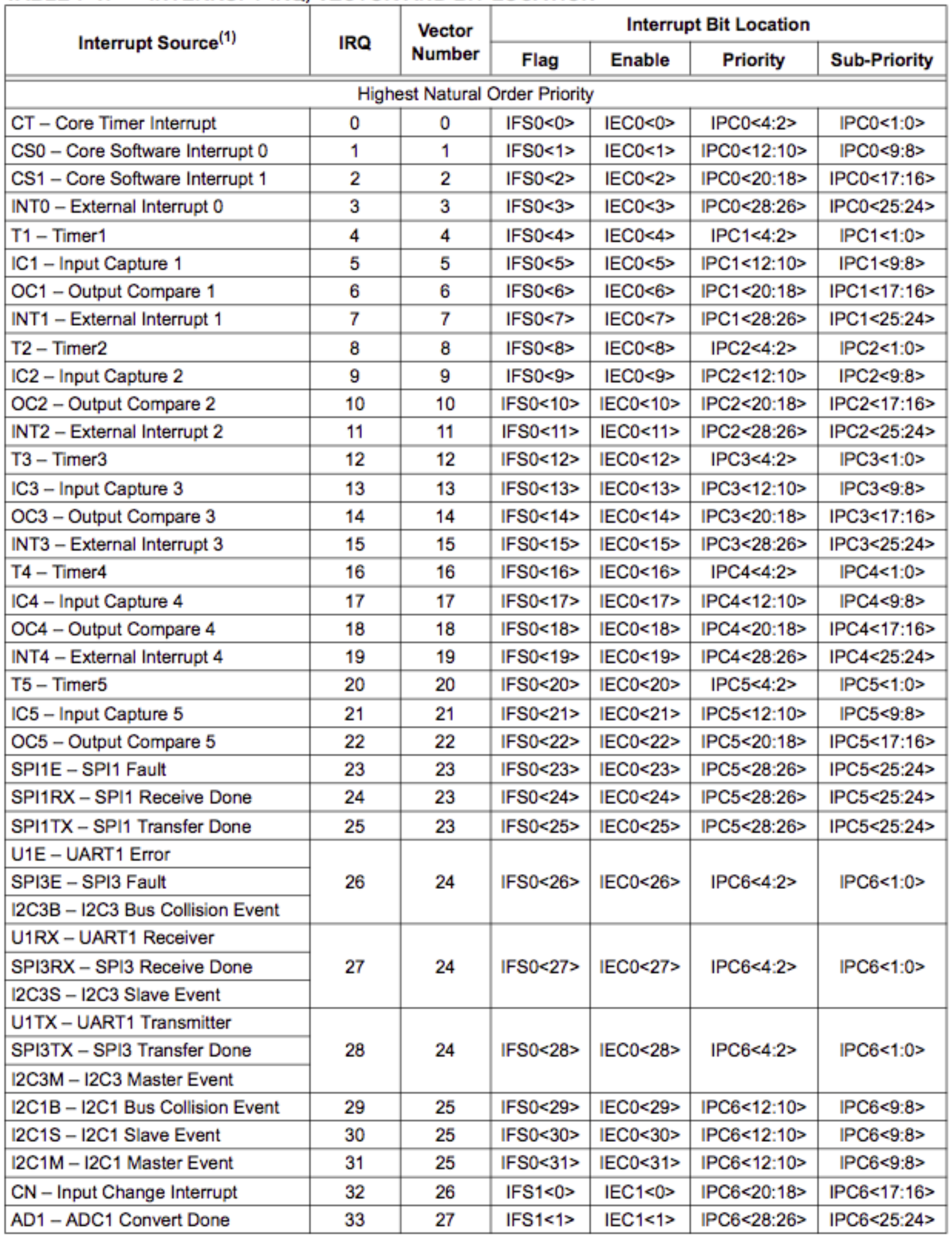
ECE 2534

- Change Notification on the PIC32
- Interrupts from the Accelerometer
- Tap Detection on the Accelerometer

Reminder

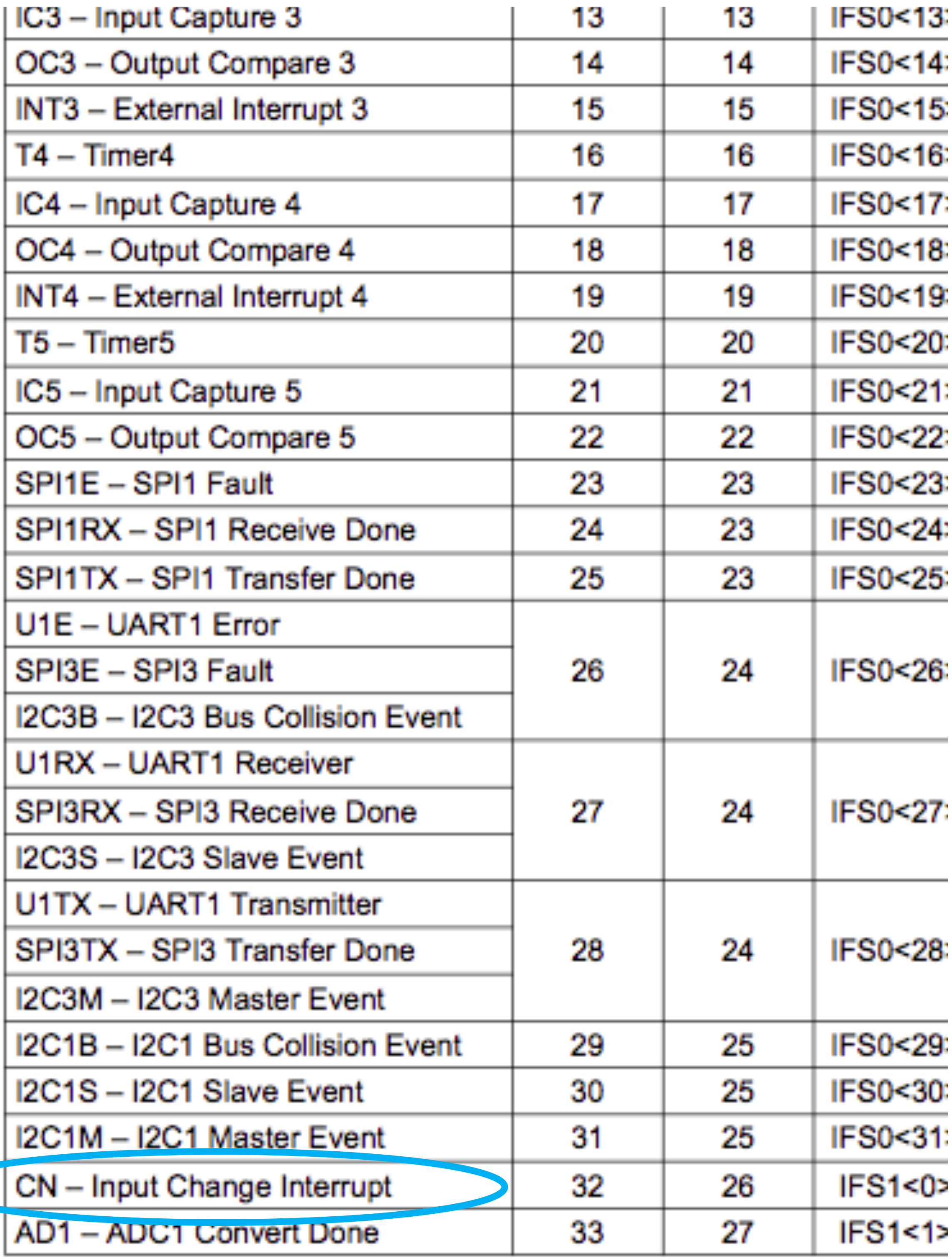
- Our PIC32 contains an interrupt controller that supports many different sources of interrupts

