to the delayed signal
adds r0, r1

// Output it
cpy_sc out0, acc32
```

This next example is more complex, it start up by using POT0 to set the delay time and reads the FLAGS register to see if the user entered a tap temp0 (bit 3 the NewTT flag). If the user did then it switches from using POT0 to using the tapped in value. The program continues to monitor the FLAGS register and if the user presses and holds tap button for longer than the "sticky" timeout then bit 4 (TapSTKY) is set and the program will switch back to using POT0 for the delay time.

```
// Example using the tap tempo - an-4_b.fxc
// This example creates an adjustable delay on channel 0
// On start the delay is set by POT0, if the user taps in
// a delay we use that and if the user want to go back to
// POT0 then the user holds the TAP button until the
// "sticky" time out

// First define a delay line of some length
.mem delay 32767

// Initialize r9 to 0 to select POT at startup
.creg r9 0

// Read IN0 and write it to the delay line
cpy_cs r0, in0
wrdel delay, r0

// registers are preset on start so if r9 is 0 use POT0
// any other value means use the tap button
```

```
// if we got a sticky on tap 1 switch to POT0
andi flags, 0x10      // bit 4 must be set
jz acc32, next        // if not jump past rest
andi flags, 0x20      // we want bit 5 to be 0 so check if it is set
jnz acc32, next       // if not 0 then was not tap 1 so jump past rest
andi r9, 0x0000       // if here we got a tap 1 sticky so clear r9
cpy_cc r9, acc32

next:
// decide POT or tap
// Do we have a new TAP count?
andi flags, 0x8        // NewTT is bit 3
jz acc32, no_tap       // if no new tap make no change
ori r9, 0x0001         // set the lsb to indicate we now use tt
cpy_cc r9, acc32       // save it
cpy_cs r8, taptempo    // get the tap count into r8
jmp do_delay

// if here no new tap but decide if we need to update count from the
// POT
no_tap:
andi r9, 0x0001        // if lsb is 1 then we are in tap mode, no pot
                       // update
jnz  acc32, do_delay   // not 0 so using tap count from above
cpy_cs r1, pot0_smth   // read in the pot value to r1
wrdld r2, delay!       // get length of the delay
multrr r1, r2          // length times pot
sr acc32, 16           // shift down to lower 16 bits
cpy_cc r8, acc32


do_delay:
// Next put the start address of the delay line in r1
wrdld r1, delay

// Since wrdld loads numbers into the upper 16-bits and
// we need them in the lower 16 do a logical shift right
sr r1, 16

// Now simply add the base address of the delay to the delay
// calculated
add r1, r8

// Use the result in acc32 as a pointer into the delay line and
// put the sample in r1
rddelx r1, acc32

// Add the dry signal to the delayed signal
adds r0, r1

// Output it
cpy_sc out0, acc32
```

Experimental Noize Inc.

Scottsdale, AZ USA

www.xnoize.com

sales@xnoize.com