CG2028

LAB 1: FAMILIARIZATION WITH HARDWARE AND DEVELOPMENT ENVIRONMENT

Objectives

1. Become familiar with LPCXpresso Integrated Development Environment (IDE)
2. Build, download and execute the following programs on NXP LPC1769 ARM Cortex-M3 system on chip on LPCXpresso board
    1. Simple assembly language program
    2. C program

Reference Materials

Can be found in Luminus under the Reference folder

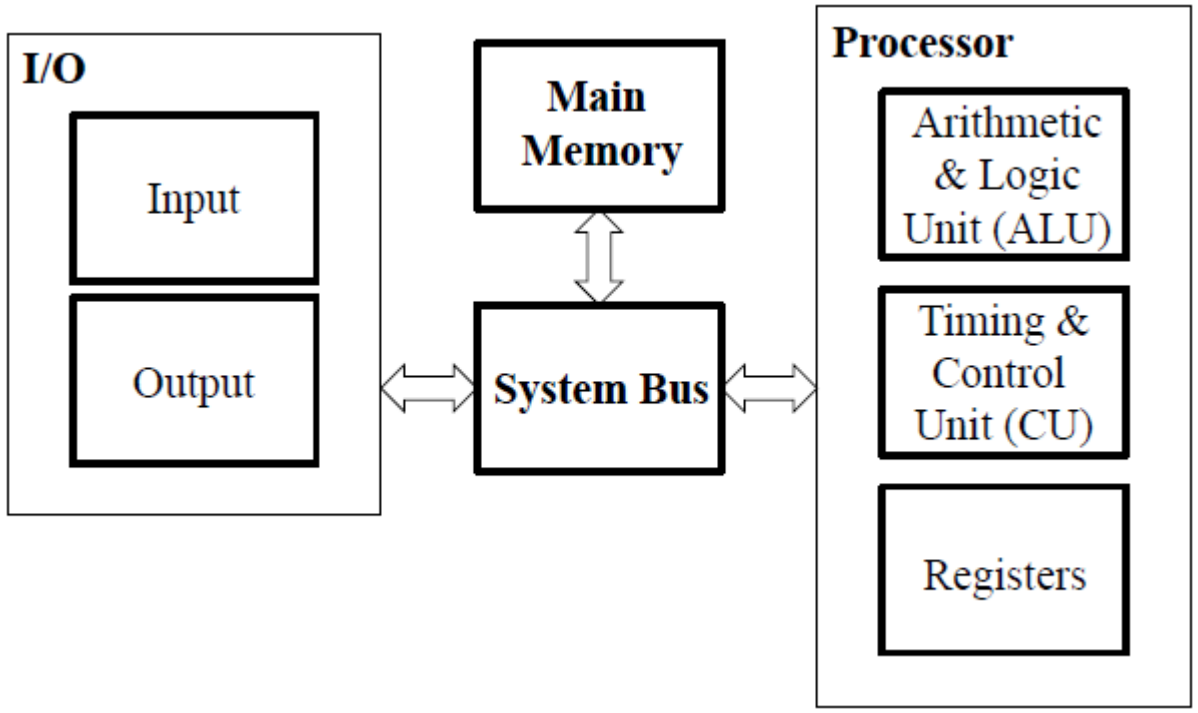
1. LPCXpresso User Guide [LUG]