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION



Interrupt Source[1]	IRQ	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>
IC1 – Input Capture 1	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>
OC1 – Output Compare 1	6	6	IFS0<6>	IEC0<6>	IPC1<20:18>	IPC1<17:16>
INT1 – External Interrupt 1	7	7	IFS0<7>	IEC0<7>	IPC1<28:26>	IPC1<25:24>
T2 – Timer2	8	8	IFS0<8>	IEC0<8>	IPC2<4:2>	IPC2<1:0>
IC2 – Input Capture 2	9	9	IFS0<9>	IEC0<9>	IPC2<12:10>	IPC2<9:8>
OC2 – Output Compare 2	10	10	IFS0<10>	IEC0<10>	IPC2<20:18>	IPC2<17:16>
INT2 – External Interrupt 2	11	11	IFS0<11>	IEC0<11>	IPC2<28:26>	IPC2<25:24>
T3 – Timer3	12	12	IFS0<12>	IEC0<12>	IPC3<4:2>	IPC3<1:0>
IC3 – Input Capture 3	13	13	IFS0<13>	IEC0<13>	IPC3<12:10>	IPC3<9:8>
OC3 – Output Compare 3	14	14	IFS0<14>	IEC0<14>	IPC3<20:18>	IPC3<17:16>
INT3 – External Interrupt 3	15	15	IFS0<15>	IEC0<15>	IPC3<28:26>	IPC3<25:24>
T4 – Timer4	16	16	IFS0<16>	IEC0<16>	IPC4<4:2>	IPC4<1:0>
IC4 – Input Capture 4	17	17	IFS0<17>	IEC0<17>	IPC4<12:10>	IPC4<9:8>
OC4 – Output Compare 4	18	18	IFS0<18>	IEC0<18>	IPC4<20:18>	IPC4<17:16>
INT4 – External Interrupt 4	19	19	IFS0<19>	IEC0<19>	IPC4<28:26>	IPC4<25:24>
T5 – Timer5	20	20	IFS0<20>	IEC0<20>	IPC5<4:2>	IPC5<1:0>
IC5 – Input Capture 5	21	21	IFS0<21>	IEC0<21>	IPC5<12:10>	IPC5<9:8>
OC5 – Output Compare 5	22	22	IFS0<22>	IEC0<22>	IPC5<20:18>	IPC5<17:16>
SPI1E – SPI1 Fault	23	23	IFS0<23>	IEC0<23>	IPC5<28:26>	IPC5<25:24>
SPI1RX – SPI1 Receive Done	24	23	IFS0<24>	IEC0<24>	IPC5<28:26>	IPC5<25:24>
SPI1TX – SPI1 Transfer Done	25	23	IFS0<25>	IEC0<25>	IPC5<28:26>	IPC5<25:24>
U1E – UART1 Error	26	24	IFS0<26>	IEC0<26>	IPC6<4:2>	IPC6<1:0>
SPI3E – SPI3 Fault						
I2C3B – I2C3 Bus Collision Event						
U1RX – UART1 Receiver	27	24	IFS0<27>	IEC0<27>	IPC6<4:2>	IPC6<1:0>
SPI3RX – SPI3 Receive Done						
I2C3S – I2C3 Slave Event						
U1TX – UART1 Transmitter	28	24	IFS0<28>	IEC0<28>	IPC6<4:2>	IPC6<1:0>
SPI3TX – SPI3 Transfer Done						
I2C3M – I2C3 Master Event						
I2C1B – I2C1 Bus Collision Event	29	25	IFS0<29>	IEC0<29>	IPC6<12:10>	IPC6<9:8>
I2C1S – I2C1 Slave Event	30	25	IFS0<30>	IEC0<30>	IPC6<12:10>	IPC6<9:8>
I2C1M – I2C1 Master Event	31	25	IFS0<31>	IEC0<31>	IPC6<12:10>	IPC6<9:8>
CN – Input Change Interrupt	32	26	IFS1<0>	IEC1<0>	IPC6<20:18>	IPC6<17:16>
AD1 – ADC1 Convert Done	33	27	IFS1<1>	IEC1<1>	IPC6<28:26>	IPC6<25:24>

- Our PIC32 contains an interrupt controller that supports many different sources of interrupts
- Some I/O port pins can generate interrupts simply by changing logic level
- PIC32 documentation refers to this as CN, or “Change Notification”, or “Change Notice”



IC3 – Input Capture 3	13	13	IFS0<13
OC3 – Output Compare 3	14	14	IFS0<14
INT3 – External Interrupt 3	15	15	IFS0<15
T4 – Timer4	16	16	IFS0<16
IC4 – Input Capture 4	17	17	IFS0<17
OC4 – Output Compare 4	18	18	IFS0<18
INT4 – External Interrupt 4	19	19	IFS0<19
T5 – Timer5	20	20	IFS0<20
IC5 – Input Capture 5	21	21	IFS0<21
OC5 – Output Compare 5	22	22	IFS0<22
SPI1E – SPI1 Fault	23	23	IFS0<23
SPI1RX – SPI1 Receive Done	24	23	IFS0<24
SPI1TX – SPI1 Transfer Done	25	23	IFS0<25
U1E – UART1 Error	26	24	IFS0<26
SPI3E – SPI3 Fault			
I2C3B – I2C3 Bus Collision Event			
U1RX – UART1 Receiver	27	24	IFS0<27
SPI3RX – SPI3 Receive Done			
I2C3S – I2C3 Slave Event			
U1TX – UART1 Transmitter	28	24	IFS0<28
SPI3TX – SPI3 Transfer Done			
I2C3M – I2C3 Master Event			
I2C1B – I2C1 Bus Collision Event	29	25	IFS0<29
I2C1S – I2C1 Slave Event	30	25	IFS0<30
I2C1M – I2C1 Master Event	31	25	IFS0<31
CN – Input Change Interrupt	32	26	IFS1<0>
AD1 – ADC1 Convert Done	33	27	IFS1<1>

