THE LAB MUST BE SUBMITTED TO BLACKBOARD BY 4PM TODAY!

University of Windsor

ELECTRICAL AND COMPUTER ENGINEERING
ELEC3270: MICROPROCESSORS

ASSIGNMENT 10: PULSE ACCUMULATION AND REAL-TIME INTERRUPTS

OBJECTIVE: To understand how the pulse accumulator and real-time interrupts features operate.

SECTIONS:

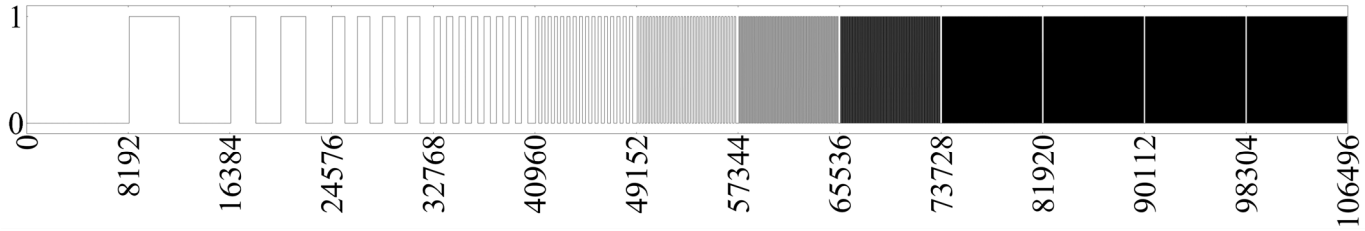
A. Pulse Accumulator (Event Counting Mode) (50%)
B. Pulse Accumulator (Gated Time Accumulation Mode) (50%)

A. PULSE ACCUMULATOR (EVENT COUNTING MODE)

The 68HC11 is equipped with a single pulse accumulator (PA) system. It offers two modes of counting; for this part we will explore the "event counting mode". In general, this mode counts the number of times an external signal changes from 0 to 1 or from 1 to 0. Although this seems like a simple operation, performing this using a polling technique would easily lead to missed pulses. This hardware feature is very accurate as it will never miss the transition of the signal. The register used to store the number of pulses counted, PACNT, is an 8-bit register. The system can therefore only count up to 255 pulses before the counter overflows. This is acceptable as we do not generally expect the pulse width of the signal to be as small as the source clock; it is usually much larger. It is reasonable to assume that the MPU will be able to handle an OVERFLOW interrupt every 256 pulses so it can count more than 256 pulses, as such it has one (PAOV). The 68HC11's PA also includes the ability to generate an interrupt each time a pulse is detected (PAI); this offers finer resolution but the system may not be able to keep up if the pulse period is too small. Unlike the input captures registers, the PACNT register is writable which allows us to set the initial value of this counter. We can therefore select the number of pulses we would like to see prior to an overflow interrupt and completely avoid using an interrupt for the detection of each individual pulse. For example, if we wanted to wait for 100 pulses, we could set PACNT to 156 (256-100) and wait for the overflow interrupt rather than actually counting the pulses through 100 separate interrupts. The PA is connected to PORTA, bit 7 (PA7) which can be either an input or an output. When set as an **output** we can control the input to the PA directly from the microcontroller (with OC1 or by writing to PA7). We can therefore generate specific timed pulses ourselves; which can be useful in some applications. We will however use it in **input** mode with an external source. THRSim11 allows us to supply our own custom signal patterns to any pin on the device by means of a "bit inserter". Open the supplied assignment ZIP file and extract the "**assign_10a.txt**", "**assign_10b.txt**", and "**SID_10a.asm**" file; open the latter file in THRSim11. You will need to connect the "bit inserter" from the "Connect", "Inserters", "Bit Inserter" menu. Select "PA7", press "Connect", press "Connect" again (it should have automatically select "number of cycles"), and then "OK". Right click in the bit inserter window, and select "Maximum Amount...", and enter 10000. Right click again in the bit inserter window, and select "Load...", and pick the "**assign_10a.txt**" file included in the assignment. Right click one more time in the bit inserter window, and uncheck "Repeat". The bit inserter window should be showing a very long list with 2 columns; the first column being the cycle number, and the next being the input to the pin. When the cycle number of the simulation is equal to the number in the left column, it will set the input pin to the value in the right column. If you examine this file, you will notice is has a period of 8192 cycles, of which the frequency starts with a period of 8192 cycles, but halves every 8192 cycles; the duty cycle is always 50% (see the figure on the top of the next page).