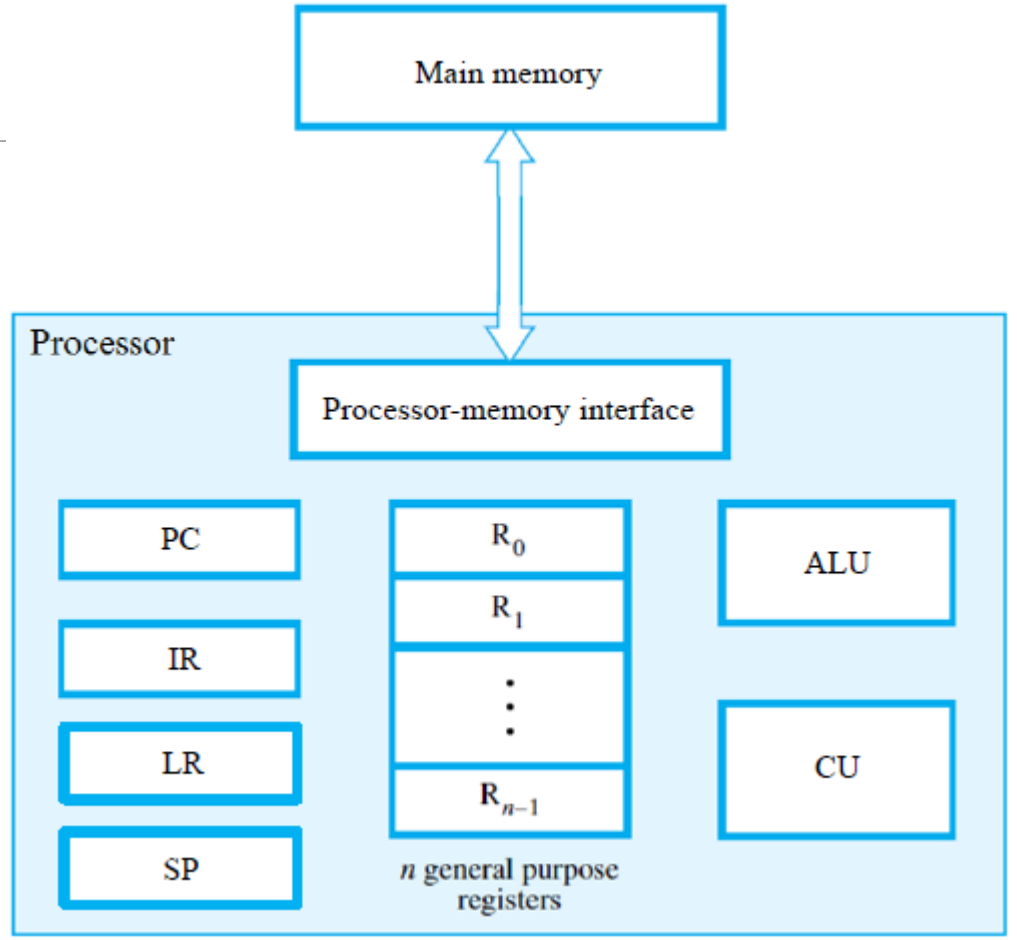
2. NXP UM10360 LPCxx User Manual

Lecture Materials

TAKEN FROM THE LECTURE MATERIALS PROVIDED IN LUMINUS

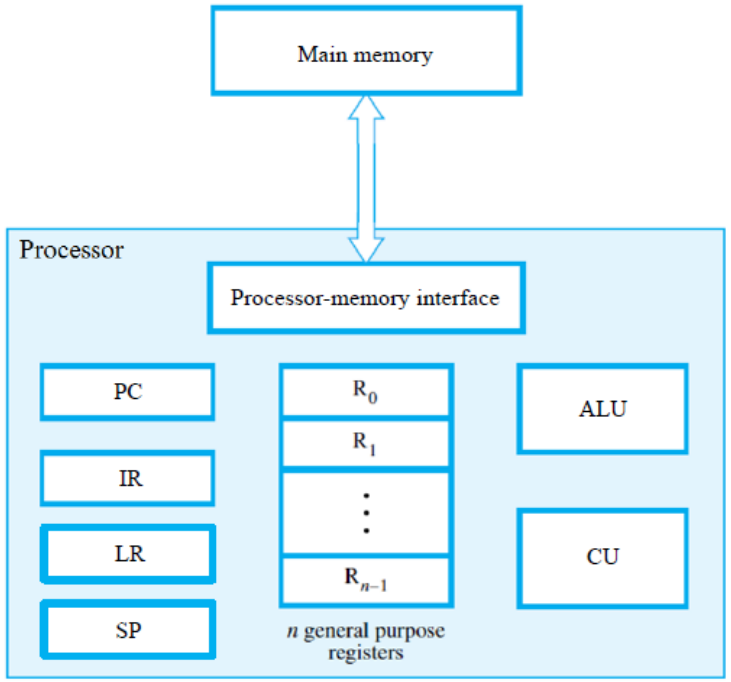
Functional Units





Processor Components in ARM Cortex-M3

- ALU: performs arithmetic & logical operations
- CU: fetch program instructions and data from memory
- Registers (32 bits)
  - General purpose low registers (R0 –R7) – to hold data and addresses
  - General purpose high registers (R8 – R12) – same as above except for interrupt cases
  - Stack Pointer or SP (R13) – hold memory address of last (most recent data)
  - Link Register or LR (R14) – stores return address when subroutine/ function is called
  - Program Counter or PC (R15) – current program address