Change Notification (CN)

- Different PIC32 versions contain up to 22 different CN pins
- CN pins are scattered over the various I/O ports
- Here are some on Port D, for example:
  - Bit 4 -- CN13
  - Bit 5 -- CN14
  - Bit 7 -- CN16
  - Bit 14 – CN20

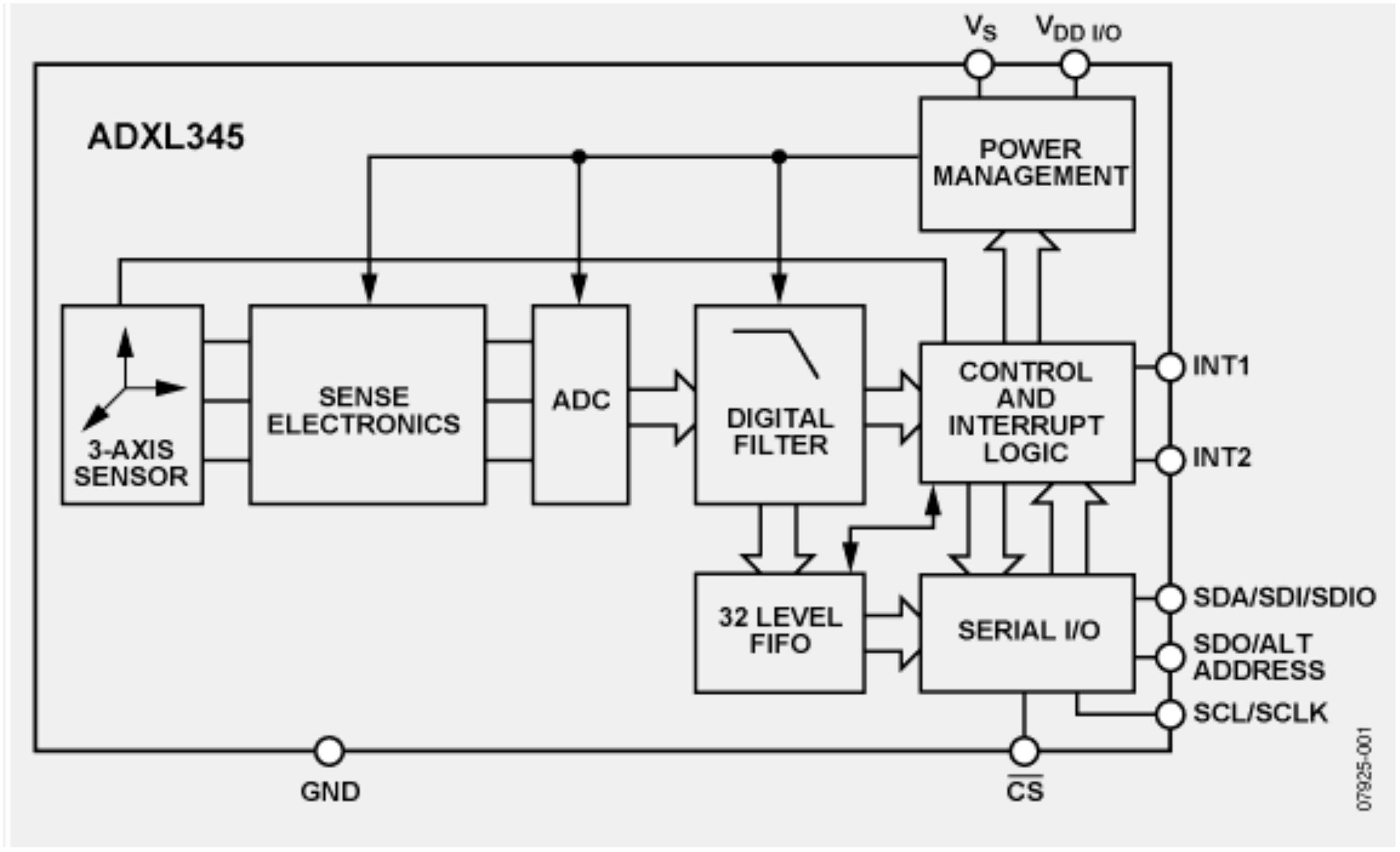
- Our SW can configure the interrupt controller to generate an interrupt in response to a change on a CNxx signal
- Interesting point: there is a single CN interrupt vector,
  but up to 22 different CN pins that can cause the interrupt

Change Notification (CN): Configuration steps

1. Make sure the relevant CN pins are inputs
   → PortSetPinsDigitalIn
2. Enable individual pins
   → mCNOpen (← *legacy?*)
3. Read each PORT pin to initialize the CN level
   → PortREAD
4. Set CN Interrupt Priority
   → INTSetVectorPriority
5. Clear CN Interrupts (pending)
   → INTClearFlag
6. Enable CN Interrupts
   → INTEnable (INT_CN, INT_ENABLED);
7. Enable CPU Interrupts
   → INTConfigureSystem(…); INTEnableInterrupts();

Interrupts on the ADXL345 accelerometer

- The ADXL345 provides 2 output pins for interrupts: INT1 and INT2
- Normally these are active-high, but can be changed to active-low
- The purpose: get the attention of the microcontroller when certain event(s) occur



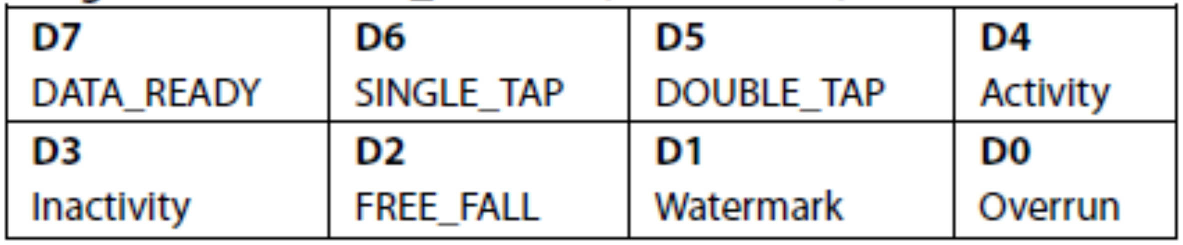
Interrupts on the ADXL345 accelerometer

- What sort of events should cause the ADXL345 to generate interrupts?

