KeyPad Component
1.0

Features

- From 2 x 2 up to 8 x 8 matrix – Keypads
- Fully debounced
- Key repetition with adjustable rate
- Optional translation table
- Adjustable buffer for keystrokes

General Description

The KeyPad Component allows an easy direct connection of a matrix keypad to the I/O-pins of a PSoC.

When to use a KeyPad Component

Use a KeyPad component when your design needs a keyboard as interface to the user. Any matrix-keypad can be connected directly to one or two ports of a PSoC 3 or 5

Input/Output Connections

ScanClock – Input

Every rising edge on this input triggers interrupt-driven a matrix read of the attached keyboard. The ScanClock should be provided at a frequency of 5 to 100 ms

KeyPressed – Output

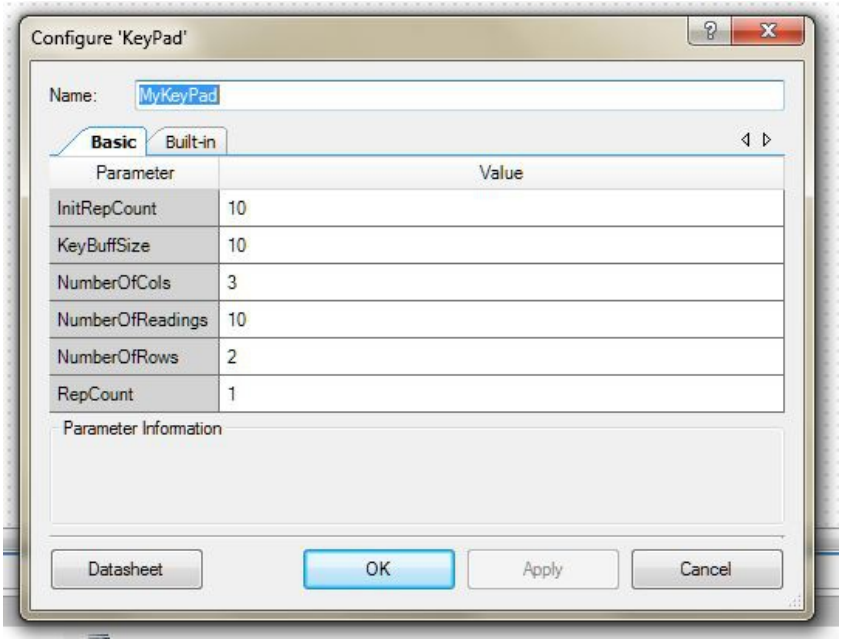
The KeyPressed output signals that a keystroke was successfully put into the buffer. The signal goes high for about one clock period of the ScanClock so when connected to an interrupt component its trigger must be set to “rising edge”.

I/O – Pins

The row- and column-lines of the keyboard are connected directly to GPIO-pins of the PSoC device. Both rows and columns I/O-pins are defined to be contiguous.

Component Parameters

Drag a KeyPad component onto your design and double-click on it to open the Configure dialog:



InitRepCount

Number of ScanClock cycles the component waits until a key-press is repeated the first time. Should be about 500 to 750ms

KeyBuffSize

Number of key-presses kept in the buffer (1 to 99).

NumberOfCols

Keypad columns to scan (2 to 8)

NumberOfReadings

Number of equal consecutive key-press-readings until a key-press is considered to be valid and debounced. Should be between 1 (no debounce) and 20.

NumberOfRows

Keypad rows to scan (2 to 8).

RepCount

ScanClock cycles between two repeated keys when a key is continuously pressed. Should be about 250 to 500 ms.

Application Programming Interface

The Keypad component has its internal interrupt triggered at each rising edge of the ScanClock. At this time the key matrix is scanned for a key-press or key-release. Only after a number of consecutive equal readings a key-press or key-release is valid and further processed. Valid keystrokes are put into a circular buffer of adjustable size and read off by the user with the help of some APIs. A key-value of 0xff (INVALIDCHAR) is not accepted and not stored in the keystroke-buffer and so can be used to define gaps in the translation-table.

Function	Description
KeyPad_Start()	Component initialization
KeyPad_IsKeyReady()	Checks for key-presses in the buffer
KeyPad_GetChar()	Get a key-press from buffer
KeyPad_Sleep()	Prepares for sleep-mode
KeyPad_Wakeup()	Power up component from sleep-mode

void KeyPad_Start(char * TranslationTable)

Description: Sets up the component. Global interrupts must be enabled for this component to function.