  - Instruction Register or IR – hold current instruction (not shown)



Instructions and Sequencing

- Data transfer to & from memory, e.g. load (LDR), store (STR)
- Arithmetic & logical operations on data, e.g. ADD, OR
- Program sequencing and control, e.g. branch, loop

- Fetch Instruction – read instruction from memory
- Decode Instruction – determine action required
- Fetch data – read data from memory
- Process data – perform arithmetic/ logical operations
- Write data – storing results to memory

Example: C = A + B

	Memory
0x40	Load R2, A
0x44	Load R3, B
0x48	Add R4, R2, R3
0x4C	Store R4, C
	...
A	2
B	3
C	

Fetch

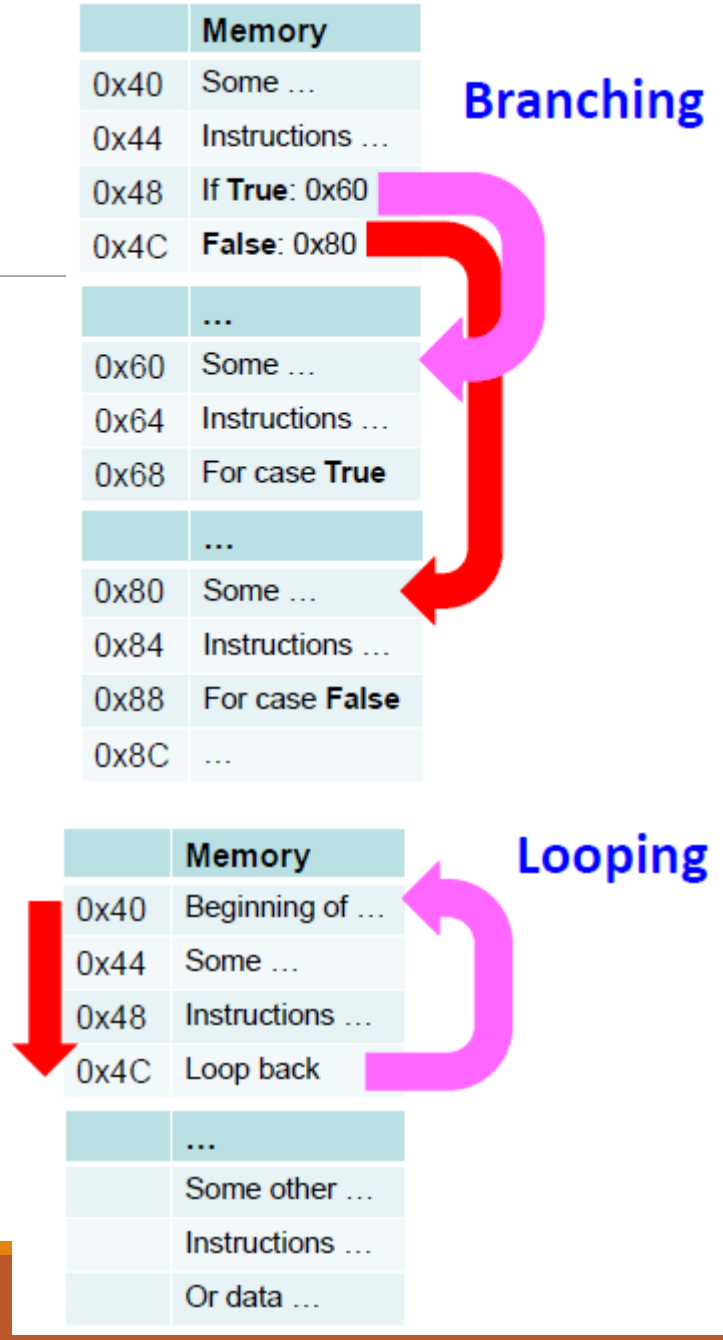
	Registers
R2	2
R3	3
R4	5
PC	0x4C
IR	Store R4, C