Interrupts on the ADXL345 accelerometer

- What sort of events should cause the ADXL345 to generate interrupts?

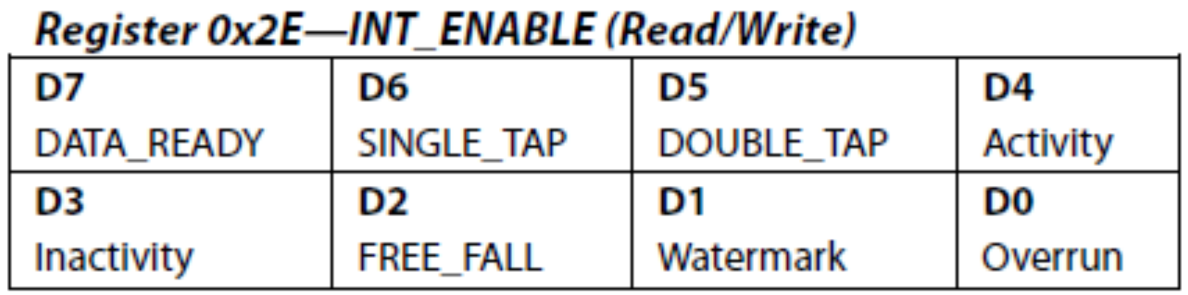
Register 0x2E—INT_ENABLE (Read/Write)



D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

- Writing 1 to any of these bit positions allows this type of event to generate interrupts

Interrupts on the ADXL345 accelerometer



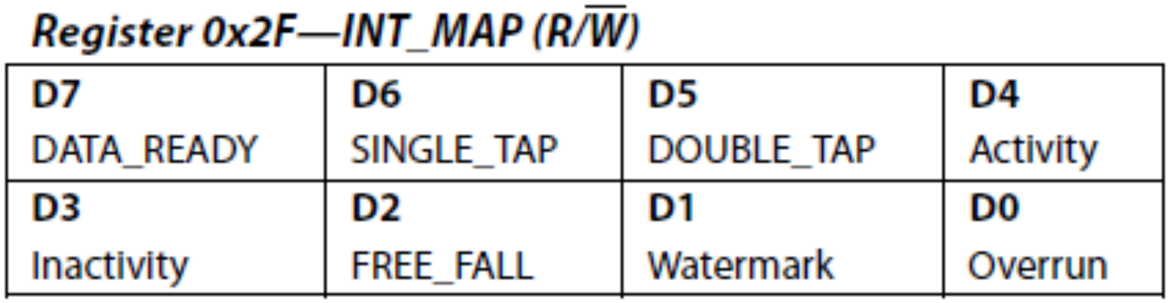
Register 0x2E—INT_ENABLE (Read/Write)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

- DATA_READY: measurements are available for the master to read
- SINGLE_TAP: a sudden, short acceleration
- DOUBLE_TAP: 2 sudden, short accelerations in quick succession
- Activity: a measurement exceeds THRESH_ACT m*g*
- Inactivity: measurements are less than THRESH_INACT m*g* for TIME_INACT seconds
- FREE_FALL: all measurements are less than THRESH_FF m*g* for TIME_FF x 5 milliseconds
- Watermark: the number of samples in the FIFO equals a given value
- Overrun: new measurement data replaces unread data

Interrupts on the ADXL345 accelerometer

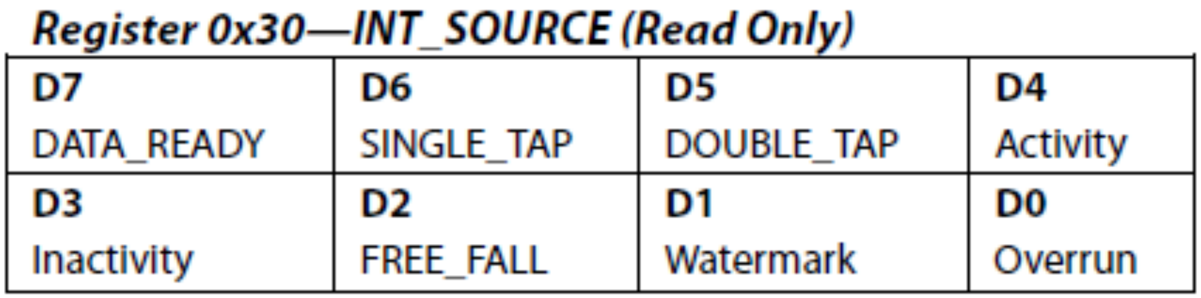
- Each enabled interrupt must be mapped to an interrupt pin
  - 0 = generate interrupt on INT1 pin
  - 1 = generate interrupt on INT2 pin



Register 0x2F—INT_MAP (R/$\overline{W}$)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

- A 1 in any of these bit positions means that this event type has occurred



Register 0x30—INT_SOURCE (Read Only)

D7 DATA_READY	**D6** SINGLE_TAP	**D5** DOUBLE_TAP	**D4** Activity
D3 Inactivity	**D2** FREE_FALL	**D1** Watermark	**D0** Overrun

Tap detection on the ADXL345 accelerometer

- A “tap” is a sudden, short acceleration
- The ADXL345 has special hardware to detect <u>single</u> and <u>double</u> taps in any direction