Parameters: char * TranslationTable. A key-press returns a number between 0 and the number of keys. If another output for any key is required a translation-table can be used. When this parameter is NULL no translation is performed.

Return Value: None

Side Effects: Noneuint8 KeyPad_IsKeyReady(void)

Description: Checks if at least one valid key-press is present in the key-buffer.

Parameters: None

Return Value: Returns a non-zero value when a key-press is present.

Side Effects: None

uint8 KeyPad_GetChar()

Description: Retrieves a key-press

Parameters: None

Return Value: This function returns the key-press from the buffer. When there is no key-press ready the function waits (blocking) until a key is pressed.

Side Effects: Blocks program execution until there is a key-press in the buffer.

void KeyPad_Sleep(void)

Description: Prepares the component for entering the sleep-mode .

Parameters: None

Return Value: None

Side Effects: Powers down the I/O – Pins and disables the KeyPad-internal interrupt.

void KeyPad_Wakeup(void)

Description: Prepares the component for normal work after exiting from sleep-mode.

Parameters: None

Return Value: None

Side Effects: Interrupt and I/O – Pins are setup for normal work.

Component Internals

The way this component operates was inspired by an article from The Ganssle Group and I decided to write a most universal but safe component for PSoC 3 and PSoC 5. Here I will unveil some of the internals the component works with.

Variables

A circular buffer is allocated for capturing keystrokes together with the required indexes.

#defines

In the corresponding .h file are definitions for the calculation of the required size for the translation-table (MAXKEYVALUE).
Another definition is for INVALIDCHAR which will not be accepted as a valid keystroke and as such not entered into the buffer.

Principals of Operation[1]

As you may have already seen this component is mostly built with software. A user supplied clock (ScanClock) generates in equal time-slices an interrupt which triggers the scanning of the keyboard matrix.

Matrix Scanning

In nearly 100% of the time there is no key-press to detect, so that branch of the interrupt-routine is kept short. The row-pins are defined as resistive pulled up while to the column-lines is written a "0". When no key is pressed the reading of the row-lines will now yield "ones". When any key is pressed the non-ones value is saved and some further investigations find out which of the column-lines is affected. The affected bit-positions are converted to two binary numbers and then concatenated to form a number in the range of

1 In the '70 I got my education on an IBM 360/370 machine with a brand new operating-system named TSS (for Time-Sharing-System). The OS manuals were named "Principals of Operation" and I started to like that name, so I am using it here.

the number of keys present. An example:
Let us assume a 3 x 4 keypad, so we have 3 rows and 4 columns. When the middle row-key is pressed we read a 3'b101 for the row and bit number 1 is affected (when counting from left-to-right starting with 0). Let us further assume the column-lines give us a 4'b1011 where now the bit number 2 is seen. Out of that is formed a binary number of the form 4'bRRCC where "R" are the row-bits and "C" the column-bits. This would give in our case a 4'b0110. So the key at the cross-point of row 1 and column 2 will give the key-number 6.

Possible Pitfall

There is a pitfall when using the KeyPad that must be avoided: To express the four bits of the column in the above example as a binary number exactly two bits are needed. The same is for the three rows, but one code (1'b11) is not used. When we exchange the rows and columns (a 4 x3 keypad) we will have "missing codes" and the numbering of the keys is no longer consecutive. This too will come into account when we use a translation-table. Although we have got only 12 keys the translation-table must have 16 entries.

Debouncing

During the period a key bounces there are different matrix-readings. The key is considered to be debounced when a number (Parameter: NumberOfReadings) of consecutive equal readings have occurred. So it is obvious that every detected bounce would start the counting anew. This algorithm has the advantage over other methods that the time to react to a valid key-press can be kept relatively short even when aging processes worsen the specs of the switch. When the component is configured appropriately even induced glitches can be calculated-out. Due to the (relatively) low input impedance and the robust algorithm used the KeyPad component gives the designer a save and sound tool to connect his interface to the PSoC.

Key-Repetition

Simple counting of the interrupts while a key is pressed and not released is used to implement a key-repetition with an adaptable initial dead-time and a key-repetition rate.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
V1.0	Initial release	