Execute

	Registers
R2	2
R3	3
R4	5
PC	0x4F
IR	Store R4, C

Branching & Looping

- To execute another program based on certain condition
- Condition code flags in a status register:
  - N (negative) 1 if result is negative, else 0
  - Z (zero) 1 if result is zero, else 0
  - V (overflow) 1 if overflow occurs, else 0 (for signed int)
  - C (carry) 1 if carry occurs, else 0 (for unsined int)



Calling Assembler from a C Program

- extern int asm_func (int x, int y);
- Input parameters: R0, R1, R2, R3 (max 4)
- Output parameter: R0
- Ends with BX LR

ARM basics

i. Label – optional to represent a memory address [label: opcode]
   - For accessing instruction in memory & when doing branching/ looping/ subroutine calls
   - Accessing data in memory as a variable, e.g. A, B, C, NUM1, etc
   - To represent a defined value, e.g. RADIUS: .word 100

ii. Options - <> alternative forms of operand, {} enclose optional operands

iii. Op2 – flexible second operand that is either a register (Rm {, shift}) or a constant (#constant)

iv. #immediate - #imm8 (0-255), #imm12 (0-4095), #imm16 (0-65535)

v. Pre- & Suffix
   - S: perform signed operation; D: perform unsigned operation
   - Op{S}: updates condition code flags – N, Z, V, C

Memory Addressing

1. Allocation for Data
   - Constants (using .word) – e.g. NUM1: .word 123
   - Static Variables (using .lcomm) – e.g. .lcomm ANSWER 4 @ reserves 4 bytes
2. Load/ Store: Offset Addressing
   - LDR/STR Rt, [Rn {, #offset}]
3. Load: PC-Relative Addressing (only for LDR)
   - LDR Rd, [PC + offset]
4. Load: Pseudo-Instruction
   - LDR R3, = 0xA123B456 (programmer: instruction)
   - LDR R3, MEMLOC (assembler: instruction)
   - MEMLOC: .word 0xA123B456 (assembler: data)

ARM Instructions

Instruction	Format	Example	Representation
MOV	MOV Rd, Op2	MOV Rd, Rm	Rm → Rd
MOVW	MOVW Rd, #imm16	MOVW Rd, #value	value → Rd
ADD	ADD {Rd,} Rn, Op2	ADD R0, R2, R4	R2 + R4 → R0
SUB	SUB {Rd,} Rn, #imm12	SUB R0, R3, #17	R3 – 17 → R0
MUL	MUL {Rd,} Rn, Rm	MUL R0, R1, R2	R1 × R2 → R0
MLA	MLA Rd, Rn, Rm, Ra	MLA R0, R4, R5, R6	(R4 × Rd) + R6 → R0
MLS	MLS Rd, Rn, Rm, Ra	MLA R0, R4, R5, R6	(R4 × Rd) – R6 → R0
AND, ORR, EOR	Op{S} {Rd,} Rn, Op2	ANDS Rd, Rn, Rm	Bit-wise AND of the Rn & Rm into Rd
MVN	MVN Rd, Op2	MVNS Rd, Rn	Bit-wise NOT on Rn into Rd
BIC, ORN	Op{S} {Rd,} Rn, Op2	BIC R0, R1	Bit-wise AND with complement of R1
LSL, LSR, ASR, ROR	Op{S} Rd, Rm, #n	LSL R0, R1, #2	R1 shifted to the left by adding 2'b00
RRX	RRX Rd, Rm	RRX R0, R1	R1 rotate starting with a carry bit (33 bit)
CMP, CMN	Op{S} Rn, Op2	CMP R0, R1	(R0 – R1) and check for condition flags
TST, TEQ	Op{S} Rn, Op2	TST R3, #1	Bit-wise AND, set Z=1 if LSB=0, else 0
B, BL, BX, BLX	B{cond} label	B {EQ} LOCATION	Branches to label LOCATION when Z =1