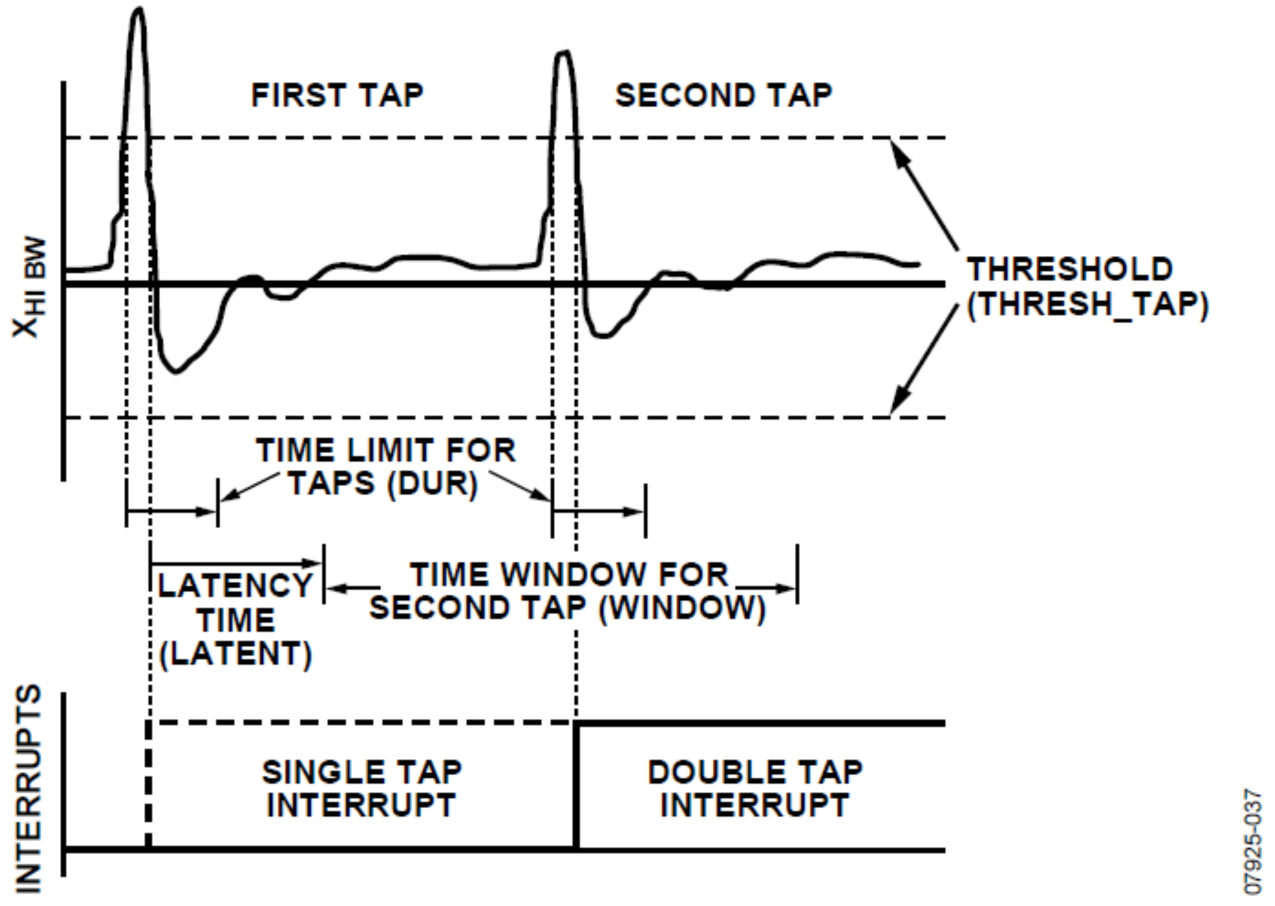


Figure 46. Tap Interrupt Function with Valid Single and Double Taps

Tap detection on the ADXL345 accelerometer

- Control registers:

THRESH_TAP

DUR

LATENT

WINDOW

TAP_AXES

ACT_TAP_STATUS

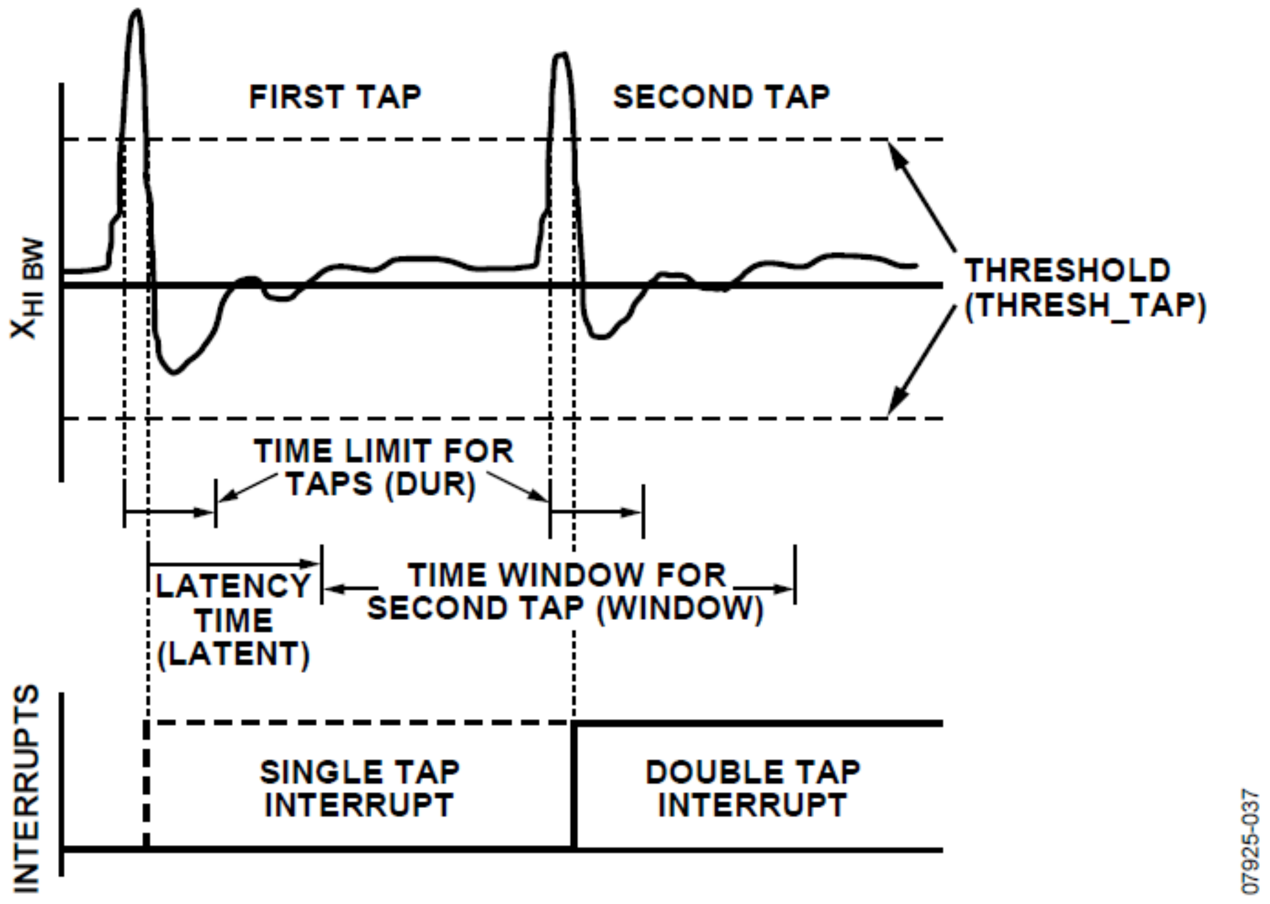
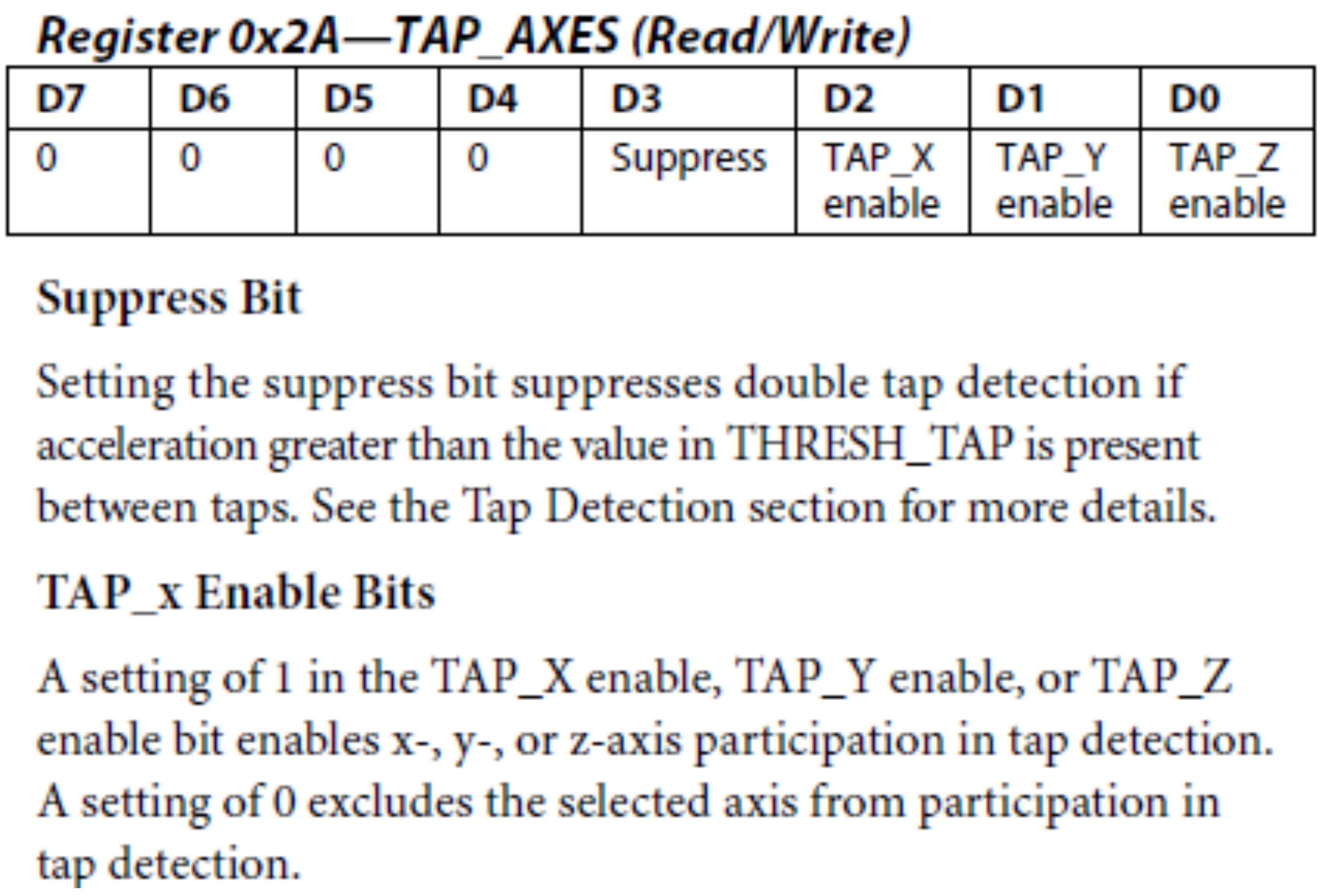


Figure 46. Tap Interrupt Function with Valid Single and Double Taps

Tap detection on the ADXL345 accelerometer



Register 0x2A—TAP_AXES (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	Suppress	TAP_X enable	TAP_Y enable	TAP_Z enable

Suppress Bit

Setting the suppress bit suppresses double tap detection if acceleration greater than the value in THRESH_TAP is present between taps. See the Tap Detection section for more details.

TAP_x Enable Bits

A setting of 1 in the TAP_X enable, TAP_Y enable, or TAP_Z enable bit enables x-, y-, or z-axis participation in tap detection. A setting of 0 excludes the selected axis from participation in tap detection.