The task of this program is to determine the number of rising edge transitions every 8192 cycles using only the MCU timing hardware. If you scroll to the bottom of the bit inserter window, you will see the

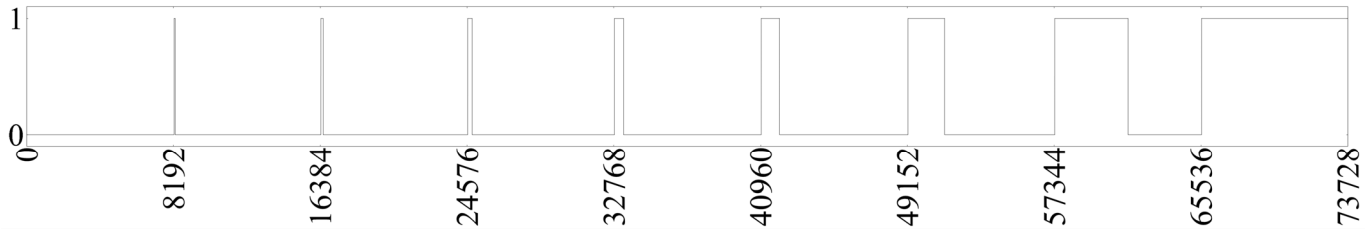
inputs to PA7 are changing almost every 2 cycles; this is far too fast for any software solution to process. The expected output should be 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, and then 0 as the pattern stops. Essentially we will read the PACNT register every 8192 cycles.



Since this is a very specific timing operation, it makes sense to use the timing hardware. Although it is possible to setup an OC operation to occur every 8192 cycles, it would be easier to the Real-Time interrupt for this task as it can generate interrupts every 8192, 16384, 32768, or 65536 cycles (which can be changed any time). So when the real-time interrupt occurs every 8192 cycles, the program will read the PACNT value, and store it in a mailbox, and flag it. The main routine will then print the value to the LCD. There is still a minor issue as PACNT can only count up to 255. We need to use the PAOV interrupt to improve the resolution of the timer by incrementing an additional 8-bit value (PACNTH). This is very similar to what we did in the previous lab. Follow the comments to setup the initial values of the MCU registers (PACTL & TMSK2), and add the appropriate code to the individual ISRs. Note that the output of the program will be in hexadecimal, so the values should appear as: 0000, 0001, 0002, 0004, 0008, 0010, 0020, etc. up to 0800.

B. Pulse Accumulator (Gated Time Accumulation Mode)

As noted earlier, the 68HC11 has two PA modes, the other mode ("gated time accumulation mode") is for counting the number of times PA7 is high or low every 64 cycles; this is useful for decoding pulse width modulation (PWM) signals. For this part you don't have to substantially change your code at all as all we are still going to read the results of PACNT (along with the higher resolution value) every 8192 cycles and display them. The only necessary change is the initial value of PACTL (iPACTL in the code) which must be set to the different counting mode; see the code comments. To begin, copy your "**SID_10a.asm**" file to "**SID_10b.asm**", and open it in THRSim11. Before running you simulation we need to change the input pattern to PA7 as it now is a PWM signal. Right click in bit inserter window, and select "Load...", and pick the "**assign_10b.txt**" file included in the assignment. This file has far fewer rows as the duration of the stable signal is important, not the number of transitions. If you examine it you will see the point of transition from 1 to 0 doubles from the 0 to 1 point every 8192 cycles (see the figure below).



For this part of the assignment, the PA hardware will measure how often the signal is **high** every 64 cycles. The expected output should be 0, 1, 2, 4, 8, 16, 32, 64, 128, and then 0 as the pattern stops. We can't go any higher than this as 8192/64 = 128. Unfortunately PWM signals don't provide a lot of resolution themselves so it is difficult to encode large numbers unless the pulse period is very long, but that limits the potential rate of change. Again, note that the output of the program will be in hexadecimal, so the values should appear as: 0000, 0001, 0002, etc. up to 0080.

Note that both of your files will be simulated with different input patterns to ensure their proper function. ZIP all your ASM files into "SID_10.zip" and upload it to blackboard before the end of the lab which is at 4pm. <u>Any late submissions will not be accepted.</u>