Table A.2 Condition Code Suffixes

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 or N != V	Less than or equal, signed ≤
AL	Can have any value	Always; default when no suffix is specified

Offset Addressing

Pre-indexing

- LDR Rd, [Rn, #offset]!

First:

- [Rn +offset] → Rd

Followed by:

- Rn +offset → Rn

Post-indexing

- LDR Rd, [Rn], #offset

First:

- [Rn] → Rd

Followed by:

- Rn +offset → Rn

Stack: efficient memory usage model specified by SP

4 implementations

- Empty: points to next free location
- Full: points to most recent item
- Ascending: starts from low memory address
- Descending: starts from high memory address

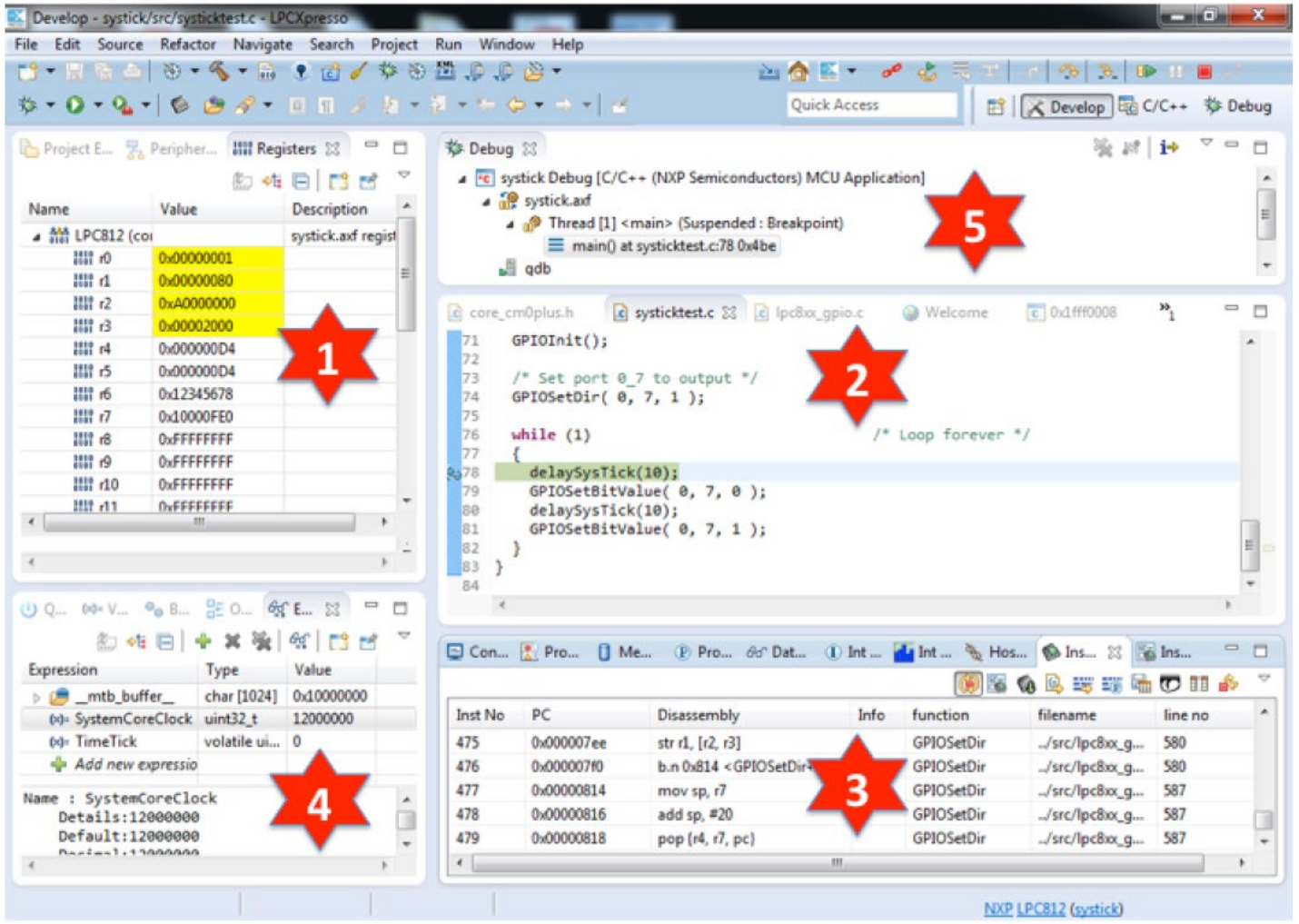
PUSH	POP
PUSH reglist	POP reglist
e.g. PUSH {R0}	e.g. POP {R0}
R13 – 4 → R13	Memory [R13] → R0
R0 → Memory [R13]	R13 + 4 → R13