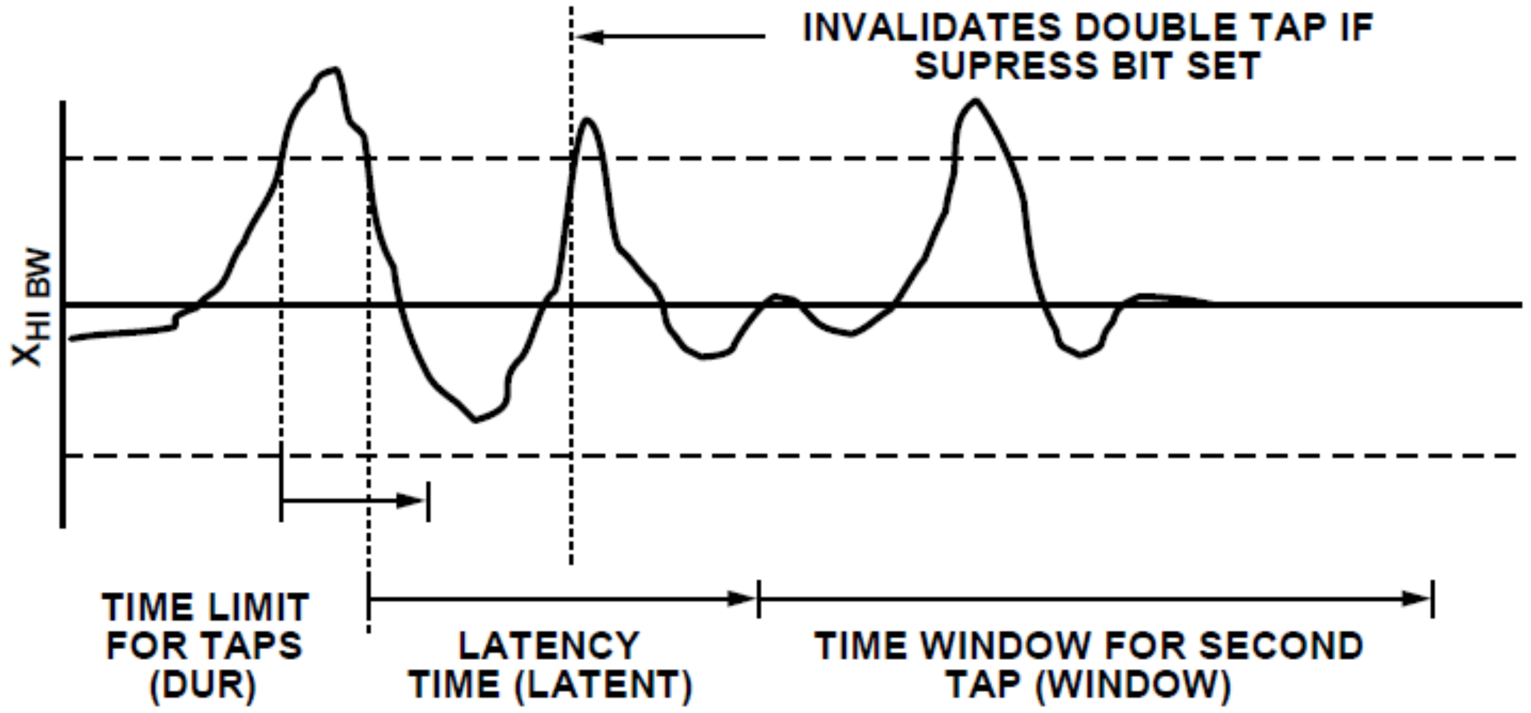

INVALIDATES DOUBLE TAP IF SUPRESS BIT SET
X_{HI} BW
TIME LIMIT FOR TAPS (DUR)
LATENCY TIME (LATENT)
TIME WINDOW FOR SECOND TAP (WINDOW)

Figure 47. Double Tap Event Invalid Due to High g Event When the Suppress Bit Is Set

Examples of invalid double-taps

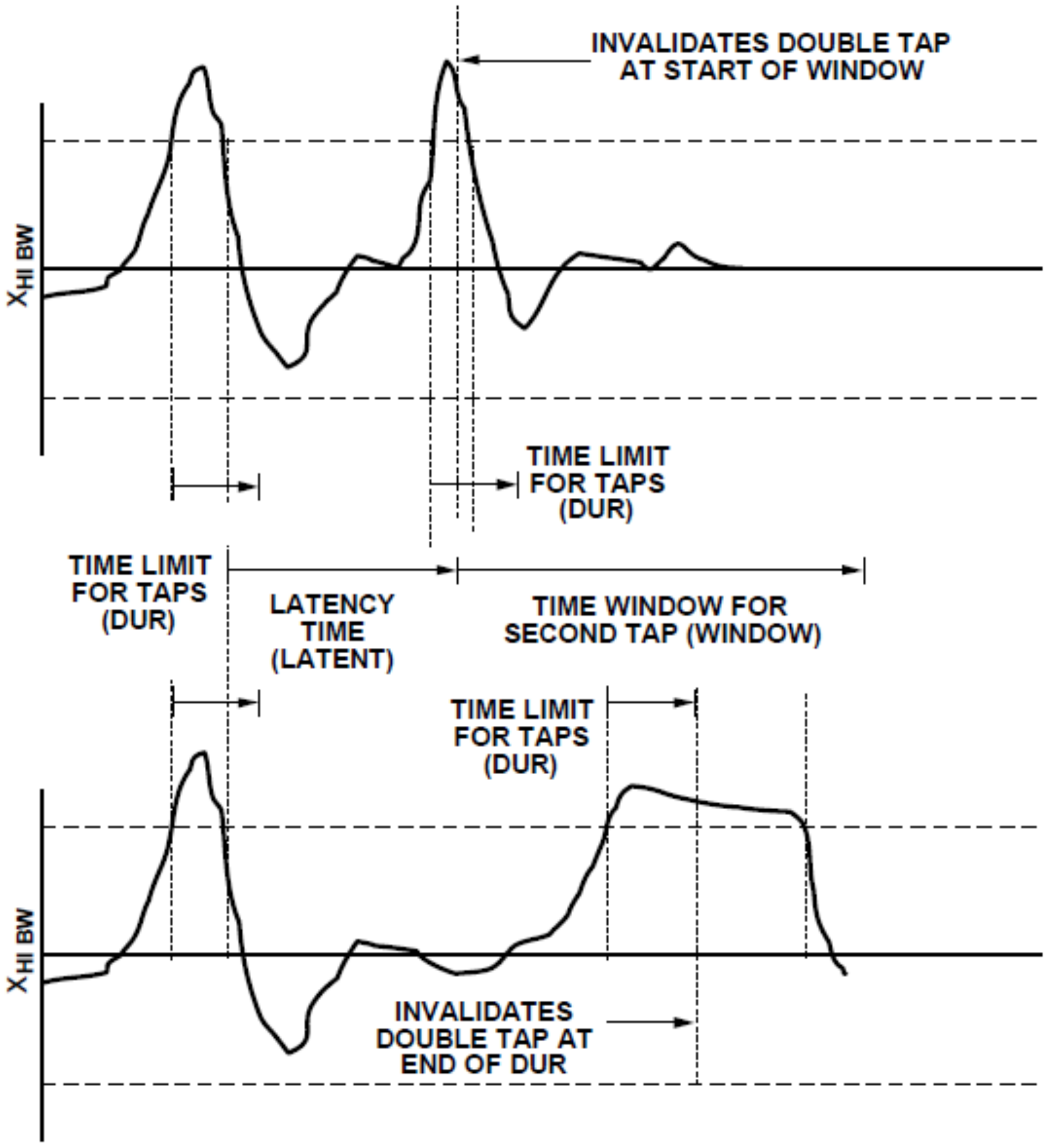
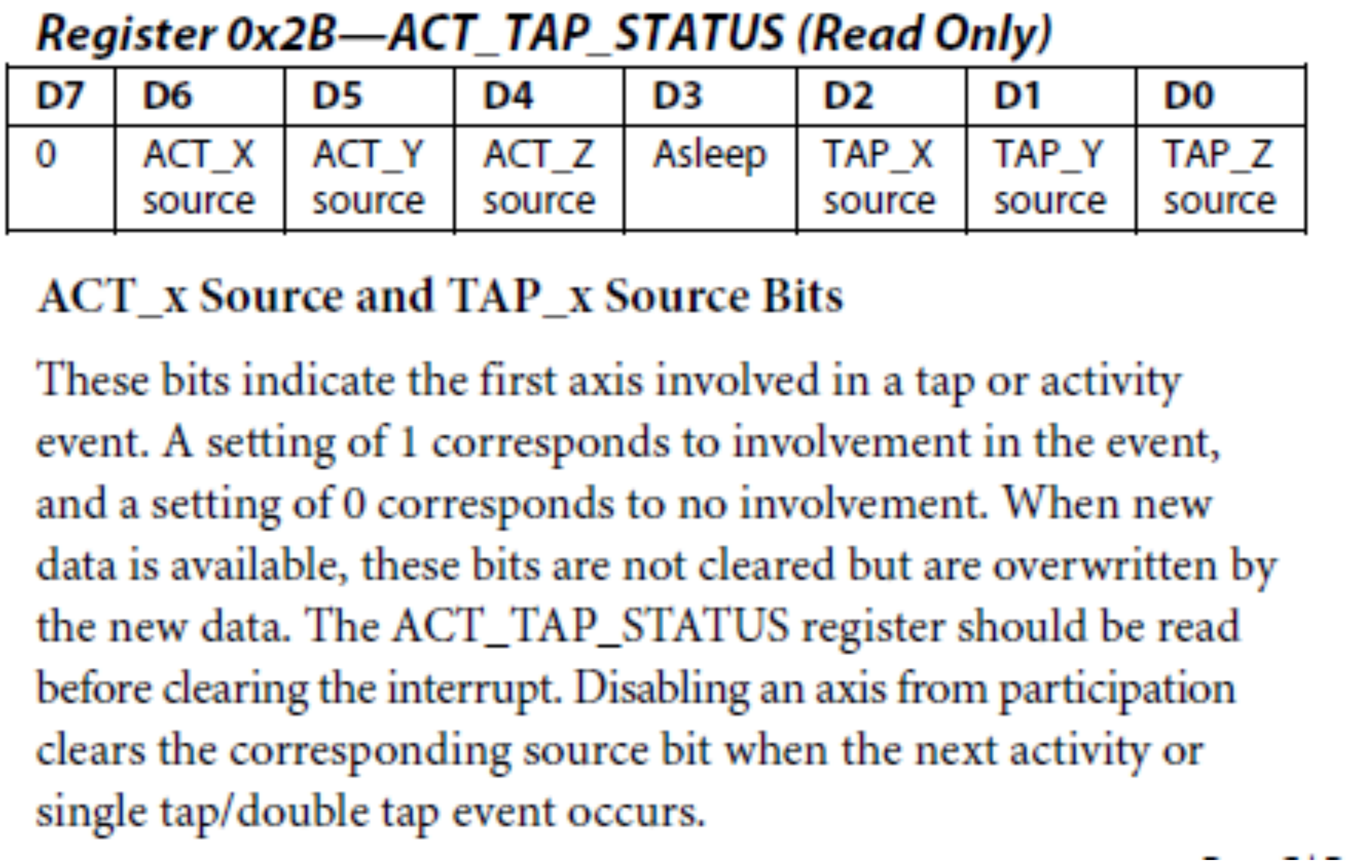

INVALIDATES DOUBLE TAP AT START OF WINDOW
X HI BW
TIME LIMIT FOR TAPS (DUR)
TIME LIMIT FOR TAPS (DUR)
LATENCY TIME (LATENT)
TIME WINDOW FOR SECOND TAP (WINDOW)
TIME LIMIT FOR TAPS (DUR)
X HI BW
INVALIDATES DOUBLE TAP AT END OF DUR

Figure 48. Tap Interrupt Function with Invalid Double Taps

Tap detection on the ADXL345 accelerometer



Register 0x2B—ACT_TAP_STATUS (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
0	ACT_X source	ACT_Y source	ACT_Z source	Asleep	TAP_X source	TAP_Y source	TAP_Z source

ACT_x Source and TAP_x Source Bits

These bits indicate the first axis involved in a tap or activity event. A setting of 1 corresponds to involvement in the event, and a setting of 0 corresponds to no involvement. When new data is available, these bits are not cleared but are overwritten by the new data. The ACT_TAP_STATUS register should be read before clearing the interrupt. Disabling an axis from participation clears the corresponding source bit when the next activity or single tap/double tap event occurs.

Putting it all together: Initialization

- Initialize the SPI or I^{2}C master on the PIC32, so that the PIC can communicate with the ADXL345
- Initialize the ADXL345 so that it will generate interrupts in response to desired events
  (Also may need to “clear” any pending interrupts on the ADXL345, as is done in the ISR)
- Make sure that the ADXL345’s INTx signal reaches the PIC32’s CNxx input
- Initialize the PIC32’s interrupt controller so that the PIC32 will respond when an interrupt occurs
  (Also may need to “clear” any pending interrupts in the interrupt controller, as is done in the ISR)
- Enable interrupts on the PIC32
- Enable the ADXL345, so that it begins taking measurements

Putting it all together: Operation

- The PIC32 begins normal operation (e.g., infinite loop w/ state machine)
- When an interrupt does occur, an ISR will begin to run on the PIC32
- The ISR should . . .
  - Verify the source of the interrupt
    → if (INTGetFlag (INT_CN)) . . .
  - Take (quick) action that depends on the application
  - "Clear" or "acknowledge" the interrupt on the peripheral, so that it is ready to generate the next interrupt
    (E.g., read a register such as INT_SOURCE on the ADXL345)
  - "Clear" or "acknowledge" the interrupt on the PIC32's interrupt controller, so that it is ready to respond to the next interrupt
    → INTClearFlag (INT_CN);
  - Return

Summary

- The ADXL345 accelerometer . . .
  - provides standard measurements of force;
  - also has special hardware for *tap detection;*
  - can generate interrupts for several different reasons
- The PIC32 microcontroller . . .
  - can detect interrupts from external devices through its *Change Notification* feature