Lab 1

LPCXPRESSO IDE

Instructions

1. Create a folder called “CG2028workspace” on your thumbdrive
2. Copy the file “CG2028 Part 1.zip” into this folder
3. Start up LXIDE
4. Locate “CG2028workspace” for the path to your workspace

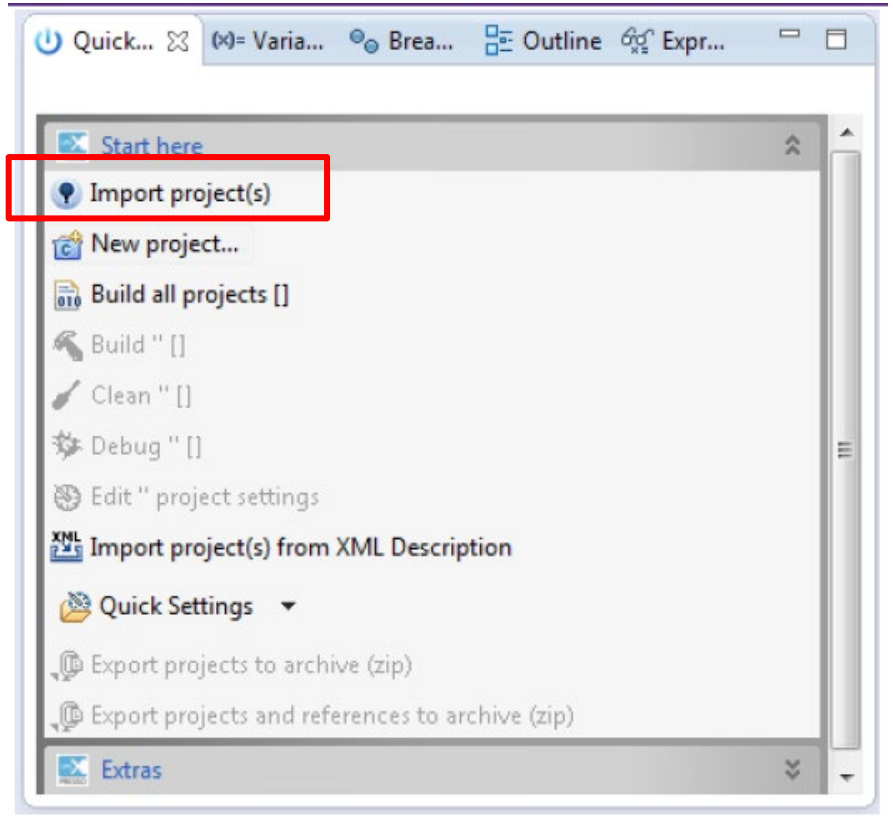


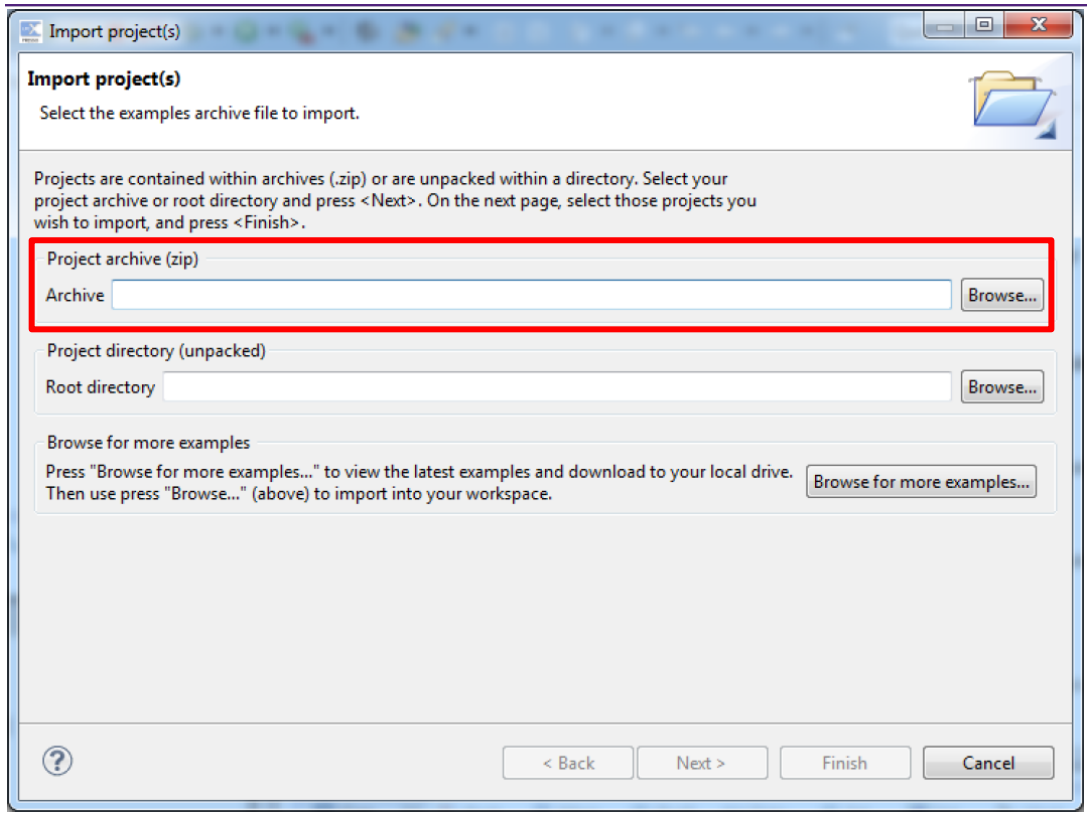
Major Components of the Develop Perspective

1. Project Explorer/ Peripherals/ Registers
2. Editor
3. Console/ Problems/ Memory
4. Quick Start/ Variables/ Breakpoints/ Expressions Views
5. Debug View

Importing and Debugging example Projects

Step-by-step instructions are provided from page 15 – 24 of [LUG]





Importing and Debugging example Projects

Step-by-step instructions are provided from page 15 – 24 of [LUG]